TTSopt/Documentation, version 0.9

Romana Jezek, Roman Kostal, Maximilian Stollmayer

April 15, 2024

Contents

1 Introduction

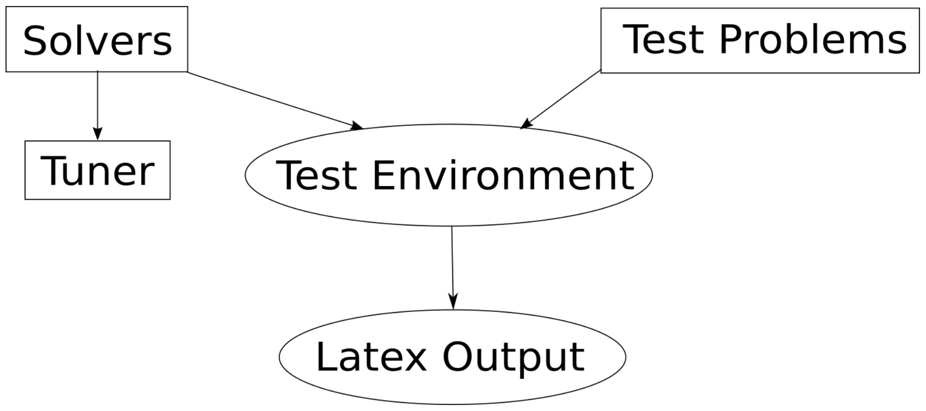
TTSopt (Testing and Tuning Software for Optimization) is a package for testing and tuning derivative free optimization software.

There are many different solvers and many problem collections available.

The Test Environment works with one or more solvers and a problem collection and saves the output into the directory from which a file displaying the results is created.

For each test problem we check whether a gradient is available and if it is we can run a sequence of gradient based solvers. In any case we can do gradient free optimization. At the end we can compare the solvers in terms of the number of sucessfully solved problems, the number of anomalies, the number of function evaluations, and the used time.

The solvers can be tuned with the best parameters for the solver.



Problem statement:

$$
\begin{aligned}
&\langle\text{type}\rangle \quad f(x) := \gamma + c^T x + \frac{1}{2} x^T G x + h(x) \\
&\text{where } \langle\text{type}\rangle \in \{\text{'min'}, \text{'max'}, \text{'0'}\}, \\
&\text{s. t.} \quad c_k(x) := a_k + A_k x + \frac{1}{2} Q_k(x) + H_k(x) \qquad (k = 1, \ldots, nc) \\
&\qquad x \in X \\
&\qquad x_i \in \begin{cases} \mathbb{R} & \text{if } \mathit{xtype}(i) = 0 \\ \mathbb{Z} & \text{if } \mathit{xtype}(i) = 1 \\ \{0,1\} & \text{if } \mathit{xtype}(i) = 2 \\ \{1, \ldots, m\} \text{ (categorical)} & \text{if } \mathit{xtype}(i) = m > 2 \end{cases} \\
&\qquad a_k \in \mathbb{R}^{d_k} \\
&\qquad A_k \in \mathbb{R}^{d_k \times n} \\
&\qquad H_k : \mathbb{R}^n \to \mathbb{R}^{d_k} \\
&\qquad Q_k(x) = \begin{cases} \frac{1}{2} x^T G_k x & \text{if } d_k = 1 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

2 Installation

2.1 Setup of TSopt

- For Octave:
  Run the file Octave_setup in the folder TEall-1.0.0
- For Matlab:
  Run the file Matlab_setup in the folder TEall-1.0.0

If you want to use the problem collection CUTEst you have to run the setup with enabling the install process of CUTEst. This will create the mex files of the problems.

The installation of CUTEst is interactive. Please answer no to the following questions:

- Do you wish to install GALAHAD
- Would you like to review and modify the system commands
- Would you like to review and modify the fortran compiler settings

- Would you like to review and modify the C compiler settings

If you use Octave answer also no to the question:

- Do you require the CUTEst-Matlab interface

Answer yes to the following questions:

- Would you like to compile SIFDecode
- Would you like to compile CUTEst

For the setup you need to have liboctave-dev and openMPI installed. If you start the setup with root rights you will be asked to install it during the installation.

