Coco suite of constrained problems: first results

Phillipe R. Sampaio

Inria-Saclay, France

May 26, 2016

Summary

1. Main features of the problems
2. Coco framework
3. Building the constrained problems
4. Current state of the implementation
5. First results
6. Next steps

Main features of the problems

- Linear constraints
- Scalable with dimension
- Non-trivial, with a few exceptions
- Mostly non-separable
- In some way comprehensible / ask a certain question or demand to be addressed by the solver / have, to some extent, known difficulties imposed to the solver
- Have a known optimal function value

Main features of the problems (cont.)

- Use the functions already implemented in Coco as objective functions of the constrained problems
- Have different number of constraints: 1, 2, 10, $n/2$, $n-1$, $n+1$
- The constraints are as independent as possible of the objective functions
- The constraints are randomly generated

Coco framework: BBOB suite

Functions, instances and problems

We consider functions, f_i, distinguished by their identifier $i = 1, 2, \ldots$.

Functions are further parametrized by the (input) dimension, n, and the instance number, j, that is, for a given q we have

$$f_i^j \equiv f(n, i, j) : \Re^n \to \Re^q \quad x \mapsto f_i^j(x) = f(n, i, j)(x) \ .$$

Coco framework: BBOB suite

Functions, instances and problems

We can think of j as an index to a continuous parameter vector setting, as it parametrizes, among other things, translations and rotations.

For instance, the function f_1 is built from the sphere function $\|x\|^2$, whose minimum is at the origin, by doing translations in the search and objetive spaces:

$$f_1(x) = \|x - x_{\text{opt}}\|^2 + f_{\text{opt}}$$

The new minimum becomes x_{opt} and the optimal function value is f_{opt}. Those two parameters are randomly generated using the instance value j.

Varying n or j leads to a variation of the same function i in the suite. By fixing n and j for function f_i, we define an optimization problem (n, i, j) that can be presented to the optimization algorithm.

Coco framework: BBOB-constrained suite

Functions, instances and problems

Each objective function f_i is tested with different numbers of linear constraints: 1, 2, 10, $n/2$, $n-1$ and $n+1$. We thus have the following structure:

Problem 1: f_1 and 1 linear constraint

Problem 2: f_1 and 2 linear constraints

...

Problem 6: f_1 and n+1 linear constraints

Problem 7: f_2 and 1 linear constraint

Problem 8: f_2 and 2 linear constraints

...

Problem 12: f_2 and n+1 linear constraints

...

Coco framework: BBOB-constrained suite

Functions, instances and problems

We know that f_1 is present in the problems ranging from 1 to 6 while f_2 is present in the problems ranging from 7 to 12, and so on.

In general, given a problem i, the identifier of the objective function used is given by the function $s(i) \equiv \lceil i/6 \rceil$.

We then denote the parametrized objective function in problem i by

$$f(n, s(i), j)(x)$$

Coco framework: BBOB-constrained suite

Functions, instances and problems

Similarly, each constraint is parametrized by the dimension, n, the number of the problem, i, the instance number, j, and its number in the problem, k (e.g. if a problem has two constraints c_1 and c_2, the values 1 and 2 are considered as parameters and are also used in the building process).

We can think of j here as a problem-related index used to define the linear constraint together with i and k. Therefore, different instances of a problem i have different linear constraints.

We denote a single constraint by $\boxed{c(n, i, j, k)(x)}$

Coco framework: BBOB-constrained suite

Functions, instances and problems

The total number of linear constraints in a problem i is a function on the number of the problem given by $h(i) \equiv ((i-1) \, mod \, 6) + 1$. For example, problem 7 has 1 linear constraint, which is confirmed by $h(7) = ((7-1) \, mod \, 6) + 1 = 1$.

