A Python/C library for bound-constrained global optimization with Continuous GRASP

TransfORMation
November 13-16, 2011
CHARLOTTE,
NORTH CAROLINA
Charlotte Convention Center

INFORMS Annual Meeting

Mauricio G. C. Resende
AT&T Labs Research
mgcr@research.att.com

Joint work with
Ricardo M. A. Silva (UFPE)
Panos M. Pardalos (Univ. of Florida)
Michael J. Hirsch (Raytheon Company)

Summary

Continuous GRASP

The libcgrpp library

An example

Paper is available

R.M.A. Silva, M.G.C.R., P.M. Pardalos, and M.J. Hirsch, "A Python/C library for bound-constrained global optimization with continuous GRASP," AT&T Labs Research Technical Report, Florham Park, 2011 (To appear in Optimization Letters, 2012).

http://www2.research.att.com/~mgcr/doc/cgrasp-gnu.pdf

Continuous GRASP

(C-GRASP)

C-GRASP

- C-GRASP is a metaheuristic to finding optimal or near-optimal solutions to
- Min f(x) subject to: $L \leqslant x \leqslant U$
  - where $x, L, U \in R^n$
  - and f(x) is continuous but can, for example, have discontinuities, be non-differentiable, be the output of a simulation, etc.

C-GRASP

C-GRASP is based on the discrete optimization metaheuristic GRASP

- M.J. Hirsch, C.N. Meneses, P.M. Pardalos, and M.G.C. Resende, "Global optimization by continuous GRASP," Optimization Letters, vol. 1, pp. 201-212, 2007.
- M.J. Hirsch, P.M. Pardalos, and M.G.C. Resende, "Speeding up continuous GRASP," European J. of Operational Research, vol. 205, pp. 507-521, 2010.
- M.J. Hirsch, "GRASP-based heuristics for continuous global optimization problems," Ph.D. Thesis, Dept. of Ind. & Syst. Eng., University of Florida, Gainesville, Florida, 2006.

C-GRASP

C-GRASP is a multi-start procedure, i.e. a major loop is repeated until some stopping criterion is satisfied.

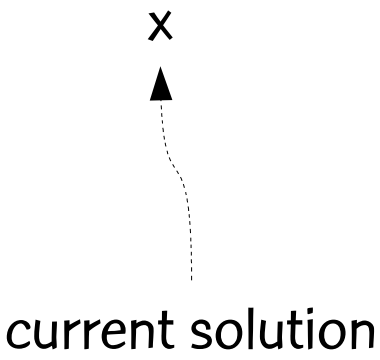
In each major iteration

- x is initialized with a solution randomly selected from the box defined by vectors L and U.
- a number of minor iterations are carried out, where each minor iterations consists of a construction phase and a local improvement phase.
- Minor iterations are done on a dynamic grid and stops when the grid has a pre-specified density.

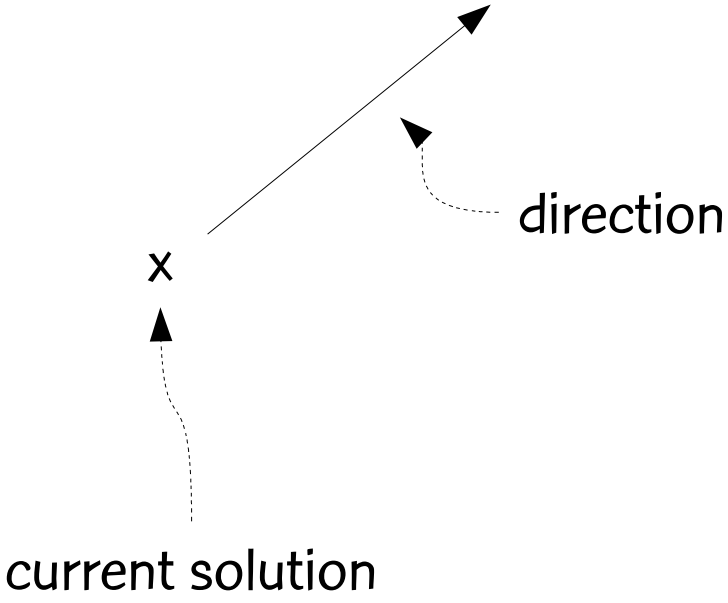
C-GRASP

```
f* = ∞
while (stopping criterion not satisfied) do
    x = random[L,U]; h = h(start);
    while ( h ≥ h(end) ) do
        x = ConstructGreedyRandomized(x)
        x = LocalImprovement(x)
        if ( f(x) < f* ) then { x* = x; f* = f(x) }
        if ( x did not improve this iteration ) then { h = h/2 }
    end while
end while
return x*
```