Please change the directories in the following files:

- TE/TUNEall.m (line 30)
- TE/TESTall.m (line 20)
- PYTHON/Octave_interfaces/runNomad.py (line 25)
- PYTHON/Octave_interfaces/run_uobyqa.py (line 26)
- PYTHON/Octave_interfaces/run_bobyqa.py (line 29)

If you want to use the Powell solvers UOBYQA or BOBYQA in Octave you must set the environment variable OCTAVE_EXECUTABLE="octave-cli" in your bash profile, source the bash profile and run Octave from the terminal.

2.2 System Requirements

This software is tested to run with one of

- Ubuntu 20.04.6
- Ubuntu 22.04.2

and one of

- MATLAB R2021a
- Octave 5.2.0
- Octave 6.4.0

Other compilers might also work, but are not tested!

For the setup you need internet connection.

For Octave you need:

- liboctave-dev

For Matlab you need a suitable gcc- and gfortran-compiler. It is dependent on the Matlab version. For Matlab R2020b and R2021a you need gcc-8 and gfortran-8. But it is also tested with gcc-9 and gfortran-9 (the setup error must be comment). If you don't have this versions you can install it before running the setup. If you have root rights you can also run the setup and it will be installed automatically.

For the solver NOMAD you need:

- python (version 3.8 or higher)
- cmake (version 3.14 or higher)
- pip (version 23 or higher)
- a c++-compiler
- libopenmpi-dev

For the problem collection CUTEst you need:

- a c-compiler
- a fortran-compiler

2.3 Trouble shooting

If you get errors while installing CUTEst the reason could be that during the installation some original files of the software of CUTEst will be changed:

- `/TEprob/Collections/CUTEst/INSTALLATION/archdefs/bin/select_arch`

  (Here the gfortran version is changed from version 8 to no specified version in line 422 and 436)

- `/TEprob/Collections/CUTEst/INSTALLATION/cutest/bin/cutest2matlab`

  (Here the corresponding error message is commended)

- `/TEprob/Collections/CUTEst/INSTALLATION/cutest/bin/cutest2octave`

  (Here the corresponding error message is commended)

- `/TEprob/Collections/CUTEst/INSTALLATION/sifdecode/bin/sifdecoder`

  (Here the environment variable SED is replaced by /usr/bin/sed)

- `/TEprob/Collections/CUTEst/INSTALLATION/cutest/bin/runcutest`

  (Here -lgfortran is added to build the mex file for octave)

If you get errors about loading files in the folder HIT or in the folder RES there might be incompatibilities between Matlab and Octave. Use only Matlab OR Octave or delete the files and create them again.

If you get the error version `GLIBCXX_<version>` not found in Matlab, add

```
export  LD_PRELOAD="/usr/lib/x86_64-linux-gnu/libstdc++.so.<reqired version number>"
```

to the bashrc file and run Matlab from the command line.

If you get the error

```
error("Please install ... and set alternatives to the new version.")
```

you can commend the error in the setup. This is not guaranteed to work.

For any other error please contact one of the following persons:

- Arnold Neumaier - arnold.neumaier@univie.ac.at
- Morteza Kimiaei - morteza.kimaei@univie.ac.at
- Romana Jezek - romana.jezek@univie.ac.at

3 Using TTSopt

In this section we describe how to use the software for testing and tuning.

In the TE folder you can find all the main programs for testing, doing statistics and tuning.

The RES folder contains all results from testing and tuning.

In the HIT folder we can find mat files for each test function of a collection (only if we have chosen `param.firsthitupdate=1` and `param.hitupdate=1` in the main file TESTall.m).

3.1 Use for testing

TESTall.m is the main function for testing. You have to choose the parameters at the beginning. Choose the values of the parameters in the structure `param` by inserting into the given structure in the file TESTall.m the desired values. The possible entries are listed below:

Table 1: Using TESTall.m

Parameter	Description
`param` (Structure)	
`.username`	full path to the folder TEall: it depends on the location of the folder on the test users's system
`.probCollect`	**Choices**: • `CUTEst` • `bcp` **Description**: collection of test problems
`.prt`	**Choices**: 0 - 3 **Description**: print level
`.noiseFun`	**Choices**: • `1` • `0` **Description**: If `noisefun = 1` the function value will be noisy, otherwise it will be noiseless.
`.noiseLevel`	**Choices**: positive real number **Description**: level of noise
`.noiseType`	**Choices**: • `absolute` • `relative` **Description**: type of noise
`.noiseDistribution`	**Choices**: • `uniform` • `Gaussian` **Description**: noise distribution
`.accf`	**Choices**: one number in the range $[1e-4, 1e-3]$ **Description**: required upper bound of $q_f := \frac{(f-f_{best})}{(f_0-f_{best})}$
`.nfmaxManual`	**Choices**: • `1` • `0` **Description**: If it is `1` define `param.nfmaxString` ignoring the value of `param.valueBudget`. If it is `0` the default of `param.valueBudget` is used.
`.secmaxManual`	**Choices**: • `1` • `0` **Description**: If it is `1` define `param.secmaxString` ignoring the value of `param.valueBudget`. If it is `0` the default of `param.valueBudget` is used.

`.valueBudget`	**Choices**: • `low` • `medium` • `large` **Description**: how often a solver can search for a better function value: If it is `low` then `nfmax`=$100 * n$ and `secmax`=180. If it is `medium` then `nfmax`=$500 * n$ and `secmax`=180. If it is `large` then `nfmax`=$1000 * n$ and `secmax`=180.
`.initPoint`	**Choices**: • `standard` • `shifted` **Description**: If `initPoint = 'standard'` the initial point will be the point recommended by default. If `initPoint = 'shifted'` you can choose another point if e.g. the problem is solved too fast.
`.con`	**Choices**: • `unconstrained` • `bound constrained` • `unconstrained and bound constrained` • `linearly constrained` **Description**: type of constraints
`.key`	**Choices**: • `N` • `D` • `E` • `T` • `F` • `DEN` • etc. **Description**: sorts the list of test problems; every combination of `N`, `D`, `E`, `T`, `F` is possible. If it is `N` the list ist sorted by the name. If it is `D` it is sorted by the dimension. If it is `E` it is sorted by the number of free variables nfree. If it is `T` the list is sorted by the needed time sec2. If it is `F` it is sorted by nf2g2. If it is `'DEN'` it is sorted first by dim, then by nfree, then by name.

`.inc`	**Choices**: • `1` • `0` **Description**: sorts the list of test problems in increasing order if it is `1`, and in decreasing order if it is `0`.
`.res`	**Choices**: • `1` • `0` **Description**: Choose `1` if saved data in the RES folder is available and choose `0` if there is no data in the res folder.
`.solvedByAll`	**Choices**: • `1` • `0` **Description**: If it is `1` only the problems solved by at least one solver are added to the tables. If it is `0` the test problems unsolved by all solvers will be added to the tables (we never use this).
`.defaultDim`	**Choices**: • `1` • `0` **Description**: This is relevant only when CUTEst is used. If it is `1` then the problems with the symbol '_' in it's name will be deleted from the problem list.
`.initHitVar`	**Choices**: • `1` • `0` **Description**: `1` should be chosen when you run TEall the first time for initialising the hitlist. Then it should be `0`. At the end when you do statistics change it to `1` again.
`.updateHitVar`	**Choices**: • `1` • `0` **Description**: Choose `1` if you have not found the optimal values `fbest`, `xbest` and `0` if you have found the optimal values and run TE for statistics.

`.TypeVar`	**Choices**: • `0` • `1` • `2` **Description**: type of variables if it is 0: continuous if it is 1: integer if it is 2: mixed-integer
`.unSol2bcProb`	**Choices**: • `0` • `1` **Description**: if solver for unconstrained problems solves bound constrained problems
`.refsolver`	**Description**: Choose the solver name of the solver you want to use for calculating the free variables: e.g 'VRBBO'. The solver must be one of `param.soList`
`.scaleProb`	**Choices**: • `none` • `0` • `1` • `2` • `3` **Description**: We always choose `none` except for the benchmark. If it is `none` `defaultSolList` must be false and you have to define `param.soList` and the `param.dimension` ignoring the default which is classified by the problem scale. If it is `0` then $n \in [1, 20]$. If it is `1` then $n \in [21, 100]$. If it is `2` then $n \in [101, 1000]$. It it is `3` then $n \in [1001, 9000]$.
`.defaultSolList`	**Choices**: • `1` • `0` **Description**: If `defaultSolList = 0` you must fill the `param.soList` with the name of the solvers, you want to use, ignoring the default. The array `param.TuneCase` contains the tune case number used by the wrapper of the solver. If `param.TuneCase[i] = 0`, the default solver parameters are used and there is no tuning.

