Nuorium Optimizer

マニュアル
V27

株式会社NTTデータ数理システム

2024年12月

MATHEMATICAL SYSTEMS INCARNATION

目　次

第 1 章 マニュアル紹介

1.1 マニュアルラインアップ紹介

Nuorium Optimizer には用途に応じて以下のマニュアルが準備されております.

1. **Nuorium Optimizer マニュアル**
   Nuorium Optimizer の詳細機能が説明されています. 読者にはある程度 Nuorium Optimizer に馴染みがあることを想定しております.
2. **Nuorium Optimizer/PySIMPLE マニュアル**
   Nuorium Optimizer 付属の Python ベースのモデリング言語 PySIMPLE のマニュアルです.
3. **Nuorium Optimizer/C++SIMPLE マニュアル**
   Nuorium Optimizer 付属の C++ベースのモデリング言語 SIMPLE のマニュアルです.
4. **Nuorium Optimizer/RSIMPLE マニュアル**
   Nuorium Optimizer 付属の R ベースのモデリング言語 RSIMPLE のマニュアルです.
5. **Nuorium Optimizer/C++SIMPLE 例題集**
   典型的な数理最適化問題に対する Nuorium Optimizer/C++SIMPLE の記述方法が記されています. 具体的な問題を手本として学ぶのに最適なマニュアルです.
6. **Nuorium Optimizer/SIMPLE チュートリアル**
   初めて Nuorium Optimizer をご利用の方におすすめのチュートリアルです.
7. **Nuorium スタートガイド**
   初めて数理最適化専用 GUI Nuorium をご利用の方におすすめのスタートガイドです.
8. **Nuorium Optimizer/Excel アドインマニュアル**
   Nuorium Optimizer と Microsoft Excel との連係機能である Excel アドインの詳細機能が説明されています.
9. **Nuorium Optimizer/C++SIMPLE 外部接続マニュアル**
   外部のプログラムから Nuorium Optimizer を呼び出して利用する方法が説明されています.

一部マニュアルに関してはオンラインマニュアルを提供しております. オンラインマニュアルは https://www.msi.co.jp/solution/nuopt/docs/index.html からご覧ください.

1.2 本マニュアルの構成

本マニュアルは以下のような内容から構成されています.

2〜6:求解ソルバ Nuorium Optimizer の解説

7:高度な利用法
付録:エラーメッセージ・アルゴリズムの解説，参考文献の紹介

第2章 標準出力

数理最適化問題が Nuorium Optimizer で解かれた場合，最終的な解情報の一部が標準出力に出力されます．以下はその一例です．

```
[Problem and Algorithm]
PROBLEM_NAME                                          a
NUMBER_OF_VARIABLES                                   5
NUMBER_OF_FUNCTIONS                                   2
PROBLEM_TYPE                                  MAXIMIZATION
METHOD                                        HIGHER_ORDER

[Progress]
<preprocess begin>.........<preprocess end>
<iteration begin>
    res=7.3e+001 .... 4.1e-003 .. 6.4e-010
<iteration end>

[Result]
STATUS                                             OPTIMAL
VALUE_OF_OBJECTIVE                                    1049
ITERATION_COUNT                                          8
FUNC_EVAL_COUNT                                         11
FACTORIZATION_COUNT                                      9
RESIDUAL                                  6.352800465e-010
ELAPSED_TIME(sec.)                                    0.01
SOLUTION_FILE                                         a.sol
```

2.1 アルゴリズム共通の出力

[Problem and Algorithm] で始まるセクションでは，以下のように問題の概要が出力されます．

```
PROBLEM_NAME                                          a
NUMBER_OF_VARIABLES                                   5
NUMBER_OF_FUNCTIONS                                   2
```

```
PROBLEM_TYPE                                    MAXIMIZATION
METHOD                                          HIGHER_ORDER
```

PROBLEM_NAME は「扱うモデルのファイル名」です．この例では a というモデルを解いています．

NUMBER_OF_VARIABLES は「変数 Variable の数」です．この例では変数が 5 個あります．

NUMBER_OF_FUNCTIONS は「関数の数」です．ここで言う関数とは，目的関数 Objective と制約式 Constraint を合わせたものになります．この例では，関数が 2 個（目的関数 1 個，制約式 1 個）あります．

PROBLEM_TYPE は問題が最小化問題（MINIMIZATION）なのか最大化問題（MAXIMIZATION）なのかを表示します．

METHOD は「最適化計算に用いたアルゴリズムの種類」です．この例では線形計画専用内点法（HIGHER_ORDER）を用いています．

[Result] で始まるセクションでは，以下のように最適化計算結果の要約が出力されます．

```
STATUS                                        OPTIMAL
VALUE_OF_OBJECTIVE                               1049
ITERATION_COUNT                                     8
FUNC_EVAL_COUNT                                    11
FACTORIZATION_COUNT                                 9
RESIDUAL                            6.352800465e-010
ELAPSED_TIME(sec.)                              0.01
SOLUTION_FILE                                  a.sol
```

最適化計算結果の要約の詳細については 2.10 をご参考ください．

2.2 内点法における出力

内点法を用いたアルゴリズムでは，[Progress] で始まるセクションに以下のような実行経過が出力されます．

```
<preprocess begin>.........<preprocess end>
<iteration begin>
    res=7.3e+001 .... 4.1e-003 .. 6.4e-010
<iteration end>
```

<preprocess begin>と<preprocess end>の間は収束計算に入る前の処理の進行を，<iteration begin>と<iteration end>の間は収束計算の進行を示しています．計算の進行中に表示される数字（7.3e+001, 4.1e-003 など）は最適性条件の残差で，この表示はそれが計算の進行とともに減少していく様子を示しています．

2.3 単体法（simplex），有効制約法，クロスオーバーにおける出力

単体法（hsimplex は除く），有効制約法，クロスオーバーを用いた場合には，[Progress] で始まるセクションに以下のような実行経過が出力されます．

```
<preprocess begin>.........<preprocess end>
<iteration begin>
      ...1...................2
<iteration end>
```

<preprocess begin>と<preprocess end>の間は単体法の反復に入る前の処理の進行を示しています．

<iteration begin>と<iteration end>の間にあるドットは単体法の反復の進行を示しています（1 つのドットにつき，数回の反復を示しています）．また，文字 1 は実行可能解を探索するフェーズに遷移したことを示し，文字 2 は最適解を探索するフェーズへの遷移を示しています．

線形計画法，二次計画法に対して内点法からのクロスオーバー（options.crossover="on"）を指定した場合には

```
<preprocess begin>.........<preprocess end>
<iteration begin>
    res=2.4e+001 .... 2.7e-005  1.4e-008
<iteration end>
<iteration begin>
        1222
<iteration end>
```

のように，内点法の経過表示の後に単体法の経過表示が現れます．

2.4 単体法（hsimplex）における出力

単体法（hsimplex）により線形計画問題を解く場合には，[Progress] で始まるセクションに以下のような実行経過が出力されます．

```
[Progress]
dual-phase1 start
    Iter.       Objective    Primal Inf.      Dual Inf.  Time(s)
       1  -1.130341e+04   1.104730e+05   0.000000e+00       0.0
      38   0.000000e+00   0.000000e+00   0.000000e+00       0.0
dual-phase2 start
      39  -2.564421e+03   6.472308e+04   0.000000e+00       0.0
      89  -7.870890e+02   1.608960e+04   0.000000e+00       0.0
     151   1.368297e+03   2.325698e+03   0.000000e+00       0.0
```

```
      236   1.487996e+03   5.042927e+01   0.000000e+00      0.0
      295   2.691577e+03   0.000000e+00   0.000000e+00      0.0
cleanup perturbation
      296   2.690013e+03   0.000000e+00   0.000000e+00      0.0
```

また，二次計画問題を解く場合には，[Progress] で始まるセクションに以下のような実行経過が出力されます．

```
[Progress]
primal-phase1 start
    Iter.      Objective    Primal Inf.      Dual Inf.  Time(s)
       1   0.000000e+00   5.748232e+04   6.502000e+01      0.0
      51   1.628902e+04   9.463961e+03   3.889075e+01      0.0
     101   8.832791e+01   8.832791e+01   1.276675e+00      0.0
     105   0.000000e+00   0.000000e+00   0.000000e+00      0.0
cleanup perturbation
     106   0.000000e+00   0.000000e+00   0.000000e+00      0.0
primal-phase2 start
    Iter.      Objective    Primal Inf.      Dual Inf.  Time(s)  Freedom
     106   6.399420e+07   0.000000e+00   4.043570e+05      0.0        0
     156   2.687252e+07   0.000000e+00   1.284500e+01      0.0        1
     180   2.686595e+07   0.000000e+00   0.000000e+00      0.0        1
```

各項目の意味は次の通りです．

項目	意味
Iter.	反復回数
Objective	目的関数値
Primal Inf.	主変数に関する実行不可能性の値
Dual Inf.	双対変数に関する実行不可能性の値
Time(s)	経過時間（秒）
Freedom	基底解の自由度（二次計画問題のときのみ表示される）

2.5 制約充足問題ソルバ（wcsp）における出力

制約充足問題ソルバ（wcsp）を用いた場合には，[Progress] で始まるセクションに進捗が表示されます．

まず，<preprocess begin>から<preprocess end>では wcsp 実行前の準備がおこなわれます．非常に大規模な問題でない限り，ここに要する時間は僅かです．

```
<preprocess begin>.........<preprocess end>
preprocessing time: 0.000465(s)
```

次に，<iteration begin>から<iteration end>では wcsp 実行の進捗が表示されます．ここでは，最良解におけるハード制約のペナルティ，セミハード制約のペナルティ，ソフト制約のペナルティと最良解を見つけたときの時間及び反復回数が逐次表示されます．問題にセミハード制約が存在しない場合，セミハード制約のペナルティは表示されません．また，最良解を更新していない場合は表示が更新されませんが，計算は行われています．試行回数 (TryCount) が 2 以上の場合は試行回数分の進捗表示が行われます．また，wcsp で並列化オプションを有効にした場合はメインスレッドの進捗のみ表示されます．以下は進捗表示の一例です．

```
<iteration begin>
--- TryCount = 1 ---
# random seed = 1
(hard/semihard/soft) penalty= 45/12/4807, time= 0.00(s)
<greedyupdate begin>..........<greedyupdate end>
greedyupdate time= 0.00066(s)
(hard/semihard/soft) penalty= 0/8/3157, time= 0.00(s), iteration= 1
(hard/semihard/soft) penalty= 0/6/2905, time= 0.00(s), iteration= 2
(hard/semihard/soft) penalty= 0/6/2831, time= 0.00(s), iteration= 3
(hard/semihard/soft) penalty= 0/6/2783, time= 0.00(s), iteration= 4
(hard/semihard/soft) penalty= 0/6/2746, time= 0.00(s), iteration= 5
(hard/semihard/soft) penalty= 0/4/2775, time= 0.00(s), iteration= 8
(hard/semihard/soft) penalty= 0/2/3320, time= 0.00(s), iteration= 14
(hard/semihard/soft) penalty= 0/2/3283, time= 0.00(s), iteration= 15
(hard/semihard/soft) penalty= 0/0/4746, time= 0.00(s), iteration= 254
--- End Phase-I iteration ---
(hard/semihard/soft) penalty= 0/0/4094, time= 0.01(s), iteration= 368
# (hard/semihard/soft) penalty= 0/0/4094
# time = 0.01/0.02(s)
# iteration = 368/1000
<iteration end>
```

各項目の意味は次の通りです．

項目	意味
TryCount	現在の試行回数
random seed	使用した乱数の種
greedyupdate time	貪欲法によって初期解を更新するのにかかった時間

項目	意味
(hard/semihard/soft) penalty	ハード，セミハード，ソフト制約のペナルティ量
time	経過した時間
iteration	反復回数
End Phase-I iteration	ハード制約とセミハード制約が 0 になったことを意味する
time = X/Y(s)	最良解の発見に X 秒，wcsp の計算開始から終了までに Y 秒かかったことを意味する
iteration = X/Y	最良解の発見に X 回，wcsp の計算開始から終了までに Y 回反復したことを意味する

TryCount が 2 以上の場合は乱数の種を変えて複数回計算が行われます．全試行が終了した後にサマリが表示されます．以下はサマリの一例です．

```
<summary>
 trycount         hard     semihard         soft    iteration  time(s)
        1            0            0         2284         9796     0.49
        2            0            0         2160         5969     0.32
        3            0            0         2370         9760     0.50
        4            0            0         2298         4029     0.21
        5            0            0         2420         9326     0.46
        6            0            0         2422         3382     0.18
        7            0            0         2297         9574     0.47
*       8            0            0         2119         7698     0.39
```

サマリの内容は各試行における最良解のハード，セミハード，ソフト制約のペナルティ量と最良解を見つけるのにかかった時間及び反復回数です．"*"は全試行の中で最も良い解を表します．この例の場合，8 回目の試行で得られた解が最も良いことを意味します．

計算終了後，[Result] セクションに求解結果が出力されます．wcsp 特有の項目は次の通りです．

項目	意味
STATUS	求解ステータス．正常終了した場合は NORMAL，異常終了した場合は ERROR が出力されます．
TERMINATE_REASON	wcsp の計算が終了した理由
PENALTY	総ペナルティ量．ハード，セミハード，ソフト制約のペナルティ量の総和が出力されます．
RANDOM_SEED	最良解を見つけた乱数の種

2.6 分枝限定法における出力

混合整数計画問題を解く場合は，通常，自動的に分枝限定法が起動されます．その場合 [Progress] で始まるセクションでの実行経過の出力は次のようになり，求解の進行状況を確認できます．

```
#sol         upper         lower   gap(%)  time(s)  list  mem(MiB)
              +inf       84121.2     +inf      2.4     1       68  cut: 44
              +inf       85090.4     +inf      2.7     1       71  cut: 12
              +inf       85570.3     +inf      3.0     1       74  cut: 6
#1          139000       88862.7   22.003      5.1   190       98  sol: rens
#2          135125       88862.7   20.654      5.2   190       98  sol: rens
#3          134125         89294   20.066      5.4   194       99  sol: rens
#4          134025         89294   20.030      5.5   195       99  sol: rens
#5          133850         89294   19.967      5.7   196       99  sol: rins
#6          129350         89294   18.320      5.9   197       99  sol: relax
```

分枝限定法の進捗は，

- 新しい暫定解が得られた
- 所要メモリが 50MiB 以上変動した
- 15 秒経過した
- （並列化機能が無効のときに）切除平面を追加した

のいずれかの条件を満たせば出力されます．

各項目の意味は次の通りです．

表示	意味
#sol	発見した実行可能解の個数
upper	目的関数の上界値
lower	目的関数の下界値
gap(%)	上下界の相対ギャップ（パーセンテージ）
time(s)	経過時間（秒）
list	探索していない分枝木の葉の数
mem(MiB)	使用メモリ（メビバイト）
cut	追加した切除平面の数（並列化機能が無効であり，切除平面を追加したときに表示）
sol	解を発見した手法（解を発見したときに表示）
#worker	求解中の worker の数（並列化機能が有効のときに表示）

分枝限定法の場合は元問題とスケーリング後の問題について，非零な係数値の絶対値の範囲が出力されます．また，スケーリング値の範囲も出力されます．

```
Coefficient Statistics (before scaling)
  Coefficient range         [min,max] : [1.00e-01,2.00e+01]
  RHS and bounds            [min,max] : [1.00e+00,6.00e+01]
  Objective                 [min,max] : [1.60e+01,4.00e+01]

Coefficient Statistics (after scaling)
  Coefficient range         [min,max] : [1.75e-01,2.42e+00]
  RHS and bounds            [min,max] : [1.00e+00,3.44e+01]
  Objective                 [min,max] : [8.60e+01,8.60e+01]
  Row scaling range         [min,max] : [3.41e-02,3.34e+00]
  Column scaling range      [min,max] : [3.56e-02,1.00e+00]
```

出力される情報は以下の 5 つです.

項目	意味
Coefficient range	係数行列において，非零要素の絶対値の範囲
RHS and bounds	制約式及び変数において，非零な上下限値の絶対値の範囲
Objective	目的関数において，非零な係数値の絶対値の範囲
Row scaling range	係数行列において，非零な行方向のスケーリング値の絶対値の範囲
Column scaling range	係数行列において，非零な列方向のスケーリング値の絶対値の範囲

スケーリング値は，目的関数，制約式及び変数に乗じられる定数値です．例えば列方向のスケーリング値が 1.0e-08〜1.0e-6 の場合は変数が「1」だけ変動すると，この変動はソルバ内部では 1.0e-08〜1.0e-6 という微小な変動として解釈されます．このため，内部の変数の上下限制約違反の閾値が 1.0e-08 であるとすると上下限制約を「1」違反する解が許容されてしまう可能性があることになります．行方向スケーリング値も同様に，制約違反の許容具合に影響します.

スケーリング値（Row scaling range / Column scaling range）が小さすぎる場合は，スケーリングオプションを off にする，あるいは問題に与える変数の単位を見直すこと等が推奨されます.

初期解の修復機能を有効にした場合，分枝限定法の計算開始前に以下のような進捗が表示されます.

```
phase       total_slack         objective  time(s)  ite.  mem(MiB)
feas.           250.392                 0      5.5     0       277
feas.           250.392                 0      6.1     1       278
 opt.           250.392      -6.01642e+09     24.3     2       302
feas.                 2       7.59344e+08     25.2     3       301
feas.                 1       7.77003e+08     25.9     4       301
feas.                 1       7.77003e+08     26.5     5       293
```

これは初期解の修復における各反復（ite.）で，制約式の総違反量（total_slack）と目的関数（objective）がどのように遷移しているかを表示しています．行頭の feas. と opt. はその反復でどのような計算が

おこなわれているかを表しています．feas. は総違反量の最小化をおこない，opt. は目的関数の最小化（あるいは最大化）をおこなっています．実行可能解が見つかるか，ある程度の反復がおこなわれると初期解の修復を終了し，分枝限定法に移行します．

2.7 重み付き局所探索法（wls）における出力

重み付き局所探索法（wls）を用いた場合には，[Progress] で始まるセクションに探索の情報が以下の順に表示されます．並列化機能の有効・無効の場合で出力が異なります．

開始時

```
<problem statistics>
# Int Vars     / Total        = 620 / 625
#   - 0-1 Vars                = 400
# 0-1 Constrs  / Total        = 150 / 540
#   - Set Multi-Cover         = 50
#   - Set Multi-Packing       = 100
#   - Set Multi-Partition     = 0
# Soft Constrs / Total        = 100 / 625
# Var Bounds
#   - Int Range Max           = 12
#   - Continuous Range Max    = 0
#   - Unbounded Vars          = 105
# Assignment Labels (V1,V2,E) = (5, 10, 50)

<iteration begin>
# Initial Sol         = given
# Obj                 = 240.00
# (Hard/Soft) Penalty = 54.00 / 430.00
```

<problem statistics>では，読み込まれた最適化問題について WLS の性能に大きく影響する情報が表示されます．具体的には，変数や制約式の中に 0-1 変数や 0-1 制約式が多く含まれるほど WLS は高い性能を発揮します．ここで 0-1 制約式とはすべての係数が 0 か 1 である線形な制約式です．0-1 制約式は，集合被覆型（例：$x_1 + x_2 \geq 2$），集合充填型（例：$x_1 + x_2 \leq 1$），集合分割型（例：$x_1 + x_2 = 2$）に分類されます．

<iteration begin>以降では初期解の情報が表示されます．上の例の場合，ユーザが指定した初期解は目的関数値が 240 であり，ハード制約違反量は 54，ソフト制約違反量は 430 であると読み取れます．並列化機能が有効のときには，初期解の設定方法が "zero" と表示されます．

それぞれの項目の意味は次のとおりです．

表示	意味
# Int Vars / Total	整数変数の個数 / 変数の個数
# - 0-1 Vars	0-1 変数の個数
# 0-1 Constrs / Total	0-1 制約式の本数 / 制約式の本数
# - Set Multi-Cover	集合被覆型制約式の本数
# - Set Multi-Packing	集合充填型制約式の本数
# - Set Multi-Partition	集合分割型制約式の本数
# Soft Constrs / Total	ソフト制約式の本数 / 制約式の本数
# - Int Range Max	整数変数の「上限 - 下限」の最大値
# - Continuous Range Max	連続変数の「上限 - 下限」の最大値
# - Unbounded Vars	上限または下限が設定されていない変数の個数
# Assignment Labels (V1,V2,E)	割当ラベルの二部グラフにおける頂点と辺の数
# Initial Sol	初期解の設定方法 "given": ユーザ指定, "random": ランダム, "zero": ゼロ初期化
# Obj	初期解の目的関数値
# (Hard/Soft) Penalty	初期解のハード制約違反量 / ソフト制約違反量
# Workers	求解する Worker の数（並列化機能が有効のときに表示）

途中経過

初期解の情報に続き，探索の途中経過が表形式で逐次的に表示されます．現在の実行時間や，現在どういった解を探索しているのか，現在までに見つけた解の中で最も良い解は何なのか，といった情報が読み取れます．

途中経過（並列化機能が無効の場合）

```
-------------------------------------------------------------------------------------
       Resources           |          Current Sol          |           Best Sol
 #Itrs  Time(s)  Mem(MiB) |    Obj      Hard       Soft   |    Obj      Hard       Soft
-------------------------------------------------------------------------------------
     1     0.67    250.23 | 688.23     11.00     123.00  | 688.23     11.00     123.00
     5     0.75    403.34 | 347.43      3.00      23.00  |    400      0.00      43.00
...
-------------------------------------------------------------------------------------
```

上の表は，反復が一定回数行われたり最良解が更新されたりする度に行が追加されていきます．解が更新されないと表示の更新も鈍くなりますが，計算は行われています．局所探索法による探索の一般的な性質として，計算の後半には解の更新は鈍くなります．

それぞれの項目の意味は次のとおりです．

表示	意味
#Itrs	反復回数
Time(s)	実行時間（秒）
Mem(MiB)	メモリ使用量（メビバイト）
Current Sol	現在探索中の解
Best Sol	最良解
Obj	目的関数値
Hard	ハード制約違反量
Soft	ソフト制約違反量

途中経過（並列化機能が有効の場合）

```
-------------------------------------------------------------
        Resources          |                 Best Sol
 #Itrs  Time(s)  Mem(MiB) |          Obj        Hard        Soft
-------------------------------------------------------------
     0     0.00       339 |       693091      109672           0
111552     4.00       339 |  3.20247e+07           0           0  sol: worker#2
272215     8.00       339 |  3.10217e+07           0           0  sol: worker#1
...
-------------------------------------------------------------
```

上の表は，一定時間が経過する度に行が追加されていきます．

それぞれの項目の意味は次のとおりです．ここで worker#n は，n 個目の Worker を表します．例えば，スレッド数を 4 に設定した場合，worker#1 〜 #4 の 4 つの Worker が並行して探索を行います．

表示	意味
#Itrs	Worker 間の最小の反復回数
Time(s)	実行時間（秒）
Mem(MiB)	総メモリ使用量（メビバイト）
Best Sol	最良解
Obj	目的関数値
Hard	ハード制約違反量
Soft	ソフト制約違反量
sol: worker#n	最良解を発見した Worker の番号

終了時

```
========================================
#  Obj                  = 230.00
#  (Hard/Soft) Penalty = 0.00 / 4.00
#  Elapsed Time (s)     = 43.54 / 100.00
#  Iterations           = 81708 / 191770
========================================
<iteration end>
```

アルゴリズムが終了すると実行時間や最良解などの情報が表示されます．上の例の場合，目的関数値 230，ハード制約違反量 0，ソフト制約違反量 4 である解が最良解として得られており，アルゴリズムの実行時間は 100 秒間であったことが読み取れます．

それぞれの項目の意味は次のとおりです．

表示	意味
# Obj	最良解の目的関数値
# (Hard/Soft) Penalty	最良解のハード制約違反量 / ソフト制約違反量
# Elapsed Time (s)	最良解発見時の実行時間 / 総実行時間
# Iterations	最良解発見時の反復回数 / 総反復回数

2.8 資源制約付きスケジューリング問題ソルバ（rcpsp）における出力

資源制約付きスケジューリング問題ソルバ（rcpsp）を用いた場合，[Progress] で始まるセクションでの実行経過の出力は以下のようになります，目的関数の設定によって 2 種類の出力があります．

2.8.1 完了時刻最小化

```
<preprocess begin>.............................<preprocess end>
<iteration begin>
(soft) penalty= 18, time= 0.00(s),iteration= 0
(soft) penalty= 17, time= 0.00(s),iteration= 2
(soft) penalty= 16, time= 0.00(s),iteration= 3
(soft) penalty= 15, time= 0.00(s),iteration= 4
(soft) penalty= 14, time= 0.03(s),iteration= 6
(soft) penalty= 13, time= 0.05(s),iteration= 8
 ...
<iteration end>
```

項目の意味は次の通りです．

表示	意味
(soft)	発見された解に関する
penalty=値 1	値 1：ソフト制約違反量
time=値 2，iteration=値 3	値 2：経過時間，値 3：反復回数

目的関数は内部では，ソフト制約として扱われていますので，目的関数の値もソフト制約違反量にふくまれています.

2.8.2 納期遅れ最小化

```
<preprocess begin>.............................<preprocess end>
<interation begin>
(objective value) value= 18, time= 0.00(s),iteration= 0
(objective value) value= 10, time= 0.03(s),iteration= 1
(objective value) value= 4, time= 0.05(s),iteration= 8
 ...
<iteration end>
```

項目の意味は次の通りです.

表示	意味
(objective value)	発見された解に関する
value=値 1	値 1：目的関数値（総納期遅れ）
time=値 2，iteration=値 3	値 2：経過時間，値 3：反復回数

上記 2 つの表示は，制約充足ソルバの時と同様，最良解が更新される度に現れます．その為，解が更新されないと表示が停止する点においても制約充足ソルバの時と同様です.

2.9 実行不可能性要因検出機能（iisDetect）の出力

デフォルトの指定では，実行不可能性を検出する iisDetect と呼ばれる仕組みが自動的に起動され，実行不可能性の原因の探索を行い，その結果を解ファイルの出力に反映させます（制約充足問題ソルバ/資源制約付きスケジューリング問題ソルバ使用時以外）．ここでは，iisDetect 機能が起動した場合の出力結果を説明します.