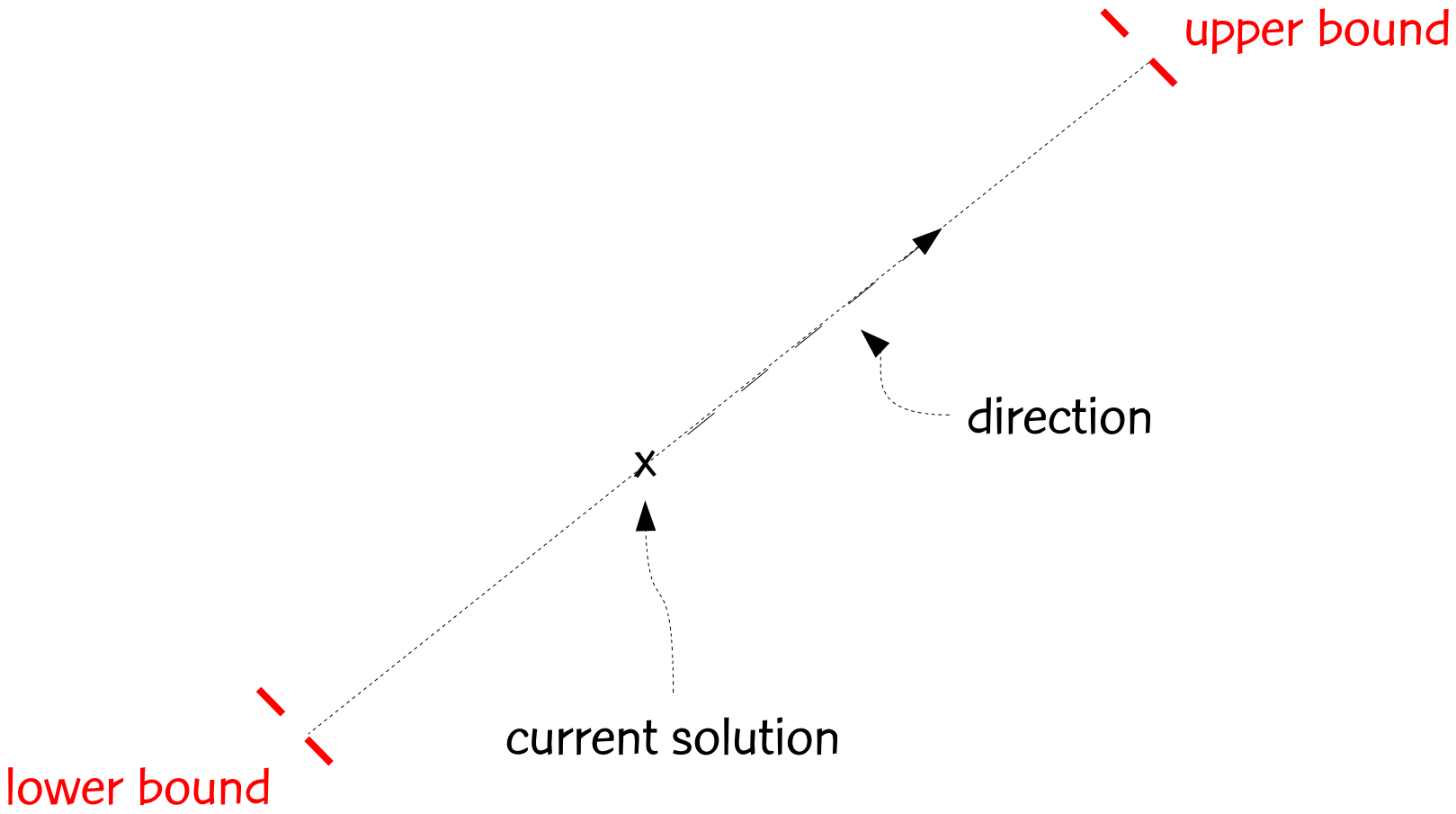
C-GRASP line search



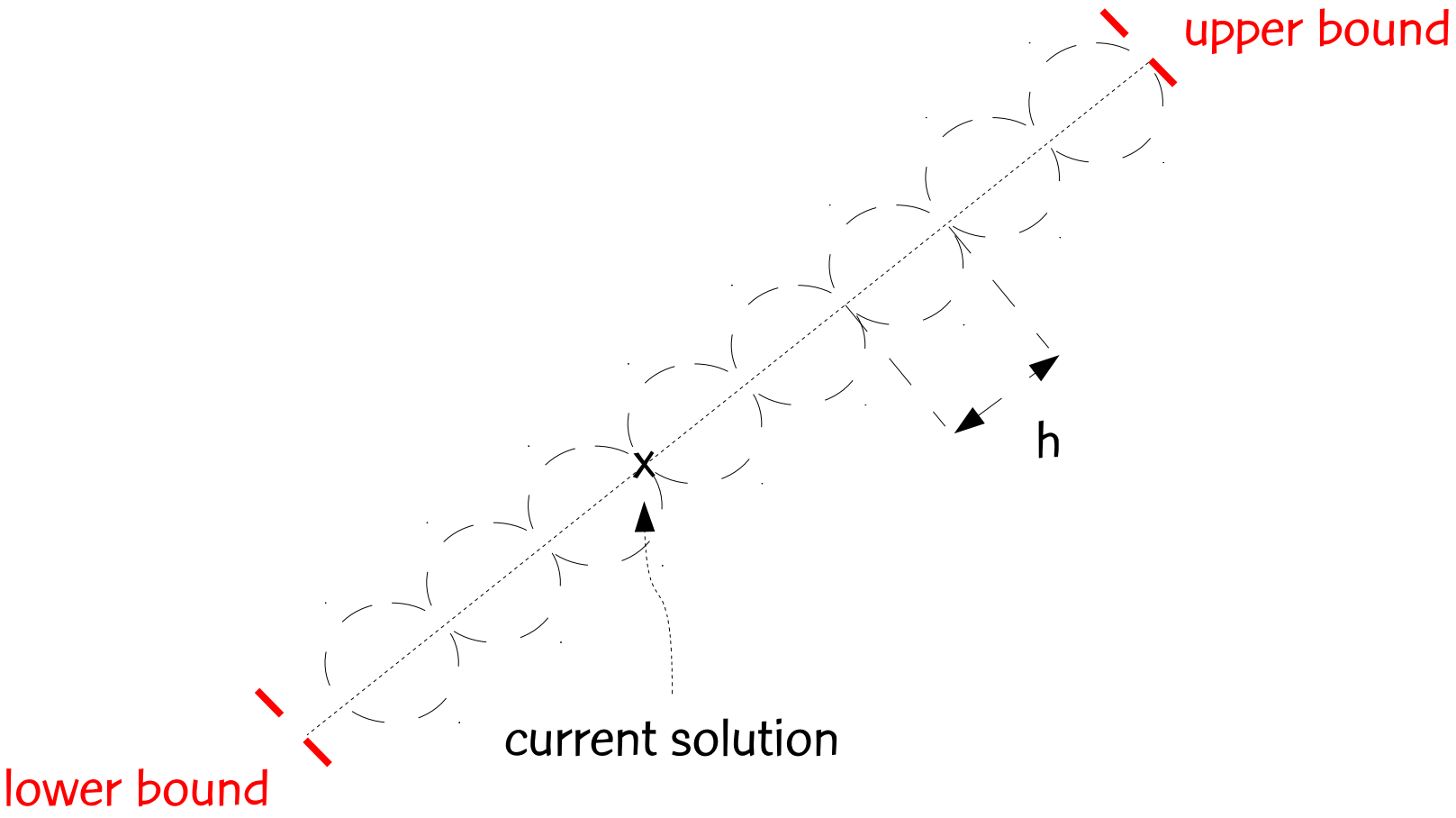
C-GRASP line search



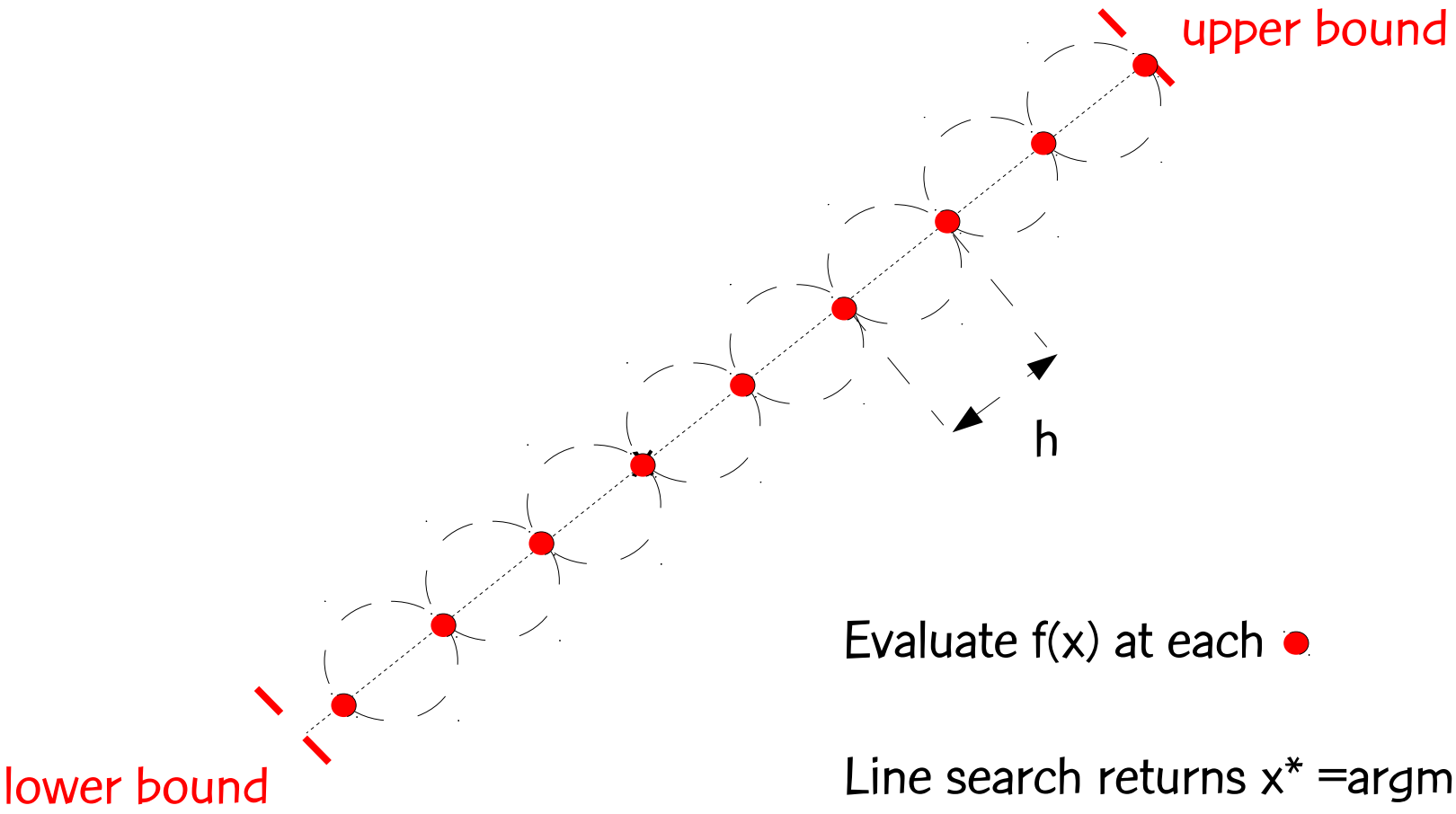