	If `TuneCase[i] = -1` the tuning is done by a mixed integer solver. If `TuneCase[i]` $\geq$ `1` the parameters in the structure `solverParams{TuneCase}`, defined in the wrapper, are used. If you want to use more than one tune case add to `param.soList` the solver name as often as you use it and to `param.TuneCase` the desired tune case numbers. If `defaultSolList = 1` the solver list is defined in ChoicesTestBBO according to the chosen scale.

The function TEpaths defines the paths which are needed automatically. After that the functions TESTChoices will run. The output parameters are used for the function TESTruns. No user input is needed.

3.2 Test Statistics

STATall is the main function for running statistics. It is used to compare the results of different solvers.
Before you can get statistical information you must have a good hitlist. This you get by running TESTall.m several times with the following settings:

1. In the first run choose only one solver because the first run is for creating a `mat` file for each problem.
   - `.initHitVar = 1`
   - `.updateHitVar = 1`
   - `param.soList` contains one solver
   - `param.TuneCase = [0]`
2. In the second run choose a sequence of competitive solvers:
   - `.initHitVar = 0`
   - `.updateHitVar = 1`
   - `param.soList` contains several solvers
   - `param.TuneCase` contains a list of n zeros, where n is the number of solvers in the `param.soList`
3. repeat the second step as long as you can find better function values
4. The last run is used for getting the statistics:
   - `.initHitVar = 1`
   - `.updateHitVar = 0`
   - `param.soList` contains the solvers we want to compare

- `param.TuneCase` contains a list of n zeros, where n is the number of solvers in the `param.soList`

The function STATcollect collects the information of every problem and saves it into a table. The function STATtable calculates the efficiency and saves all relevant informations for the LATEX output into the table. The `pdf` file with the results is saved into RES/PDF (see Section 3.4). In this file you can find the table with the dimension (dim), the number of free variables (nact) and the number of function evaluations (nf) for every problem and the efficiency for every solver. There are also many graphs for illustrating the performance of the solvers.

3.3 Use for tuning

The aim of the tuning process is to find the best parameters for every solver. The main file for tuning is TUNEall.m. The t<solver>.m files in TEallFinal/TUNERS include the current values and the upper and lower bound of the solver parameters. TUNEgetf computes the objective function of the tuning problem. The objective function is the mean of the ratios

$$r := \frac{f_{best} - f_{hit}}{f_{init} - f_{hit}}.$$

This ratio is computed for every problem in the function `getf` and saved in the output of the solver wrapper. The table of the ratios of all considered problems are created in the function `STATcollect`. The function `TUNEgetf` calculates the mean of the ratios.
In `TUNEall` the tuner can be chosen and the best function value and the best parameter values are saved into the file tunerResults.mat in the

`RES/<Collection>/PDF/<solver>` folder.

Choose the values of the parameters in the structure `param` by inserting into the given structure in the file TUNEall.m the desired values desribed above like for TESTall.m.. There are three more parameters:

Table 2: Using TESTall.m

Parameter	Description
`param` (Structure)	
`.discretizeParams`	**Choices**: • `1` • `0` **Description**: If the tuner is an integer solver choose 1, otherwise choose 0.

`.fixedParameter Indices`	**Choices**: array of integers, can be empty **Description**: if you want to fix some paramters enter the index of the parameter. The indices can be found in the tuner file in .../TEallFinal/TUNERS.
`.reducedBench`	**Choices**: • `1` • `0` **Description**: if you want to improve the performance you can reduce the number of problems. The problems will be ordered with the algorithm found in the paper. TODO: cite the paper

The output in the console is dependent on the print level chosen in TUNEall.

3.4 RES folder

The RES folder contains all results from testing and tuning.

1. The results of testing are collected in /RES/<Collection>/. The folder name contains:
   - resQ1 for derivative free
   - the stopping criterion (number)
   - resN⟨number⟩ where number is the upper bound of the number of function evaluations
   - resS⟨number⟩⟨point⟩ where number is the upper bound of the time and point is 'E' if the initial point is standard and 'N' if the initial point is shifted.