以下のモデル記述（モデルファイル名 lp.smp とします）に書かれた線形計画問題は，実行不可能（制約を満たす解なし）です.

```
Variable x, y, z;
Objective f(type = minimize);
```

```
f = x + y + z;
x >= 2 * y;       // IIS
1 + 2 * z >= x;   // IIS
y >= 2 + z;       // IIS
x >= z;
y >= 0;
z >= 0;
```

よく見るとモデルで“IIS”のマークが付いた制約式群のどの一つを除去しても実行不可能性は解消しますが，すべてを満たす x, y, z は存在しません．また，マークされていない最後の制約式は実行不可能性とは無関係で，除去する，しないにかかわらず，問題は実行不可能であることもわかります．

iisDetect 機能はこのように，実行不可能性の原因となっている行の組（Irreducible Infeasible Set：IIS と呼ばれます）を特定して出力します．一般に実行不可能な問題について IIS は複数存在しますが，このアルゴリズムは可能な限り小さなもの（含まれている行が少ない）ものを求めるようなヒューリスティクスが導入されています．

この問題を解かせたとき，[Result] で始まるセクションに以下のような出力がなされます．

```
ERROR_TYPE                          (NUOPT 11) infeasible.
DETECTED_IIS_SIZE                                        3
(#IIS_RELATED_VAR)                                       3
INFEASIBILITY_OF_IIS                                   1.5
```

それぞれの出力の意味は，以下のようになります．

タイトル	解説	備考
DETECTED_IIS_SIZE	検出された IIS に含まれる行の数	成功時のみ
(#IIS_RELATED_VAR)	IIS に含まれている行に含まれる変数の数	成功時のみ
INFEASIBILITY_OF_IIS	IIS 全体での実行不可能性	成功時のみ
NO_IIS_FOUND_BY	IIS 検出失敗の原因	失敗時のみ
(#NONLINEAR_CONSTR.)	IIS 検出失敗の原因の可能性がある非線形制約の数	失敗時のみ

モデルに非線形の式が含まれていた場合，IIS の正確な検出はできません．その場合には，ヘッダー部には IIS の検出が非線形性のために失敗したというメッセージが現れ，非線形な制約がいくつあるかを示します．例えば，次のモデルに対する出力は以下のようになります．

```
Variable x, y, z;
Objective f(type = minimize);
f = x + y + z;


x * x >= 2 * y * y;
1 + z >= x;
```

```
y >= 2 + z;
x + y + z >= 0;
```

出力

```
ERROR_TYPE                          (NUOPT 11) infeasible.
NO_IIS_FOUND_BY                             NON_LINEARLITY
(#NONLINEAR_CONSTR.)                                     1
```

2.10 最適化計算結果標準出力内容

以下は，標準出力に出力される内容の一覧です.

タイトル	解説	備考
STATUS	最適化計算終了時の状態	NORMAL/OPTIMAL/NON_OPTIMAL/ERROR のいずれかを取る
(#IIS_RELATED_VAR)	IIS に含まれている行に含まれる変数の数	IIS 特定成功時のみ
(#INTEGER/DISCRETE)	整数変数の総数	
(#NONLINEAR_CONSTR.)	IIS 検出失敗の原因の可能性がある非線形制約の数	IIS 特定失敗時のみ
BOUND_INFEASIBILITY	変数の上下限制約違反量の最大値	値が小さい場合は出力が省略される
CONSTRAINT_INFEASIBILITY	制約式違反量の最大値	値が小さい場合は出力が省略される
DETECTED_IIS_SIZE	検出された IIS に含まれる行の数	IIS 特定成功時のみ
ELAPSED_TIME(sec.)	計算時間	SIMPLE の展開時間を含みません
ERROR_TYPE	エラー番号とエラーメッセージ	STATUS が NON_OPTIMAL や ERROR のときに出力される
FACTORIZATION_COUNT	行列の分解回数	内点法のみ
FUNC_EVAL_COUNT	関数の評価回数	内点法のみ
GAP	上界値と下界値の差	分枝限定法のみ
INFEASIBILITY_OF_IIS	IIS 全体での実行不可能性	IIS 特定成功時のみ
ITERATION_COUNT	アルゴリズム内の反復回数	内点法/wcsp/rcpsp/wls のみ
METHOD	適用した最適化手法	

タイトル	解説	備考
NO_IIS_FOUND_BY	IIS 検出失敗の原因	IIS 特定失敗時のみ
NUMBER_OF_ACTIVITIES	アクティビティの総数	rcpsp のみ
NUMBER_OF_FUNCTIONS	関数（目的関数を含む）の総数	
NUMBER_OF_GENERAL_CONSTRAINT	一般の考慮制約の総数	rcpsp のみ
NUMBER_OF_IMPRECEDENCE	直前先行制約の総数	rcpsp のみ
NUMBER_OF_MODES	モードの総数	rcpsp のみ
NUMBER_OF_PRECEDENCE	先行制約の総数	rcpsp のみ
NUMBER_OF_RESOURCES	資源の総数	rcpsp のみ
NUMBER_OF_VARIABLES	変数の総数	
PARTIAL_PROBLEM_COUNT	部分問題数	分枝限定法のみ
PENALTY	重みつき制約違反量	wcsp/rcpsp 適用時のみ
PROBLEM_NAME	問題名	SIMPLE 版:モデル名 MPS 版:TITLE の内容
PROBLEM_TYPE	MINIMIZATION（最小化） MAXIMIZATION（最大化）	
RANDOM_SEED	乱数の種	wcsp のみ
RESIDUAL	最適性条件の充足度合	分枝限定法/wcsp/rcpsp/wls 以外
SIMPLEX_PIVOT_COUNT	単体法の反復回数	単体法のみ
SOLUTION_FILE	解ファイルのパス名	出力される解ファイルの名前は，環境や設定により異なる
TERMINATE_REASON	計算終了理由	wcsp/rcpsp 適用時のみ
THREADS	並列化実行において使用したスレッド数	分枝限定法/制約充足問題ソルバ wcsp など並列化実行可能なアルゴリズムにおいて出力される
VALUE_OF_OBJECTIVE	目的関数値	実行不可能（infeasible）の場合，出力される目的関数の値は不定となる

STATUS は「最適化計算終了時の状態」です.

- OPTIMAL は最適解が得られたことを意味します．ただし，「4.1 数理最適化問題一覧」内の NLP（非線形計画問題）・NLSDP（非線形半正定値計画問題）に対しては局所最適解が得られていることを意味します.
- NON_OPTIMAL は何かしらの理由で局所最適解が得られなかったことを意味します．詳細は

ERROR_TYPE を参照します.

- NORMAL は wcsp,rcpsp,wls が正常終了したことを意味します.
- ERROR は wcsp,rcpsp,wls が異常終了したことを意味します.

2.11 標準出力の抑制

以下の設定を行う事で，標準出力に出力される情報を出力しないように設定できます．設定方法は 2 つあります.

一つは，求解オプションファイル nuopt.prm に記述する方法です．nuopt.prm 内に以下のように記述します.

```
output:mode = silent
```

もう一つは，SIMPLE モデルファイル内に記述する方法です．C++SIMPLE の場合，モデルファイル内に以下のように記述します.

```
options.outputMode = "silent";
```

第3章 解ファイル

Nuorium Optimizer は最適化計算の詳細情報を解ファイルというファイルに出力します．コマンドラインで Nuorium Optimizer を起動した場合，モデル名.sol という解ファイルが作成されます．

具体例として，次の例題に対する解ファイルを見ていくことにします．

$$
\begin{aligned}
\text{最小化} \quad & -3x_1 - 2x_2 - 4x_3 \\
\text{条件} \quad & x_1 + x_2 + 2x_3 \leq 4 \\
& 2x_1 + 2x_3 \leq 5 \\
& 2x_1 + x_2 + 3x_3 \leq 7 \\
& x_1 \geq 0, x_2 \geq 0, x_3 \geq 0
\end{aligned}
$$

3.1 冒頭部分

解ファイルの冒頭部分には

```
%%
%%
%%
%% RESULT OF NUOPT #1
%%
%%
%%
%%
```

と表示されます．これは 1 回目の求解結果であることを示しています．`solve()` を複数回記述した場合，1 回目の求解結果の後に 2 回目以降の求解結果が順に表示されます．

各求解結果では，標準出力の [Problem and Algorithm] および [Result] で始まるセクションに出力される内容と，同等の情報が出力されます．

例えば，上記の例題を線形計画専用内点法 higher で解いた際の，解ファイルの冒頭部分は，次のようになります．

```
PROBLEM_NAME                                    sample
NUMBER_OF_VARIABLES                                  3
NUMBER_OF_FUNCTIONS                                  4
```

```
PROBLEM_TYPE                                MINIMIZATION
METHOD                                      HIGHER_ORDER
STATUS                                           OPTIMAL
VALUE_OF_OBJECTIVE                                -10.5
ITERATION_COUNT                                       7
FUNC_EVAL_COUNT                                      10
FACTORIZATION_COUNT                                   8
RESIDUAL                              3.903545017e-009
ELAPSED_TIME(sec.)                                 0.02
```

上記例題を，単体法 simplex で解いた際，解ファイルの冒頭部分は次のようになります．ITERATION_COUNT，FUNC_EVAL_COUNT，FACTORIZATION_COUNT という行が表示されないかわりに SIMPLEX_PIVOT_COUNT という行に単体法の反復回数が表示されます．

```
PROBLEM_NAME                                   sample
NUMBER_OF_VARIABLES                                 3
NUMBER_OF_FUNCTIONS                                 4
PROBLEM_TYPE                                MINIMIZATION
METHOD                                          SIMPLEX
STATUS                                           OPTIMAL
VALUE_OF_OBJECTIVE                                -10.5
SIMPLEX_PIVOT_COUNT                                 3
RESIDUAL                              2.340507908e-016
ELAPSED_TIME(sec.)                                 0.02
```

上記例題において変数をすべて整数変数に直した問題を，分枝限定法＋単体法 simplex を用いて解いた場合，分枝限定法の部分問題の数 PARTIAL_PROBLEM_COUNT が出力されます．分枝限定法を行った場合に RESIDUAL が表示されないのは，整数解においては最適性条件（連続変数を仮定）が充足しているとはいえないので，RESIDUAL の値が目的関数値の正しさを示す尺度とはならないためです．

```
PROBLEM_NAME                                   sample
NUMBER_OF_VARIABLES                                 3
(#INTEGER/DISCRETE)                                 3
NUMBER_OF_FUNCTIONS                                 4
PROBLEM_TYPE                                MINIMIZATION
METHOD                                          SIMPLEX
STATUS                                           OPTIMAL
VALUE_OF_OBJECTIVE                                  -10
SIMPLEX_PIVOT_COUNT                                 4
```

```
PARTIAL_PROBLEM_COUNT                                    1
ELAPSED_TIME(sec.)                                    0.01
```

3.2 解ファイルの変数値表示部

```
%%
%% VARIABLES
%%
```

で始まる部分には最適化アルゴリズム停止時における変数の値，及びその上下限が記述されています．

```
%%
%% VARIABLES
%%
     NAME     VALUE     STATUS      SLACK      [      BOUND TYPE   ]
V# 1 x1             2.5  FREE  2.50000000e+000 [ 0 <=    x[1]      ]
V# 2 x2             1.5  FREE  1.50000000e+000 [ 0 <=    x[2]      ]
V# 3 x3    3.51218e-010  LOWER 3.51217955e-010 [ 0 <=    x[3]      ]
```

VALUE は「変数の値」，STATUS は「変数値が上下限のいずれに付着しているかの状態」，SLACK が「上下限値からの余裕量」，BOUND TYPE は「変数の上下限」を示しています．

例えば，`V# 1` から始まる行は x1 という名前の変数の値（2.5）と，それが下限にも上限にも等しくなっていないこと（"FREE"），続いて x1 に課された制約の種類（[] 内）が示されています．

変数の状態を示す STATUS の意味を以下に示します．

状態を示す文字列	状態
LOWER	下限制約に付着（下限制約が active）
UPPER	上限制約に付着（上限制約が active）
FREE	上下限，いずれにも付着していない
REMVD	前処理によって削除された
INFS	上下限を違反している

INFS は得られた解が，変数の上下限あるいは制約式を違反している場合に出力されます．問題は正常に解けていません．

また，資源制約付きスケジューリング問題ソルバ（rcpsp）を用いた場合には，変数の部分は，以下に相当します．

```
%%
%% ACTIVITIES
%%
```

例えば,

```
%%
%% ACTIVITIES
%%
        NAME       MODE    STATUS SLACK [       BOUND TYPE       ]
V#  1 sourceActivity     "DummyMode"   ACT        [  0 <=  sourceActivity  <= 0  ]
V#  2 act[1]    "A_does"     ACT        [  6 <=  act[1]  <= 12 ]
V#  3 act[2]    "C_does"     ACT        [  0 <=  act[2]  <= 11 ]
V#  4 act[3]    "B_does"     ACT        [  0 <=  act[3]  <= 8  ]
```

のような出力があった場合には，MODE がアクティビティが処理されるモード，BOUND TYPE が（アクティビティの開始時刻）<=（アクティビティ）<=（アクティビティの終了時刻）を表します.

3.3 解ファイルの関数値表示部

```
%%
%% FUNCTIONS
%%
```

で始まる部分には最適化アルゴリズム停止時における関数[1]の値が記述されています.

```
%%
%% FUNCTIONS
%%
          NAME     VALUE STATUS      SLACK      [     BOUND TYPE       ]
F# 1 obj          -10.5  FREE                   [    OBJECTIVE (MINIMIZE)       ]
F# 2 g1               4  UPPER 1.75611081e-010 [             g1 <= 4 ]
F# 3 g2               5  UPPER 7.02441660e-010 [             g2 <= 5 ]
F# 4 g3             6.5  FREE  5.00000001e-001 [             g3 <= 7 ]
```

VALUE が「関数の値」，STATUS が「関数値が上下限のいずれに付着しているかの状態」，SLACK が「上下限値からの余裕量」，BOUND TYPE が「関数の上下限」を示しています.

F# 1 から始まる行には F という名前の関数（目的関数）の値（-10.5）と，それが最小化された（"MINIMIZE"）目的関数である旨が示されています．F# 2 から始まる行は g1 という名前の制約式の値が 4 であり，それが上限に等しくなっていること（"UPPER"），続いて g1 に課された制約の種類

[1] 目的関数と制約式を総称してこう呼んでいます.

（[] 内）が示されています.

関数（制約式）の状態を示す STATUS の意味を以下に示します.

状態を示す文字列	状態
LOWER	下限制約に付着（下限制約が active）
UPPER	上限制約に付着（上限制約が active）
FREE	上下限，いずれにも付着していない
INFS	上下限を違反している
TGIN	ソフト制約が制約を違反していない
TGOUT	ソフト制約が制約を違反し，ペナルティが発生している.

INFS は数値的な性質の悪い問題や最適化が途中で失敗した際に出力されます.

また，半正定値制約を課した対称行列の各要素の値に関しては以下のような出力がされます.

```
      NAME   VALUE  STATUS SLACK [   BOUND TYPE   ]
F# 1 X[1,1] 25.6995              [   MATRIXELEM   ]
F# 2 X[2,1]       1              [   MATRIXELEM   ]
F# 3 X[2,2] 25.6995              [   MATRIXELEM   ]
```

3.4 解ファイルの上下限，制約と対応する双対変数表示部

解ファイルの

```
%%
%% BOUNDS
%%
```

から続く部分には変数の上下限と双対変数，また

```
%%
%% CONSTRAINTS
%%
```

から続く部分には制約式の上下限と双対変数がそれぞれ表示されます.

```
%%
%% BOUNDS
%%
     [            BOUND TYPE            ]     DUAL VALUE
B# 1 [     0      <=  x1                ]   4.891837406e-010
B# 2 [     0      <=  x2                ]   8.151927281e-010
B# 3 [     0      <=  x3                ]         1.000000001
```

```
%%
%% CONSTRAINTS
%%
     [    CONSTRAINT/OBJECTIVE TYPE     ]     DUAL/WGT
C# 1 [       OBJECTIVE (MINIMIZE)       ]                   0
C# 2 [                 g1  <=     4      ]       -1.999999998
C# 3 [                 g2  <=     5      ]      -0.4999999986
C# 4 [                 g3  <=     7      ]  -2.443858094e-009
```

BOUND TYPE 及び CONSTRAINT/OBJECTIVE TYPE が「上下限の種類」，DUAL VALUE 及び DUAL/WGT が「双対変数の値」を示しています．

B# 1 の行では，x1 に対する制約が下限制約 0 <= x1 であること，またその双対変数がほぼ 0 である旨が示されています．

双対変数は制約式や変数の上下限を単位あたり変化させたときの目的関数の変動を示しています．双対変数は別名，シャドウプライス，または reduced cost とも呼ばれ，その正負や大きさから上下限のいずれが active かを次のように判断することができます．

解ファイルの双対変数値	状態
正	下限制約に付着（下限制約が active）
負	上限制約に付着（上限制約が active）
零/零に近い値	上下限，いずれにも付着していない

目的関数の双対変数値に対応する部分には零が出力されます．

3.5 解ファイルの実行不可能性要因出力部

解ファイルには，実行不可能性検出機能によって判定された「実行不可能な制約式の組」が出力されます．次のモデルに対しては，以下の出力が解ファイルになされます．

```
Variable x, y, z;
Objective f(type = minimize);
f = x + y + z;
x >= 2 * y;       // IIS
1 + 2 * z >= x;   // IIS
y >= 2 + z;       // IIS
x >= z;
y >= 0;
z >= 0;
```

解ファイル出力

```
%%
%% IIS
%%
--------------------------------------------------------------------------------
#2      sample.smp:4      :   -1*x+2*y
                                          <=          0   (-1.11e-016)
--------------------------------------------------------------------------------
#3      sample.smp:5      :   -1+1*x-2*z
                                          <=          0   (         0)
--------------------------------------------------------------------------------
#4      sample.smp:6 INFS :   2-1*y+1*z
                                          <=          0   (       1.5)
--------------------------------------------------------------------------------
```

上記のように，IIS に含まれる行の制約式の名前が出力されます．内部表現に従って移項されていますので，x, y, z の順番はモデル記述とは異なります．() 内は現在の変数値の設定（以降に出力されます）における制約式の値です．INFS とマークがある行は現在の変数値の設定において，上下限を破っている行です．これを解消しようとすると，IIS の定義から，ここに現れている行のほかのいずれかに波及します．IIS 検出に成功した場合の変数の設定は IIS に含まれる行の違反の合計が最も小さくなるように行われます．その際の IIS に含まれる行の違反の合計が，標準出力に現れる

INFEASIBILITY_OF_IIS

という値です．

実行不可能性が検出された場合は，実行不可能性の要因と関連する変数，関数のみが解ファイルに出力されます．

3.6 解ファイルのハード制約，セミハード制約およびソフト制約表示部

アルゴリズムとして制約充足問題ソルバ wcsp を用いている場合，解ファイルには，最終的に満たすことのできていないハード制約，ソフト制約が出力されます．次が出力サンプルです．

解ファイル出力

```
%%
%% WCSP_PENALTY
%%
           NAME                 TYPE    VALUE   BOUND    AMOUNT WEIGHT PENALTY
F#   246   model.smp:83[6]       HARD    0    >=    1        1
F#   432   model.smp:91[13]      S.HARD  0    >=    1        1
F#   946   model.smp:119[17,E]   S.HARD  7    <=    6        1
F#  7080   model.smp:152[1](u)   SOFT    3    <=    2        1      100    100
```

```
F#  7586   model.smp:156[7,3]   SOFT   5   <=   2   3   10   30
F#  7675   model.smp:156[21,6]  SOFT   3   <=   2   1   10   10
F#  7756   model.smp:156[2,10]  SOFT   4   <=   2   2   10   20
F#  7780   model.smp:156[1,11]  SOFT   3   <=   2   1   10   10
F#  8293   model.smp:162[19,25] SOFT   0   ==   5   5    1    5
                               . . .
```

上記のように，ハード制約，ソフト制約で満たされていないもののみがハード制約，セミハード制約，ソフト制約の順に表示されます．各行は，制約式の名前，タイプ（ハード：HARD，セミハード：S.HARD，ソフト：SOFT），現在の値と制約，違反量，ウエイト（ソフト制約のみ），ペナルティ量（ソフト制約のみ）を示しています．

この出力例ではハード制約である 246 番目の制約（model.smp:83[6]）が 0 ですが，本来 1 以上とならねばならないので，ハード制約違反の違反量が 1 であることなどがわかります．

上記で上から 4 つめの制約（model.smp:152[1]）の名前の後に (u) とあるのは，この制約が上下限制約であり，違反しているのは制約の上限であることを示しています（2 という上限に対して 3 という値を取っています）．上下限制約の下限に違反しているのであれば制約の名前の後に (l) と表示されます．ソフト制約については，違反量に設定されたウエイトを掛けた値である「ペナルティ」があわせて表示されます．制約充足問題ソルバ wcsp は，ハード制約，セミハード制約の違反量，ソフトペナルティのペナルティの合計値を最小化します．

第4章 Nuorium Optimizerの適用範囲とアルゴリズム

本章では，Nuorium Optimizer が扱う事のできる数理最適化問題の範囲及び備えているアルゴリズムを列挙し，それらの対応関係を与えます．また，アルゴリズムの設定方法も示します．

4.1 数理最適化問題一覧

Nuorium Optimizer では，以下のような数理最適化問題を取り扱うことができます．

- LP (Linear Programming：線形計画問題)
  目的関数と制約式がすべて線形である問題で，整数変数を含まないものです．
- MILP (Mixed Integer Linear Programming：混合整数線形計画問題)
  目的関数と制約式がすべて線形で，整数変数を含むものです．MIP（Mixed Integer Programming）と呼ばれることも多いです．
- CMIQP (Convex Mixed Integer Quadratic Programming：凸混合整数二次計画問題)
  制約式がすべて線形，目的関数が二次関数で，整数変数を含むものです．
- MINLP (Mixed Integer Nonlinear Programming：混合整数非線形計画問題)
  制約式および目的関数が非線形で整数変数を含むものです．
- CQP (Convex Quadratic Programming：凸二次計画問題)
  目的関数が凸な二次関数，制約式がすべて線形であるもの（ただし，目的関数の符号の変更で下に凸な目的関数の最小化に帰着できるもの）です．
- CP (Convex Programming：凸計画問題)
  目的関数，制約式に非線形なものが含まれていますが，実行可能領域が凸で，目的関数の符号の変更で下に凸な目的関数の最小化に帰着できる問題です．ここでは整数変数は含まないものを言います．
  半正定値計画問題も凸計画問題の一部ですが，ここには含めません．
- NLP (Nonlinear Programming：非線形計画問題)
  上記以外で，整数変数を含まない一般の非線形計画問題です．
- SDP (SemiDefinite Programing：半正定値計画問題)
  行列の半正定値制約を含む線形計画問題です．
- NLSDP (NonLinear SemiDefinite Programing：非線形半正定値計画問題)
  行列の半正定値制約を含み，なおかつ目的関数・制約式に非線形項が含まれる問題です．
- WCSP (Weighted Constraint Satisfaction Problem：重み付き制約充足問題)
  各々重みの付いた制約条件をなるべく満足するためには値をどのように割り当てると良いかを決定する問題です．制約充足問題ソルバ wcsp により高速に解を得ることができます．

- RCPSP (Resource Constrained Project Scheduling Problem：資源制約付きスケジューリング問題）
  一定の資源制約の下で，決められた作業の開始・終了時刻を決定する問題です．一般の整数計画問題（MILP）として記述することも可能ですが，特殊な記法を行うと資源制約付きスケジューリング問題ソルバ rcpsp により高速に実行可能解を得ることができます．

4.2 アルゴリズム一覧

Nuorium Optimizer は以下のようなアルゴリズムを備えています．見出しの解法名は，アルゴリズムを指定する際に用います．カッコ内の解法名は，標準出力に METHOD として出力されるものです．

- simplex：単体法（SIMPLEX）
  線形計画法の解法として古くから知られている方法です．大規模問題では内点法に速度的に劣りますが，可能基底解が求まり原理的に内点法/外点法よりも高精度です．
  整数変数を含む問題に対して指定すると，単体法を分枝限定法（Branch and bound method）という枠組のなかで繰り返し行って，最適性の保証のある整数解を求めます．大規模問題において基底解が必要な場合には，"cross:on"と指定して内点法からのクロスオーバーを用いるのが有利です．
- hsimplex：単体法 (HSIMPLEX)
  PySIMPLE から呼び出して使える単体法です．線形計画問題および凸二次計画問題に適用できます．simplex と比較して，よりスパース性を活用しているので，多くのケースで高速に動作します．整数変数を含んだ問題に対しては整数性を緩和した問題を解くのでご注意ください．
- asqp：有効制約法（ACTIVE_SET_QP）
  単体法と同様，古典的な凸二次計画問題の厳密解法です．1 万変数以上の大規模問題では，一般に内点法（直線探索法（Line Search Method））に劣りますが，
  - 変数に比べて制約式の数が非常に少ない（1/10 以下）場合
  - 目的関数のヘッセ行列が密行列である場合

  には内点法よりも高速かつ高精度です．また，整数計画法に対応しているので，整数変数が含まれている凸二次計画問題を解くことができます．`"cross:on"`と指定することで内点法からのクロスオーバーを用いることができるので，大規模問題に対して高精度な解を求めることができます．
- higher：線形計画問題専用内点法（HIGHER_ORDER）
  線形計画法に特化した内点法で，大規模な線形計画問題の解法としては最も高速です．単体法と違い，可能基底解は求まりません．
- lipm：直線探索法（LINE_SEARCH_IPM）
  一般の凸計画問題に適用可能な内点法です．問題が凸であることがわかっている場合には信頼領域法よりも高速です．幅広い範囲の問題に対して有効です．
- bfgs：準ニュートン法（BFGS_LINE_SEARCH）
  準ニュートン法を用いた直線探索に基づく内点法です．ヘッセ行列の近似行列を密行列として保持します．小規模（50〜500 変数以下）かつ非線形性の強い問題に対して tipm よりも有効な場合があります．
- tipm：信頼領域内点法（TRUST_REGION_IPM）

大規模なものを含む一般の非線形計画問題に適用可能な内点法です．幅広い範囲の問題に対して有効です．

- lsqp：直線探索法に基づく逐次二次計画法（LINE_SEARCH_SQP）
  準ニュートン法によって二階微係数を求める逐次二次計画法です．小規模（50〜100 変数以下）な非線形計画問題に適しています．
  問題によっては直線探索内点法（lipm）よりも安定的により精度の良い解を導くことができます．
- tsqp：信頼領域法に基づく逐次二次計画法（TRUST_REGION_SQP）
  二階微係数をそのまま用いる逐次二次計画法です．大規模なものを含む一般の非線形計画問題に適用可能な方法です．一般に内点法よりも低速ですが，問題によっては内点法よりも安定的に，より精度の良い解を導くことができます．
  変数の数よりも制約式数が多い場合には内点法（tipm）よりも高速な場合があります．
- lsdp：線形半正定値計画問題に対する主双対内点法
  線形の半正定値計画問題に対する主双対内点法です．目的関数・制約式に出現する項は線形である必要があります．内部でメリット関数の計算を行いません．
- trsdp：信頼領域法を用いた非線形半正定値計画問題に対する主双対内点法
  目的関数・制約式に非線形項が出現する半正定値計画問題に対する主双対内点法です．メリット関数の降下を保証するために，信頼領域法を利用しています．
- wcsp：制約充足問題ソルバ（WCSP）
  京都大学「問題解決エンジン」グループの開発による制約充足問題に対するアルゴリズムです．必ずしも厳密解が求まるわけではありませんが，大規模な整数計画問題に対し，非常に高速に実行可能解（近似解）を求めることができます．
  整数変数のみを含み，かつすべての変数に上限と下限がある問題に対してのみ有効です．目的関数，制約式に重みを設定することができます．制約の重みには，ハード制約，セミハード制約，ソフト制約の三種類があります．
- wls：重み付き局所探索法 (WLS)
  PySIMPLE から呼び出して使える近似解法です．0-1 係数のみを含む制約式に対して特別な処理をおこなっています．そのため，集合被覆や集合分割といった特定の問題を得意としています．線形及び二次の制約式・目的関数を扱うことができます．問題が二次式を含む場合は整数変数のみを扱うことができ，すべて線形式の場合は連続変数も扱うことができます．
- rcpsp：資源制約付きスケジューリング問題ソルバ（RCPSP）
  京都大学「問題解決エンジン」グループの開発による資源制約付きスケジューリング問題に対するアルゴリズムです．資源制約の下，決められた作業の開始・終了時刻を決定する問題の実行可能解を高速に求めることができます．rcpsp の記述にあたっては問題を C++SIMPLE の特殊なクラスを用いて記述する必要があります．完了時刻の最小化問題と，納期遅れ最小化問題を扱うことができます．前者を扱う際にはソフト制約，後者を扱う際にはハード制約のみが使用できます．

Nuorium Optimizer のアルゴリズムは，SIMPLE で記述される目的関数，制約で四則演算および数学関数を用いて記述されたものをサポートします．

4.3 数理最適化問題とアルゴリズムの対応

以下は，Nuorium Optimizer で取り扱い可能な数理最適化問題と，Nuorium Optimizer が備えているアルゴリズムの対応一覧です.