C-GRASP line search



C-GRASP line search



C-GRASP line search



Evaluate f(x) at each ●

Line search returns x* =argmin{ f(x) }

C-GRASP greedy randomized construction

unset = {1, 2, 3, ..., n }; $x = x^0$

for (k = 1, 2, ..., n) **do**

for (all i ∈ unset) **do**

z_i = line search in direction $e_i = (0,0,...,1,....,0)$

i-th component

end for

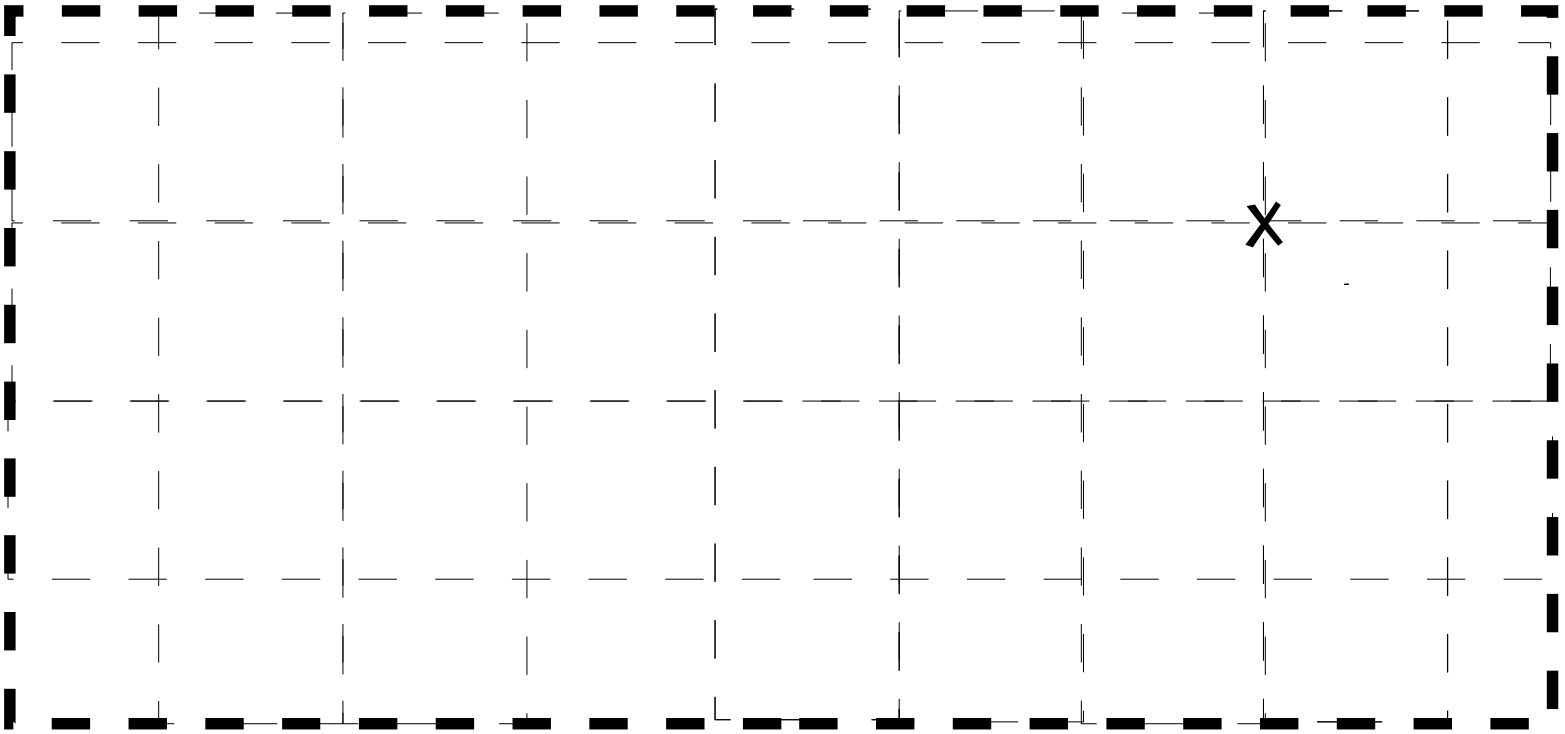
RCL = { i ∈ unset | $f(z_i) < CUTOFF$ }

Select at random i* ∈ RCL

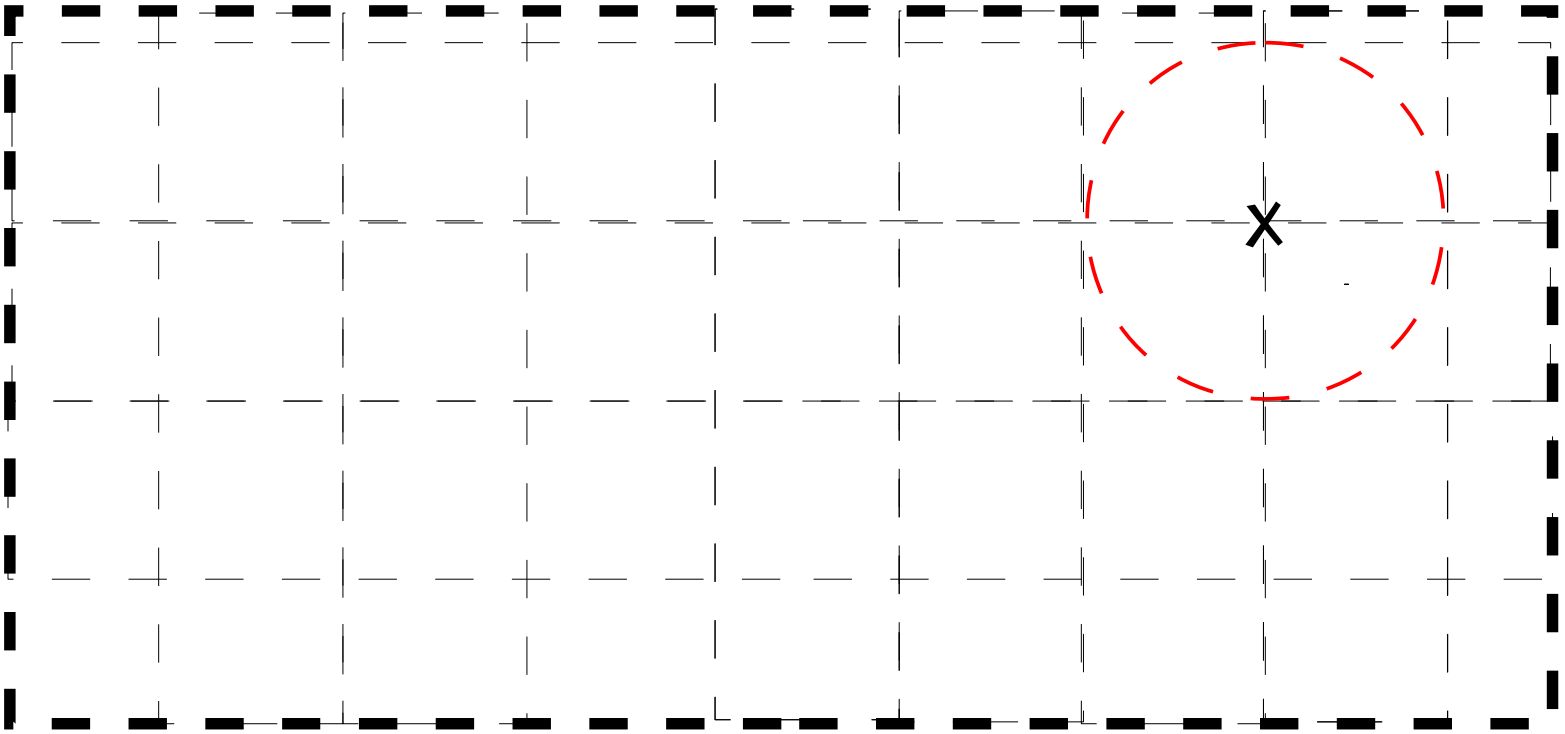
Set $x_{i^*} = z_{i^*}$; unset = unset \ {i*}

