SACOBRA_Py

Release 0.8

Wolfgang Konen

May 28, 2025

CONTENTS:

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER

ONE

OVERVIEW

this doc gives an overview over this documentation.

CHAPTER

TWO

CONSTRAINT OPTIMIZATION PROBLEMS

This document defines COPs (Constraint Optimization Problems) and introduces the G-problem benchmark.

2.1 COPs

A constrained optimization problem (COP) for numerical and continuous quantities in $\mathbb{R}^d$ is defined as:

$$Min \quad f(\vec{x}), \quad \vec{x} \in [\vec{a}, \vec{b}] \subset \mathbb{R}^d$$

$$\text{subject to} \quad g_i(\vec{x}) \leq 0, \quad i = 1, 2, \ldots, m$$

$$h_j(\vec{x}) = 0, \quad j = 1, 2, \ldots, r$$

2.2 G-problem benchmark

The G-problem benchmark suite originates from a CEC 2006 competition [LiangRunar]. It is a set of 24 constrained optimization problems (COPs, G-problems) G01, ..., G24 with various properties like dimension, number of equality / inequality constraints, feasibility ratio, etc. Eight of the 24 COPs have equality constraints. Although these problems were introduced as a suite in the technical report [LiangRunar] at CEC 2006, many of them have been used by different authors earlier.

The G-problems are available in **SACOBRA_Py** as objects of class `GCOP`:

class `gCOP.GCOP`(*name*, *dimension=None*)

Constraint Optimization Problem Benchmark (G Function Suite)

[LiangRunar] J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. Suganthan, C. C. Coello, and K. Deb, "Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization," Journal of Applied Mechanics, vol. 41, p. 8, 2006. http://www.lania.mx/~emezura/util/files/tr_cec06.pdf

Example: Instantiate problem G01 or G02 with

- `G01 = GCOP("G01")`.
- `G02 = GCOP("G02", dimension=5)`.

Only problems G02 and G03 have the extra parameter `dimension`. All other problems G01, G04, ..., G24 have fixed dimensions.

Objects of class GCOP have the following useful attributes:

- **name** name of the problem, given by the user as 1st argument
- **dimension** input space dimension of the problem. For the scalable problems `G02` and `G03`, the dimension should be given by the user, otherwise it will be set automatically
- **lower** lower bound vector, length = input space dimension
- **upper** upper bound vector, length = input space dimension
- **fn** the COP functions which can be passed to **SACOBRA_Py** (see parameter `fn` in `CobraInitializer`).
- **nConstraints** number of constraints
- **x0** the suggested optimization starting point, may be `None` if not available
- **solu** the best known solution(s), (only for diagnostics purposes). Can be `None` (not known) or a vector in case of a single solution or a matrix in case of multiple equivalent solutions (each row of the matrix is a solution)
- **fbest** the objective at the best known solution(s), (only for diagnostics purposes)
- **info** information about the problem, may be `None` if not available

CHAPTER

THREE

INITIALIZATION

this doc describes initialization

3.1 CobraInitializer

The initialization in **SACOBRA_Py** consists of the following steps:

- pass in the specification of the COP and all the options in *SACoptions* `s_opt`
- (optional, if `s_opts.ID.rescale==True`) rescale the problem in input space
- create the initial design, see *InitDesigner*
- adjust several elements according to constraint range, see *CobraInitializer.adCon()*
- calculate for each initial design point `numViol`, the number of violated constraints, and `maxViol`, the maximum constraint violation. If equality constraints are involved, calculate μ_{init} for an artificial feasibility band around each equality constraint and base `numViol` and `maxViol` on this artificial feasibility.
- calculate the so-far best (artificial) feasible point. If no point fulfills (artificial) feasibility, take from the set of points with minimum `numViol` the one with the best objective.
- set up result dictionary `self.sac_res`
- adjust DRC according to objective range, see *CobraInitializer.adDRC()*

class cobraInit.**CobraInitializer**(*x0*, *fn: object*, *fName: str*, *lower: ~numpy.ndarray*, *upper: ~numpy.ndarray*, *is_equ: ~numpy.ndarray*, *solu=None*, *s_opts: ~opt.sacOptions.SACoptions = <opt.sacOptions.SACoptions object>*)