表の内容は

◎　最も適している
○　適している
R　整数性を緩和した問題を解く

を示します.

	LP	MILP	CMIQP	WCSP※5	RCPSP	MINLP	CQP	CP	NLP	SDP	NLSDP
simplex	◎	◎									
hsimplex	◎	R					◎				
asqp	○	○	◎				◎				
higher	◎	R									
lipm	○	R	R			R	◎	◎	○		
bfgs	○	R	R			R	○	○	◎		
tipm	○	R	R			R	○	○	◎		
lsqp	○	R	R			R	○	◎	○		
tsqp	○	R	R			R	○	○	◎		
lsdp	○	R※4	R※4							◎	○
trsdp	○	R※4	R※4							○	◎
wcsp		◎※1	◎※1	◎		○※1					
wls		◎※2	◎※2	◎※3							
rcpsp					◎※6						

- ※ 1 について，0-1 整数変数と離散変数（DiscreteVariable）のみを含む問題のみ扱うことができます.
- ※ 2 について，PySIMPLE から呼び出すことができます.
- ※ 3 について，wls が扱えるのは制約式・目的関数がそれぞれ二次までの問題です．また，ハード制約とソフト制約のみ扱うことができます.
- ※ 4 について，SymmetricMatrix を用いて定式化した場合に扱うことができます.
- ※ 5 について，本表では WCSP は「ソフト制約・セミハード制約・ハード制約が定義された連続変数を含まない問題」を指します.
- ※ 6 について，RCPSP は C++SIMPLE のみ扱うことができます.

4.4 アルゴリズムの設定方法

アルゴリズムを設定する方法には，

- モデルファイル内で指定する方法
- 求解オプションファイル nuopt.prm 内で指定する方法

の二通りがあります.

詳細については 5.2.1 最適化計算における解法選択を参照してください.

4.5 アルゴリズムの自動選択

アルゴリズムの指定を明示的に行わない場合には，入力された問題の内容から自動的にアルゴリズムを選択します.（**アルゴリズムの自動選択**）

適用アルゴリズム	判定基準
simplex	関数がすべて線形で整数変数を含む
asqp	目的関数が非線形で整数変数を含む
higher	関数がすべて線形で整数変数を含まない
wcsp	DiscreteVariable, selection を含む
rcpsp	Activity, ResourceRequire, ResourceCapacity を含む
lsdp	関数が全て線形で半正定値制約を含む
trsdp	上記以外の半正定値制約を含む
tipm	上記以外

次のような場合，個別に設定するとよりよい結果が得られる可能性があります.

4.5.1 整数変数が含まれている非線形計画問題

凸混合整数二次計画問題（CMIQP）であれば asqp が利用可能です．混合整数非線形計画問題（MINLP）の場合，アルゴリズムを明示的に指定していない場合はエラーを返します．そのような場合は以下をお試しください.

- 非線形項を線形化して混合整数線形計画問題（MIP）として解く.
- 変数がすべて 0-1 整数変数であれば wcsp で解く.
- 整数性を無視した問題を tipm や tsqp で解く.

4.5.2 整数変数が含まれない非線形計画問題

整数変数が含まれない非線形計画問題の場合，デフォルトでは内点法による信頼領域法（tipm）となりますが，問題が二次計画問題（目的関数のみ二次関数）の場合，特に変数に比べて一般の制約式の数が少ない問題には有効制約法 asqp が有利な場合もあります.

逐次二次計画法 tsqp は若干時間を所要するケースもございますが，最も精度良く非線形最適化を行う方法です.

bfgs は小規模（50〜500 変数以下）かつ非線形性の強い問題に対して tipm よりも有効な場合があり

ます.

4.5.3　凸計画問題

凸計画問題に対しては直線探索法を用いた方が一般に高速ですので，非線形ながら問題が凸であるとわかっている場合には（SIMPLE は凸であるかを自動判定できません）としていずれかの直線探索法を指定してください．通常お勧めできるのが lipm です.

4.6　実行不可能性要因検出機能 iisDetect

実行不可能性要因の検出を行うアルゴリズム iisDetect は，実行不可能性の原因となっている制約式の組の内，できるだけ式の少ないものを特定します．与えられた問題が実行不可能と判定されたら自動的に起動します.

大規模問題で実行不可能と判定された場合，iisDetect で多大な計算時間を要する場合があります．その時には求解オプションでオフにすることが可能です.

4.7　クロスオーバー

線形計画問題（LP）あるいは二次計画問題（QP）に対しては，内点法（higher/lipm/bfgs/tipm）によって得られた解の情報をもとにして単体法を起動する，クロスオーバーと呼ばれる手法を用いることができます．大規模問題に関して精度の良い解を得るにはこの方法が最も有効です.

クロスオーバーを行うためには内点法の手法を設定した後に，以下のように指定します.

モデルファイルに記述する方法

```
options.crossover = "on";
```

求解オプションファイル nuopt.prm に記述する方法

```
cross:on
```

第5章 求解オプション設定

求解オプションは，Nuorium Optimizer の求解動作をより細かく制御するためのものです．求解オプションを利用することにより，アルゴリズムの選択や，標準出力の制御，終了条件の調整などを行う事ができます．

以下は各オプションの説明に記載されている表の見方の説明になります．

「オプション名」には各モデリング言語あるいはで参照する際のオプションが記述されています．一部オプションについては「型」という列が付随しています．これは該当オプションがモデリング言語によって記述する型が異なることを意味します．

「設定値」には，各オプションの値範囲あるいは値の意味が記述されています．

- 値範囲にはデフォルト値および値範囲（最小値・最大値など）が記述されています．
- 値の意味には各設定値の意味する動作が記述されます． 「任意値」とある場合は，値範囲（最小値・最大値含む）を満たした範囲でとる任意の値を意味します．
- 最小値・最大値に「> 数値」とあるのは，それよりも大きい値を表します．例えば最小値が「> 0.0」となっている場合は，0.0 よりも大きい値をとることを表します．
- 「詳細」には各オプションの詳細が記述されています．**[ADVANCED]** とあるのは，上級者向けの説明です．

以下は，求解オプションの一覧表です．なお，オプション名列は PySIMPLE での名称を記載しています．「なし」となっているオプションについては C++SIMPLE 等で設定可能です．

共通求解オプション

オプション	オプション名
5.2.1 最適化計算における解法選択	method
5.2.2 反復回数上限	maxIteration
5.2.3 計算時間上限	maxTime
5.2.4 求解情報の表示制御	solve(silent=...)
5.2.5 スケーリング	scaling
5.2.6 実行不可能性要因検出機能	solve(iis=...)

内点法及び逐次二次計画法用求解オプション

オプション	オプション名
5.3.1 線形計画専用内点法から単体法へのクロスオーバー	higherCrossover
5.3.2 KKT 条件残差停止条件	kktEps
5.3.3 内点法における初期値からの探索	なし
5.3.4 線形計画専用内点法における連立一次方程式を反復法で解く	なし

単体法用求解オプション

オプション	オプション名
5.4.1 主変数実行可能性判定値	simplexPrimalTolerance
5.4.2 双対変数実行可能性判定値	simplexDualTolerance
5.4.3 単体法解法選択（解法 hsimplex のみ）	hsimplexMethod

解法 wcsp タブーサーチ（wcsp）用求解オプション

オプション	オプション名
5.5.1 乱数シード値	randomSeed
5.5.2 試行回数	tryCount
5.5.3 スレッド数上限	threads
5.5.4 制約充足フェーズにおける計算時間上限	wcspPhaseOneMaxTime
5.5.5 制約充足フェーズにおける反復回数上限	wcspPhaseOneMaxIteration
5.5.6 解更新間隔計算時間上限	wcspPhaseTwoMaxIntervalTime
5.5.7 解更新間隔反復回数上限	wcspPhaseTwoMaxIntervalIteration
5.5.8 初期値からの探索	wcspInitialValueActivation

解法 wls 用求解オプション

オプション	オプション名
5.6.1 メモリ量上限	maxMemory
5.6.2 目的関数の目標値	objectiveTarget
5.6.3 試行回数	tryCount
5.6.4 スレッド数上限	threads

分枝限定法用求解オプション

オプション	オプション名
5.7.1 切除平面の強度	branchCut
5.7.2 前処理	branchPresolve
5.7.3 発見的探索 diving	branchDiving
5.7.4 発見的探索 feasibility pump	branchFeasPump
5.7.5 発見的探索 RINS	branchRins
5.7.6 発見的探索 RENS	branchRens
5.7.7 ノード選択	branchNodeSelect
5.7.8 wcsp タブーサーチの起動	branchUseWcsp
5.7.9 wls の起動	branchUseWls
5.7.10 足切り点	branchCutoff
5.7.11 分枝変数スコアの算出方法	branchVariableSelectScore
5.7.12 実行可能解の個数上限	branchMaxSolutionCount
5.7.13 探索問題数上限	branchMaxNode
5.7.14 メモリ上限	maxMemory
5.7.15 上下界値ギャップ閾値（絶対値）	branchGapTolerance
5.7.16 上下界値ギャップ閾値（相対値）	branchRelativeGapTolerance
5.7.17 目的関数の目標値設定	objectiveTarget
5.7.18 スレッド数上限	threads
5.7.19 並列化手法	branchParallelMethod
5.7.20 初期解修復	branchRepairSolution
5.7.21 初期解修復の回数上限	branchRepairIteration
5.7.22 非連結成分検出	branchDisconnected

5.1 求解オプションの設定方法について

求解オプションの設定方法は，モデリング言語あるいは求解オプションファイル nuopt.prm で異なります．本章では求解オプションファイル nuopt.prm での設定方法，及び各モデリング言語での設定方法の参照先を説明します．

5.1.1 求解オプションファイル nuopt.prm

求解オプションファイルは nuopt.prm という名前でなければなりません．

求解オプションファイルの冒頭の行は `begin`，最後の行は `end` である必要があります．`end` の後は必ず改行しなければなりません．次の例は，特に何も指定がなされていない，最も簡単な求解オプションファイルです．

```
begin
end
```

begin と end の間に各種求解オプションの設定を行うことができます．次の例ではアルゴリズムとして単体法 simplex を指定しています．

```
begin
method:simplex
end
```

求解オプションは:ではなく，=で指定する場合もあります．次の例では，停止条件を 10^{-12} に設定しています．

```
begin
crit:eps=1.0e-12
end
```

求解オプションファイル中には半角スペースを適宜入れる事ができます．次の例は，上記の例と同じ意味です．

```
begin
crit : eps = 1.0e-12
end
```

求解オプションファイルは，複数の求解オプションを指定する事もできます．以下の例では，停止条件を 10^{-4} に設定し，アルゴリズムに線形計画専用内点法 higher を設定しています．

```
begin
crit:eps = 1.0e-4
method:higher
end
```

行頭の半角アスタリスク*は，その行がコメント文であることを意味します．例えば次の例では 3 行目の method:higher が読み込まれません．

```
begin
crit:eps = 1.0e-4
*method:higher
end
```

求解オプションファイルで設定する値には整数や小数のほか，

```
3.4e-3   4.562384E-2
```

のような浮動小数点表記が許されています．

求解オプションファイルが読み込まれた場合，標準出力には求解オプションファイルを読み込んだ

ことを示すメッセージと内容が表示されます．求解オプションファイル中の空白，コメントは無視されます．以下は，求解オプションファイルが読み込まれた場合の出力例です．

求解オプションファイル

```
begin
maximize
method:tipm
scaling:on
crit:eps = 1.0e-8
end
```

標準出力

```
<reading solver option file: nuopt.prm>
nuopt.prm:1:     begin
nuopt.prm:2:     maximize
nuopt.prm:3:     method:tipm
nuopt.prm:4:     scaling:on
nuopt.prm:5:     crit:eps = 1.0e-8
nuopt.prm:6:     end


        ...
```

5.1.2 PySIMPLE マニュアルにおける求解オプション

モデリング言語 PySIMPLE では問題クラスの options 属性を介して求解オプションを設定します．詳細は PySIMPLE マニュアル「3.6. 求解オプション」（https://www.msi.co.jp/solution/nuopt/docs/pysimple/guide/options.html）をご参考ください．

5.1.3 C++SIMPLE マニュアルにおける求解オプション

モデリング言語 C++SIMPLE では options を介して求解オプションを設定します．詳細は C++SIMPLE マニュアル「9.2 求解オプション」(https://www.msi.co.jp/solution/nuopt/docs/SIMPLE/html/09-02-00.html) をご参考ください．

特に，options のメンバのうち，C++SIMPLE を制御する以下のオプションに関しては C++SIMPLE マニュアルをご参照ください．

- options.noDefaultSolout
- options.noDefaultSolve
- options.outfilename
- outputExpression

- outputParameter

5.1.4 RSIMPLE マニュアルにおける求解オプション

モデリング言語 RSIMPLE では関数 nuopt.options() を介して求解オプションを設定します．詳細は RSIMPLE マニュアル「8.1 nuopt.options() を使って NUOPT をカスタマイズする」をご参考ください．

5.2 共通求解オプション

本節では，求解アルゴリズムに依存しない求解オプションについて解説します．

5.2.1 最適化計算における解法選択

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.method	シンボル
C++SIMPLE	options.method	文字列値
RSIMPLE	options.method	文字列値
nuopt.prm	method:[value]	文字列値

設定値

型	文字列値	シンボル
デフォルト値	auto	Method.AUTO
値範囲	{auto, lipm, higher, tipm, bfgs, simplex, hsimplex, asqp, lsqp, tsqp, lsdp, trsdp, wcsp, wls, rcpsp}	{Method.AUTO, Method.LIPM, Method.HIGHER, Method.TIPM, Method.BFGS, Method.SIMPLEX, Method.HSIMPLEX, Method.ASQP, Method.LSQP, Method.TSQP, Method.WCSP, Method.WLS}

文字列値	シンボル	意味
auto	Method.AUTO	解法を自動で選択する
auto 以外	Method.AUTO 以外	指定値で解法を選択する

詳細

- 本オプション値が auto/Method.AUTO の場合は解法を自動で選択します．本オプション値が auto/Method.AUTO 以外の場合は指定された値で解法を選択します．
- 指定値と解法（アルゴリズム）の対応は以下となります．

asqp	Method.ASQP	有効制約法（+ 分枝限定法）
bfgs	Method.BFGS	準ニュートン法
higher	Method.HIGHER	線形計画専用内点法
hsimplex	Method.HSIMPLEX	スパース単体法
lipm	Method.LIPM	直線探索内点法
lsdp	-	線形半正定値計画問題に対する主双対内点法
lsqp	Method.LSQP	直線探索を利用した逐次二次計画法
rcpsp	-	資源制約付きスケジューリング問題ソルバ
simplex	Method.SIMPLEX	単体法（+ 分枝限定法）
tipm	Method.TIPM	信頼領域内点法
trsdp	-	信頼領域法を用いた非線形半正定値計画問題に対する主双対内点法
tsqp	Method.TSQP	信頼領域を利用した逐次二次計画法
wcsp	Method.WCSP	wcsp タブーサーチ
wls	Method.WLS	重み付き局所探索法 WLS

- 本オプション値で指定された解法と与えられた最適化問題の種類は符合しない場合はエラーとなります．例えば非線形計画問題に単体法（ simplex ）を指定すると以下のようなエラーメッセージが出力されます.

```
(NUOPT 15) simplex/asqp misapplied to NLP.
```

5.2.2 反復回数上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.maxIteration
C++SIMPLE	options.maxitn
RSIMPLE	options.maxitn
nuopt.prm	crit:maxitn

設定値

型	整数値
デフォルト値	-10
最小値	無制限
最大値	無制限

値	意味
< 0	無制限あるいは自動設定
0	反復を行わない
> 0	指定値を反復回数上限とする

詳細

- 停止条件として用いる反復計算の回数の上限です.
- 反復回数を負の値に設定すると，無制限あるいは自動設定されます．解法毎に，無制限になる・自動設定されるのいずれになるかは異なり，自動設定される場合はその値も異なります.

5.2.3　計算時間上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.maxTime
C++SIMPLE	options.maxtim
RSIMPLE	options.maxtim
nuopt.prm	crit:maxtim

設定値

型	実数値
デフォルト値	-1
最小値	-1
最大値	無制限

値	意味
-1	無制限
>= 0	計算時間上限（秒）

詳細

- 停止条件となる計算時間上限を定めます．最適化計算が指定値を超過すると計算が停止します.
- 計算時間は CPU 時間ではなく実時間で計測されます.
- 負の値が設定された場合は無制限となります.
- 本オプション値を正の値で設定した場合でも，指定された計算時間上限を僅かに超えて停

止する場合があります． このような場合は以下が想定されます．

- 計算上限超過で最適化解法が停止した場合でも，その後処理をする必要がある場合があります （得られた実行可能解を解ファイルに出力する等）．この後処理で時間を要する場合があります．
- 最適化解法において，計算時間超過の確認がない箇所において時間を要している場合があります．
- 並列化処理の場合は，全てのスレッドが停止するのを待つ必要があります． そのため停止にオーバーヘッドが生じる場合があります．

設定した計算時間上限と実際の計算時間に大きな隔たりがある場合は nuopt-support@ml.msi.co.jp までご相談ください．

5.2.4 求解情報の表示制御

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.solve(silent=...)	真偽値
C++SIMPLE	options.outputMode	文字列
RSIMPLE	solve(trace=...)	真偽値
nuopt.prm	output:mode	文字列

設定値

型	文字列	真偽値
デフォルト値	normal	True
値範囲	{normal, silent}	{True, False}

文字列	真偽値（PySIMPLE）	真偽値（RSIMPLE）	意味
normal	True	FALSE	求解情報の表示を行う
silent	False	TRUE	求解情報の表示を行わない

詳細

- 本オプション値で "normal"（PySIMPLE では True, RSIMPLE では FALSE）を設定すると，最適化計算で求解情報が標準出力に表示されます．出力される内容は主に以下です．
  - Nuorium Optimizer のバージョン情報
  - 変数の数や制約式の数等の問題情報
  - 最適化計算の進捗
  - 最適化計算の結果
- 本オプション値で "silent" を設定すると求解情報の表示が抑制できます．

- 本オプションは最適化計算の表示抑制を行います．モデリング言語自体の表示（ファイルの読み込み表示等）は抑制できない場合がありますのでご注意ください．
- 最適化計算において入出力のオーバーヘッドが大きい場合は，本オプションで表示を抑制することにより実行時間を短縮できる場合があります．
- PySIMPLE においてはオプションではなく solve 関数の引数 silent で設定します．詳細は PySIMPLE マニュアルでご確認ください．
- RSIMPLE においてはオプションではなく solve 関数の引数 trace で設定します．詳細は RSIMPLE マニュアルでご確認ください．

5.2.5　スケーリング

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.scaling	シンボル
C++SIMPLE	options.scaling	文字列値
RSIMPLE	options.scaling	文字列値
nuopt.prm	scaling:[value]	文字列値

設定値

型	文字列値	シンボル
デフォルト値	auto	Scaling.AUTO
値範囲	{auto, off, on, minmax, cr}	{Scaling.AUTO, Scaling.OFF, Scaling.ON, Scaling.MINMAX, Scaling.CR}

値	意味
auto	内部で自動選択する
off	スケーリング処理は行わない
on	minmax と等価
minmax	min-max スケーリング
cr	係数行列全体について，非零要素の絶対値の対数の 2 乗和を最小化する

詳細

- 最適化計算を効率よく安定に動作させるため，目的関数，制約式，変数に定数値を乗じる処置がスケーリングです．
- 本オプションによってスケーリングの種類を指定することができます．
- 本オプション値を "auto" にすることにより，内部で自動的にスケーリング処理が選択されます．

- 本オプション値を "off" にすることにより，スケーリングを避けることができます．スケーリングによって問題が変形しすぎて数値エラーを起こしている場合は "off" にすると解決する場合があります．
- 本オプション値を "on" とすると "minmax" と設定した場合と等価なスケーリングが施されます．
- 本オプション値を "minmax" とすると "min-max スケーリング" が施されます．係数行列の各行と列について，非零要素の絶対値の最大値と最小値との幾何平均が 1 になるように施されます．
- 本オプション値を "cr" とすると "Curtis-Reid スケーリング" が施されます．係数行列全体について，非零要素の絶対値の対数の 2 乗和を最小になるようスケーリングが施されます．
- スケーリングの技術詳細については [20] の「7.2 Scaling」をご参考ください．

5.2.6 実行不可能性要因検出機能 iisDetect

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.solve(iis=...)	真偽値
C++SIMPLE	options.iisDetect	文字列
RSIMPLE	なし	-
nuopt.prm	param:iis	文字列

設定値

型	文字列	真偽値
デフォルト	on	True
値範囲	{on, off}	{True, False}

文字列	真偽値	意味
on	True	実行不可能性要因検出機能を動作する
off	False	実行不可能性要因検出機能を動作しない

詳細

- 実行不可能性要因検出機能 iisDetect は実行不可能性の原因の探索を行います．
- 以下の解法においては iisDetect 機能は動作しません．
  - wcsp タブーサーチ
  - 資源制約付きスケジュール問題ソルバ（rcpsp ソルバ）
  - 重み付き局所探索法 WLS
- 実行不可能性要因検出機能 iisDetect はソフト制約を含むモデルでは動作しません．

- 実行不可能性要因検出機能 iisDetect は，大規模問題で起動された場合時間を要する可能性があります．この場合は本オプションを off/False に設定することを推奨します．
- 実行不可能性要因検出機能 iisDetect の出力については「2.9 実行不可能性要因検出機能（iisDetect）の出力」をご参考ください．
- PySIMPLE においては iisDetect 機能はオプションではなく solve 関数の引数 iis で設定します．

関連

- 2.9 実行不可能性要因検出機能（iisDetect）の出力

5.3 内点法及び逐次二次計画法用求解オプション

本節では内点法及び逐次二次計画法で有効な求解オプションについて解説します．
以下は本節で対象となる解法一覧です．

- 内点法（higher/lipm/tipm）
- 逐次二次計画法（lsqp/tsqp）
- 半正定値計画専用内点法（lsdp/trsdp）

5.3.1 線形計画専用内点法から単体法へのクロスオーバー

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.higherCrossover	真偽値
C++SIMPLE	options.crossover	文字列
RSIMPLE	options.crossover	文字列
nuopt.prm	cross:[value]	文字列

設定値

型	文字列	真偽値
デフォルト	off	False
値範囲	{off,on}	{False, True}

文字列	真偽値	意味
off	False	単体法へのクロスオーバーを起動しない
on	True	単体法へのクロスオーバーを起動する

詳細

- 本オプションは解法として線形計画専用内点法（ higher ）を選んだ場合のみで有効です．

- 本オプションでは，線形計画専用内点法（higher）によって得られた解の情報をもとにして単体法を起動することができます.
- 線形計画専用内点法（higher）では大規模な線形計画問題が単体法と比較して容易に解ける反面，基底を得ることができません．本オプションを用いることにより内点法で得られた解を起点として単体法を実行するという「単体法へのクロスオーバー」を実行することができます.
- 大規模線形計画問題で非零要素ができるだけ少ない解を得る等の場合，本オプションが有効です.

関連

- 5.2.1 最適化計算における解法選択

5.3.2 内点法及び逐次二次計画法における KKT 条件残差停止条件

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.kktEps
C++SIMPLE	options.eps
RSIMPLE	options.eps
nuopt.prm	crit:eps

設定値

型	実数値
デフォルト値	自動設定
最小値	> 0.0
最大値	1.0e-4

値	意味
任意値	最適性条件の残差が指定値を下回ったら反復を停止する

詳細

- 内点法及び逐次二次計画法における停止条件として用いる最適性条件の残差です.
- 最適性条件の残差が本オプション値以下になったときに，計算が収束したとみなして反復計算を終了します.
- デフォルトでは，解法ごとに内部で自動決定されます.
- 線形計画専用内点法（higher）では「最適性条件の残差」及び「双対ギャップ」の大きい方の値が本オプション値以下になった場合，反復計算を終了します.

5.3.3 内点法における初期値からの探索

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	なし
C++SIMPLE	options.ipmInitialValueActivation
RSIMPLE	なし
nuopt.prm	ipm:initialValueActivation

設定値

型	文字列値
デフォルト	off
値範囲	{off,on}

値	意味
off	内点法において初期値からの探索を有効にしない
on	内点法において初期値からの探索を有効にする

詳細

- ユーザ指定による初期値を内点法の初期値として用いるかどうかを指定できます.
- 本オプションは内点法のみ有効です.
- 本オプションのデフォルト値は "off" となっているため，ユーザ指定による初期値から探索をスタートする場合は明示的に "on" （有効にする）という設定をする必要があります.

5.3.4 Matrix Free ／線形計画専用内点法における連立一次方程式を反復法で解く

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	なし
C++SIMPLE	options.mtxfree
RSIMPLE	なし
nuopt.prm	linear:mtxfree

設定値

型	文字列
デフォルト値	off
値範囲	{on, off}

値	意味
off	反復法で解く設定をしない
on	反復法で解く設定をする

詳細

- 本オプションは解法として線形計画専用内点法（ higher ）を選んだ場合のみで有効です.
- 本オプションを有効にし，反復法で解く設定をすると，内点法において探索方向を求めるために解く連立一次方程式を反復法（クリロフ部分空間法）を用いて解きます.
- 本オプションを有効にすることにより，メモリ使用量を減らすことができる可能性があります.
- 本オプションは下記の制限が伴います.
  - 単体法へのクロスオーバーと併用することができません.
  - 上界も下界も設定されていない変数（自由変数）がある場合は使用できません.
  - 実行不可能性要因検知機能（iisDetect）と併用することができません.

関連

- 5.2.1 最適化計算における解法選択

5.4 単体法用求解オプション

本節では線形計画法（線形計画問題に対する解法）の一つである単体法で有効な求解オプションについて解説します.

以下は本節で対象となる解法一覧です.