   In the subfolder there is a file results.txt where the summaries of the results are saved, and another subfolder called PDF where the graphs of the statistics are saved, if you have run STATall.m for doing statistics. In this case there is also a file results.tex which contains tables of summaries and comparisons of the different solver outputs.
2. The tuning results can be found in /RES/<Collection>/PDF/<solver>. There is a file tunerResults.pdf which contains the summary of the results.

3.5 HIT folder

In the HIT folder we can find mat files for each testfunction of a collection, only if we have chosen `param.firsthitupdate=1` and `param.hitupdate=1` in the main file TESTall.m. The .mat files contain the following information:

- `x`: the best point
- `f`: the best function value
- `checked`: `1` means the information will be updated
- `acc`: the maximum norm of the gradient
- `acc2`: the 2-norm of the gradient
- `nfree`: the number of free variables
- `normx`: the norm of the optimum point

We should update the files by running the solvers as long as we can find better points. If the point does not change anymore we can stop and do statistics or tuning.

4 Solvers

4.1 Current state

The software is tested with the following solvers:

1. solvers for unconstrained problems:
    - for noisy DFO problems with continous variables and dimension $\leq$ 30:
        - UOBYQA [28], [22], [21]
    - for noisy DFO problems with continous variables and dimension $\leq$ 500:
        - ADSMAX [7]
        - MADFO [17]
        - MCS [19]
        - MDSMAX [7]
        - NMSMAX [7]
    - for DFO problems with continous variables and dimension $\leq$ 10000:
        - SDBOX [24]
        - SSBBO [27]
        - VRBBO [29]
        - VRBBON [30]
2. solvers for bound constrained problems:

- for noisy DFO problems with continous variables ≤ 30:
  - BOBYQA [12], [21], [22]
- for noisy DFO problems with integer variables for dimension ≤ 10000:
  - DFLint [14]
  - DFLBOX [13]
- for noisy DFO problems with integer variables for dimension ≤ 30:
  - MATRS [18]
  - IMATRS [16]
- for noisy DFO problems with mixed-integer variables ≤ 10000:
  - DFLBOX [13]
- for noisy DFO problems with mixed-integer variables ≤ 30:
  - MATRS [18]

The solvers can be found in the folder SOLVERS.

4.2 Further untested solvers

The solvers that have no tuning parameters are not tested:

- FMINUNC [15]
- HOOKE [25]

Linearly constrained problems are not tested:

- PSWARM [23]

The following solvers are not finished:

- MATRS [18]
- LINCOA,BOBYQA,NEWUOA [21], [22]: is not finished because the names of the matrix and the vector of the linear constraints differ from the names in the TE code

The following solvers are not working at the moment:

- BFO [8], [9], [10], [11]: does not accept problem lists with different dimensions
- MDS [25]: tuning paramter dc should not be 0 or 1
- NOMAD [20]: runs only as tuner in Octave. It does not accept the same upper and lower bound, the wrapper is not finished.

- FMINUNCL [15]: in wrapper the function optimoptions is used, which is not known by Octave
- SNOBFIT [26]: does not accept problems with different dimensions (see Wrapper: params.npoint), in the wrapper getf is used without handling the finfo.error message
- DFLINT [14]: uses haltonset

4.3 Adding a new solver

If you want to add a solver to the software

1. go to the folder tunerWrapperGen and write a file <solver>.py (version >= 3.6, tested on 3.9) like the others found in the folder
2. open the terminal in this directory and type the command:

   ```
   python3 <solver>.py
   ```

   The generated wrapper and tuner files can be found in the folders WRAPPERS and TUNERS respectively.
3. update Masterdoc by adding the solver information in Masterdoc.tex and compiling it to pdf.