Initialize SACOBRA optimization:

- problem formulation: x0, fn, lower, upper, is_equ, solu
- parameter settings: s_opts via `SACoptions`
- create initial design: A, Fres, Gres via `InitDesigner`

Parameters

- **x0** (*np.ndarray or None*) – start point, if given, then its dim has to be the same as `lower`. If it is/has NaN or None on input, it is replaced by a random point from `[lower, upper]`.
- **fn** – function returning `(1+nConstraints)`-dim vector: [objective to minimize, constraints]
- **fName** – function name
- **lower** – lower bound, its dimension defines input space dimension

- **upper** – upper bound (same dim as lower)
- **is_equ** – boolean vector with dim nConstraints: which constraints are equality constraints?
- **solu** (*np.ndarray or None*) – (optional, for diagnostics) true solution vector or solution matrix (one solution per row): one or several feasible x that deliver the minimal objective value
- **s_opts** (SACoptions) – the options

adCon()

Adjust several elements according to constraint range.

The following elements of self.sac_res may be changed: 'fn', 'Gres', 'Grange', 'GrangeEqu'

adDRC()

Adjust DRC (distance requirement cycle), based on range of Fres

class initDesigner.**InitDesigner**(*x0: ndarray*, *fn*, *rng*, *lower: ndarray*, *upper: ndarray*, *s_opts:* SACoptions)

Create matrix self.A with shape (P, d) of sample points in (potentially rescaled) input space [lower, upper] $\subset \mathbb{R}^d$, where P = initDesPoints and d = input space dimension.

Apply fn to these points and split the result in objective function values self.Fres (with shape (P,)) and constraint function values self.Gres (with shape (P,nC) where nC = number of constraints).

Parameters

- **x0** – the last point self.A[-1,:] is x0
- **fn** – see parameter fn in *cobraInit.CobraInitializer*
- **lower** – vector of shape (d,)
- **upper** – vector of shape (d,)
- **s_opts** (SACoptions) – object of class SACoptions. Here we use from element IDoptions s_opts.ID the elements initDesign and initDesPoints.

__call__() → tuple[ndarray, ndarray, ndarray]

Return the three results A, Fres and Gres of the initial design

Returns
(self.A, self.Fres, self.Gres)

Return type
(np.ndarray, np.ndarray, np.ndarray)

3.2 Options

All paramters (options) for SACOBRA_Py have sensible defaults defined. The user has only to specify those parameters where a setting different from the defaults is desired.

class opt.sacOptions.**SACoptions**(*feval=100*, *XI=None*, *skipPhaseI=True*, *DOSAC=1*, *saveIntermediate=False*, *saveSurrogates=False*, *verbose=1*, *verboseIter=10*, *important=True*, *cobraSeed=42*, *ID=<opt.idOptions.IDoptions object>*, *RBF=<opt.rbfOptions.RBFoptions object>*, *SEQ=<opt.seqOptions.SEQoptions object>*, *ISA=<opt.isaOptions.ISAoptions object>*, *EQU=<opt.equOptions.EQUoptions object>*, *MS=<opt.msOptions.MSoptions object>*, *TR=<opt.trOptions.TRoptions object>*)

The collection of all parameters (options) for **SACOBRA_Py**. Except for some general parameters defined in this class, they are hierarchically organized in nested option classes.

