Fonctionnement de COCO

Introduction

COCO considère les problèmes d'optimisation continue de type black box.

La fonction a optimiser f:R^n->R^m avec n,m>=1

Notations

Espace de recherche : X

Ensemble de contraintes g

Δ (f) : précision de l'algo

Fopt : valeur optimale de la fonction à optimiser

Ftarget = Δ (f) + Fopt : valeur de la fonction à atteindre

D : dimension de l'espace de recherche

Ntrial : nombre d'essais pour chaque configuration

X est connu mais on ne dispose d'aucune information sur f

Evaluation

A single performance measure is used—and thereafter aggregated and displayed in several ways — namely runtime, measured in number of f-evaluations

Initialisation

Run_optimizer() : prend en paramètre:

- la fonction du problème
- la dimension de l'espace de recherche
- Le nombre maximum d'évaluations de la fonction (=coût)
- Valeur optimale de la fonction

Initialise les paramètres propres à notre optimiser puis appelle l'algo.

Résolution du problème

def NOTRE_ALGO(fun, x, maxfunevals, ftarget):

Notre optimiser. La fonction doit renvoyer la meilleure solution possible dans [-5,5]^D.

La fonction doit s'arrêter si :

- La valeur optimale est atteinte
- Le nombre max d'évaluations est atteint
- Ou qu'on dépasse le coup d'évaluation de la fonction (1^8 * D)

Paramètres généraux

minfunevals = 'dim + 2' # PUT MINIMAL SENSIBLE NUMBER OF EVALUATIONS for a restart

Maxrestarts : nombre max de fois ou le problème est relancé pour une config donnée

Suite_options : les dimensions des espaces de solution à tester

A faire :

Modifier run_optimiser pour appeler notre optimiser changer le nom et paramètres de l'algo

augmenter maxfunevals en fonction du CPU

Générations des résultats

python path_to_postproc_code/exampleexperiment.py
Résultats générés dans le dossier
“PUT_MY_BBOB_DATA_PATH”

Génération des graphes pour UN seul algo:

python
path_to_postproc_code/bbob_pproc/rungeneric.py
DATAPATH

Génération des graphes pour plusieurs algos:

python
path_to_postproc_code/bbob_pproc/rungeneric.py
ALG1 ALG2 ALG3

Code

Exampleexperiment.py => exemple sur un ensemble de fonctions

Exampletiming : calcul la complexité de l'optimiser

Exampleexperiment.py :

```
import time
import numpy as np
import cocoexp
import bbobbenchmarks as bn

datapath = 'PUT_MY_BBOB_DATA_PATH'
opt = dict(algid='PUT ALGORITHM NAME',
           comments='PUT MORE DETAILED INFORMATION, PARAMETER SETTINGS ETC')
maxfunevals = '10 * dim'  # 10*dim is a short test-experiment taking a few minutes
                          # INCREMENT maxfunevals successively to larger value(s)
```

Code 2

```
def run_optimizer(fun, dim, maxfunevals, ftarget=-np.Inf):
    """start the optimizer, allowing for some preparation.
    This implementation is an empty template to be filled

    """
    # prepare
    x_start = 8. * np.random.rand(dim) - 4

    # call
    PURE_RANDOM_SEARCH(fun, x_start, maxfunevals, ftarget)

def PURE_RANDOM_SEARCH(fun, x, maxfunevals, ftarget):
    """samples new points uniformly randomly in [-5,5]^dim and evaluates
    them on fun until maxfunevals or ftarget is reached, or until
    1e8 * dim function evaluations are conducted.

    """
    dim = len(x)
    maxfunevals = min(1e8 * dim, maxfunevals)
    popsize = min(maxfunevals, 200)
    fbest = np.inf

    for iter in range(0, int(np.ceil(maxfunevals / popsize))):
        xpop = 10. * np.random.rand(popsize, dim) - 5.
        fvalues = fun(xpop)
        idx = np.argsort(fvalues)
        if fbest > fvalues[idx[0]]:
            fbest = fvalues[idx[0]]
            xbest = xpop[idx[0]]
        if fbest < ftarget:  # task achieved
            break

    return xbest

minfunevals = 'dim + 2'  # PUT MINIMAL SENSIBLE NUMBER OF EVALUATIONS for a restart
maxrestarts = 10000      # SET to zero if algorithm is entirely deterministic

t0 = time.time()
np.random.seed(int(t0))
```

Code 3

```
e = cocoexp.Logger(datapath, **opt)
for dim in (2, 3, 5, 10, 20, 40):  # small dimensions first, for CPU reasons
    for f_name in bn.nfreenames: # or bn.noisynames
        for iinstance in range(1, 16):
            e.setfun(*bn.instantiate(f_name, iinstance=iinstance))

            # independent restarts until maxfunevals or ftarget is reached
            for restarts in range(0, maxrestarts + 1):
                run_optimizer(e.evalfun, dim,  eval(maxfunevals) - e.evaluations,
                              e.ftarget)
                if (e.fbest < e.ftarget
                    or e.evaluations + eval(minfunevals) > eval(maxfunevals)):
                    break

            e.finalizerun()

            print('  f%d in %d-D, instance %d: FEs=%d with %d restarts, '
                  'fbest-ftarget=%.4e, elapsed time [h]: %.2f'


                  % (f_name, dim, iinstance, e.evaluations, restarts,
                     e.fbest - e.ftarget, (time.time()-t0)/60./60.))

        print '      date and time: %s' % (time.asctime())
    print '---- dimension %d-D done ----' % dim
```