- 単体法（ simplex ）
- 有効制約法（ asqp ）
- スパース単体法（ hsimplex ）

5.4.1 主変数実行可能性判定値

概要

- 主変数実行可能性の判定値

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.simplexPrimalTolerance
C++SIMPLE	options.tolx
RSIMPLE	options.tolx
nuopt.prm	simplex:tolx

設定値

型	実数値
デフォルト値	1.0e-8
最小値	0.0
最大値	1.0e-4

値	意味
任意値	主変数について上下界を指定値の範囲内で満たしていたら実行可能と判定する

詳細

- 本オプション値は主変数の上下限違反判定及び制約式違反に関する閾値です．変数値に関する上下限違反は本オプション値以下まで許容されます．制約式に関する違反も本オプション値以下まで許容されます．
- スケーリング適用後の問題に対して適用されます．

関連

- 5.4.2 双対変数実行可能性判定値
- 5.2.5 スケーリング

5.4.2　双対変数実行可能性判定値

概要

- 双対変数実行可能性の判定値

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.simplexDualTolerance
C++SIMPLE	options.told
RSIMPLE	options.told
nuopt.prm	simplex:told

設定値

型	実数値
デフォルト値	1.0e-8
最小値	0.0
最大値	1.0e-4

値	意味
任意値	双対変数について上下界を指定値の範囲内で満たしていたら実行可能と判定する

詳細

- 本オプション値は双対変数の上下限違反判定に関する閾値です．変数値に関する上下限違反は本オプション値以下まで許容されます．
- 主変数の実行可能性判定値と同様，スケーリング適用後の問題に対して適用されます．

関連

- 5.4.1 主変数実行可能性判定値
- 5.2.5 スケーリング

5.4.3 単体法解法選択（解法 hsimplex のみ）

概要

- スパース単体法における主単体法・双対単体法を選択する．

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.hsimplexMethod	シンボル
C++SIMPLE	なし	-
RSIMPLE	なし	-
nuopt.prm	hsimplex:method	整数値

設定値

型	整数値	シンボル
デフォルト値	-1	HsimplexMethod.AUTO
最小値	-1	-
最大値	1	-
値範囲	-	{HsimplexMethod.PRIMAL, HsimplexMethod.DUAL, HsimplexMethod.AUTO}

整数値	シンボル	意味
-1	HsimplexMethod.AUTO	解法 hsimplex において解法を自動選択する
0	HsimplexMethod.PRIMAL	解法 hsimplex において主単体法を選択する
1	HsimplexMethod.DUAL	解法 hsimplex において双対単体法を選択する

詳細

- スパース単体法 hsimplex では主単体法と双対単体法の二種類があります.
  - 主単体法はまず（主）実行可能解を求め，その後最適性を満たす解を探索します.
  - 双対単体法では（主）実行可能解をまず求めることはしません．双対実行可能となる解を求め，その後（主）実行可能となる解を探索します.
- 本オプションによりユーザはスパース単体法における主単体法と双対単体法を切り替えることができます.

5.5 解法 wcsp タブーサーチ（ wcsp ）に有効な求解オプション

本節では解法 wcsp タブーサーチ（ wcsp ）で有効な求解オプションについて解説します.

- 解法 wcsp タブーサーチは制約をできるだけ充足する解をいずれか一つ求めるという手法で，厳密な最適解を求めるものではありません.
- 解法 wcsp タブーサーチは「制約充足フェーズ」と「ソフト制約違反最小化フェーズ」で求解します.
  - 「制約充足フェーズ」は，ハードペナルティ及びセミハードペナルティが残っているフェーズです.
  - 「ソフト制約違反最小化フェーズ」は，ハードペナルティ及びセミハードペナルティが 0 としたままソフト制約違反を最小化するフェーズです.
  - 「制約充足フェーズ」においてハードペナルティ及びセミハードペナルティが 0 になると，「ソフト制約違反最小化フェーズ」に移行します.

5.5.1 解法 wcsp タブーサーチにおける乱数シード値

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.randomSeed
C++SIMPLE	options.wcspRandomSeed
RSIMPLE	options.wcspRandomSeed
nuopt.prm	wcsp:randomSeed

設定値

型	整数値
デフォルト値	1
最小値	1
最大値	無制限

値	意味
任意値	解法 wcsp タブーサーチにおける乱数シード値

詳細

- 解法 wcsp タブーサーチでは初期解生成時に乱数を使用します．本オプション値で乱数のシード値を設定できます．
- 解法 wcsp タブーサーチでは初期解が最終的に得られる解の精度に影響します．初期解を変更することによって，より良い解が得られる場合があります．

関連

- 5.5.2 解法 wcsp タブーサーチにおける試行回数

5.5.2 解法 wcsp タブーサーチにおける試行回数

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.tryCount
C++SIMPLE	options.wcspTryCount
RSIMPLE	options.wcspTryCount
nuopt.prm	wcsp:tryCount

設定値

型	整数値
デフォルト値	1
最小値	1
最大値	無制限

値	意味
>= 1	解法 wcsp タブーサーチにおける試行回数

詳細

- 解法 wcsp タブーサーチでは初期解を変更することによって最終的に得られる解が変化することがあります．初期解生成時に使用する乱数を変更し何回か解くことによって解の精度向上が期待されます．
- 本オプション値を用いて計算回数を複数回に設定した場合には，初期解生成時に用いる乱数を変更し，指定した回数だけ求解を行います．その中で最良解が解出力されます．

関連

- 5.5.1 解法 wcsp タブーサーチにおける乱数シード値

5.5.3　解法 wcsp タブーサーチにおけるスレッド数上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.threads
C++SIMPLE	options.wcspthreads
RSIMPLE	なし
nuopt.prm	wcsp:threads

設定値

型	整数値
デフォルト値	1
最小値	-1
最大値	無制限

値	意味
-1	適切なスレッド数を自動的に設定する
0	単一のスレッドで実行する
1	単一のスレッドで実行する
>= 2	指定された値のスレッド数で実行する

詳細

- 解法 wcsp タブーサーチは並列処理を行うことができ，異なる初期値から探索を独立に行います．探索同士での情報のやりとり（通信）はありません．この並列処理により，マルチコア環境において複数試行を効率よく行うことができます．
- 本オプション値は解法 wcsp タブーサーチにおける並列処理のスレッド数上限（並列数上限）を定めます．
- 本求解オプションは「試行回数」と合わせて利用します．例えば，計算回数を 8 回，スレッド数上限を 2 に設定する場合，C++SIMPLE では以下のよう記述します．

```
options.wcspTryCount = 8;
options.wcspthreads = 2;
```

上記のように設定をすることにより，各スレッドで試行回数を 4（= 8 / 2） 回とする実行が行われます．

- 推奨されるスレッド数は，実行する計算機の物理コア数以下となります．同時マルチスレッディング技術により論理コア数が物理コア数よりも多くなっている場合がありますが，この場合も物理コア数以下に設定することが推奨されます．
- 本求解オプションのデフォルト値は 1 となっているため，並列化を有効にする場合は 2 以上の値を設定してください．

関連

- 5.5.2 解法 wcsp タブーサーチにおける試行回数

5.5.4 解法 wcsp タブーサーチにおける制約充足フェーズにおける計算時間上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.wcspPhaseOneMaxTime
C++SIMPLE	options.wcspPhaseOneMaxTime
RSIMPLE	なし
nuopt.prm	なし

設定値

型	整数値
デフォルト値	-1
最小値	-1
最大値	無制限

値	意味
-1	無制限
>= 0	制約充足フェーズの計算時間上限（秒）

詳細

- 本オプションが設定された場合，その設定値に経過時間が達した時点で，強制的に「ソフト制約違反最小化フェーズ」に移行します．すなわち，ハード・セミハードペナルティが残存していたとしても「ソフト制約違反最小化フェーズ」に移行します．
- 「ソフト制約違反最小化フェーズ」に移行した後の探索は，ハード・セミハードペナルティ

の改善よりも，ソフトペナルティの改善を目指して行われます.

- ある一定の時間が経過した後にハード・セミハードペナルティが改善することはないとわかっている問題に対して本オプションを使用することで，よりソフトペナルティの小さい解が得られる可能性があります.

5.5.5　解法 wcsp タブーサーチにおける制約充足フェーズにおける反復回数上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.wcspPhaseOneMaxIteration
C++SIMPLE	options.wcspPhaseOneMaxIteration
RSIMPLE	なし
nuopt.prm	なし

設定値

型	整数値
デフォルト値	-1
最小値	-1
最大値	無制限

値	意味
-1	無制限
>= 0	制約充足フェーズの反復回数上限

詳細

- 本オプションを設定した場合，反復回数が設定値に達した時点で，強制的に「ソフト制約違反最小化フェーズ」に移行します．すなわち，ハード・セミハードペナルティが残存していたとしても「ソフト制約違反最小化フェーズ」に移行します.
- 「ソフト制約違反最小化フェーズ」に移行した後の探索は，ハード・セミハードペナルティの改善よりも，ソフトペナルティの改善を目指して行われます.
- ある一定の時間が経過した後にハード・セミハードペナルティが改善することはないとわかっている問題に対して本オプションを使用することで，よりソフトペナルティの小さい解が得られる可能性があります.

5.5.6 解法 wcsp タブーサーチにおける解更新間隔計算時間上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.wcspPhaseTwoMaxIntervalTime
C++SIMPLE	options.wcspPhaseTwoMaxIntervalTime
RSIMPLE	なし
nuopt.prm	なし

設定値

型	整数値
デフォルト値	-1
最小値	-1
最大値	無制限

値	意味
<= 0	無制限
> 0	解更新間隔計算時間上限（秒）

詳細

- 本オプション値は制約充足フェーズの終了時から有効となります．すなわち，ハードペナルティ及びセミハードペナルティが 0 になった段階で有効になります．
- 本オプション値は解更新間隔計算時間の上限を定めます．ソフト制約違反最小化フェーズにおいて，解が更新されてから計算時間が本オプション値を超過すると計算を終了します．
- 本オプション値が 0 または負の場合は，解更新間隔計算時間に上限は設定されません．
- 大規模問題に対して wcsp タブーサーチを適用すると，計算時間の適切な設定が難しい場合がありますが，本オプションを用いることによって柔軟に計算を行うことができます．

関連

- 5.5.7 解法 wcsp タブーサーチにおける解更新間隔反復回数上限

5.5.7　解法 wcsp タブーサーチにおける解更新間隔反復回数上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.wcspPhaseTwoMaxIntervalIteration
C++SIMPLE	options.wcspPhaseTwoMaxIntervalIteration
RSIMPLE	なし
nuopt.prm	なし

設定値

型	整数値
デフォルト値	-1
最小値	-1
最大値	無制限

値	意味
<= 0	無制限
> 0	解更新間隔反復回数上限

詳細

- 本オプション値は制約充足フェーズの終了時から有効となります．すなわち，ハードペナルティ及びセミハードペナルティが 0 になった段階で有効になります．
- 本オプション値は解更新間隔反復回数の上限を定めます．ソフト制約違反最小化フェーズにおいて，解が更新されてから反復回数が本オプション値を超過すると計算を終了します．
- 本オプション値が 0 または負の場合は，解更新間隔反復回数に上限は設定されません．
- 大規模問題に対して wcsp タブーサーチを適用すると，計算時間の適切な設定が難しい場合がありますが，本オプションを用いることによって柔軟に計算を行うことができます．

関連

- 5.5.6 解法 wcsp タブーサーチにおける解更新間隔計算時間上限

5.5.8 解法 wcsp タブーサーチにおける初期値からの探索

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.wcspInitialValueActivation	真偽値
C++SIMPLE	options.wcspInitialValueActivation	文字列値
RSIMPLE	なし	-
nuopt.prm	wcsp:initialValueActivation	文字列値

設定値

型	文字列値	真偽値
デフォルト	off	False
値範囲	{off, on}	{False, True}

文字列値	真偽値	意味
off	False	初期値からの探索を有効にしない
on	True	初期値からの探索を有効にする

詳細

- ユーザ指定による初期値から wcsp の探索をスタートするか否かを指定することができます.
- 本オプションのデフォルト値は off/False となっているため，ユーザ指定による初期値から探索をスタートする場合は明示的に on/True （有効にする）という設定をする必要があります.
- 解法 wcsp タブーサーチにおける試行回数を 2 以上に設定した場合，全ての回においてユーザ指定による初期値から探索を出発します.
- モデルが離散変数（DiscreteVariable）を含む場合，本オプション値の指定は無視されます.

関連

- 5.5.2 解法 wcsp タブーサーチにおける試行回数

5.6 解法 wls に有効な求解オプション

本節では重み付き局所探索法 wls で有効な求解オプションについて解説します.

5.6.1　解法 wls におけるメモリ量上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.maxMemory
C++SIMPLE	なし
RSIMPLE	なし
nuopt.prm	wls:maxmem

設定値

型	整数値
デフォルト値	-1
最小値	-1
最大値	無制限

値	意味
< 0	無制限
>= 0	メモリ量上限（MiB）

詳細

- 本オプション値は解法 wls において最大メモリ量を制限します.
- 本オプション値は MiB 単位で指定します．例えば 1024 とすると 1GiB を上限とすることを意味し，使用メモリ量が 1GiB を超えた場合に実行を停止します.

5.6.2　解法 wls における目的関数の目標値

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.objectiveTarget
C++SIMPLE	なし
RSIMPLE	なし
nuopt.prm	wls:objectiveTarget

設定値

型	実数値
デフォルト値	未定義
最小値	無制限
最大値	無制限

値	意味
任意値	目的関数の目標値

詳細

- 本オプション値は解法 wls における目的関数の目標値を設定します.
- 設定した場合，探索中に「すべての制約条件を満たし，かつ目的関数値が目標値より良い解（最小化問題であれば目標値以下の解)」が得られたら探索を終了します.

5.6.3 解法 wls における試行回数

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.tryCount
C++SIMPLE	なし
RSIMPLE	なし
nuopt.prm	wls:tryCount

設定値

型	整数値
デフォルト値	1
最小値	1
最大値	無制限

値	意味
>= 1	指定値だけ解法 wls が試行を行う

詳細

- 本オプションによって解法 wls の試行回数を定めることができます．各試行では初期解が変更され，これにより最終的に得られる解が変化する可能性があります.
- 試行回数を増やすことによって，解の精度向上が期待されます.

5.6.4　解法 wls におけるスレッド数上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.threads
C++SIMPLE	なし
RSIMPLE	なし
nuopt.prm	wls:threads

設定値

型	整数値
デフォルト値	1
最小値	-1
最大値	無制限

値	意味
−1	適切なスレッド数を自動的に設定する
0	単一のスレッドで実行する
1	単一のスレッドで実行する
>= 2	指定された値のスレッド数で実行する

詳細

- 解法 wls は並列処理を行うことができ，異なる初期値，異なる内部パラメータ設定を用いて探索を行います．探索同士での情報のやりとり（通信）も行い，マルチコア環境において複数試行を効果的に行うことができます．
- 本オプション値は解法 wls における並列処理のスレッド数上限（並列数上限）を定めます．
- 本求解オプションは「試行回数」と合わせて利用できます．例えば，「試行回数」を 8 回，「スレッド数上限」を 2 に設定する場合，PySIMPLE では以下のよう記述します．

```
problem.options.tryCount = 8;  # problem は Problem クラスのインスタン
ス
problem.options.threads  = 2;
```

上記のように設定することにより，各スレッドで試行回数を $4 = \frac{8}{2}$ 回とする実行が行われます．終了条件を「計算時間上限」とする場合，単一のスレッドでの実行と比べおよそ半分の実行時間で計算が完了します．

- 推奨されるスレッド数は，実行する計算機の物理コア数以下となります．同時マルチスレッディング技術により論理コア数が物理コア数よりも多くなっている場合があります，この

場合も物理コア数以下に設定することが推奨されます.

- 本求解オプションのデフォルト値は 1 となっているため，並列化を有効にする場合は 2 以上の値を設定してください.
- 解法 wls の並列化計算は基本的に非決定的（計算する度に計算過程が異なるため結果も異なる）であるため，再現性が重要となるアプリケーションには推奨されません.
- **[ADVANCED]** 解法 wls の並列化計算における各スレッドの探索は，「初期値」と「探索戦略」の組合せが自動的に設定されます.
  - 「探索戦略」とは探索における内部パラメータ設定を表し，「実行可能解の発見を促進する」等，探索ごとの性格に対応します．各スレッドで異なる探索戦略を用いることで，並列化計算における探索性能の向上を図ります.
  - 現時点では 3 パターンの探索戦略が実装されています.
  - 「試行回数」の設定値が 1 の場合，3 パターンの探索戦略が各スレッドで実行されます．スレッドが残っている場合は，初期値を変えて探索戦略を割り振ります.
    - 例えば，「試行回数」を 1 回，「スレッド数上限」を 5 に設定する場合，各スレッドにおける探索は以下のように設定されます.

```
スレッド 1：[T1, S1]
スレッド 2：[T1, S2]
スレッド 3：[T1, S3]
スレッド 4：[T2, S1]
スレッド 5：[T2, S2]
```

ここで [T, S] は，「初期値が T」であり，「探索戦略が S」であることを表します．例えばスレッド 1 では，初期値は T1，探索戦略は S1 となります.

  - 「試行回数」の設定値が 2 以上の場合，「試行回数」による複数試行を優先して割り振ります．スレッドが残っている場合は，まだ試していない探索設定を割り振ります.
    - 例えば，「試行回数」を 6 回，「スレッド数上限」を 5 に設定する場合，以下のとおりに探索設定が割り振られます.

```
スレッド 1：[T1, S1] -> [T6, S1]
スレッド 2：[T2, S1] -> [T1, S2]
スレッド 3：[T3, S1] -> [T1, S3]
スレッド 4：[T4, S1] -> [T7, S1]
スレッド 5：[T5, S1] -> [T2, S2]
```

ここで [T, S] -> [T', S'] は，[T, S] の実行完了後に [T', S'] が実行されることを表します．各実行は全スレッドの探索終了を待ち，全スレッドで同時に完了します.

関連

- 5.6.3 解法 wls における試行回数
- 5.2.3 計算時間上限
- 5.2.2 反復回数上限

5.7 分枝限定法用求解オプション

本節では整数計画法（整数計画問題に対する解法）の一つである分枝限定法で有効な求解オプションについて解説します.

分枝限定法は，整数変数の整数条件を一部緩和した問題（部分問題）を繰り返し解きながら，整数条件を満たす実行可能解の発見と及び解の最適性の保証を行います.

連続変数のみからなる同程度の規模の問題に比べて，数倍あるいは数十倍の計算時間を要する場合があります．また計算時間（最適性の証明に要するまでの時間）が問題の規模のみならず，問題の性質に大きく左右されるということがあります．数万変数の問題が数秒で解けてしまったり，一方では数百変数の問題を解くのに一時間以上を所要したりということがあります.

Nuorium Optimizer に実装されている分枝限定法は多様な問題が安定にとけるようにチューニングしていますが，あらゆる性質の問題に対して常に良い設定を見出すとは限りません．このため求解オプションの適切な設定により，効率良く求解ができる可能性があります．本節ではこのようなチューニングに役立つ求解オプションをご紹介します.

説明の便宜のため，本節では解いている問題が最小化問題だと仮定している場合があります．最大化問題は目的関数の符号を逆にして最小化を行っていることと等価なので，意味は同じですが以下に出てくる下界値と上界値という言葉を逆転させて解釈する必要があります.

以下は本節で対象となる解法一覧です.

- 単体法 + 分枝限定法 (simplex)
- 有効制約法 + 分枝限定法 (asqp)

5.7.1 分枝限定法における切除平面の強度

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.branchCut	シンボル
C++SIMPLE	options.clevel	整数値
RSIMPLE	options.clevel	整数値
nuopt.prm	branch:clevel	整数値

設定値

型	整数値	シンボル
デフォルト値	1	Branch.Cut.ON
最小値	0	-
最大値	2	-
値範囲	-	{Branch.Cut.OFF, Branch.Cut.ON, Branch.Cut.AGGRESSIVE}

整数値	シンボル	意味
0	Branch.Cut.OFF	切除平面を追加しない
1	Branch.Cut.ON	切除平面を追加する
2	Branch.Cut.AGGRESSIVE	切除平面をより多く追加する

詳細

- 切除平面法は実行可能解が満たす線形な制約式を内部で生成することにより，緩和問題の目的関数値を改良することにより分枝限定法の限定操作を強める手法です.
- 切除平面法により分枝限定法は早期に停止することが期待されますが，加えすぎると扱う問題の規模が増大するため，緩和解の計算コストがかさみ，結局実行時間の増大につながる可能性もあります.
- 切除平面が多く生成されていて分枝限定法が遅くなる場合には本オプションを 0 に設定することを推奨します.
- 切除平面が抑制されている場合には本オプションを 2 に設定することにより，分枝限定法の速度が向上する可能性があります.

5.7.2 分枝限定法における前処理

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.branchPresolve	シンボル
C++SIMPLE	options.branchPresolve	整数値
RSIMPLE	options.branchPresolve	整数値
nuopt.prm	branch:presolve	整数値

設定値

型	整数値	シンボル
デフォルト値	-1	Branch.Presolve.AUTO
最小値	-1	-
最大値	1	-
値範囲	-	{Branch.Presolve.AUTO, Branch.Presolve.OFF, Branch.Presolve.ON}

整数値	シンボル	意味
-1	Branch.Presolve.AUTO	自動決定
0	Branch.Presolve.OFF	前処理を行わない
1	Branch.Presolve.ON	前処理を行う

詳細

- 前処理は，与えられた問題を変形することによって，分枝限定法の速度向上を図る手法です.
- 前処理によって問題サイズが小さくなることなどにより，実行時間が短くなることが期待されますが前処理による数値誤差が数値的問題を引き起こす可能性があります．例えば，実行可能な問題が実行不可能と判定されたり，最適値と異なる値が最適値と判定されたりする可能性があります.
- 数値的問題が引き起こされる場合は，本オプションを 0/Branch.Presolve.OFF に設定することにより回避できる可能性があります，ただしそのような設定は速度低下を招くこともあるので注意が必要です.

5.7.3　分枝限定法における発見的探索 diving

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.branchDiving	シンボル
C++SIMPLE	options.rounding	整数値
RSIMPLE	なし	-
nuopt.prm	branch:rounding	整数値

設定値

型	整数値	シンボル
デフォルト値	-1	Branch.Diving.AUTO
最小値	-1	-
最大値	3	-
値範囲	-	{Branch.Diving.AUTO, Branch.Diving.OFF, Branch.Diving.ON, Branch.Diving.AGGRESSIVE, Branch.Diving.SUPERAGGRESSIVE}

整数値	シンボル	意味
-1	Branch.Diving.AUTO	自動決定
0	Branch.Diving.OFF	diving を実行しない
1	Branch.Diving.ON	diving を実行する
2	Branch.Diving.AGGRESSIVE	diving を高い強度で実行する
3	Branch.Diving.SUPERAGGRESSIVE	diving をより高い強度で実行する

詳細

- 分枝限定法における発見的探索 diving は，分枝限定法において変数固定を逐次的に行うことによって実行可能解を得る手法です.

- 本オプション値が -1/Branch.Diving.AUTO の場合は実行する・しないおよび diving の強度が自動決定されます.
- 本オプション値が 0/Branch.Diving.OFF の場合は diving は実行されません.
- 本オプション値が 1 以上の場合（あるいは Branch.Diving.AGGRESSIVE/Branch.Diving.SUPERAGGRESSIVE の場合）は，diving が実行されます．指定の強度で探索が行われ，本オプション値が大きいほど，より頻繁に探索が行われます.

5.7.4 分枝限定法における発見的探索 feasibility pump

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.branchFeasPump	シンボル
C++SIMPLE	options.feasPump	整数値
RSIMPLE	なし	-
nuopt.prm	branch:feas	整数値

設定値

型	整数値	シンボル
デフォルト値	-1	Branch.FeasPump.AUTO
最小値	-1	-
最大値	1	-
値範囲	-	{Branch.FeasPump.OFF, Branch.FeasPump.ON, Branch.FeasPump.AUTO}

整数値	シンボル	意味
-1	Branch.FeasPump.AUTO	自動決定
0	Branch.FeasPump.OFF	feasibility pump を実行しない
1	Branch.FeasPump.ON	feasibility pump を実行する

詳細

- 分枝限定法における発見的探索 feasiblity pump は整数性を満たさない解に対して，逐次的に連続緩和問題を解くことによって実行可能解を得る手法です.
- 本オプション値が -1/Branch.FeasPump.AUTO の場合は feasibility pump の実行する・しないが自動決定されます.

5.7.5 分枝限定法における発見的探索 RINS

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.branchRins	シンボル
C++SIMPLE	options.rins	整数値
RSIMPLE	なし	-
nuopt.prm	branch:rins	整数値

設定値

型	整数値	シンボル
デフォルト値	-1	Branch.Rins.AUTO
最小値	-1	-
最大値	1	-
値範囲	-	{Branch.Rins.OFF, Branch.Rins.ON, Branch.Rins.AUTO}

整数値	シンボル	意味
-1	Branch.Rins.AUTO	自動決定
0	Branch.Rins.OFF	RINS を実行しない
1	Branch.Rins.ON	RINS を実行する

詳細

- 分枝限定法における発見的探索 RINS は，連続緩和解と実行可能解（整数解）を組み合わせて，分枝限定法を再帰的に実行することにより実行可能解を得る手法です．連続緩和解と実行可能解を比較し，値が一致している変数を固定した上で分枝限定法を実行します．
- 本オプション値が -1/Branch.Rins.AUTO の場合は RINS の実行する・しないが自動決定されます．

5.7.6 分枝限定法における発見的探索 RENS

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.branchRens	シンボル
C++SIMPLE	options.rens	整数値
RSIMPLE	なし	-
nuopt.prm	branch:rens	整数値

設定値

型	整数値	シンボル
デフォルト値	-1	Branch.Rens.AUTO
最小値	-1	-
最大値	1	-
値範囲	-	{Branch.Rens.OFF, Branch.Rens.ON, Branch.Rens.AUTO}

整数値	シンボル	意味
-1	Branch.Rens.AUTO	自動決定
0	Branch.Rens.OFF	RENS を実行しない
1	Branch.Rens.ON	RENS を実行する

詳細

- 分枝限定法における発見的探索 RENS は，連続緩和解の情報から一部変数を固定した上で，分枝限定法を再帰的に実行することにより実行可能解を得る手法です.
- 本オプション値が -1/Branch.Rens.AUTO の場合は RENS の実行する・しないが自動決定されます.

5.7.7 分枝限定法におけるノード選択

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.branchNodeSelect	シンボル
C++SIMPLE	branchNodeSelect	文字列値
RSIMPLE	なし	-
nuopt.prm	branch:nodeSelect	文字列値

設定値

型	文字列値	シンボル
デフォルト値	auto	Branch.NodeSelect.AUTO
値範囲	{auto, bestDepth, bestEstimate}	{Branch.NodeSelect.AUTO, Branch.NodeSelect.BESTDEPTH, Branch.NodeSelect.BESTESTIMATE}

文字列値	シンボル	意味
auto	Branch.NodeSelect.AUTO	自動決定
bestDepth	Branch.NodeSelect.BESTDEPTH	ノード選択で深さ優先探索を行う
bestEstimate	Branch.NodeSelect.BESTESTIMATE	ノード選択で最良優先探索を行う

詳細

- 本オプションは分枝限定法のノード探索における戦略を選択します.
- 本オプション値で深さ優先探索を選択すると，ノード選択において深い位置にあるノード（ルートノードから最も遠いノード）が選択されます．実行可能解が早期に見つかる可能性が高い反面，下界（最大化問題であれば上界）が上がるのが遅くなる可能性があります.
- 本オプション値で最良優先探索を選択すると，ノード選択において最も目的関数値が良い（最小化問題であれば小さい）ノードを選択します．下界（最大化問題であれば上界）が上がるのが早くなる反面，実行可能解を見つけるのが遅くなったり，分枝木が大きくなりメモリ使用量が増える可能性があります.