Specify tuning parameters by creating a list with instances of the **Parameter** class using the following arguments:

```
name : str
  Required argument that specifies the name of the parameter.

value : float | int | str
  Required argument with a predefined value of the parameter.

tuning_range : list(float | int | str)
  Optional argument that specifies the interval of continuous or integer
  tuning variables or all the options of categorical ones.
  Defaults to empty list.

input_type : type
  Optional argument that specifies what kind of values the
  parameter attains.
  Must be ‘float‘, ‘int‘ or ‘str‘.
  Defaults to type of given ‘value‘.

optimization_type : str
  Optional argument that specifies how the variable is treated
  when tuning.
```

```
  Must be ‘"continuous"‘, ‘"integer"‘ or ‘"categorical"‘.
  Default depends on type of given ‘value‘ as follows:
  ‘float‘ -> ‘"continuous"‘,
  ‘int‘ -> ‘"integer"‘,
  ‘str‘ -> ‘"categorical"‘

description : str
  Optional argument that specifies a short description of the parameter
  and should fit into half a line.
  Defaults to empty string.

info : str
  Optional argument that provides more information than the description.
  Will be broken into multiple lines if necessary.
  Defaults to empty string.
```

Create an instance of <Solver> using the following arguments:

```
name : str
  Required argument that specifies the name of the solver.

params : list(Parameter)
  Optional argument that defines all the tuning parameters.
  Defaults to empty list.

paths : list(str)
  Optional argument that specifies all paths of the solver
  which are needed for calling it and will be added to the workspace.
  Defaults to empty list.

preparation : str
  Optional argument that is valid MATLAB code, which is needed
  to prepare all input options of the solver.
  Defaults to empty string.

call : str
  Optional argument that is valid MATLAB code, which calls the solver
  and processes the output as needed.
  Defaults to empty string.

tune_cases : list(list(Parameter))
  Optional argument that specifies more predefined parameter options
  that can be selected when calling the wrapper.
  Defaults to empty list.
```

Call the desired wrapper or tuner writing method on the defined solver instance:

```
Solver.write_wrapper(save_path)
```

```
  save_path : str
    Optional argument that specifies path to which the wrapper
    will be saved.
    Defaults to path that the script is executed from.

Solver.write_tuner(save_path)
  save_path : str
    Optional argument that specifies path to which the tuner
    will be saved.
    Defaults to path that the script is executed from.
```

To make structural changes to all wrappers and tuners edit the WRAPPER_TEMPLATE.txt or TUNER_TEMPLATE.txt. Make sure to not alter the keywords, unless you intend to edit them in data.py too. All solver scripts that require this change then need to be run again.

See example.py for a showcase of an ideal structure for such a script. The files wSOLVER.m and tSOLVER.m were generated with this script.

5 Problem collections

There are various problem collections sourced from both academia and industry:

- bcp [2]
- bcpInt [2]
- bcpMint [2]
- CUTEst [3]
- global [1]
- globalInt [1]
- globalMint [1]
- ibm [4]
- ibmInt [4]
- ibmMint [4]
- inf [5]
- infInt [5]
- infMint [5]
- minlp [1]
- minlpInt [1]

- minlpMint [1]
- MIPLIB [6]
- MIPLIBint [6]
- morg [1]
- morgInt [1]
- morgMint [1]
- prince [1]
- princeInt [1]
- princeMint [1]

5.1 Current state

The tested collections are:

- CUTEst [3]
- bcp [2]

5.2 The universal data structure

For the purposes of the project, the problem collections were translated into a single common MATLAB/Octave format ('the format'), to avoid having to write separate code for each problem collection when solving the actual problems. The universal structure S is stored as a .mat file (or created on the fly if this is faster) for each problem. It contains the fields:

- S.info
  - .names
    - .prob
    - .obj
    - .vars
    - .con{k}
  - .table
- S.prob
  - .obj
    - .const [scalar]

    - .sense ['max' / 'min' / '0']
    - .lin [vector]
    - .quad [matrix]
    - .nol [function handle]
  - .con{k}
    - .const [scalar]
    - .nrows [int]
    - .lin [matrix]
    - .quad [symmetric matrix]
    - .nol [function handle]
    - .lb [vector]
    - .ub [vector]
  - .domain
    - .xtype [vector] (e.g. [11100042] 0 real, 1 int, 2 bin, $k \geq 3$ categorical with k cat.)
    - .lb [vector]
    - .ub [vector]
    - .startp{q} [vector] (q-th starting point)
- S.hits{k} = $m \times 1$ cell array of structs where m is the number of available hits
  - .names (names of variables)
  - .values (values of variables)
  - .obj (value of objective at hit)

5.3 TEprob folder

The folder TEprob contains:

1. Collections - contains a folder 'name' for each of the interfaced collections with sub-folder and files:
   - given - contains the problem collection as it was downloaded from the source
   - DotMat - contains a .mat file for each problem in the format
   - 'name'_Table.mat - a table with problems and their parameters