Parameters

- **feval** – number of function evaluations
- **XI** – Distance-Requirement-Cycle (DRC) that controls exploration: Each infill point has a forbidden-sphere of radius `XI[c]` around it. `c` loops cyclically through `XI`'s inidices. If `XI==None`, then CobraInitializer will set it, depending on objective range, to short DRC `[0.001, 0.0]` or long DRC `[0.3, 0.05, 0.001, 0.0005, 0.0]`.
- **skipPhaseI** – whether to skip **SACOBRA_Py** phase I or not
- **DOSAC** – controls the default options for `ISAoptions ISA`. 0: take plain COBRA settings, 1: full SACOBRA settings, 2: reduced SACOBRA settings
- **saveIntermediate** – whether to save intermediate results or not (TODO)
- **saveSurrogates** – whether to save surrogate models or not (TODO)
- **verbose** – verbosity level: 0: print nothing. 1: print only important messages. 2: print everything
- **verboseIter** – an integer value, after how many iterations to print summarized results.
- **important** – controls the importance level for some `verboseprint`'s in `updateInfoAndCounters`
- **cobraSeed** – the seed for RNGs. **SACOBRA_Py** guarantees the same results for the same seed
- **ID** (*IDoptions*) – nested options for initial design
- **RBF** (*RBFoptions*) – nested options for radial basis functions
- **SEQ** (*SEQoptions*) – nested options for sequential optimizer
- **ISA** (*ISAoptions*) – nested Internal SACOBRA options
- **EQU** (*EQUoptions*) – nested options for equality constraints
- **MS** (*MSoptions*) – nested options for model selection
- **TR** (*TRoptions*) – nested options for trust region

Distance Requirement Cycle

The Distance Requirement Cycle (DRC) is the vector `XI` that controls exploration: Each already evaluated infill point is surrounded by a forbidden-sphere of radius `XI[c]` with `c = i mod XI.size` (`c` loops cyclically through `XI`'s inidices`, that's where the name *cycle* comes from). A new infill point is searched under the additional constraint that it has to be a distance `XI[c]` away from all other already evaluted infill points. The larger `XI[c]`, the more exploration.

If `XI==None`, then `CobraInitializer` will set it, depending on objective range, to short DRC `[0.001, 0.0]` or long DRC `[0.3, 0.05, 0.001, 0.0005, 0.0]`. Both vectors contain `XI[c] = 0` which enforces exploitation. (If all entries were `XI[c] > 0` then the search could never continue in the close vicinity of an already good infill point.)

class opt.idOptions.**IDoptions**(*initDesign='RANDOM'*, *initDesPoints: int = None*, *initDesOptP: int = None*, *initBias=0.005*, *rescale=True*, *newLower=-1*, *newUpper=1*)

Options for the initial design (d = input dimension of problem).

Parameters

- **initDesign** – options: "RANDOM", "RAND_R", "RAND_REP", "LHS", … (see initDesigner)

- **initDesPoints** – number of initial design points. If None, cobraInit will set it to $d + 1$ if RBF.degree=1 or to $(d + 1)(d + 2)/2$ if RBF.degree=2
- **initDesOptP** – if None, cobraInit will set it to initDesPoints
- **initBias**
- **rescale** – if True, rescale input space from [lower, upper] to $[$ newLower, newUpper $]^d$
- **newLower** – common new lower bound for each of the d input dimensions
- **newUpper** – common new upper bound for each of the d input dimensions

class opt.rbfOptions.**RBFoptions**(*model='cubic'*, *degree=None*, *rho=0.0*, *rhoDec=2.0*, *rhoGrow=0*, *width=-1*, *widthFactor=1.0*, *gaussRule='One'*)

Options for the RBF surrogate models

Parameters

- **model** – RBF kernel type (currently only "cubic", but others will follow soon)
- **degree** – degree of polynomial tail for RBF kernel. If None, then RBFInterpolator will set it depending on kernel type
- **rho** – 0: interpolating RBFs, > 0: approximating (spline-like) RBFs. The larger rho the smoother
- **rhoDec** – exponential decay factor for rho
- **rhoGrow** – every rhoGrow (e.g. 100) iterations, re-enlarge rho. If 0, then re-enlarge never
- **width** – only relevant for scalable (e.g. Gaussian) kernels. Determines the width σ
- **widthFactor** – only for scalable kernels. Additional constant factor applied to each width σ
- **gaussRule** – only relevant for Gaussian kernels, see trainGaussRBF

class opt.seqOptions.**SEQoptions**(*optimizer='COBYLA'*, *feval=1000*, *tol=1e-06*, *conTol=0.0*, *penaF=[3.0, 1.7, 300000.0]*, *sigmaD=[3.0, 2.0, 100]*, *epsilonInit=None*, *epsilonMax=None*, *finalEpsXiZero=True*, *trueFuncForSurrogates=False*)