関連

- 5.7.17 分枝限定法における目的関数の目標値設定

5.7.8　分枝限定法における wcsp タブーサーチの起動

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.branchUseWcsp	シンボル
C++SIMPLE	options.useWcsp	整数値
RSIMPLE	なし	-
nuopt.prm	branch:useWcsp	整数値

設定値

型	整数値	シンボル
デフォルト	-1	Branch.UseWcsp.AUTO
最小値	-1	-
最大値	1	-
値範囲	-	{Branch.UseWcsp.AUTO, Branch.UseWcsp.OFF, Branch.UseWcsp.ON}

整数値	シンボル	意味
-1	Branch.UseWcsp.AUTO	自動決定
0	Branch.UseWcsp.OFF	分枝限定法内で wcsp タブーサーチを起動しない
1	Branch.UseWcsp.ON	分枝限定法内で wcsp タブーサーチを起動する

詳細

- 本オプションにより分枝限定法内で wcsp タブーサーチを利用することができます.
- 本オプション値が -1 の場合は，wcsp タブーサーチを起動するかどうかは自動決定されます.
- 分枝限定法内における wcsp タブーサーチは通常の wcsp タブーサーチと以下が異なります.
  - 0-1 変数以外の変数は適当な刻み幅の DiscreteVariable として扱われる
  - 目的関数の目標値は，変数の上下限を考慮した目的関数の下限あるいは上限が使用される
  - 内部で収束判定が行われ，自動的に終了する

5.7.9 分枝限定法における wls の起動

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.branchUseWls	シンボル
C++SIMPLE	options.useWls	整数値
RSIMPLE	なし	-
nuopt.prm	branch:useWls	整数値

設定値

型	整数値	シンボル
デフォルト値	-1	Branch.UseWls.AUTO
最小値	-1	-
最大値	1	-
値範囲	-	{Branch.UseWls.AUTO, Branch.UseWls.ON, Branch.UseWls.OFF}

整数値	シンボル	意味
-1	Branch.UseWls.AUTO	自動決定
0	Branch.UseWls.OFF	分枝限定法内で wls を起動しない
1	Branch.UseWls.ON	分枝限定法内で wls を起動する

詳細

- 本オプションにより分枝限定法内で wls を利用することができます.
- 本オプション値が -1 の場合は，wls を起動するかどうかは自動決定されます.

5.7.10　分枝限定法における足切り点

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.branchCutoff
C++SIMPLE	options.cutoff
RSIMPLE	options.cutoff
nuopt.prm	branch:cutoff

設定値

型	実数値
デフォルト値	未定義
最小値	無制限
最大値	無制限

値	意味
任意値	指定値で足切り点を設定する

詳細

- 本オプションで足切り点を設定した場合，足切り点より悪い解しか与えないことが解った部分問題は探索の対象から外します．最小化問題においては，緩和問題における目的関数値（緩和値）が足切り点より大きい場合，該当の部分問題で枝刈りが行われます.
- 本オプションは，最小化問題において目的関数を f，足切り点を α とした場合，$f \leq \alpha$ という制約式を追加することと類似しています.
- 本オプションを適切に設定することにより計算の無駄を省くことができます．例えば，明らかに大きい目的関数値を持つ実行可能解を避けたい場合などに有効です.
- 厳しく設定しすぎると実行可能解がないというエラーになりますので，注意が必要です.
- 分枝限定法における目的関数の目標値設定（5.7.17）とは異なるので注意が必要です．目標値の場合は，実行可能解が見つかったら終了となるため，設定する／しないで探索プロセスは変わりません．足切り点は設定されることにより，探索プロセスが変わります.

関連

- 5.7.17 分枝限定法における目的関数の目標値設定

5.7.11 分枝限定法における分枝変数スコアの算出方法

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.branchVariableSelectScore	シンボル
C++SIMPLE	options.branchVariableSelectScore	文字列値
RSIMPLE	なし	-
nuopt.prm	branch:variableSelectScore	文字列値

設定値

型	文字列値	シンボル
デフォルト値	auto	Branch.VariableSelectScore.AUTO
値範囲	{auto, sum, product}	{Branch.VariableSelectScore.AUTO, Branch.VariableSelectScore.SUM, Branch.VariableSelectScore.PRODUCT}

文字列値	シンボル	意味
auto	Branch.VariableSelectScore.AUTO	自動決定
sum	Branch.VariableSelectScore.SUM	和
product	Branch.VariableSelectScore.PRODUCT	積

詳細

- 分枝限定法では，各変数で決められている分枝スコアに従って，分枝スコアが高い変数を選択し分枝を行います．変数選択で用いられるのが擬コストです．これは分枝時によってどの程度目的関数が改善するかという推定値です．
- 分枝時に生成される二つの子問題から得られる擬コストを合算して，各変数の分枝スコアとしています．合算方法には和（+）および積（×）の二通りがあります．本オプションはその合算方法を制御します．

5.7.12　分枝限定法における実行可能解の個数上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.branchMaxSolutionCount
C++SIMPLE	options.maxintsol
RSIMPLE	options.maxintsol
nuopt.prm	branch:maxintsol

設定値

型	整数値
デフォルト値	-1
最小値	-1
最大値	無制限

値	意味
–1	個数の上限を設けない
0	個数の上限を設けない
> 0	指定値を超えたら分枝限定法は終了する

詳細

- 本オプション分枝限定法において実行可能解の個数（整数解の個数）の上限を定めます.
- 本オプション値を -1 あるいは 0 に設定した場合，実行可能解の個数に上限は定めません.
- 本オプション値を 1 とすれば，実行可能解を 1 つだけ求めて終了する，ということが可能になります.
- 計算開始から見つかった実行可能解の数が本オプション値を越えると，以下のエラーメッセージとともに，現在までの実行可能解を出力して実行を終了します.

```
(NUOPT 37) Branch-and-bound method terminated with given number of
feasible solutions.
```

5.7.13　分枝限定法における探索問題数上限

概要

- 分枝限定法における部分問題個数の上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.branchMaxNode
C++SIMPLE	options.maxnod
RSIMPLE	options.maxnod
nuopt.prm	branch:maxnod

設定値

型	整数値
デフォルト値	-1
最小値	-1
最大値	無制限

値	意味
-1	部分問題の個数に上限を設けない
0	部分問題の個数に上限を設けない
> 0	部分問題の個数が指定値を超えたら分枝限定法は終了する

詳細

- 本オプションは分枝限定法において部分問題の個数の上限を定めます.
- 本オプション値を -1 あるいは 0 に設定した場合，部分問題の個数に上限を設けません.
- 部分問題の数が本オプション値を越えた場合のエラーメッセージは実行可能解が得られているかどうかで変わります．得られている場合は以下の出力が得られます.

```
(NUOPT 17) Branch-and-bound node limit reached (with feasible
solution).
```

得られていない場合は以下の出力が得られます.

```
(NUOPT 19) Branch-and-bound node limit reached (no feasible
solution found).
```

5.7.14　分枝限定法におけるメモリ上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.maxMemory
C++SIMPLE	options.maxmem
RSIMPLE	options.maxmem
nuopt.prm	branch:maxmem

設定値

型	整数値
デフォルト値	-10
最小値	無制限
最大値	無制限

値	意味
< 0	利用可能なメモリ量が指定値を下回った場合停止する
0	利用可能なメモリ量を制限しない
> 0	利用しているメモリ量が指定値を上回った場合停止する

詳細

- 本オプションは分枝限定法において最大メモリ量を制限します.
- 本オプションで対象となるメモリ量は仮想メモリ量になります．スワップ領域なども含めたメモリ利用量に対して上限が設定されます.
- 本オプション値は MiB 単位で指定します．例えば本オプション値を 1000 とすると 1GiB を上限とすることを意味し，1 GiB を超えた場合に実行を停止します.
- 負の値を設定すると，システムで利用可能なメモリ量が残り指定値以下になったときに実行を停止します．例えば本オプション値を -10 とすると残り 10MiB になったら停止します.
- メモリ上限によって実行が停止した場合には NUOPT43 エラーが，実行可能解が見つかっていない場合には NUOPT44 エラーが出力されます．この場合，現在までの最適解を出力して実行を終了します（実行可能解が見つかっていない場合には緩和解のみの出力となります）.

5.7.15 分枝限定法における上下界値ギャップ閾値（絶対値）

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.branchGapTolerance
C++SIMPLE	options.gaptol
RSIMPLE	なし
nuopt.prm	branch:gaptol

設定値

型	実数値
デフォルト値	-1
最小値	-1
最大値	無制限

値	意味
< 0	ギャップ閾値を設定しない
>= 0	上下界のギャップ値が指定値を下回ったら停止する

詳細

- 上下界のギャップ値が指定した値を下回る場合に解の探索を停止します．停止した際は，以下のエラーが出力されます．

```
(NUOPT 45) Gap in branch-and-bound method reaches below the limit.
```

- 実行可能解が求まってはじめて上下界値のギャップは意味を持ちます．したがってこのエラーで停止した場合には必ず実行可能解の出力がされます．
- ギャップ閾値の設定方法として，絶対値で指定する方法と相対値で指定する方法の二つがあります．本オプションは絶対値で指定します．

関連

- 5.7.16 分枝限定法における上下界値ギャップ閾値（相対値）

5.7.16　分枝限定法における上下界値ギャップ閾値（相対値）

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.branchRelativeGapTolerance
C++SIMPLE	options.relgaptol
RSIMPLE	なし
nuopt.prm	branch:relgaptol

設定値

型	実数値
デフォルト値	-1
最小値	-1
最大値	無制限

値	意味
< 0	ギャップ閾値を設定しない
>= 0	上下界のギャップ値の相対値が指定値を下回ったら停止する

詳細

- 分枝限定法は実行中の上界と下界がもとまります．例えば，最小化問題の場合は実行可能解の目的関数値が上界となります．上下界ギャップ値とは，この上界と下界の差を表します．
- 上下界ギャップ値の**相対値**が指定した値を下回る場合に解の探索を停止します．停止した際は，以下のエラーが出力されます．

```
(NUOPT 45) Gap in branch-and-bound method reaches below the limit.
```

- 上下界値のギャップは実行可能解が求まったときにのみ意味を持ちます．したがってこのエラーで停止した場合には必ず実行可能解の出力がされます．
- ギャップ閾値の設定方法として，絶対値で指定する方法と相対値で指定する方法の二つがあります．本オプションは相対値で指定します．
- ギャップの相対値は，上界値を Z_p，下界値を Z_d とした時，以下で定義されます．

$$relgap := \begin{cases} 0, & Z_p = Z_d = 0 \\ \frac{|Z_p - Z_d|}{\max(|Z_p|, |Z_d|)}, & Z_p \cdot Z_d \geq 0 \\ 1, & Z_p \cdot Z_d < 0 \end{cases}$$

5.7.17　分枝限定法における目的関数の目標値設定

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.objectiveTarget
C++SIMPLE	options.branchObjTarget
RSIMPLE	なし
nuopt.prm	branch:objTarget

設定値

型	実数値
デフォルト値	不定
最小値	無制限
最大値	無制限

値	意味
任意値	分枝限定法の目的関数値に指定値で目標を設定する

詳細

- 分枝限定法において，設定された目標値を超える目的関数値を有する実行可能解が得られたら停止します．
- 最小化問題の場合は目標値以下，最大化問題の場合は目標値以上の実行可能解が得られたら停止します．
- ギャップ閾値が設定されている場合，目標値の判定にはギャップ閾値が考慮されます．すなわち，目標値が，ギャップ閾値の範囲で目的関数値を下回る場合，停止します．最大化問題の場合は，ギャップ閾値の範囲で目的関数値を上回る場合，停止します．
- 分枝限定法における足切り点（5.7.10）とは異なるので注意が必要です．目標値の場合は，実行可能解が見つかったら終了となるため，設定する／しないで探索するプロセスは変わりません．足切り点は設定されることにより，探索プロセスが変わります．

関連

- 5.7.15 分枝限定法における上下界値ギャップ閾値（絶対値）
- 5.7.10 分枝限定法における足切り点

5.7.18　分枝限定法におけるスレッド数上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.threads
C++SIMPLE	options.bbthreads
RSIMPLE	なし
nuopt.prm	branch:threads

設定値

型	整数値
デフォルト値	1
最小値	-1
最大値	無制限

値	意味
-1	適切なスレッド数を自動的な設定する
0	単一のスレッドで実行する
1	単一のスレッドで実行する
>= 2	指定された値のスレッド数で実行する

詳細

- 分枝限定法は並列化で実行することによりユーザのマルチコア環境を生かすことができます，
- 推奨されるスレッド数は，実行する計算機の物理コア数以下となります．同時マルチスレッディング技術により論理コア数が物理コア数よりも多くなっている場合があります，この場合も物理コア数以下に設定することが推奨されます．
- 並列化手法を「5.7.19 分枝限定法における並列化手法」にて設定することができます．
- 分枝限定法の並列化計算は基本的に非決定的（計算する度に計算過程が異なるため結果も異なる）であるため，再現性が重要となるアプリケーションには推奨されません．
- 並列化手法の Racing 及び Subtree においては，設定したスレッド数全てで最適化計算が行われます．一方，Determinstic Racing では決定性を担保するためのスレッドが 1 つ存在するため，最適化計算用のスレッド数は「設定したスレッド数-1」となります．
- Windows 環境での実行では 4 スレッド以下の実行が推奨です．
- 本求解オプションは初期解修復でも有効です．

関連

- 5.7.19 分枝限定法における並列化手法

- 5.7.20 分枝限定法における初期解修復

5.7.19　分枝限定法における並列化手法

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.branchParallelMethod	シンボル
C++SIMPLE	options.branchParallelMethod	文字列値
RSIMPLE	なし	-
nuopt.prm	branch:parallelMethod	文字列値

設定値

型	文字列値	シンボル
デフォルト値	racing	Branch.ParallelMethod.RACING
値範囲	{racing, deterministicRacing, subtree}	{Branch.ParallelMethod.RACING, Branch.ParallelMethod.DETERMINISTIC_RACING, Branch.ParallelMethod.SUBTREE}

文字列値	シンボル	意味
racing	Branch.ParallelMethod.RACING	並列化手法 Racing を選択する
deterministicRacing	Branch.ParallelMethod.DETERMINISTIC_RACING	並列化手法 Deterministic Racing を選択する
subtree	Branch.ParallelMethod.SUBTREE	並列化手法 Subtree を選択する

詳細

- 並列分枝限定法の手法は以下の三つから選択できます.
  - Racing
  - Deterministic Racing
  - Subtree
- 各手法の特徴については付録の「B.2.4 並列分枝限定法」をご参考ください.

5.7.20　分枝限定法における初期解修復

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.branchRepairSolution	シンボル
C++SIMPLE	options.branchRepairSolution	文字列値
RSIMPLE	なし	-
nuopt.prm	branch:repairSolution	文字列値

設定値

型	文字列値	シンボル
デフォルト値	off	Branch.RepairSolution.OFF
値範囲	{off, on, aggressive}	{Branch.RepairSolution.OFF, Branch.RepairSolution.ON, Branch.RepairSolution.AGGRESSIVE}

文字列値	シンボル	意味
off	Branch.RepairSolution.OFF	初期解修復機能をオフにする
on	Branch.RepairSolution.ON	初期解修復機能をオンにする
aggressive	Branch.RepairSolution.AGGRESSIVE	初期解修復機能をオンにして積極的な探索を行う

詳細

- 本オプションは初期解修復機能を制御します.
- 初期解修復機能とは，与えた初期解から，制約違反量及び目的関数値を最適化して良質な実行可能解を得る機能です．例えば，制約条件を満たさない実行不可能な解が与えられたときも，これを修復して実行可能解を得ることができます.
- 初期解修復機能は分枝限定法の前段階（前処理を行う前）で実行されます．本機能が停止した段階で実行可能解が得られた場合，解を初期解として分枝限定法が実行されます.
- 初期解修復機能では解を修復するためには 2 段階の求解を繰り返します（[21]）.
  - 1 段階目では，実行不可能性（制約条件の違反量）を最小化します．これは元の数理最適化問題と同等に難しいため，一部の整数変数を固定することで問題規模を縮小して求解します.
  - 2 段階目では，実行不可能性に上限を与え，その範囲内で元の目的関数を最適化します．ここでも一部整数変数を固定して問題規模を縮小します.
  - 2 段階の求解を繰り返すことで実行不可能性を最小化することにより実行可能解を得ます.
- 初期解修復機能で用いる初期解について，連続変数には初期値を与える必要はありません.

整数変数には上下限制約を満たす初期値を設定します.

- 「分枝限定法におけるスレッド数上限」を 2 以上に設定した場合は本機能も並列実行されます．その場合，より少ない反復で解が修復されます．スレッド数を増やすと初期解修復機能によって実行可能解が得られる可能性も高まります.
- 初期解修復機能は以下の停止条件を満たすと停止します.
  1. 2 つ以上の実行可能解を発見した
  2. 制約違反量または目的関数値が小さくなる解を 3 回連続して発見できなかった
  3. 反復回数が，設定された「分枝限定法における初期解修復の回数」を超過（設定されていない場合はデフォルト値を超過）
  4. 設定された「計算時間上限」を超過
  5. 設定された「分枝限定法におけるメモリ上限」を超過
- 本オプション値を aggressive/Branch.RepairSolution.AGGRESSIVE に設定した場合，停止条件 1. と 2. を無視します.

関連

- 5.7.18 分枝限定法におけるスレッド数上限
- 5.2.3 計算時間上限
- 5.7.14 分枝限定法におけるメモリ上限
- 5.7.21 分枝限定法における初期解修復の回数上限

5.7.21　分枝限定法における初期解修復の回数上限

オプション名

モデリング言語/nuopt.prm	オプション名
PySIMPLE	Problem.options.branchRepairIteration
C++SIMPLE	options.branchRepairIteration
RSIMPLE	なし
nuopt.prm	branch:repairIteration

設定値

型	整数値
デフォルト値	20
最小値	0
最大値	無制限

値	意味
>= 0	初期解修復の回数上限を指定値で設定する

詳細

- 「分枝限定法における初期解修復」が有効になっている場合，その反復回数上限を設定します．

関連

- 5.7.20 分枝限定法における初期解修復

5.7.22　分枝限定法における非連結成分検出

オプション名

モデリング言語/nuopt.prm	オプション名	型
PySIMPLE	Problem.options.branchDisconnected	シンボル
C++SIMPLE	options.branchDisconnected	整数値
RSIMPLE	なし	-
nuopt.prm	branch:disconnected	整数値

設定値

型	整数値	シンボル
デフォルト値	-1	Branch.Disconnected.AUTO
最小値	-1	-
最大値	1	-
値範囲	-	{Branch.Disconnected.OFF, Branch.Disconnected.ON, Branch.Disconnected.AUTO}

整数値	シンボル	意味
−1	Branch.Disconnected.AUTO	自動決定
0	Branch.Disconnected.OFF	非連結成分検出を実行しない
1	Branch.Disconnected.ON	非連結成分検出を実行する

詳細

- 整数計画問題は，複数の完全に独立した問題に分割できることがあります．
- 本オプションは，この構造を自動検出し，分枝限定法の高速化を図ります．
- 値が 0/OFF の場合はこの構造を検出しません．
- 値が 1/ON の場合には構造が自動検出されます．
- 値が -1/AUTO の場合には，自動検出するかどうか内部で自動決定されます．

5.8 MPS ファイルに関する設定

この節では MPS ファイルから問題を入力する際の付加情報の指定方法を解説します．MPS ファイルに対する付加情報に関しては，求解オプションファイル nuopt.prm を用いて指定する方法のみが提供されています．

- 最小化，最大化の指定

  MPS ファイルから読み込んだ問題を目的関数の最小化問題/最大化問題のいずれとして解くかを指定します．初期設定は最小化です．

```
maximize
```

- 各種ラベル名

  これらは MPS ファイル中に複数の RHS/BOUNDS/RANGE/目的関数行があるとき，実際の計算で用いるものを指定します．

  デフォルトでは最初に現れたものとなります．

```
mpsfile:rhs = 文字列   (RHS ラベル名)
mpsfile:bou = 文字列   (BOUNDS ラベル名)
mpsfile:ran = 文字列   (RANGE ラベル名)
mpsfile:obj = 文字列   (目的関数行ラベル名)
```

  上記に関して該当するラベルを持つものが存在しない場合には，以下のようなエラーが出力されます．

```
(MPS FILE 13) Specified rhs: RHS データラベル not found
(MPS FILE 11) Specified bound: BOUND データラベル  not found
(MPS FILE 15) Specified range data: RANGE データラベル not found.
(MPS FILE 12) Specified objective: 目的関数行名 not found
```

第6章 MPSファイル・LPファイル

Nuorium Optimizer は MPS ファイル形式で記述された数理最適化問題を解くことができます．コマンドラインでの使用法は，以下の通りです．

```
prompt% nuopt ファイル名
```

上記の場合，拡張子が.lp であるファイルを LP ファイル形式として，それ以外を Free-MPS ファイル形式として読み込みます．

コマンドラインで使用する場合，ファイル形式をオプションとして指定することが可能です．この場合ファイル拡張子は無視されます．

Fix-MPS ファイル形式で読み込む場合には以下のようにします．

```
prompt% nuopt -fix-mps ファイル名
```

Free-MPS ファイル形式で読み込む場合には以下のようにします．

```
prompt% nuopt -free-mps ファイル名
```

LP ファイル形式で読み込む場合には以下のようにします．

```
prompt% nuopt -lp ファイル名
```

6.1 MPS ファイルに対する標準出力

MPS ファイルに対する求解コマンド nuopt を実行すると標準出力に計算の進行が表示されます．

```
prompt% nuopt ex1.mps
[About Nuorium Optimizer]
Nuorium Optimizer x.x.x  (NLP/LP/IP/SDP module)
         <with META-HEURISTICS engine "wcsp"/"rcpsp">
         <with Netlib BLAS>
        , Copyright (C) 1991 NTT DATA Mathematical Systems Inc.

[Reading MPS file: ex1.mps]
MPS_FILE_NAME                                          ex1.mps
PROBLEM_NAME(TITLE)                                   example1
```

```
ROWS                                                    4
COLUMNS                                                 3
NONZEROS                                               11
OBJECTIVE                                               f
RHS                                                     b

[Problem and Algorithm]
NUMBER_OF_VARIABLES                                     3
NUMBER_OF_FUNCTIONS                                     4
PROBLEM_TYPE                                   MINIMIZATION
METHOD                                         HIGHER_ORDER

[Progress]
<preprocess begin>.........<preprocess end>
<iteration begin>
    res=2.6e+001 .... 1.6e-004 . 3.9e-009
<iteration end>

[Result]
STATUS                                               OPTIMAL
VALUE_OF_OBJECTIVE                                     -10.5
ITERATION_COUNT                                            7
FUNC_EVAL_COUNT                                           10
FACTORIZATION_COUNT                                        8
RESIDUAL                                   3.903545017e-009
ELAPSED_TIME(sec.)                                      0.00
SOLUTION_FILE                                        ex1.sol
```

この出力中以下の部分：

```
[Reading MPS file: ex1.mps]
PROBLEM_NAME(TITLE)                                  example1
ROWS                                                       4
COLUMNS                                                    3
NONZEROS                                                  11
OBJECTIVE                                                  f
RHS                                                        b
```

は MPS ファイルを読み込むインタフェース部分からのメッセージで，MPS ファイルの NAME セクションにあるタイトル example1，ROWS セクションで指定された行の数（4），COLUMNS セクションで指

定された変数の数（3）と総非零要素数（11），目的関数の行の名前（F）と右辺ラベル（B）を示しています．

以降の標準出力は SIMPLE モデルで最適化計算を行った場合と同じです．詳しくは 2 標準出力をご覧ください．

6.2 MPS ファイル及び LP ファイルに対する解ファイル

nuopt は計算終了と共に，解ファイル出力します．これらには最適化アルゴリズム停止時における，変数や関数（目的関数及び制約式），双対変数（シャドウプライス）の値が記されています．解ファイルは入力ファイルの拡張子を.sol に変えたものが作成されますが，特殊な場合は次のルールに則って解ファイルの名前が決定されます．

入力ファイル名	解ファイル名	備考
ex1	ex1.sol	
ex1.mps, ex1.lp	ex1.sol	. 以降が.sol に
ex1.4.mps, ex1.3.lp	ex1.4.sol	最後の. 以降のみが.sol に
/nuopt/samples/ex1.mps	ex1.sol	パス名は反映されない

6.3 MPS ファイルに対する求解オプション設定

MPS ファイルに対しても，求解オプションを用いて各種設定をすることができます．MPS ファイルに対する情報を求解オプションで指定するには，求解オプションファイル nuopt.prm を用いる方法のみが提供されています．MPS ファイルに特有の求解オプションとして，以下が提供されています．

- 最小化，最大化の指定

  MPS ファイルから読み込んだ問題を目的関数の最小化問題/最大化問題のいずれとして解くかを指定します．MPS ファイルの初期設定は最小化ですので，最大化問題として解くには，明示的に指定する必要があります．

```
maximize
```

- 各種ラベル名

  これらは MPS ファイル中に複数の RHS/BOUNDS/RANGE/目的関数行があるとき，実際の計算で用いるものを指定します．

  デフォルトでは最初に現れたものとなります．

```
mpsfile:rhs = 文字列   (RHS ラベル名)
mpsfile:bou = 文字列   (BOUNDS ラベル名)
mpsfile:ran = 文字列   (RANGE ラベル名)
mpsfile:obj = 文字列   (目的関数行ラベル名)
```

  上記に関して該当するラベルを持つものが存在しない場合には，以下のようなエラーが出力され

ます.

```
(MPS FILE 13) Specified rhs: RHS データラベル not found
(MPS FILE 11) Specified bound: BOUND データラベル  not found
(MPS FILE 15) Specified range data: RANGE データラベル not found.
(MPS FILE 12) Specified objective: 目的関数行名 not found
```

6.4 MPS ファイルの具体例

MPS ファイルは次のような一般形の線形/二次計画問題/二次制約二次計画問題を記述するためのものです.

$$
\begin{array}{ll}
\text{最小／最大化} & f(x) \\
\text{条　件} & c_{L_i} \le g_i(x) \le c_{U_i}, \quad i = 1, \cdots, m \\
 & b_{L_j} \le x_j \le b_{U_j}, \quad j = 1, \cdots, n \\
\text{初期値} & x_j = x_j{}^0 \quad j = 1, \cdots, n
\end{array}
$$

ここで $f(x)$, $g_i(x)$ は二次関数で

$$f(x) = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n + \frac{1}{2} x^t H_0 x$$

$$g_i(x) = a_{i1} x_1 + a_{i2} x_2 + \cdots + a_{in} x_n + \frac{1}{2} x^t H_i x$$

と表されます.

MPS ファイルは MPS フォーマットと呼ばれる形式で，次の情報を記述したものです.