- 'name'_Plot.jpg - a scatter plot of the collection (# of variables vs. # of proper constraints)
- 'name'_Plot.fig - same as above but as .fig

2. CommonCode
   - 'setBasePath.m' this is where the base path is defined as a global variable. All other routines then refer to the base path. It should be executed before running other programs.
   - 'readProb.m' is the main interface program of the whole collection. It is used by the solver suite and depending on the number of arguments, returns information about a problem collection, loads a concrete problem or a just a part thereof. Please see description inside 'readProb.m'
   - 'addCollection.m' a program intended to provide a template for future users who would like to add another collection to package, documented in detail in 'How_to_add_a_collection.pdf'
   - 'createStruct.m' a subroutine called by 'addCollection.m', this is a template just like 'addCollection.m'
   - 'changeNames.m' naming conflicts or characters not being allowed to use with solvers are common issues which come up. This file provides a template for quickly changing names of a subset of problems in a given collection
   - 'makeTables.m' a program for generating tables as it is written it loops through all problem collections and all problems, but a the user can easily modify it to loop through subsets thereof.
3. CommonInfo contains a 'overview_Table.mat' - a table of the collections and their parameters

5.4 Adding a new problem collection

To add a new collection to the package:

1. set the path to the folder with the problem collection `addCollection.m`;
2. edit `createStruct.m` to do its job. A template is provided but the user has to fill in the details;
3. run `addCollection.m`;
4. use `makeTables` to generate tables and graphs.
5. update Masterdoc.pdf by adding the new problem collection in the Masterdoc.tex file and compile it to Masterdoc.pdf.

This translates whatever format is given to the common format.

References

[1] N. V. Sahinidis, *Optimization test problems.* Webdocument (`https://minlp.com/optimization-test-problems`), accessed 04.04.2024.

[2] Y. Puranik and N. V. Sahinidis, *Bounds tightening on optimality conditions for nonconvex box-constrained optimization.* Journal of Global Optimization, 67, 59-77, 2017, `https://minlp.com/downloads/testlibs/mlibs/bcp.zip`.

[3] N. I. M. Gould, D. Orban and P. L. Toint, *CUTEst: a Constrained and Unconstrained Testing Environment with safe threads for mathematical optimization*, Computational Optimization and Applications, 60, 545-557, 2013, `https://github.com/ralna/CUTEst`.

[4] *CMU/IBM MINLP Project.* Webdocument (`http://egon.cheme.cmu.edu/ibm/page.htm`), accessed 04.04.2024, downloaded from `https://minlp.com/downloads/testlibs/barlibs/ibm.zip`.

[5] Y. Puranik and N. V. Sahinidis, *Deletion presolve for accelerating infeasibility diagnosis in optimization models.* INFORMS Journal on Computing, 29, 754-766, 2017, `https://minlp.com/downloads/testlibs/barlibs/inf.zip`.

[6] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. M. Christophel, K. Jarck, T. Koch, and J. Linderoth, M. Lübbecke, H. D. Mittelmann, D. Ozyurt, T. K. Ralphs, D. Salvagnin and Y. Shinano, *MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library.* Mathematical Programming Computation, 2021, `https://miplib.zib.de/download.html`

[7] N. Higham, *The Matrix Computation Toolbox.* `https://www.mathworks.com/matlabcentral/fileexchange/2360-the-matrix-computation-toolbox`, MATLAB Central File Exchange. Retrieved 04.04.2024.

[8] M. Porcelli and Ph. L. Toint, *BFO, a trainable derivative-free Brute Force Optimizer for nonlinear bound-constrained optimization and equilibrium computations with continuous and discrete variables*, ACM Transactions on Mathematical Software, 44:1 , Article 6, 2017.

[9] M. Porcelli and Ph. L. Toint, *A note on using performance and data profiles for training algorithms*, ACM Transactions on Mathematical Software, 45:2, Article 20, 2019.

[10] M. Porcelli and Ph. L. Toint, *Global and local information in structured derivative free optimization with BFO*, arXiv:2001.04801, 2020.

[11] M. Porcelli and Ph. L. Toint, *BFO*, `https://github.com/m01marpor/BFO`.