Options for the sequential optimizer

Parameters

- **optimizer** – string defining the optimization method for SACOBRA_Py phase I and II. One out of ["COBYLA","ISRESN"]
- **feval** – maximum number of function evaluations on the surrogate model
- **tol** – convergence tolerance for sequential optimizer
- **conTol** – constraint violation tolerance
- **penaF** – (TODO)
- **sigmaD** – (TODO)
- **epsilonInit** – initial constant added to each constraint to maintain a certain margin to boundary
- **epsilonMax** – maximum for constant added to each constraint
- **finalEpsXiZero** – if True, set in final iteration EPS and XI to zero for best exploitation
- **trueFuncForSurrogates** – if True, use the true (constraint & fitness) functions instead of surrogates (only for debug analysis)

CHAPTER

FOUR

OPTIMIZATION

this doc gives an overview over the optimization phases.

4.1 Phase I

TODO

4.2 Phase II

class cobraPhaseII.**CobraPhaseII**(*cobra:* CobraInitializer)

SACOBRA phase II executor.

Information is communicated via object *CobraInitializer* `cobra` (with elements sac_opts and sac_res) and via object *Phase2Vars* `p2` (internal variables needed in phase II).

get_cobra()

Returns
COBRA variables

Return type
CobraInitializer

get_p2()

Returns
phase II variables

Return type
Phase2Vars

start()

Start the main optimization loop of phase II

Returns
`self`

class phase2Vars.**Phase2Vars**(*cobra:* CobraInitializer)

Variables needed by `CobraPhaseII` (in addition to `CobraInitializer` `cobra`). These variables include:

- **EPS**
- **currentEps** number, the current safety margin EPS in constraint surrogates
- **num** number, the current equality margin mu, see equHandling.py

- **globalOptCounter** counter of the global optimization steps in phase II, excluding repair and trust region
- **Cfeas** how many feasible infills in a row (see adjustMargins, updateInfoAndCounters)
- **Cinfeas** how many infeasible infills in a row (see adjustMargins, updateInfoAndCounters)
- **fitnessSurrogate** the objective surrogate model
- **constraintSurrogates** the constraint surrogate models

Example: `p2 = Phase2Vars; print(p2.num);`

CHAPTER

FIVE

USAGE

this doc shows how to use **SACOBRA_Py**.

CHAPTER

SIX

PLAYGROUND RST

To retrieve a list of random ingredients, you can use the `lumache.get_random_ingredients()` function:

lumache.**get_random_ingredients**(*kind=None*)

Return a list of random ingredients as strings.

Parameters
kind (*list[str] or None*) – Optional "kind" of ingredients.

Raises
lumache.InvalidKindError – If the kind is invalid.

Returns
The ingredients list.

Return type
list[str]

##.. autoclass:: cobraInit.CobraInitializer ## :members: __init__, adCon, get_xbest, get_fbest ## :no-index:

cobraInit.CobraInitializer.**__init__**(*self*, *x0*, *fn: object*, *fName: str*, *lower: ~numpy.ndarray*, *upper: ~numpy.ndarray*, *is_equ: ~numpy.ndarray*, *solu=None*, *s_opts: ~opt.sacOptions.SACoptions = <opt.sacOptions.SACoptions object>*) → object

cobraInit.CobraInitializer.**get_fbest**(*self*)

Return the original objective function value at the best feasible solution.

Note: We cannot take the best function value via `sac_res['fn']`, because this may be modified by PLOG or others.

$\mathbb{R}^n$

This: $(x + a)^3$

$$\frac{1}{\left(\sqrt{\phi\sqrt{5}} - \phi\right) e^{\frac{2}{5}\pi}} = 1 + \frac{e^{-2\pi}}{1 + \frac{e^{-4\pi}}{1 + \frac{e^{-6\pi}}{1 + \frac{e^{-8\pi}}{1 + \cdots}}}}$$

BIBLIOGRAPHY

[LiangRunar] J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. Suganthan, C. C. Coello, and K. Deb, "Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization," Journal of Applied Mechanics, vol. 41, p. 8, 2006. http://www.lania.mx/~emezura/util/files/tr_cec06.pdf

INDEX