目的関数，制約式の線形部分の係数	c_j, a_{ij}
制約式の上下限	c_{L_i}, c_{U_i}
目的関数，制約式の Hessian の要素	H_0, H_i
変数の上下限	b_{L_j}, b_{U_j}
変数の初期値	$x_j{}^0$

本マニュアルでは MPS ファイルのフォーマットに対する定義は述べず，具体的な問題に対する MPS ファイルの対応を記述するに留めます．MPS ファイルフォーマットの詳細をご所望の方は nuopt-support@ml.msi.co.jp までご連絡ください.

$$
\begin{aligned}
&\text{最小化} && 4x_1 - x_3 + {x_2}^2 && \\
&\text{条件} && x_1 + x_4 && = 4 \\
& && x_1 + 2x_2 - x_3 + x_1 x_2 && \leq 10 \\
& && x_2 + x_3 && \geq 2 \\
& && 10 \geq x_1 \geq 3 && \\
& && 4 \geq x_2 \geq 1 && \\
& && 10 \geq x_3 \geq 0 && \\
& && 10 \geq x_4 \geq 0 && \\
&\text{初期値} && {x_1}^0 = 4, {x_2}^0 = 2 && \\
& && {x_3}^0, {x_4}^0 = 0 &&
\end{aligned}
$$

次は上記の二次計画問題を記述した MPS ファイルの例です.

```
NAME          solver
ROWS
 E  F1
 L  F2
 G  F3
 N  F4
COLUMNS
    X1        F1         1
    X1        F2         1
    X1        F4         4
    X2        F2         2
    X2        F3         1
    X3        F2        -1
    X3        F3         1
    X3        F4        -1
    X4        F1         1
RHS
    B         F1         4
    B         F2         10
    B         F3         2
BOUNDS
 LO BND       X1         3
 UP BND       X1         10
 LO BND       X2         1
 UP BND       X2         4
```

```
 UP BND       X3          10
 UP BND       X4          10
HESSIAN
    F2
    X2        X1          1
    F4
    X2        X2          2
INITIAL
     X1                                4
     X2                                2
ENDATA
```

6.5 LP ファイルの具体例とファイルフォーマット

LP ファイルとは，下記のようなフォーマットで数理最適化問題を記述したものです.

```
MIN
 2x1 + 3x2
SUBJECTTO
  x1 + x4           =  4
 -x1 + x2 - 0.5x3 <= 10
  x2 + 0.25x3      >=  2
BOUND
 x1 < -2
GEN
 x2
END
```

ここでは Nuorium Optimizer が対応している LP ファイルフォーマットについて簡単に説明します.

6.5.1 命名規則

変数，目的関数，制約式の名前に用いることができる文字は以下通りです.

- アルファベット：a-z A-Z
- 数字：0-9
- 記号：!"#$%&/,.;?@_‘´{}()|~’

```
4x2
```

のような記述は x2 の部分を名前とみて 4 × x2 と解釈します.

6.5.2 コメントと空行

\から行末までをコメントとします．また，空行は読み飛ばします．

6.5.3 半角スペースおよび式中の改行

"数 変数"および"変数 変数"という記述は，積と解釈します．式中の改行は，半角スペースと同様に扱われます．

6.5.4 lp ファイルの節

数理最適化問題を構成する節の記述順は以下の通りです．

1. 問題名（省略可）
2. 目的関数
3. 制約式
4. 境界条件（省略可）
5. 変数型（省略可）
6. 初期値（省略可）

各節の内容は，対応する指示語を行頭から記述した次の行から記述します．指示語は大文字・小文字を問いません．問題記述が終わった後に

```
END
```

と記述する必要があります．次節以降の lp ファイルの節についての説明では，下記の通りの書式を用います．

- `[ ]`：[] 内は省略可能
- `(a,b,..)`：`a b ..` のいずれか
- `{ }**`：{}内の繰り返し
- `number`：数値
- `name`：名前
- `term`：(number, [number] name [([*] name, ^2)])
- `expression`：[(+,-)] term { [(+,-) term] }**
  目的関数における定数項は 1 つまで可

6.5.5 問題名節

指示語：`prob`, `problem`

6.5.6 目的関数節

指示語：minimize, maximize, min, max, mininum, maximum

書式：

```
[name:] expression
```

name 前後の半角スペースは読み飛ばします．name が省略された場合"Objective"が目的関数名となります．二次式を記述する場合には

```
0.5 x1^2 + 6 x1 x2 + 2x2^2
```

もしくは

```
+ [ x1^2 + 3x1 * x2 + 2x2 ^2 ] / 2
```

と記述します．

6.5.7 制約式節

指示語：subject to, subject to:, such that, st, s.t., st., subjectto, suchthat, such

書式：

```
[name:] expression (<,<=,=<,>,>=,=>,=) number
```

name 前後の半角スペースは読み飛ばします．Name が省略された場合"co(開始行数)"が制約式名となります．不等号・等号と右辺値の間に改行を挟んではいけません．

6.5.8 境界条件節

指示語：bounds, bound

書式：

```
number (<,<=,=<,>,>=.=>,=) name
name (<,<=,=<,>,>=.=>,=) number
number (<,<=,=<,>,>=.=>,=) name (<,<=,=<,>,>=.=>,=) number
name free
(-inf, -infinite) (<,<=,=<) name
name (>,>=.=>) (-inf, -infinite)
(inf, infinite) (>,>=.=>) name
name (<,<=,=<) (inf, infinite)
```

<, >はそれぞれ<=, =>と解釈します．境界条件において，重複した定義はエラーとなります．上界値は，未設定の場合には+inf と設定されます．下界値が未設定の場合は

1. 上界値が 0 未満であれば-inf
2. そうでない場合 0

と設定されます.

6.5.9 変数型節

書式：

```
{name}**
```

指示語：generals, general, gens, gen

指示語に続く名前の変数を一般整数変数とします．境界条件を与えていない変数に対しては，境界条件を [0，+inf) とします.

指示語：integers, integer, ints, int

指示語に続く名前の変数を整数変数とします．境界条件が与えられていない場合は，境界条件を [0，1] とします.

指示語：binaries, binary, bins, bin

指示語に続く名前の変数を 0-1 整数変数とします.

6.5.10 初期値節

指示語：init, initial

書式：

```
name = number
```

6.6 変数の境界条件について

MPS ファイルにおいて ‘INTORG’ マーカーで指定した整数変数に対して，境界条件が与えられない場合は 0-1 整数変数と推定し，計算を実行します．また，MPS ファイルおよび LP ファイルのいずれについても，下記のルールが適用されます.

- 下界値のみが指定された場合，上界値+inf とする.
- 上界値のみが指定された場合，これが 0 未満であれば下界値を-inf とする.
- 上界値のみが指定された場合，これが 0 以上であれば下界値を 0 とする.

第 7 章　高度な利用法

本章では，Nuorium Optimizer の高度な利用法を扱います．

7.1　変数の初期値

変数の初期値の設定は，分枝限定法，wcsp，wls 及び rcpsp において有効です．

分枝限定法では設定された初期値が制約を満たす場合は，実行可能解として与えられます．与えられた実行可能解は枝刈りや実行可能化の更新等に用いられ，実行可能解が見つかりづらいが別のロジックでは見つけやすい場合や多段階求解の場合などに有効です．一方初期値が制約を満たさない場合は，初期値の設定は無視されます．この場合，微小な違反により初期値の設定が有効にならないケースがあります．そのような場合は求解オプション `branchRepairSolution` による「初期解の修復機能」を用いると初期値から実行可能解を構成できる可能性があります．

求解アルゴリズム wcsp タブーサーチでは設定された初期値を探索の出発点として採用することが可能です．以下は利用にあたっての注意点です．

- 求解オプション `wcspInitialValueActivation` のデフォルト値が"off"のため，ユーザ指定による初期値から探索をスタートする場合は本オプションを"on"に設定する必要があります．
- 求解オプション `tryCount` で計算回数を 2 以上に設定した場合，全ての試行回においてユーザ指定による初期値から探索を出発します．
- 離散変数 `DiscreteVariable` の定義域に文字列を用いる場合は，該当の変数に初期値を設定することはできません．
- 以下のケースなどでは初期値の設定は無視されます．
    - 初期値が整数性を満たさない
    - 初期値が変数の固定条件を違反している
    - 初期値が `selection` 制約を違反している

求解アルゴリズム wls では設定された初期値を探索の出発点として採用します．ただし試行回数（`tryCount`）が 2 以上で設定された場合は 2 回目以降の試行では設定された初期値ではなくランダムに構成された初期値が採用されます．

内点法では設定された初期値を探索の出発点として採用することが可能です．ただし，求解オプション ipmInitialValueActivation のデフォルト値が"off"のため，ユーザ指定による初期値から探索をスタートする場合は本オプションを"on"に設定する必要があります．

C++SIMPLE を用いる場合は，関数 "SimpleSetInitialValues()" を用いる必要がある場合があります．詳細は C++SIMPLE マニュアル「9.5 初期値設定 SimpleSetInitialValues()」をご参考ください．

7.2 制約違反によるエラー

最適化計算実行後，制約式あるいは変数の上下限を違反するというエラーが出ることがあります．

```
(NUOPT 33) Variable bound is violated.
(NUOPT 34) Both variable bound and constraint are violated.
(NUOPT 35) Constraint is violated.
(NUOPT 36) Equality constraint is violated.
```

これは最適化計算においては，スケーリング処理を施した後の問題を解き，計算の終了条件をスケーリング後の数値で判定しているためです．例えば内点法系統の解法であれば，スケーリング後の数値で KKT 条件の残差を見て終了判定を行なっています．

そのため最適化計算が正常に終了したように見えても，スケーリングを戻した後に制約式あるいは変数の上下限を違反してしまう場合があります．

このような場合，対応方法としては以下の 3 つが考えられます．

1. 問題のスケーリングを見直す
2. 解法の停止条件を変更する
3. スケーリング処理をオフにする

1. について，ここでいう「スケーリング」は問題にあらわれる数値のばらつきの大きさになります．例えば係数行列に 10^6 など大きい値があらわれる一方 10^{-4} など小さい値があらわれる場合，値がばらついていると言えます．

このような場合，例えば問題に含まれる数値について単位が揃っていない等によってばらつきが生じている場合は，単位を揃えることによって改善する場合があります．例えば「重さ」を表す定数の場合，「kg」と「g」など単位が揃っていない場合は，問題のスケーリングを悪くすることが考えられます．この場合，単位を「kg」などに統一することにより改善する可能性があります．

2. の停止条件については，対応は解法によって異なります．

例えば内点法系統の解法であれば，終了条件となる KKT 条件の残差をより小さく設定することによってエラーが解消される可能性があります．分枝限定法であれば，実行可能性閾値（tolx）をより小さく設定することにより回避できる可能性はあります．

3. についてはスケーリング処理を回避することにより違反を解消できる場合があります．ただし，そもそもスケーリング処理は最適化計算の安定性を増すためにしている処置です．このため，スケーリング処理を回避した場合，最適化計算自体が不安定になる可能性はあります．

上記の方策が不明な場合，あるいはそれでも解決が難しい場合は nuopt-support@ml.msi.co.jp までご相談ください．

付録 A Nuorium Optimizerのエラー/警告メッセージ

A.1 Nuorium Optimizer のエラー/警告メッセージ

Nuorium Optimizer 本体の出力するエラー/警告メッセージは,

- Nuorium Optimizer 本体部の計算で検出されたエラー/警告

```
(NUOPT 番号)　　エラー/警告メッセージ
```

- 求解オプションに関するエラー/警告

```
(SOLVER OPTION 番号)　エラーメッセージ
```

- MPS ファイル解釈時に検出されたエラー/警告

```
(MPS FILE 番号)　　エラーメッセージ
```

- LP ファイル解釈時に検出されたエラー/警告

```
(LP FILE 番号)　　エラーメッセージ
```

の 4 種類です．エラーメッセージは，標準出力に出力されます．SIMPLE ロードモジュールの場合には

```
ex2.smp:10:error: (SIMPLE 193) アルゴリズム実行時の問題
ex2.smp:10:error: (NUOPT 10) Interior Point Method iteration limit exceeded.
```

などのように，SIMPLE のエラーメッセージの一部として表示されますが，これはモデリング言語 SIMPLE が Nuorium Optimizer を起動している形を取っているためです.

A.1.1 Nuorium Optimizer のエラー/警告メッセージ

エラーに関する解説の最後に [解出力なし] とあるエラーの場合には，解ファイルへの変数値や関数値に関する出力は行われません．[解出力あり] とあるエラーの場合には最適性の保証がない解を一応は出力します.

エラー番号	エラーメッセージ / 説明
1	(NUOPT 1) Memory allocation error occurred during preprocessing phase.
	前処理部で所要メモリが，使用可能な量をオーバーしました．[解出力なし]

エラー番号	エラーメッセージ 説明
2	(NUOPT 2) infeasible (linear constraints and variable bounds). 線形制約や変数の上下限制約のために問題が infeasible となっていることが前処理部で検出されました．[解出力なし]
3	(NUOPT 3) Neither a valid objective function nor valid constraints. モデル（問題）に目的関数および制約式がありません．[解出力なし]
5	(NUOPT 5) Variable bounds are infeasible. (NUOPT 5) Integer variable bounds are infeasible. 連続変数もしくは整数変数の上下限制約のために問題が infeasible となっていることが前処理部で検出されました．（整数変数が整数性に違反する様な上下限を課されている場合等に発生します．）[解出力なし]
6	(NUOPT 6) Unbounded solution due to linear constraints and variable bounds. 線形制約と変数の上下限から問題の最適解は有界とはならないことが前処理部で判定されました．[解出力なし]
7	(NUOPT 7) internal error. [内部ルーチン名] 内部エラーが発生しました． （nuopt-support@ml.msi.co.jp にご連絡ください）．[解出力なし]
8	(NUOPT 8) Memory allocation error occurred during optimization phase. 計算部において所要メモリが使用可能な量をオーバーしました．[解出力なし]
9	(NUOPT 9) Exceeded the limit for step reductions. 直線探索アルゴリズムにおいて step reduction の失敗が起き，最適化実行が止まってしまいました（凸でない問題に直線探索アルゴリズムを適用した場合や，問題が infeasible である場合に起きます）．[解出力あり]
10	(NUOPT 10) Interior Point Method iteration limit exceeded. 内点法の反復回数が上限を越えました（上限は特に指定しない場合には 150 回です．求解オプションファイルからは criteria:maxitn = 300 として設定することができます）．[解出力あり] 内点法の収束状況が悪化しており，反復回数が上限を越えた場合に本エラーメッセージが出力されます． このような現象は，問題のスケールの悪さなど，数値的な問題に起因することが多いことが経験上知られています． この問題に対して万能な解決策を挙げることは難しいのですが，以下の方策により回避できることがあります． 初期点を変更する 自動スケーリング機能を off にする options.scaling = "off"; 反復回数上限を上げる（例えば，デフォルトで 150 であれば 300 にする） options.maxitn = 300;
11	(NUOPT 11) infeasible. 問題が実行不可能であると判定されました．[解出力あり]

エラー番号	エラーメッセージ 説明
12	(NUOPT 12) Heap memory exhausted for optimization process.
13	(NUOPT 13) unbounded.
	問題の最適解が有界でないことが判定されました．[解出力あり]
14	(NUOPT 14) Integrality constraint is violated.
	整数変数を含む問題に単体法以外を適用したため，整数変数が整数となっていない解が出力されています．[解出力あり]
15	(NUOPT 15) 手法名 misapplied to 問題種類.
	非線形計画問題に単体法が適用されようとしています．[解出力なし]
16	(NUOPT 16) Infeasible MIP.
	混合整数計画問題に整数解が存在しないことがわかりました．[解出力あり]
17	(NUOPT 17) Branch-and-bound node limit reached (with feasible solution).
	分枝限定法において生成する部分問題数が上限を越えました．最適解である保証はありませんが，整数解は得られています．（branch:maxnod を設定した場合）．[解出力あり]
19	(NUOPT 19) Branch-and-bound node limit reached (no feasible solution found).
	17 番と同じですが，整数解が得られていません．[解出力あり]
20	(NUOPT 20) Some subproblems remain unsolved in branch-and-bound method (no feasible solution found).
	分枝限定法で数値的理由によりいくつかの部分問題が解かれずに残りました．実行可能解の出力はありません．スケーリングオプションを変更してもう一度お試しください．[解出力あり]
21	(NUOPT 21) Branch-and-bound method iteration timeout (with feasible solution).
	整数計画法を解いている場合の Nuorium Optimizer の起動時間が上限を越えました．最適である保証はありませんが，整数解は得られています（crit:maxtim を設定した場合）．[解出力あり]
22	(NUOPT 22) Branch-and-bound method iteration timeout (no feasible solution found).
	21 番と同じですが，整数解が得られていません．[解出力あり]
23	(NUOPT 23) Branch-and-bound method objective value reaches below the limit.
	分枝限定法において目的関数値が設定値に達しました．[解出力あり]
25	(NUOPT 25) Cannot open file in current directory [no ファイル種類 created].
	（警告）解ファイル（.sol ファイル等）のオープンに失敗したので解ファイルが出力されていません（計算は行われます）．[解出力なし]
27	(NUOPT 27) Simplex iteration limit exceeded.
	単体法の反復回数の上限をオーバーしました．[解出力あり]
28	(NUOPT 28) Higher-order method can only be applied to Linear Programming (LP) problems.
	非線形計画問題に線形計画法専用の内点法が適用されようとしています．[解出力なし]

エラー番号	エラーメッセージ 説明
29	(NUOPT 29) Iteration process diverged. 内点法が発散しました．ペナルティパラメータが増大しており，以下の可能性が考えられます． 実行不可能 実行不可能に近い状態 実行可能領域になかなか近づけない 制約想定が満たされていない [解出力あり]
30	(NUOPT 30) Terminated by user. ユーザの指示により計算の中断を行いました．[解出力あり]
31	(NUOPT 31) Branch-and-bound method terminated by user (with feasible solution). 分枝限定法の演算中ユーザの指示により計算の中断を行いました．最適解である保証はありませんが，整数解は得られています．[解出力あり]
32	(NUOPT 32) Branch-and-bound method terminated by user (no feasible solution found). 31 番と同じですが，整数解が得られていません．[解出力あり]
33	(NUOPT 33) Variable bound is violated. 変数の上下限を有意に違反している解が出力されています．違反している個所は，解ファイル（*.sol）で"INFS"と表示されています．スケールの悪い（制約式や変数の値のオーダーの差が激しい）問題と思われます．内点法で解いている場合には停止条件を小さくして実行する必要があります．7.2 をご参考ください．[解出力あり]
34	(NUOPT 34) Both variable bound and constraint are violated. 変数および制約式の上下限を有意に違反している解が出力されています．違反している個所は，解ファイル（*.sol）で"INFS"と表示されています．スケールの悪い（制約式や変数の値のオーダーの差が激しい）問題と思われます．内点法で解いている場合には停止条件を小さくして実行する必要があります．7.2 をご参考ください．[解出力あり]
35,36	(NUOPT 35) Constraint is violated. (NUOPT 36) Equality constraint is violated. 制約式の上下限を有意に違反している解が出力されています．違反している個所は，解ファイル（*.sol）で"INFS"と表示されています．スケールの悪い（制約式や変数の値のオーダーの差が激しい）問題と思われます．内点法で解いている場合には停止条件を小さくして実行する必要があります．7.2 をご参考ください．[解出力あり]
37	(NUOPT 37) Branch-and-bound method terminated with given number of feasible solutions. options.maxintsol で指定した数だけの整数解が発見されたので分枝限定法を停止しました．[解出力あり]

エラー番号	エラーメッセージ 説明
38	(NUOPT 38) Dual infeasible. リスタート時に起動される双対単体法のプロセスで実行不可能性が検出されました．[解出力あり]
39	(NUOPT 39) Interior Point Method (IPM) iteration timeout. options.maxtim で設定した時間に対して，内点法の反復がタイムアウトしました．[解出力あり]
40	(NUOPT 40) Sequential Quadratic Programming (SQP) iteration limit exceeded. SQP（lsqp,tsqp）の反復が上限を超えました．他のアルゴリズムをお試しください．[解出力あり]
41	(NUOPT 41) Internal error in Sequential Quadratic Programming (SQP). SQP（lsqp,tsqp）の実行中に内部的なエラーが発生しました．他のアルゴリズムをお試しください．[解出力なし]
42	(NUOPT 42) Crossover method cannot be applied to Mixed-Integer Programming (MIP).
43	(NUOPT 43) Memory error in branch-and-bound method (with feasible solution). 分枝限定法の実行中に，options.maxmem で設定したメモリオーバーが起こり，実行を停止しましたが，実行可能解は得られています．[解出力あり]
44	(NUOPT 44) Memory error in branch-and-bound method (no feasible solution found). 分枝限定法の実行中に，options.maxmem で設定したメモリオーバーが起こり，実行を停止しました．実行可能解は得られていません [解出なし]
45	(NUOPT 45) Gap in branch-and-bound method reaches below the limit. 上下界の差が options.gaptol または options.relgaptol で与えられたよりも小さくなりましたので，分枝限定法を停止します．[解出力あり]
47	(NUOPT 47) IntegerVariable 変数名 should be declared as "binary". 0-1 整数変数以外の整数変数が表れています．wcsp は一般の整数変数を扱うことができません．[解出力なし]
48	(NUOPT 48) Variable 変数名 appear in two selection (). 二つ以上の selection にまたがって現れている 0-1 整数変数が表れました．[解出力なし]
49	(NUOPT 49) Variable 変数名 is fixed to infeasible value. 0-1 整数変数が 0，1 以外の値に固定されました．[解出力なし]
50	(NUOPT 50) Both of two variables 変数名 1 and 変数名 2 cannot be 1. 変数名 1 と変数名 2 の両方は（単一の `selection` に現れているので）両方 1 になることができません．[解出力なし]
51	(NUOPT 51) 手法名 is currently not available without SIMPLE. wcsp は SIMPLE によって定義された問題に対してのみ有効です．[解出力なし]

エラー番号	エラーメッセージ 説明
52	(NUOPT 52) All variables in selection statement#数字 fixed to 0. 対応する変数がすべて 0 に固定されているような `selection()` 文が「数字」番目に現れました．[解出力なし]
53	(NUOPT 53) Constraint 制約式名's weight is 値 should be -1 or non-negative value. 重みとしては-1 もしくは非負の数を与えねばなりません．[解出力なし]
54	(NUOPT 54) Constraint 制約式名's weight is 値 should be -1 or non-negative value. 重みとしては-1 もしくは非負の数を与えねばなりません．[解出力なし]
57	(NUOPT 57) You cannot use any method but "rcpsp" for model with Activity. Activity の定義があるにも関わらず，rcpsp 以外のメソッドを適用しようとしました．[解出力なし]
58	(NUOPT 58) You cannot use any method but "rcpsp" for model with ResourceRequire. ResourceRequire の定義があるにも関わらず，rcpsp 以外のメソッドを適用しようとしました．[解出力なし]
59	(NUOPT 59) You cannot use any method but "rcpsp" for model with ResourceCapacity. ResourceCapacity の定義があるにも関わらず，rcpsp 以外のメソッドを適用しようとしました．[解出力なし]
60	(NUOPT 60) You cannot use any method but "rcpsp" for model with tardiness and completionTime. 目的関数が tardiness/completionTime に設定されているのにも関わらず，rcpsp 以外のメソッドを適用しようとしました．[解出力なし]
63	(NUOPT 63) You cannot use Variable for "rcpsp" . rcpsp は `Variable` は用いる事が出来ません．[解出力なし]
64	(NUOPT 64) You cannot use IntegerVariable for "rcpsp" . rcpsp は `IntegerVariable` は用いる事が出来ません．[解出力なし]
67	(NUOPT 67) DiscreteVariable （変数名）'s bound: [a,b] and domain: {..} conflicts. (Regard Bound as a definition). `DiscreteVariable` の定義された範囲では満たすことのできない上下限があたえられました．[解出力なし]
70	(NUOPT 70) initial order is invalid between XX and YY . activities related imprecedene have to continue. rcpsp において初期解として不正な順番が与えられました. 直前先行制約に関係する `Activity` の順番は連続していなければなりません．[解出力なし]
71	(NUOPT 71) initial order is invalid. XX's order have to be previous to YY by way of precedence. rcpsp において初期解として不正な順番が与えられました．先行関係がある `Activity` はその順番が守られなければなりません．[解出力なし]

エラー番号	エラーメッセージ 説明
72	(NUOPT 72) Mixed-Integer Programming (MIP) problem detect an infeasibility during preprocessing. 前処理において実行可能解が無いと判定しました.
73	(NUOPT 73) Continuous Variable 変数名 cannot be included in model for wcsp. 連続変数を含む問題を wcsp で解こうとしました．wcsp は連続変数を扱うことができません．[解出力なし]
81	(NUOPT 81) You cannot use any method but "wcsp" for model with DiscreteVariable. `DiscreteVariable` を含む問題に wcsp 以外の方法は適用できません．[解出力なし]
82	(NUOPT 82) The trust region is too small. 関数の二次近似に失敗しつづけて信頼領域が小さくなりすぎましたので実行を停止します．[解出力あり]
83	(NUOPT 83) Some subproblems remain unsolved in branch-and-bound method (with feasible solution). 分枝限定法で数値的理由によりいくつかの部分問題が解かれずに残りました．実行可能解が出力されます．[解出力あり]
84	(NUOPT 84) Parallel branch-and-bound method was terminated due to numerical issues (with feasible solution). 数値的理由により実行可能解の最適性を証明できずに並列分枝限定法が終了されました．スケーリングオプションを変更してもう一度お試しください．[解出力あり]
85	(NUOPT 85) Parallel branch-and-bound method was terminated due to numerical issues (no feasible solution found). 数値的理由により実行可能解を発見できずに並列分枝限定法が終了されました．スケーリングオプションを変更してもう一度お試しください．[解出力なし]
100	(NUOPT 100) Cannot open NUOPT License file: "XX". UNIX/Linux 版のライセンスファイルを開くことができません.
101	(NUOPT 101) Invalid License file: "XX". UNIX/Linux 版のライセンスファイルの内容に問題があります.
102	(NUOPT 102) Machine key (YY) is not consistent with the license file: "XX". UNIX/Linux 版のライセンスファイルがご利用のマシンと整合していません.
103	(NUOPT 103) License expired XX days ago. 試用版の有効期間が終了しています.
104	(NUOPT 104) Invalid license limit.