[12] M. J. D. Powell, *The BOBYQA algorithm for bound constrained optimization without derivatives*, Technical Report DAMTP 2009/NA06, Department of Applied Mathematics and Theoretical Physics, Cambridge University, Cambridge, UK, 2009.

[13] G. Liuzzi, S. Lucidi and F. Rinaldi, *Derivative-free methods for bound constrained mixed-integer optimization*, Kluwer Academic Publishers, 53, 2, USA, 2012, `https://github.com/DerivativeFreeLibrary/DFL`.

[14] G. Liuzzi, S. Lucidi and F. Rinaldi, *An algorithmic framework based on primitive directions and nonmonotone line searches for black box problems with integer variables*, 2018, `https://github.com/gliuzzi/DFLINT`.

[15] Matlab Optimization Toolbox. Webdocument (`https://de.mathworks.com/help/optim/ug/fminunc.html`). Accessed 04.04.2024.

[16] M. Kimiaei, A. Neumaier. *Efficient composite heuristics for integer bound*, constrained noisy optimization, unpublished manuscript, 2022, `https://github.com/GS1400/IMATRS`.

[17] M. Kimiaei and A. Neumaier, *Effective matrix adaptation strategy for noisy derivative-free optimization*, Math Program Comput, 2023, `https://github.com/GS1400/MADFO`.

[18] M. Kimiaei and A. Neumaier, *Heuristic methods for noisy derivative-free bound-constrained mixed-integer optimization*, manuscript, Math Program Comput, 2023, `https://github.com/GS1400/MATRS/`.

[19] W. Huyer and A. Neumaier, *Global optimization by multilevel coordinate search*, J. Global Optimization 14, 331-355, 1999, `https://arnold-neumaier.at/software/mcs/`.

[20] C. Audet, S. Le Digabel, V. Rochon Montplaisir and C. Tribes. *NOMAD version 4: Nonlinear optimization with the MADS algorithm.* ACM Transactions on Mathematical Software, 48, 3, 35:1-35:22, 2022, `https://github.com/bbopt/nomad/releases/tag/v.4.4.0`.

[21] Z. Zhang, *PRIMA: Reference Implementation for Powell's Methods with Modernization and Amelioration*, DOI: 10.5281/zenodo.8052654, 2023, `https://github.com/libprima/prima/releases/tag/v0.7.2`.

[22] T. M. Ragonneau and Z. Zhang. *PDFO: a cross-platform package for Powell's derivative-free optimization solvers*, 2023, `https://github.com/pdfo/pdfo/releases/tag/v2.1.0`.

[23] A. I. F. Vaz and L. N. Vicente, *PSwarm: a hybrid solver for linearly constrained global derivative-free optimization*, Optimization Methods

and Software, 24, 669-685, 2009, `http://www.norg.uminho.pt/aivaz/pswarm/`.

[24] S. Lucidi and M. Sciandrone, *A Derivative-Free Algorithm for Bound Constrained Optimization*, Computational Optimization and Applications 21, 119–142, 2002, `https://github.com/DerivativeFreeLibrary/DFL`

[25] C. T. Kelley, *Iterative Methods for Optimization*, Frontiers in Applied Mathematics, 1999, `https://ctk.math.ncsu.edu/matlab_darts.html`.

[26] W. Huyer and A. Neumaier, *SNOBFIT – Stable Noisy Optimization by Branch and Fit*, ACM Trans. Math. Softw., 35, 9:1-9:25, 2008, `https://arnold-neumaier.at/software/snobfit/`.

[27] M. Kimiaei, A. Neumaier and P. Faramarzi, *New Subspace Method for Unconstrained Derivative-Free Optimization*, ACM Trans. Math. Softw. , 49, 4, Association for Computing Machinery, New York, NY, USA, 2023, `https://github.com/GS1400/SSDFO`.

[28] M. J. D. Powell, *UOBYQA: unconstrained optimization by quadratic approximation*, Math. Program., 92(B):555–582, 2002.

[29] M. Kimiaei and A. Neumaier, *Efficient global unconstrained black box optimization*, unpublished manuscript, 2022, `https://github.com/GS1400/VRBBO/`.

[30] M. Kimiaei, *A developed randomized algorithm with noise level tuning for large-scale noisy unconstrained DFO problems*, Numerical Algorithms, 2023, `https://github.com/GS1400/VRDFON/`.