The constraints of the problem i are then denoted by

$$c(n,i,j)(x) \equiv (c(n,i,j,1)(x), \ldots, c(n,i,j,h(i))(x))$$

Coco framework: BBOB-constrained suite

Functions, instances and problems

Finally, we denote the parametrized constrained problem i by

$$p(n,i,j)(x) \equiv (\, f(n,s(i),j)(x),\ c(n,i,j)(x)\,)$$

Varying the instances j, while n and i are fixed, implies on variations of the same objective function but it also implies on different linear constraints. The latter is due to the fact that the generation of the linear constraints takes into account the value of j as mentioned before.

By fixing n and j for problem i, we define an optimization problem (n,i,j) that can be presented to the optimization algorithm.

Coco framework

Runtime and targets

In order to measure the runtime of an algorithm on a problem, we establish a hitting time condition. For a single run, when an algorithm reaches or surpasses a **target value** t on problem (n, i, j), we say that it has solved this problem - it was successful.

Unconstrained case:
The runtime is the *evaluation count* when the target value t was reached or surpassed for the first time. That is, runtime is the *number of f-evaluations* needed to solve the problem (n, i, j).

Constrained case:
The runtime is the evaluation count when the target value t was reached or surpassed for the first time by a **feasible solution**.

In the current state, the evaluations count is based on the number of calls to the routine that evaluates the objective function only (to be discussed).

Building the constrained problems

Generating the linear constraints

Important: when generating the linear constraints, we must be sure that the resulting feasible set is *not empty*.

Proposed solution: consider the linear constraints in the form $Ax \leq 0$, where A is a $m \times n$ matrix. Steps to build the matrix A:

1) Sample a number m of vectors $a_1, a_2, \ldots, a_m$.
2) Choose a point p that we would like to be feasible ($Ap \leq 0$).
3) For each vector a_i such that $a_i^T p > 0$, redefine $a_i = -a_i$.
4) Define the rows of A using the vectors $a_1, a_2, \ldots, a_m$.

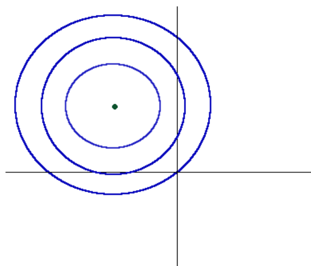
This ensures a "feasible half-line" defined by $\{\alpha p \,|\, \alpha \geq 0\}$.

The sampling of the vectors a_i is carried out using normal distribution.

Defining the constrained minimum

We build the problems in a way such that the constrained minimum is initially at the origin and it is different from the unconstrained minimum. We describe next how this is done.

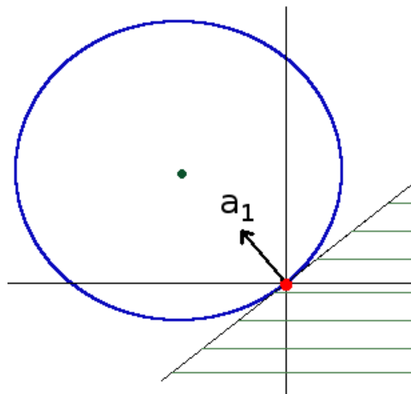
Step 1: we consider a function from the BBOB suite to be the objective function of the constrained problem.



Note: if the function contains nonlinear transformations, these are removed and are applied to the whole constrained problem in the end.

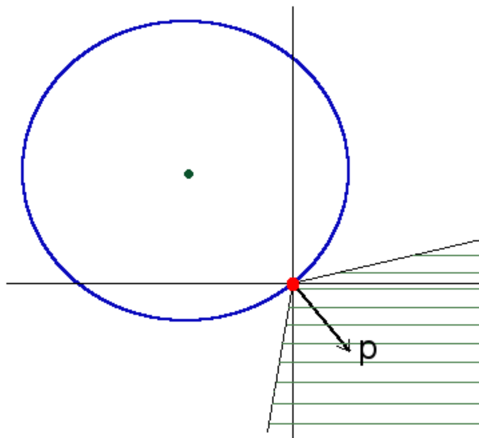
Defining the constrained minimum

Step 2: in order to have the *constrained minimum* at the origin, define the gradient of the first constraint, a_1, as $-\nabla f(0)$.