エラー番号	エラーメッセージ 説明
105	(NUOPT 105) Cannot retrieve license information; please check machine configuration.
110	(NUOPT 110) Internal error (dpotrf failed).
	正定値で無い行列が与えられ，コレスキー分解に失敗しました．[解出力なし]
111	(NUOPT 111) Internal error (dpotrf failed).
	メリット関数計算時において，正定値でない行列が与えられ，コレスキー分解に失敗しました．[解出力なし]
112	(NUOPT 112) Internal error (dpotrs failed).
	逆行列を求める計算に失敗しました．[解出力なし]
113	(NUOPT 113) Internal error (dsygst failed).
	一般化固有値問題を標準固有値問題に変換する事ができませんでした．[解出力なし]
114	(NUOPT 114) Internal error (dsytrd failed).
	三重対角化に失敗しました．[解出力なし]
115	(NUOPT 115) Internal error (dstebz failed).
	最小固有値の導出に失敗しました．[解出力なし]
120	(NUOPT 120) The primal-dual gap is too large.
	主双対ギャップが十分小さくならない内に，エラーが発生し計算を終了致しました．主問題・双対問題の実行可能解は満たされています．[解出力あり]
121	(NUOPT 121) The primal-dual gap is too large.
	主双対ギャップが十分小さくならない内に，エラーが発生し計算を終了致しました．主問題の実行可能解は満たされています．[解出力あり]
122	(NUOPT 122) Negative stepsize detected.
	正であるべきステップサイズに負の値が検出されました．解法を tipm に変更してお試しください．[解出力なし]
123	(NUOPT 123) The SDP constraint cannot be handled by the specified algorithm.
	半正定値制約が存在しますが，半正定値制約を解釈できるアルゴリズムが選択されていません．[解出力なし]
124	(NUOPT 124) No SDP constraints detected.
	半正定値計画法用のアルゴリズムが起動されましたが，半正定値制約が存在しません．[解出力なし]
132	(NUOPT 132) Overflow error.
	wcsp の target や hard penalty の概算値，soft penalty の概算値が INT_MAX を超えてしまったことを表します．また変数の上下限が INT_MAX を超えていた際もこのエラーが出力されます．
133	(NUOPT 133) No feasible solution found.
	wcsp が求解を行ったが制約を満たす解が得られなかったことを表します．

エラー番号	エラーメッセージ 説明
134	(NUOPT 134) hard penalty overflow
	wcsp 求解後の hard penalty が overflow していることを表します．正しく求解が行われている保証はありません.
135	(NUOPT 135) soft penalty overflow
	wcsp 求解後の soft penalty が overflow していることを表します．正しく求解が行われている保証はありません.
141	(NUOPT 141) mtxfree parameter error
	mtxfree のための求解オプションの設定に誤りがあります．nuopt.prm をご確認ください.
142	(NUOPT 142) mtxfree option is valid only for higher-order method
	mtxfree は higher order でのみ使うことができます.
143	(NUOPT 143) mtxfree failed to solve linear equations
	反復法で連立一次方程式が解けませんでした．メモリが十分にある場合は mtxfree を使用せずに求解してください．いくつかの求解オプションを調整することで解けるようになるかもしれません．詳細は nuopt-support@ml.msi.co.jp にお問い合わせください.
144	(NUOPT 144) mtxfree cannot deal with free variables
	`mtxfree` では上限も下限も存在しない変数に対応していません．モデルを調整するか，`mtxfree` を使用せずに求解してください.
150	(NUOPT 150) asqp misapplied to nonconvex QP
	非凸二次計画問題に有効制約法が適用されようとしています．目的関数の二次項に対応する対称行列が半正定値ではありません．モデルと入力データを確認してください. 例えば二次の目的関数 `x*x + y*y` は凸関数で，対応する対称行列が半正定値です．一方で `x*x - 2*y*y` は半正定値ではありません．非凸二次計画問題を解く場合は信頼領域内点法 `tipm` といった別の解法を指定してください．整数変数を含む非凸二次計画問題を解く場合は問題を線形化して単体法ベースの分枝限定法 `simplex` を指定してください．整数変数のみの場合は解法 `wcsp/wls` も対応可能です.
151-165	（例）(NUOPT 156) irowA[0] = 9 should be in [1,3]
	`solveLP` や `solveQP` 実行時に与えた引数に問題があります．詳しくは「Nuorium Optimizer/C++SIMPLE 外部接続マニュアル」をご覧ください.
166	(NUOPT 166) Access error in calling "XX(YY)". Argument should be in [0,ZZ] (index for variable)
167	(NUOPT 167) Access error in calling "XX(YY)". Argument should be in [0,ZZ] (index for constraint)

エラー番号	エラーメッセージ 説明
168	(NUOPT 168) Invalid [int/double/string] value 求解オプション値 for nuoptPrm options. 求解オプション名 . options. 求解オプション名 に誤った値が設定されています.
171	(NUOPT 171) upper or lower bound of integer variable is too large. 整数変数の上限値もしくは下限値の絶対値が INT_MAX（int 型の最大値）を超えています.
172	(NUOPT 172) 手法名 is currently not available with SIMPLE. [手法名] は SIMPLE では利用することができません．[解出力なし]
177	(NUOPT 177) Nonlinear constraints cannot be included in model for wls. 解法 WLS は非線形制約を含めることができません．[解出力なし]
182	(NUOPT 182) no feasible solution found wls による求解時に制約を満たす解が得られなかったことを表します．[解出力なし]
190	(NUOPT 190) SIMPLEX time limit exceeded. 単体法が計算時間上限を超過しました．[解出力あり]
191	(NUOPT 191) dual infeasible(infeasible or unbounded). 実行不可能あるいは非有界となりました．[解出力あり]
193	(NUOPT 193) hsimplex misapplied to QCQP. 解法 hsimplex は QCQP に適用することができません．[解出力なし]

A.1.2 求解オプションのエラー/警告メッセージ

nuopt.prm または options を用いた Nuorium Optimizer の求解オプション設定のエラーです.

エラー番号	エラーメッセージ 説明
1	(SOLVER OPTION 1) Syntax error in solver option file. 求解オプションファイルの記述にエラーが検出されました.
2	(SOLVER OPTION 2) Solver option file is empty. 求解オプションファイルが全く空になっています.
4	(SOLVER OPTION 4) Internal error [内部ルーチン名] 求解オプションの解釈を行うルーチンにて内部エラーが発生しました（nuopt-support@ml.msi.co.jp にご連絡ください）.

求解オプションファイルの記述にエラーが発生した場合（エラー番号 1）には求解オプションファイル読み込み部から標準出力に補助的なメッセージが表示されますので，併せて参考にしてください.

エラーのある求解オプションファイル：

```
begin
maximize
method:tipm
criteria:eps = 1.0e-8
```

標準出力：

```
<reading solver option file: nuopt.prm >
nuopt.prm:1:     begin
nuopt.prm:2:     maximize
nuopt.prm:3:     method:tipm
nuopt.prm:4:error: Unknown category     criteria:eps = 1.0e-8（行名が違う）
nuopt.prm:5:error: end command is needed.                    （end で終わっていない）
(SOLVER OPTION 1) Syntax error in solver option file.
```

A.1.3 MPS ファイルのエラー/警告メッセージ

エラー番号	エラーメッセージ 説明
1	(MPS FILE 1) Failed to open mps file: ファイル名. MPS ファイルのオープンに失敗しました.
2	(MPS FILE 2) Undefined row name: 行名. COLUMNS/RHS/RANGES セクションに未定義の行が現れました.
3	(MPS FILE 3) Internal error. 内部エラーが発生しました. （nuopt-support@ml.msi.co.jp にご連絡ください.）
4	(MPS FILE 4) Syntax error in セクション名 section. 「セクション名」の示すセクションで文法エラーが発生しました.
5	(MPS FILE 5) Too many 'INTORG' markers. 'INTORG' マーカー行と'INTEND' マーカー行の対応が取れていません.（'INTORG' マーカー行が多すぎます.）
6	(MPS FILE 6) Too many 'INTEND' markers. 'INTORG' マーカー行と'INTEND' マーカー行の対応が取れていません.（'INTEND' マーカー行が多すぎます.）
7	(MPS FILE 7) Unknown marker: フィールド 3 の内容 'INTORG','INTEND' 以外のマーカー行が現れました. （Nuorium Optimizer の MPS ファイル解釈部は'INTORG','INTEND' 以外のマーカー行を解釈しません.）

エラー番号	エラーメッセージ 説明
8	(MPS FILE 8) Undefined row: 行名 in HESSIAN section. HESSIAN セクションの二次の項を付加する行名として，定義されていないものが現れました.
9	(MPS FILE 9) Undefined column: 変数名 in HESSIAN section. HESSIAN セクションの非零要素の場所を示す際の変数名として，定義されていないものが現れました.
10	(MPS FILE 10) row: 行名 appeared more than once. ROWS セクションに同一の行名が二度以上現れました.
11	(MPS FILE 11) Specified bound: BOUND データラベル not found. 求解オプションファイルで指定された（mpsfile: bound = BOUND データラベル）ラベルを持つ BOUNDS データが MPS ファイルには存在しません.
12	(MPS FILE 12) Specified objective: 目的関数行名 not found. 求解オプションファイルで指定された（mpsfile: objective = 目的関数行名）名前を持つ目的関数行が MPS ファイルには存在しません.
13	(MPS FILE 13) Specified rhs: RHS データラベル not found. 求解オプションファイルで指定された（mpsfile: rhs = RHS データラベル）ラベルを持つ RHS データが MPS ファイルには存在しません.
14	(MPS FILE 14) Range data: RANGE データラベル contains unsuitable row. 「RANGE データラベル」を持つ RANGE データが目的関数行に対しての指定を行っています.
15	(MPS FILE 15) Specified range data: RANGE データラベル not found. 求解オプションファイルで指定された（mpsfile: range = RANGE データラベル）ラベルを持つ RANGE データが MPS ファイルには存在しません.
17	(MPS FILE 17) Undefined column name: 変数名 in INITIAL section. INITIAL セクションに定義されていない変数名が現れました.
18	(MPS FILE 18) Bound spec. on column : 変数名 should appear earlier. （警告）「変数名」に関する上下限設定の現れるのはより前でなければなりません.（最初に発見されたものに対してのみこのメッセージが出力されます.）
19	(MPS FILE 19) 数字 column(s) appeared disorderly in BOUNDS section （警告）BOUNDS section において「数字」個の変数の現れる順番が違法です.（エラー 18 と組になって出力されます.）
20	(MPS FILE 20) Memory allocation error. ファイルの読み込み時にメモリエラーが発生しました.
21	(MPS FILE 21) Undefined column name: 変数名 in BOUNDS section. BOUNDS section において定義されていない変数名が現れました.

エラー番号	エラーメッセージ 説明
22	(MPS FILE 22) Hessian is implicitly bound for hlinexistent objective. HESSIAN セクションでは暗黙のうちに（行名を指定せず）目的関数に対して二次の項が設定されていますが，MPS ファイルには全く目的関数が存在しません.
23	(MPS FILE 23) Same bound spec. on column: 変数名 appeared more than once. BOUNDS section で「変数名」に関して同一の制約指定が二度以上行われました.
24	(MPS FILE 24) Column : 変数名 has bound specification FX and other. BOUNDS section で「変数名」に関して FX 制約と他の制約指定が同時に行われました.
25	(MPS FILE 25) Column : 変数名 has bound specification FR and other. BOUNDS section で「変数名」に関して FR 制約と他の制約指定が同時に行われました.
26	(MPS FILE 26) Column : 変数名 has bound specification LO and MI. BOUNDS section で「変数名」に関して LO 制約と MI 制約指定が同時に行われました.
27	(MPS FILE 27) Column : 変数名 has bound specification UP and PL. BOUNDS section で「変数名」に関して UP 制約と PL 制約指定が同時に行われました.
28	(MPS FILE 28) Unknown bound specification フィールド 1 の内容 BOUNDS セクションに LO, LI, UP, UI, BV, PL, MI, FR, FX 以外のラベルが現れました.
29	(MPS FILE 29) row : 行名 appreaed more than once in セクション名 section. RHS セクションおよび RANGE セクションにおいて同一の行名が二度以上現れました.
30	(MPS FILE 30) Unsupported section. フィールド 1 の内容 未対応のセクション名が現れました.
31	(MPS FILE 31) Bound of column 列名 infeasible. 列名に対する境界条件が実行不可能です.
32	(MPS FILE 32) Invalid mps file. その他の理由で読み込むことができないファイルです.

A.1.4 LP ファイルのエラー/警告メッセージ

エラー番号	エラーメッセージ 説明
1	(LP FILE 1) Failted to open lp file : ファイル名. ファイルのオープンに失敗しました.
2	(LP FILE 2) Memory allocation error. ファイルの読み込み時にメモリエラーが発生しました.
3	(LP FILE 3) Internal error. 内部エラーが発生しました.（nuopt-support@ml.msi.co.jp にご連絡ください.）
4	(LP FILE 4) Syntax error. 構文エラーが見つかりました.

エラー番号	エラーメッセージ 説明
5	(LP FILE 5) Non-ascii char appeared. 非 ASCII 文字が現れました.
6	(LP FILE 6) The order of sections is wrong. セクションの記述順が間違っています.
7	(LP FILE 7) Variable 変数名 appeared more than once in 式名. 同一の変数が一つの式に二度以上現れました.
8	(LP FILE 8) Term 変数名 * 変数名 appeared more than once in 式名. 同一の項が一つの式に二度以上現れました.
9	(LP FILE 9) Undefined variable name : 変数名. 未定義の変数名が現れました.
10	(LP FILE 10) Lower/Upper bound of variable 変数名 appeared more than once. 同一の変数に対する境界条件が二度以上現れました.
11	(LP FILE 11) Bound of variable 変数名 is infeasible. 変数について矛盾した境界条件が与えられました.
12	(LP FILE 12) Length of name 名前... is too longer. 変数や式の名前が長すぎます.（255 文字までです.）
13	(LP FILE 13) セクション名 section unsupported. 未対応のセクション名が現れました.
14	(LP FILE 14) general/integer/binary section appeared more than once. 同一のセクションが二度以上現れました.
15	(LP FILE 15) Invalid lp-format. 対応できないファイルです.

付録 B Nuorium Optimizer アルゴリズム概説

B.1 内点法

B.1.1 問題

次の最適化問題

$$\begin{aligned} &\text{最小化} \quad f(x), \qquad x \in R^n, \\ &\text{条　件} \quad g(x) = 0, \quad x \geq 0 \end{aligned}$$

を考えます[2]．ここで，変数 $x = (x_1, \cdots, x_n)^t$ は n 次元ベクトルで，関数 f は目的関数です．また，$g : R^n \to R^m$ は m 次元のベクトル値関数です．この問題のラグランジュ関数を

$$L(x, y, z) = f(x) - y^t g(x) - z^t x$$

としたとき，Karush-Kuhn-Tucker（KKT）条件（最適性の一次の必要条件）は次式で与えられます：

$$\begin{aligned} \nabla_x L(x, y, z) &= 0, \\ g(x) &= 0, \\ XZe &= 0, \quad x \geq 0, z \geq 0. \end{aligned}$$

ただし，$X = diag(x_1, \cdots, x_n) \in R^n, Z = diag(z_1, \cdots, z_n) \in R^n, e = (1, \cdots, 1)^t \in R^n$ です．ここで，相補性条件 $XZe = 0$ を $XZe = \mu e$（$\mu > 0$）で置き換えたものを修正 KKT 条件と呼びます．

Nuorium Optimizer ではバリヤペナルティ関数：

$$F(x, z) = f(x) - \mu \sum_{i=1}^{n} \log(x_i) + \rho \sum_{i=1}^{m} |g_i(x)| + \upsilon \left(x^T z - \mu \sum_{i=1}^{n} \log(x_i z_i) \right)$$

をメリット関数として採用します．ここで，$\mu > 0$ はバリヤパラメータ，$\rho > 0$ はペナルティパラメータ，$\upsilon > 0$ は主双対項の考慮度合いを表すパラメータです．

非線形最適化問題に対する内点法では，「修正 KKT 条件を満たす点を求めて，修正 KKT 条件を更新する」という作業を逐次行います．この際に，メリット関数 $F(x, z)$ の一次近似 $F_l(x, z)$ あるいは二次近似 $F_q(x, z)$ の変化量が重要な手がかりとなります[3]．次項以降で示すように，修正 KKT 条件を求める方法は複数存在します（直線探索を利用する方法・信頼領域を利用する方法）．

この作業を通して，最終的に元来の KKT 条件を満たす点を求めます．

---

[2] ここでは，説明の便宜上このような形式を考えますが，Nuorium Optimizer では制約関数に上下限が存在する場合，等式条件，変数に上下限が存在する場合などの一般形を扱うことができます．また，制約条件が存在しない問題も扱うことができます．

[3] 詳細は論文 [13][14][15] を参照

B.1.2 直線探索を利用する方法

修正 KKT 条件に対するニュートン法は次のようになります.

$$\begin{bmatrix} G + X^{-1}Z & -A^t \\ -A & 0 \end{bmatrix} \begin{bmatrix} \Delta x_N \\ \Delta y_N \end{bmatrix} = \begin{bmatrix} -r_L - X^{-1}r_C \\ r_E \end{bmatrix},$$

$$\Delta z_N = -X^{-1}Z\Delta x_N - X^{-1}r_C.$$

ここで, $(\Delta x_N, \Delta y_N, \Delta z_N)$ は各変数に対する探索方向ベクトルで, G はラグランジュ関数のヘッセ行列あるいはその近似行列です. このとき,「ρ が十分大きく, G が半正定値行列である場合 $\Delta F_l(x, z, \Delta x_N, \Delta z_N) < 0$ である」という性質が成り立ちます.

この性質を利用して, 直線探索を利用して大域的に収束するアルゴリズムを構築することができます. 主双対変数に対するステップ幅 α は以下のように計算されます. まず, 次の三式から α の上限値 $\alpha_{\max}$ を求めます.

$$\alpha^x{}_{\max} = \min_i \left\{ \frac{-x_i}{(\Delta x_N)_i} \mid (\Delta x_N)_i < 0 \right\},$$

$$\alpha^z{}_{\max} = \min_i \left\{ \frac{-z_i}{(\Delta z_N)_i} \mid (\Delta z_N)_i < 0 \right\},$$

$$\alpha_{\max} = \min\{\alpha^x{}_{\max}, \alpha^z{}_{\max}\}$$

次に,

$$\alpha = \bar{\alpha}\beta^l, \quad \bar{\alpha} = \min\{\gamma\alpha_{\max}, 1\}$$

と定義します. ここで, $\gamma \in (0,1), \beta \in (0,1)$ です. そして, l は

$$F(x + \bar{\alpha}\beta^l\Delta x_N, z + \bar{\alpha}\beta^l\Delta z_N) - F(x,z) \leq \varepsilon_0\bar{\alpha}\beta^l\Delta F_l(x, z, \bar{\alpha}\beta^l\Delta x_N, \bar{\alpha}\beta^l\Delta z_N)$$

をみたす最小の正整数として定義されます[4]. $\varepsilon_0 \in (0,1)$ です.

このように, 各種パラメータを適切に設定すれば, メリット関数 $F(x,z)$ が確実に減少するような探索方向ベクトル $(\Delta x_N, \Delta y_N, \Delta z_N)$ 及びステップ幅 α を定める事ができます. ここからアルゴリズムの大域的収束性が保証されます.

ラグランジュ関数のヘッセ行列が非負定値となるのは

- 線形計画問題（ヘッセ行列は 0）
- 一般の凸計画問題

です. また, 一般の非線形計画問題ではヘッセ行列を準ニュートン法で近似することによって行列 G を常に正定値に保つことができます. 従って, このような場合に直線探索を利用した手法を使用することができます.

B.1.3 信頼領域を利用する方法

行列 G が非負定値でないとき直線探索法を利用することは困難です. そこで, G が不定であるとき

[4] この一連のステップ幅の設定ルールを Armijo's Rule と呼びます.

も利用できる方法として主双対変数に対する信頼領域法を採用します．このとき，探索方向ベクトル w，サイズ $\delta > 0$，ステップ幅 α は次のような関係を満たします．

$$\|w\| \le \delta,$$
$$\alpha \le \min\left\{\frac{\delta}{\|w\|}, \gamma\alpha_{\max}\right\}$$

$\alpha_{\max}$ の導出方法は「直線探索を利用する方法」同様です．信頼領域のサイズ調整は通常の方法で行います．

大域的収束性を得るために基準となる最急降下方向ベクトル $(\Delta x_{SD}, \Delta y_{SD}, \Delta z_{SD})$ を

$$\begin{bmatrix} D + X^{-1}Z & -A^t \\ -A & 0 \end{bmatrix}\begin{bmatrix} \Delta x_{SD} \\ \Delta y_{SD} \end{bmatrix} = \begin{bmatrix} -r_L - X^{-1}r_C \\ r_E \end{bmatrix},$$
$$\Delta z_{SD} = -X^{-1}Z\Delta x_{SD} - X^{-1}r_C,$$

によって定義します．ここで $D \succ 0$ は対角行列です．α^* を方向 Δ_{SD} に沿って信頼領域で与えられる区間内で，メリット関数変化量の二次近似 $\Delta F_q(x, z, \Delta x, \Delta z)$ を最小化するステップ幅として定義します．

$$\alpha^* = \arg\min\{\Delta F_q(x, z, \alpha\Delta x_{SD}, \alpha\Delta z_{SD}) \mid \|\alpha(\Delta x_{SD} + \Delta z_{SD})\| \le \delta, \alpha \in [0, \bar{\alpha}]\}$$
$$\bar{\alpha} = \min\{1, \gamma\alpha_{\max}\}, \gamma \in (0, 1),$$

信頼領域のステップ幅 α は以下の条件をみたすように設定します．

$$\Delta F_q(x, z, \alpha\Delta x, \alpha\Delta z) \le \frac{1}{2}\Delta F_q(x, z, \alpha^*\Delta x, \alpha^*\Delta z) < 0,$$
$$\|\Delta x_{Nk}\| \le M\|\Delta x_{SDk}\|,$$
$$\|\Delta z_{Nk}\| \le M\|\Delta z_{SDk}\|$$

「信頼領域を利用する方法」では探索方向ベクトル $(\Delta x, \Delta z)$ は

$$\begin{pmatrix} \Delta x \\ \Delta z \end{pmatrix} = \nu\begin{pmatrix} \Delta x_{SD} \\ \Delta z_{SD} \end{pmatrix} + (1 - \nu)\begin{pmatrix} \Delta x_N \\ \Delta z_N \end{pmatrix},$$

として計算されます．ここで，パラメータ $\nu \in [0, 1]$ は $\nu = 0, 0.1, 0.2, \cdots, 0.9, 1.0$ の中で条件：

$$\Delta F_q(x, z, \alpha\Delta x, \alpha\Delta z) \le \frac{1}{2}\Delta F_q(x, z, \alpha^*\Delta x, \alpha^*\Delta z) < 0$$

をみたす最小の数です．このように，「信頼領域を利用する方法」では探索方向ベクトルの設定に異なる二方向 $(\Delta x_{SD}, \Delta z_{SD})$，$(\Delta x_N, \Delta z_N)$ を利用します．この結果，大域的収束性が保証されます．

B.1.4 線形計画問題専用内点法

問題が線形の場合，KKT 条件

$$\nabla_x L(x, y, z) = 0,$$

$$g(x) = 0,$$
$$XZe = 0 \tag{1}$$

の第 1 式，第 2 式は線形であり，非線形なのは第 3 式のみとなります．この方程式系に関するステップ幅 1 のニュートン法を適用すると線形な式の残差は原理的に零になり，非線形な第 3 式のみに

$$\Delta X_N \cdot \Delta Z_N e$$

なる形の残差が現れます（ΔX_N，ΔZ_N はニュートン法のステップ方向を対角に並べた行列）．この式にこの残差が発生することを見越してニュートン法のステップ方向を修正する（高次方向（Higher Order）の修正を加える）ことによって，より良いステップ方向を得ることができます．ニュートン法のステップ方向修正分を計算するためにはニュートン法のステップ方向の計算時に得られる副産物を有効に利用できるため，少ない計算コストでニュートン方向を改善することができ，計算を効率化することができます．

Nuorium Optimizer にはこのことを利用し，さらに問題が線形であることに特化したチューニングを加えた手法が組み込まれています．

B.1.5 半正定値計画問題専用内点法

次の最適化問題

$$\begin{array}{lll} \text{最小化} & f(x) & x \in R^n, X \in S^p \\ \text{条件} & g(x) = 0, X_0(x) = X, & x \geq 0, X \geq 0 \end{array}$$

を考えます．ここで, 変数 $x = (x_1, \cdots, x_n)^t$ は n 次元ベクトルで, 関数 f は目的関数です．また, $g : R^n \to R^m$ は m 次元のベクトル値関数です．変数 X は p 次元対称正方行列で，$X \geq 0$ は行列 X の半正定値制約を意味するものとします．$X_0 : R^n \to S^p$ は n 次元ベクトルを対称正方行列空間に写す写像です．問題が線形の場合，X_0 は $X_0(x) = \sum_{i=1}^{n} A_i x_i - B$ と表現しても構いません．

この問題のラグランジュ関数を $w = (y, Y, Z)$ として

$$L(w) = f(x) - y^T g(x) - \langle X_0(x) - X, Y \rangle - \langle X, Z \rangle$$

としたとき，Karush-Kuhn-Tucker(KKT) 条件（最適性の一次の必要条件）は次式で与えられます：

$$\nabla_x L(w) = \nabla f(x) - \nabla g(x)^T \Delta y - A^*(x)Z = 0$$
$$\nabla_X L(w) = Y - Z = 0$$
$$g(x) = 0$$
$$X_0(x) - X = 0$$
$$X \circ Z = 0, X \geq 0, Z \geq 0$$

ここでは，$A_i(x) = \dfrac{\partial X_0(x)}{\partial x_i}$，$A^*(x)Z = \begin{pmatrix} \langle A_1, Z \rangle \\ \cdots \\ \langle A_n, Z \rangle \end{pmatrix}$，$X \circ Z = \dfrac{1}{2}(XZ + ZX)$ であるものとします．

Nuorium Optimizer では，この KKT 条件を満たす点を，Newton 法を利用して反復解法で求めます．

線形な半正定値計画問題の場合，大域的収束性は保証されますが，そうでない問題を扱う場合，大域的収束を保証するには，メリット関数を定義する必要があります．

非線形半正定値計画問題を扱う場合，バリヤペナルティ関数：

$$F(x,X,Z) = F_{BP}(x,X) + \nu F_{PD}(x,X,Z)$$

$$F_{BP}(x,X) = f(x) - \mu \log(\det X) + \rho\|g(x)\|_1 + \rho'\|X_0(x) - X\|_1$$

$$F_{PD}(X,Z) = \langle X,Z\rangle - \mu \log(\det X \det Z)$$

をメリット関数として採用します．ここで，$\mu > 0$ はバリヤパラメータ，$\rho, \rho' > 0$ はペナルティパラメータ，$\nu > 0$ は主双対項の度合いを表すパラメータです．

探索方向を求める Newton 方程式は以下のように記述されます．

$$\begin{pmatrix} G+H & -\nabla g(x) \\ -\nabla g(x)^T & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = - \begin{pmatrix} \nabla f(x) - \nabla g(x)^T y - \mu A^*(x) X^{-1} + A^*(x)(ZX_0X^{-1} - Z) \\ g(x) \end{pmatrix}$$

$$\Delta X = X_0(x + \Delta x) - X$$

$$\Delta Z = \mu X^{-1} - Z - \frac{1}{2}(Z\Delta X X^{-1} + X^{-1}\Delta X Z)$$

ここでは，$H_{ij} = \langle A_i, X^{-1}A_j Z\rangle$ であるものとします．A_i, A_j の構造に応じて H_{ij} を計算する方法は異なります．

修正 KKT 条件を満たす点を逐次求め反復するという枠組み自体は，一般の非線形用内点法と同等です．Newton 方程式を解く際には，探索方向を対称化するため，KSH 方向へのスケーリングを行っています．

問題が非凸な場合，大域的収束性を確保するための工夫が必要になります．Nuorium Optimizer では信頼領域法に基づく方法を利用する場合，大域的収束を保証する方向と，局所的収束に有利な方向を混ぜ合わせて探索方向を決定します．

B.2 単体法・有効制約法

単体法は線形最適化問題

$$\begin{aligned} &\text{最小化} \quad c^t x && x \in R^n \\ &\text{条　件} \quad b_U \geq Ax \geq b_L \end{aligned}$$

に対して，有効制約法は二次計画問題

$$\begin{aligned} &\text{最小化} \quad \frac{1}{2}x^t Q x + c^t x && x \in R^n \\ &\text{条　件} \quad b_U \geq Ax \geq b_L \end{aligned}$$

に対して，それぞれ有効な方法です．一度可能基底解が得られれば，問題に対して小さな変更を行った際の解を比較的高速に求めることができるなど，内点法にはない特徴を備えています．

B.2.1 改訂単体法

Nuorium Optimizer に実装されている単体法は大規模問題用の改訂単体法と呼ばれる手法です．単体

法自身に関する解説は例えば [3] を参照してください.