Defining the constrained minimum

Step 3: define the point $p = -\beta a_1 = \beta \nabla f(0)$, for some $\beta > 0$, as our feasible point and generate the other constraints randomly while making sure that p remains feasible.



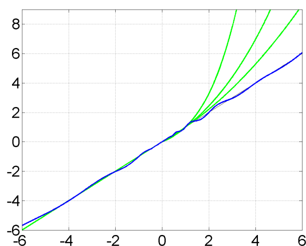
Implementation

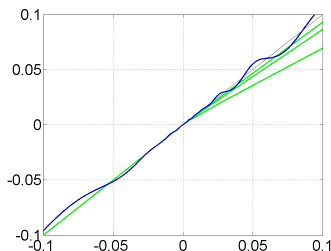
How the constrained problems are implemented in Coco:

```
p_f = objective_function_problem(...)
coco_evaluate_gradient(p_f, origin, feasible_direction)
p_c = single_linear_constraint(-feasible_direction,...)
for i=2...m
  # randomly generate the other gradients
   p_c2 = single_linear_constraint(...)
  # make sure that "feasible_direction" is feasible for p_c2
   p_c2 = guarantee_feasible_point(p_c2, feasible_direction)
   p_c = stacked_problem(p_c, p_c2)
endfor
p_f = stacked_problem(p_f, p_c)
# move the constrained minimum away from the origin
p_f = transform_vars_shift(p_f, xshift)
```

Applying nonlinear transformations to constrained problems

Coco makes use of two nonlinear transformations, T_{asy}^{β} and T_{osz}, that can be applied to the problems in order to make them less regular.





Question: is it possible to apply such transformations to constrained problems without affecting the constrained minimum's location?

Applying nonlinear transformations to constrained problems

Consider the constrained problem:

$$\begin{array}{ll} \min\limits_{x} & f(x) \\ \text{s.t.:} & a^T x \leq 0. \end{array}$$

Let $g : \Re^n \to \Re^n$ be an injective function. By "applying" g to the problem above, we obtain

$$\begin{array}{ll} \min\limits_{x} & f(g(x)) \\ \text{s.t.:} & a^T g(x) \leq 0. \end{array}$$

Assume that x^* is a global minimum to the first problem. Since g is injective, it has an inverse g^{-1}. It follows that $g^{-1}(x^*)$ is a global minimum to the second problem.

Applying nonlinear transformations to constrained problems

Since the transformations T_{asy}^{β} and T_{osz} in Coco, denoted by g in the previous slide, are strictly increasing functions, they are both injective, thus having inverse functions.

Answer to the question: using the definition of the transformations, their injectivity and the construction of the constrained problems in Coco, it is possible to apply these transformations into the constrained problems while keeping the minimum.

Since the constrained minimum x^* is initially at the origin in Coco, this means that $g^{-1}(x^*) = g^{-1}(0) = 0$. Therefore, the constrained minimum does not change if any of these transformations is applied.

Current state of the implementation

A first set of 48 linearly-constrained problems is implemented. The problems are composed by the following objective functions:

- Sphere
- Ellipsoid
- Linear slope
- Ellipsoid rotated
- Discus
- Bent cigar
- Different powers
- Rastrigin

Each one of the above functions is tested with 1, 2, 10, n/2, n-1 and n+1 linear constraints.

Current state of the implementation

Number of problems with

Separable objective function: 18

- Sphere
- Ellipsoid
- Linear slope

Ill-conditioned objective function: 24

- Ellipsoid rotated
- Discus
- Bent cigar
- Sum of different powers

Multi-modal objective function: 6

- Rastrigin

First results on the 48 problems

- Algorithm used: random search
- Budget: $200 * n$
- Plots shown on the pages HTML generated by the postprocessing code

Next steps

Extensions:

- Handle the cases where the objective function has neither convex level sets nor a known distribution of local minima.
- Should we store information about infeasible solutions? It would allow the assessment of the convergence to feasibility.
- Nonlinear constraints.