B.2.2 有効制約法

Nuorium Optimizer に実装されている有効制約法は改訂単体法に基づいています．有効制約法に関する解説は例えば [12] を参照してください．この手法は 5 千変数以上の大規模問題では，一般に内点法（直線探索法（Line Search Method））に劣りますが,

- 変数に比べて制約式の数が非常に少ない（1/10 以下）場合
- 目的関数のヘッセ行列が密行列である場合

には内点法よりも高速かつ高精度です.

B.2.3 分枝限定法

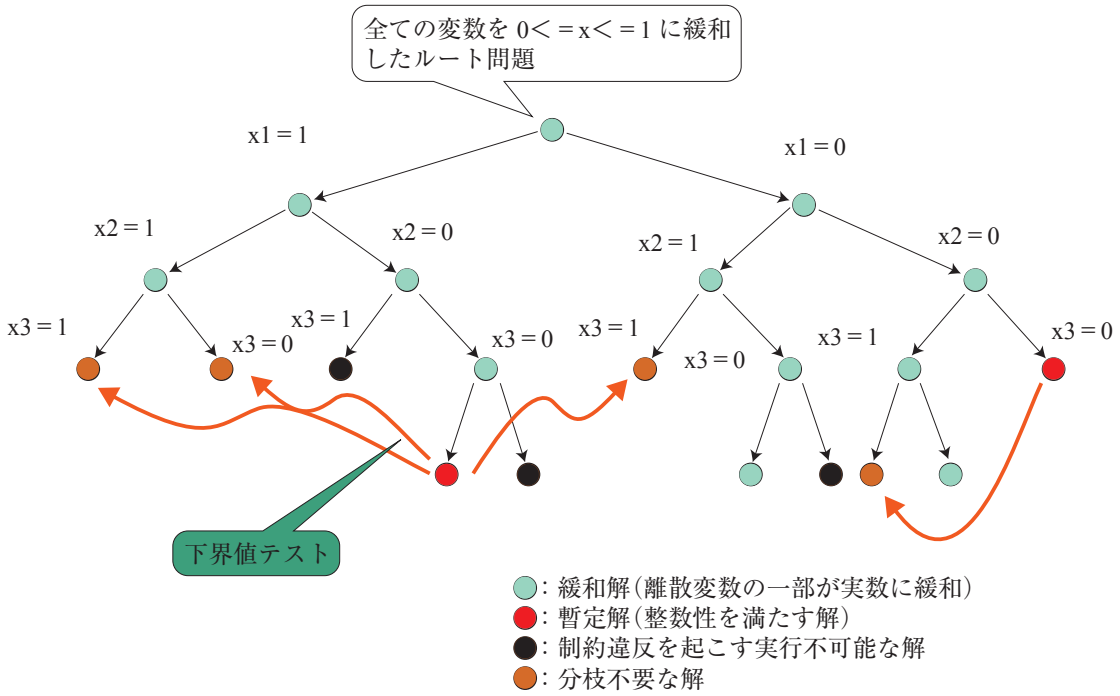
Nuorium Optimizer は，整数変数を含む線形/二次計画問題に対して以下のアルゴリズムが指定された場合，分枝限定法と呼ばれるアルゴリズムを用います.

- 単体法 （options.method = "simplex"）
- 有効制約法（options.method = "asqp"）

分枝限定法とは一般に元の問題の制約を一部緩和した問題（緩和問題）を繰り返し解くことにより厳密解を求めるアルゴリズムです．特に整数計画問題に適用される場合，緩和問題は整数変数を連続変数に緩和した問題（連続緩和問題）を考えます.

ここでは 0-1 整数変数を含む線形計画問題（目的関数を最小化する）の例を説明します．0-1 整数変数は連続緩和すると上限値が 1，下限値が 0 となる連続変数になることにご注意ください.

分枝限定法では緩和問題の解から整数になっていない変数を一つ選択し，0 または 1 に固定する，という操作（分枝操作）を行います．この過程は次の図のような木（分枝木）の形に表すことができ，整数制約を満たす解はこの木の「葉」の部分（図の下側の先端部分）に現れます.



しかしながら，分枝操作だけでは「葉」が整数変数の数に対して指数的に増える可能性があり，現実的な規模の問題に対して効率の良い解法となりません．このため分枝限定法は，分枝木の「分枝するごとに緩和問題の目的関数値が同じあるいは増大する」という性質を用いて枝の削減を行います（限定操作）．具体的には，分枝時に緩和解がその時点で知られている暫定解（整数性を満たす解）よりも目的関数値が大きければ，それ以上分枝する必要がありません．例えば緩和解の目的関数値が 11 で，暫定解として 10 が得られていればそれ以上分枝して探索する必要がありません（上の図の橙色の解に相当）．分枝限定法はこのような分枝操作と限定操作を繰り返し最適解を得るアルゴリズムです．

Nuorium Optimizer は更なる工夫により分枝限定法を高速化しています．

ヒューリスティクス

通常の分枝限定法では緩和解が整数性を満たす時のみ暫定解が得られます．Nuorium Optimizer ではそれに加え，実行可能解探索を別のロジックで行うことにより効率よく暫定解を求めます．例えばヒューリスティクス RINS では，緩和解と暫定解を比較し，共通して整数となっている変数は固定し，解が非共通の変数を非固定にした状態で分枝限定法を新たに解くことで，実行可能解を得ようとします．

切除平面

整数計画問題では，整数性を考慮することにより「切除平面」と呼ばれる制約式を生成できます．この制約式を元の制約式に付与すると緩和解が改善するため，分枝木を小さくできます．ただし緩和問題のサイズは大きくなるため，トレードオフの調整が必要です．

前処理

整数計画問題は，最初の緩和問題を解く前に「前処理」により問題変換あるいは問題削減を行います．例えば「検針（ probing ）」と呼ばれる操作は 0-1 変数を 0 あるいは 1 に固定しそこから実行不可能性が導かれるのであればどちらかに固定します．

これらの手法の具体的な求解オプションの設定方法については「整数計画法（simplex/asqp）に有効

な求解オプション」の項に解説があります.

B.2.4 並列分枝限定法

並列分枝限定法では「Supervisor」および「Worker」と呼ばれる役割が協調して処理を行います [17]. Supervisor は並列化動作を統制し，Worker は与えられたタスクを処理します.

Nuorium Optimizer は次の 3 つの並列分枝限定法を提供しています.

1. Racing（デフォルト）
2. Deterministic Racing
3. Subtree

Racing と Subtree は [18] を参考にしています.

Racing

Racing は複数の Worker が同一の整数計画問題を並行して解く手法です．各 Worker では異なる求解オプションが設定されています．同一の問題を解くため，冗長な計算を多く含みますが，デフォルトの求解オプションでは発見することが容易でない解を得ることができます．短時間で優良な解を得ることが重視される場合に有効な手法です．非決定性並列処理のため，同じ計算環境かつ同じ入力データであっても，異なる計算プロセスを経る可能性があります.

Deterministic Racing

Deterministic Racing は Racing に決定性を持たせた手法です．ここで言う決定性とは，同じ計算環境かつ同じ入力データであれば同じ結果が得られる（同じ計算プロセスを経る）ことを意味します.

デバッグやテストがしやすく，システムの計算エンジンとして利用するなど，より安定的に利用したい場合に有効です．注意点として決定性を持たせている分計算性能が犠牲になっているため，問題によっては通常の分枝限定法よりも遅くなるケースがあります．また，計算性能はスレッド数を多くするほど悪化する可能性があるため，適切なスレッド数を事前に調べておくことが重要です.

Subtree

Subtree は並列に分枝木を探索する手法です．Racing と比べると探索の効率が良く，小から中規模の問題で最適解が必要となる場合に有効です．注意点として大規模な問題（特に分枝限定法の前処理や単体法に時間がかかる問題）を本手法で解かせると，他の Worker に探索させる分枝木のルート問題の生成に時間がかかるため，通常の分枝限定法よりも遅くなる可能性があります．このような問題の場合は Subtree よりも Racing が推奨されます.

並列分枝限定法を実行する際の求解オプションについては「整数計画法（simplex/asqp）に有効な求解オプション」の項に解説があります.

B.3 逐次二次計画（SQP）法

逐次二次計画法では，次のような等式・不等式制約つきの非線形最適化問題の解を求めることがで

きます[5].

$$\begin{aligned} \text{目的関数} \quad & f(x) \quad \rightarrow \quad \text{最小化} \\ \text{制約条件} \quad & g_j(x) = 0, j \in J_E \\ & g_j(x) \geq 0, j \in J_I \end{aligned}$$

SQP 法とは，元の問題を現在の反復点において二次計画問題で近似し，その二次計画問題の解を探索方向としながら解を求める手法です．以下，Nuorium Optimizer で用いることができる SQP 法について説明します．なお，説明で用いる k は反復の回数を表します．

B.3.1 準ニュートン法を用いる方法

本手法では，元の問題を次のような二次計画問題で近似します．

$$\begin{aligned} \text{目的関数} \quad & \frac{1}{2}\Delta x^T B_k \Delta x + \nabla f(x_k)^T \Delta x \quad \rightarrow \quad \text{最小化} \\ \text{制約条件} \quad & g_j(x_k) + \nabla g_j(x_k)^T \Delta x = 0, j \in J_E \\ & g_j(x_k) + \nabla g_j(x_k)^T \Delta x \geq 0, j \in J_I \end{aligned}$$

ここで B_k は元の問題のラグランジュ関数のヘッセ行列を準ニュートン法によって近似した行列です．この問題の解 Δx を探索方向とし，直線探索を行って，次の反復点を定める方法となります．

B.3.2 信頼領域法を用いる方法

本手法では，二つの二次計画問題を解くことで点列を生成していきます．まず，一つ目の二次計画問題は次の通りです．

$$\begin{aligned} \text{目的関数} \quad & \frac{1}{2}(\Delta x_k^{SD})^T D_k \Delta x_k^{SD} + \nabla f(x_k)^T \Delta x_k^{SD} \quad \rightarrow \quad \text{最小化} \\ \text{制約条件} \quad & g_j(x_k) + \nabla g_j(x_k)^T \Delta x_k^{SD} = 0, j \in J_E \\ & g_j(x_k) + \nabla g_j(x_k)^T \Delta x_k^{SD} \geq 0, j \in J_I \end{aligned}$$

ここで D_k とは，要素が正の値であるような対角行列とします．この問題の解 Δx_k^{SD} は，本手法に大域的収束性を与える上で大きな役割を果たします．

この問題の解を求めたとき，効いている制約の集合を J_A^k とします．すなわち

$$J_A^k = \{j \in J_E \cup J_I \mid g_j(x_k) + \nabla g_j(x_k)^T \Delta x_k^{SD} = 0\}$$

となります．このとき，もう一つの二次計画問題は次のように定められます．

$$\begin{aligned} \text{目的関数} \quad & \frac{1}{2}(\Delta x_k^N)^T G_k \Delta x_k^N + \nabla f(x_k)^T \Delta x_k^N \quad \rightarrow \quad \text{最小化} \\ \text{制約条件} \quad & g_j(x_k) + \nabla g_j(x_k)^T \Delta x_k^N = 0, j \in J_A^k \end{aligned}$$

ここで G_k は元の問題のラグランジュ関数のヘッセ行列です．この問題の解 Δx_N は，反復点が元の問題の解に近づいたときに速い収束をするための方向になっています．また，この問題の KKT 条件は，

[5]内点法の説明の箇所でも説明しましたが，ここに挙げた数理最適化問題の定式化はアルゴリズム説明のために挙げた一例であり，Nuorium Optimizer は変数の上下限制約などを含んだより一般的な問題を扱うことができます．

次のように線形方程式系として表すことができます.

$$\begin{pmatrix} G_k & -\nabla g_{J_A^k}(x_k) \\ \nabla g_{J_A^k}(x_k)^T & 0 \end{pmatrix} \begin{pmatrix} \Delta x_k^N \\ y^N_{k+1,J_A^k} \end{pmatrix} = \begin{pmatrix} -\nabla f(x_k) \\ -g_{J_A^k}(x_k) \end{pmatrix}$$

ここで，y^N_{k+1,J_A^k} はこの問題の制約条件に対するラグランジュ乗数とします．一般に，問題規模が同程度であれば，線形方程式系は二次計画問題に比べ速く解くことができるので，この問題に対する計算コストは，先程の Δx_{SD} を求める問題と比べて小さいものと考えることができます.

本手法では，Δx_{SD} と Δx_N の凸結合

$$\Delta x_k(\nu_k) = \nu_k \Delta x_k^{SD} + (1-\nu_k)\Delta x_k^N$$

を探索方向とし，定められた信頼領域の中を探索し，次の反復点を生成します.

B.4 制約充足問題ソルバ wcsp

このアルゴリズムは離散変数，0-1 整数変数を変数とした制約充足問題：

$$\begin{aligned} &\text{変数} \quad x_j \in X_j, && j = 1, \cdots, n \\ &\text{条件} \quad c_i^U \geq g_i(x_1, \cdots, x_j, \cdots, x_n) \geq c_i^L, && i = 1, \cdots, m \end{aligned}$$

をメタヒューリスティクスの解法の一つであるタブー・サーチを用いて解くものです．ここで，X_j は各変数の定義域に対応する有限集合です.

本アルゴリズムは，各制約の違反量の合計を最小化する問題を解きます．近似解法であるため，制約式をすべて満たす解が存在しない場合でもできるだけ制約を満たす解を得ることができます．各制約の違反量の合計を計算する際，各制約の違反量には「重み」と呼ばれる正の定数を掛けて合算します.

この際，重みの大きな制約式は優先的に満たされるように解の探索が行われます.

重要度の高い制約式の重みを大きくすることで，より有用な解が期待できます.

制約には重みを ∞ として設定することもできます．このように設定された制約は，何よりも優先して満たすべき制約として扱われます．重みが ∞ に設定されたものをハード制約，正の有限の値に設定されたものをソフト制約と呼びます.

制約の他に最大化，最小化すべき目的関数 $f(x_1, \cdots, x_j, \cdots, x_n)$ を定義することもできますが，その際には目標値 μ を定めて

- 最小化問題であれば $\mu - f(x) \geq 0$
- 最大化問題であれば $f(x) - \mu \geq 0$

というソフト制約を定義して目的関数を扱います．つまり，目的関数自身も，あくまで満たすべき制約式のひとつとして扱われるため，目的関数に対しても重みを設定する必要があります.

制約充足問題ソルバは，以下のいずれかの条件を満たせば停止します.

1. すべてのハード制約およびソフト制約を満たす解が求まった（目的関数に関しては目標値を満たす解が求まった）
2. 反復回数が指定の上限を超えた
3. 計算時間が指定の上限を超えた

4. ユーザが停止を命じた

本アルゴリズムは「各制約は，それ単体でみれば容易に満たすことができる（greedy な方法でペナルティを 0 にできる）」ことを前提としています．例えば等式制約よりも不等式制約の方で記述したほうが，単体で満たしやすいため有利になる場合があります．

本アルゴリズムは，京都大学「問題解決エンジン」グループの開発によるものです．詳細については [9], [10] をご参照ください．

B.5　タブー・サーチによる資源制約スケジューリング問題解法

このアルゴリズムは以下の資源制約付きスケジューリング問題:

- 限られた資源の下，仕事の処理に用いる資源の分配，作業の開始時刻を決定する

をタブー・サーチを用いたリストの探索を用いて解くものです．

アルゴリズムは wcsp と同じ京都大学「問題解決エンジン」グループによるものです．詳細については [16] をご覧ください．

B.6　重み付き局所探索法 WLS

このアルゴリズムは最適化問題を局所探索により近似的に解くものです．問題が二次式を含む場合は整数変数のみを扱うことができ，すべて線形式の場合は連続変数も扱うことができます．近似解法のため得られた解の最適性の保証はありません．

本アルゴリズムは各制約式に疑似的に「重み」をもたせ，制約違反量の合計を目的関数と合わせて最小化します．

WLS の対象となる問題を以下のように定義します．

$$
\begin{aligned}
\text{minimize} \quad & c^\top x + \frac{1}{2} x^\top Q x & \\
\text{subject to} \quad & f_i(x) \le b_i, & i \in M_L, \\
& f_i(x) \ge b_i, & i \in M_G, \\
& f_i(x) = b_i, & i \in M_E, \\
& l_j \le x_j \le u_j, & x_j \in N_I \cup N_C, \\
& x_j \in \mathbb{Z}, & j \in N_I, \\
& x_j \in \mathbb{R}, & j \in N_C.
\end{aligned}
$$

ここで，$f_i(x)$ は線形式あるいは二次式を表します．

上の問題に対して重み w^+, w^- を導入し，以下のような緩和問題を考えます．

$$
\begin{aligned}
\text{minimize} \quad & c^\top x + \frac{1}{2} x^\top Q x + \sum_{i \in M_L \cup M_E} w_i^+ y_i^+ + \sum_{i \in M_G \cup M_E} w_i^- y_i^- & \\
\text{subject to} \quad & f_i(x) - y_i^+ \leq b_i, & i \in M_L, \\
& f_i(x) + y_i^- \geq b_i, & i \in M_G, \\
& f_i(x) - y_i^+ + y_i^- = b_i, & i \in M_E, \\
& l_j \leq x_j \leq u_j, & x_j \in N_I \cup N_C, \\
& x_j \in \mathbb{Z}, & j \in N_I, \\
& x_j \in \mathbb{R}, & j \in N_C, \\
& y_i^+ \geq 0, & i \in M_L \cup M_E, \\
& y_i^- \geq 0, & i \in M_G \cup M_E.
\end{aligned}
$$

アルゴリズムの各反復では，重み w^+, w^- を当該制約式の違反量が大きいものほど大きくなるよう内部で自動調整します．探索時は自動調整された重み w^+, w^- に基づき，上記緩和問題において目的関数が減少する方向へ解が遷移します．これにより制約条件に違反している解から違反を解消する方向や目的関数値が減少する方向へ解が遷移し，より良い解が得られます．

本アルゴリズムは緩和問題に対して局所探索を行うため，制約式をすべて満たす解が存在しない場合においてもできるだけ制約を満たす解が得られます．

本アルゴリズムはソフト制約付きの最適化問題を扱えます．ここでソフト制約とは，制約違反に対して所与の重みに比例したペナルティがかかる制約のことです．ソフト制約に違反した解も実行可能解となります．したがってソフト制約は通常の制約に比べ満たすべき優先度が低く，ソフト制約同士であれば所与の重みが大きいほど優先度が高いと解釈できます．本アルゴリズムでは所与の重みを重み w^+, w^- の自動調整の際の上限値として扱うことで，上記の優先度に合わせた制約違反の解消を実現します．

類似アルゴリズムである制約充足問題ソルバ wcsp との関連に触れながら，技術的な詳細を簡単に説明します．WLS と wcsp はいずれも局所探索の性能向上のための工夫をもつ手法です．wcsp はタブー・サーチと呼ばれるアルゴリズムを用いることで得られる局所最適解の質の向上を狙います．一方で WLS は近傍操作の種類を増やすことで同様の効果を狙います．各近傍操作において wcsp は 1 つの離散変数の値しか変化させないのに対し，WLS は最大 4 つの整数変数の値を変化させます．wcsp が複数回の近傍操作をしないと得られないような解も WLS は一回の近傍操作で得られます．

WLS は近傍操作の計算時間の削減を「隣接リスト」というデータ構造を用いて実現します．ここで隣接リストとは，変数同士の関連度の推定値に基づき，強く関連する変数の組を保持するリストです．2 つ以上の整数変数の値を変化させる際にはこの隣接リストを用い，改善解を得る見込みが薄い近傍操作をスキップします．

本アルゴリズムは，係数が 1 である線形な制約式に対する高速化手法を取り入れています．したがってそのような制約式が多い最適化問題，特に大規模な集合被覆問題や集合分割問題に対して高い性能を発揮します．

本アルゴリズムは [19] に基づき設計されています．

高度な機能として，変数に対し「割当ラベル」が設定された場合，WLS は設定から二部グラフ（頂

点が 2 つのグループに分けられ、異なるグループの頂点同士だけが辺で結ばれるグラフ）を構築することによって，探索範囲を拡大します．4 つの整数変数の値を変化させる近傍操作において割当ラベルの情報が考慮され，割当の交換などの有望な近傍操作に基づき，より深く探索を行います．

本アルゴリズムは複数スレッドを用いた並列計算に対応しています．並列計算では「Supervisor」および「Worker」と呼ばれる役割が協調して処理を行います．Supervisor は並列化動作を統制し，Worker は与えられたタスクを処理します．本並列計算により，次の 2 点の性能向上が見込めます．

- 複数の Worker が異なる探索戦略に基づき求解することで，優良な解を発見しやすくなる．
- 発見した優良な解をすべての Worker で共有することで，各 Worker の探索効率が向上する．

本並列計算は非決定性処理のため，同じ計算環境かつ同じ入力データであっても，異なる計算プロセスを経る可能性があります．並列計算を実行する際の求解オプションについては「解法 wls に有効な求解オプション」の項に解説があります．

付録 C 使い方に関するサポート

C.1 ユーザーサポートのページ

ユーザーサポートのページ（https://www.msi.co.jp/solution/nuopt/support.html）にお客様からよく寄せられるご質問をまとめました．お問い合わせの前に，是非一度ご確認ください．

C.2 使い方サポートサービス

年間保守にご加入の方は，使い方サポートサービスをご利用いただけます．以下のページの「製品サポート」フォームからお問い合わせください．

https://www.msi.co.jp/solution/nuopt/top.html#info

なお，データおよびプロジェクトファイルをお送りいただく場合には，いったんお送りいただく旨をフォームの通信欄にてお知らせいただければこちらよりセキュアなデータ転送サービスご利用についてご案内をいたします．

「製品サポート」フォームをご利用いただけない場合，下記アドレスに E-Mail でお問い合わせください．

nuopt-support@ml.msi.co.jp

E-Mail でのお問い合わせの際には下記を明記してください．

- ご利用の製品名
- バージョン
- シリアル ID
- ご登録者様のお名前
- ご質問事項

ご質問に関わるデータやプロジェクトファイルなどは，直接メール添付をしないようお願いいたします．（容量により，エラーとなる場合がございます．）

データおよびプロジェクトファイルをお送りいただく場合には，いったんお送りいただく旨を E-Mail にてお知らせいただければこちらよりセキュアなデータ転送サービスご利用についてご案内をいたします．

フォームおよび E-Mail でのお問い合わせについては，回答は一営業日以内に行います．もし回答がない場合，送信いただいた E-Mail がエラーとなっている等の場合があります．お手数ではございますが，今一度，宛先のメールアドレス等をご確認ください．どうしても原因が分からない場合は，下記

までお電話にてご連絡下さい.（使い方のご質問そのものは，お電話ではお受けしておりませんので，ご注意ください.）

（株）NTT データ数理システム　営業部　03-3358-6681

参考文献

[1] J. E. Beasley(ed.), Advances in Linear and Integer Programming, Oxford University Press, 1996.

[2] I. Bongartz, A. Conn, N. Gould and Ph. L. Toint, CUTE: Constrained and Unconstrained Testing Environment, Research Report RC 18860, IBM, T. J. Watson Research Center, Yorktown, U.S.A., 1993.

[3] V・フバータル著/阪田省二朗・藤野和建訳，線形計画法（上/下），啓学出版，1983.

[4] I. S. Duff and J. K. Reid, The Multifrontal solution of indefinite sparse symmetric linear systems, ACM Transaction on Mathematical Software, Vol.9,No.3 ,302-325,1983.

[5] P. E. Gill, W. Murray, M.A.Saunders and M.H.Wright, A practical anti-cycling procedure for linearly constrained optimization, Mathematical Programming, 45:437-474, 1989.

[6] P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright, Inertia-controlling methods for general quadratic programming, SIAM Review, 33:1-36, 1991.

[7] W. Hock and K. Shittkowski, Test examples for nonlinear programming codes, Springer Verlag, 1981.

[8] 田辺隆人，山下浩，主双対外点法とそのパラメトリック最適化への応用，2005 年日本オペレーションズ・リサーチ学会秋季研究発表会アブストラクト集 50-51.

[9] K. Nonobe and T. Ibaraki, A tabu search approach for the constraint satisfaction problem as a general problem solver, European Journal of Operational Research 106, 599-623, 1998.

[10] K. Nonobe and T. Ibaraki, An improved tabu search method for the weighted constraint satisfaction problem, INFOR 39, 131-151, 2001.

[11] 伊理正夫，今野浩，刀根薫監訳，最適化ハンドブック，朝倉書店，1995.

[12] 矢部博，八巻直一，非線形計画法，朝倉書店，1999.

[13] H. Yamashita, A globally convergent primal-dual interior point method for constrained optimization, Optimization Methods and Software, Vol. 10, 443-469, 1998.

[14] H. Yamashita, H. Yabe and T. Tanabe, A globally and superlinearly convergent primal-dual interior point trust region method for large scale constrained optimization, Mathematical Programming vol. 102, 111-151 2005.

[15] H. Yamashita and H. Yabe, Superlinear and quadratic convergence of some primal-dual interior point methods for constrained optimization, Mathematical Programming vol. 75, 377-397 1996.

[16] K. Nonobe and T. Ibaraki, Formulation and tabu search algorithm for the resource constrained project scheduling problem, in: C.C. Ribeiro and P. Hansen (eds.): Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, pp.557-588, 2002.

[17] T. Ralphs, et al., Parallel solvers for mixed integer linear optimization, In Handbook of parallel constraint reasoning, Springer, 2018, pp. 283-336.

[18] Y. Shinano, et al., Solving open MIP instances with ParaSCIP on supercomputers using up to 80,000 cores. In: Parallel and Distributed Processing Symposium, 2016 IEEE International. IEEE. 2016, pp. 770-779.

[19] S. Umetani, Exploiting variable associations to configure efficient local search in large-scale set partitioning problems, Twenty-Ninth AAAI Conference on Artificial Intelligence, February 2015.

[20] Maros, István. Computational techniques of the simplex method. Vol. 61. Springer Science & Business Media, 2002.

[21] Munguía, LM., Ahmed, S., Bader, D.A. et al. Alternating criteria search: a parallel large neighborhood search algorithm for mixed integer programs. Comput Optim Appl 69, 1 – 24 (2018).

索　引

記号・数字

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

あ

お

か

き

く

こ

さ

し

せ

そ