end for

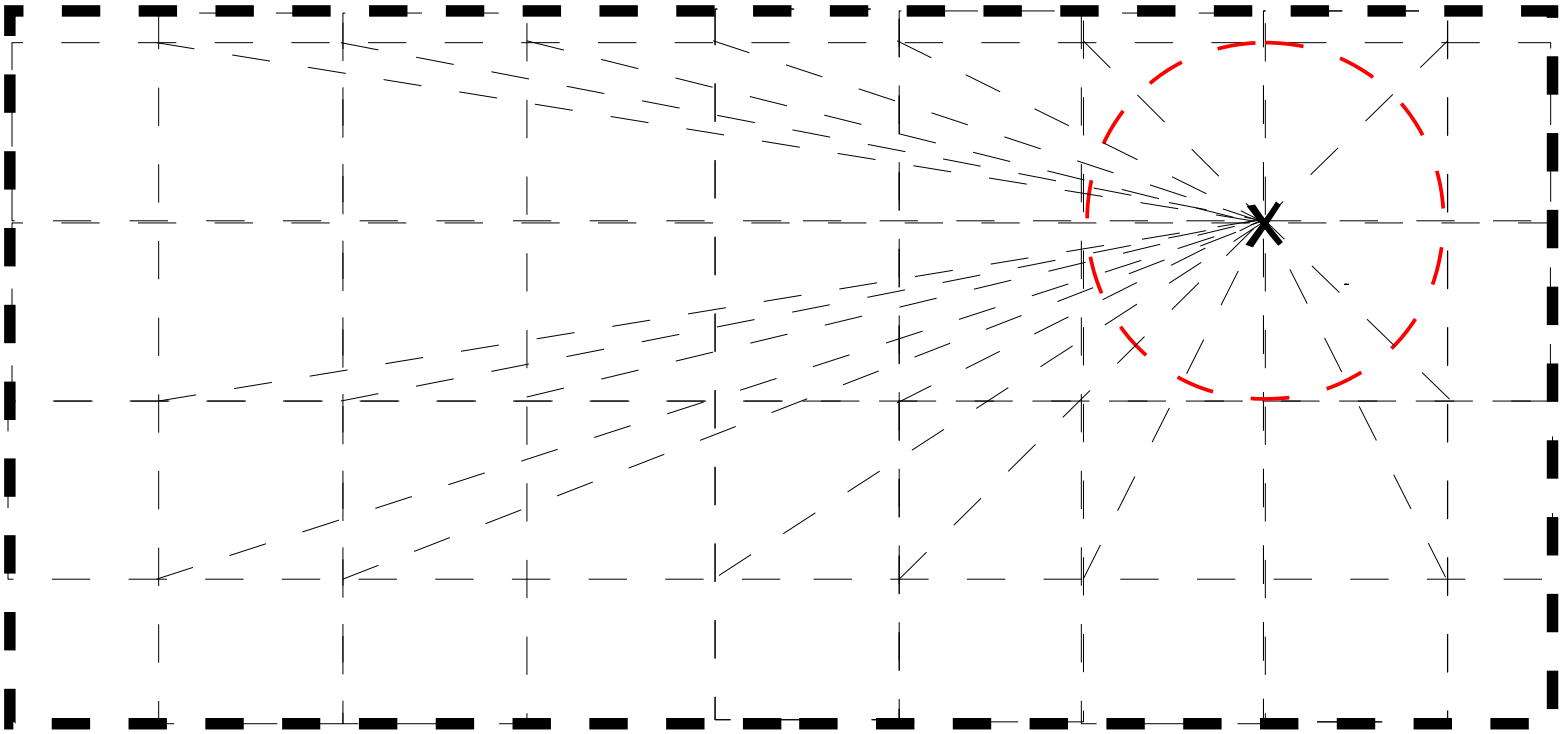
C-GRASP local improvement



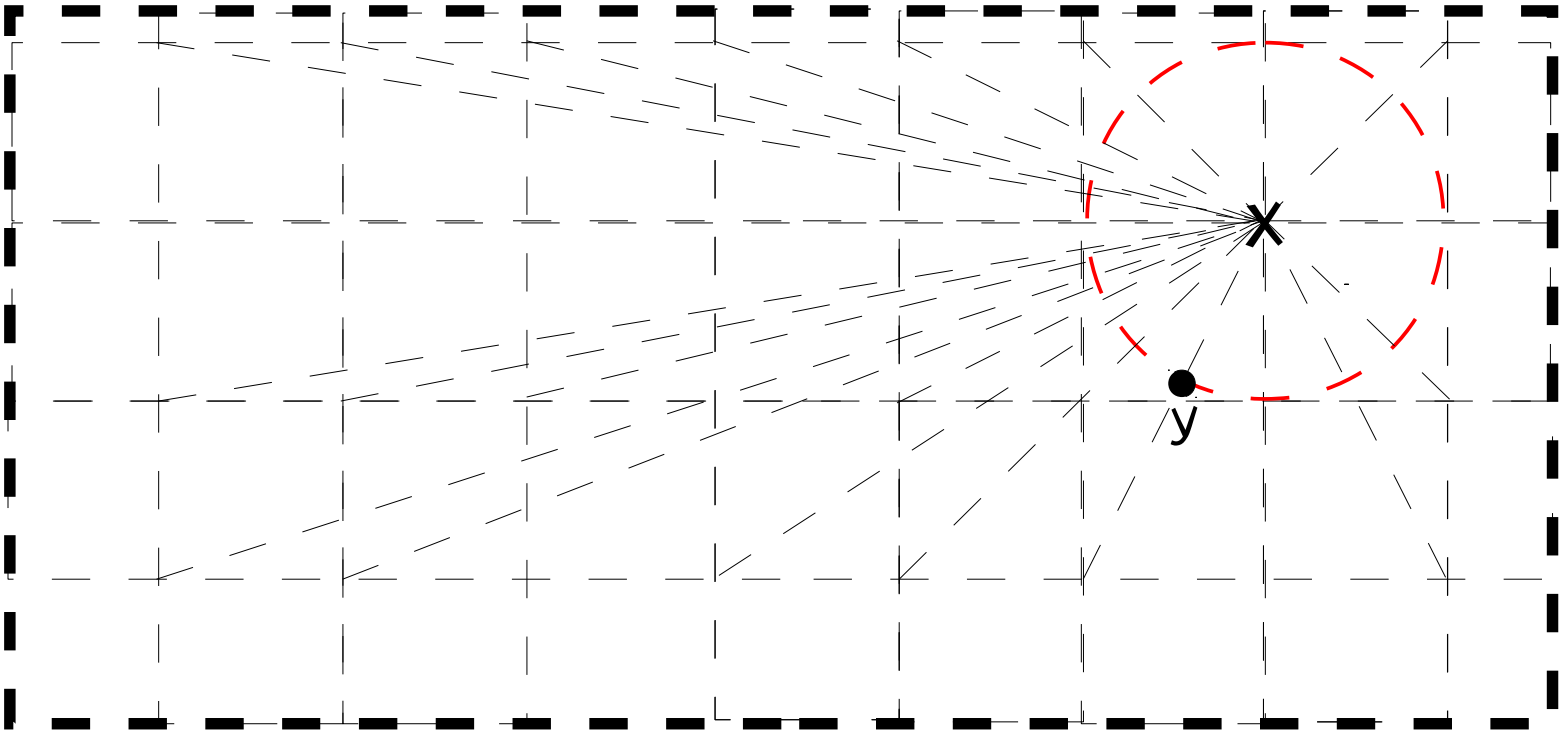
C-GRASP local improvement



C-GRASP local improvement



C-GRASP local improvement



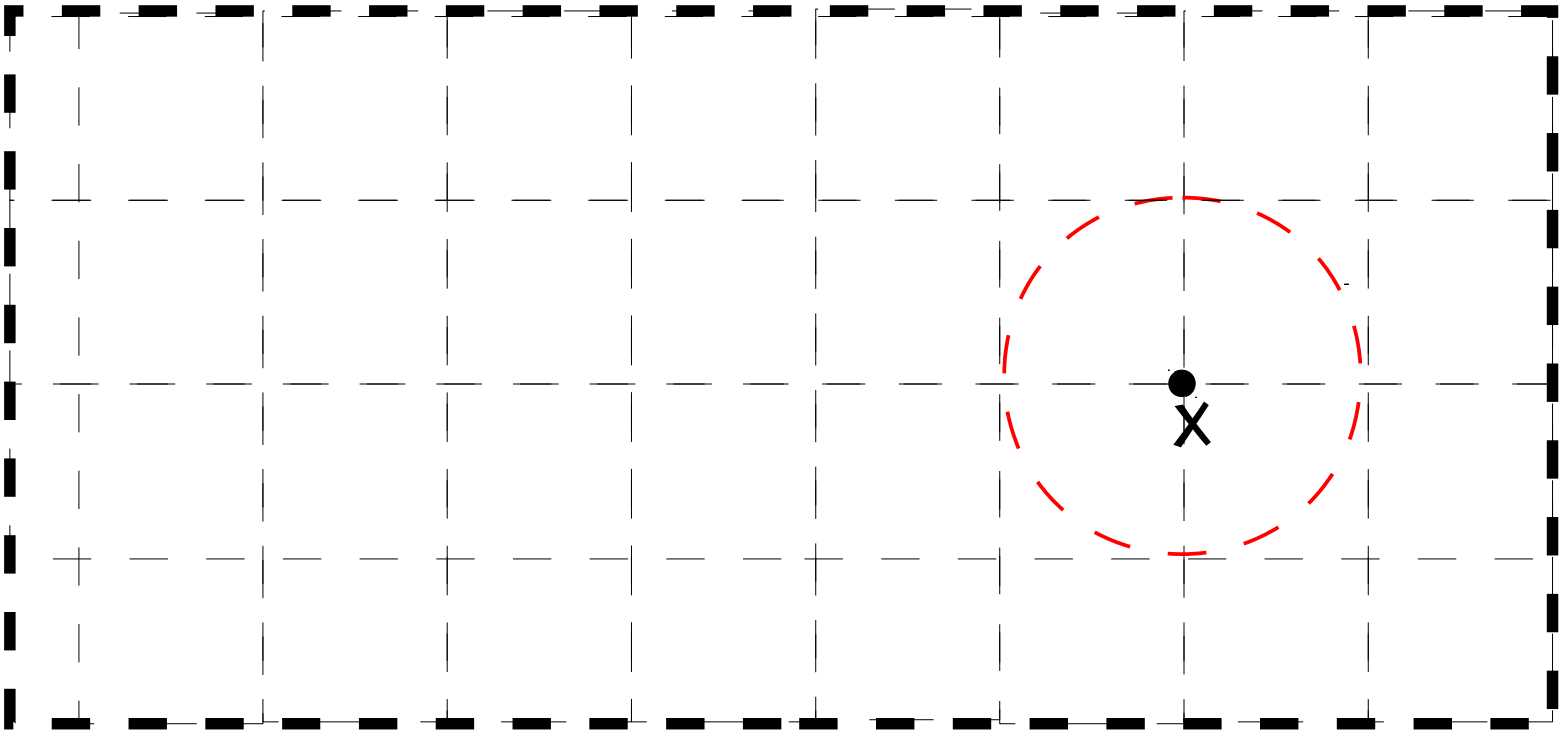
Sample projected point y on circle and evaluate f(y)

If f(y) < f(x) then set x = y, translate grid to intersect x

and restart local search from x

If max-points are examined without improvement: x is h-local min

C-GRASP local improvement



Sample projected point y on circle and evaluate f(y)

If f(y) < f(x) then set x = y, translate grid to intersect x

and restart local search from x

If max-points are examined without improvement: x is h-local min

C-GRASP

M.J. Hirsch, P.M. Pardalos, and M.G.C. Resende, "Sensor registration in a sensor network by continuous GRASP," IEEE Military Communications Conference (MILCOM), 2006.

- Sensor registration is the process of removing (accounting for) non-random errors, or biases, in sensor data.
- We solve the sensor registration problem when some data is not seen by all sensors, and the correspondence of data seen by the different sensors is not known.
- We outperform previous methods in the literature and have been granted two U.S. Patents.

C-GRASP

M.J. Hirsch, C.N. Meneses, P.M. Pardalos, M.A. Ragle, and M.G.C. Resende, "A continuous GRASP to determine the relationship between drugs and adverse reactions," in "Data Mining, Systems Analysis and Optimization in Biomedicine," O. Seref, O.Erhun Kundakcioglu, and P.M. Pardalos (eds.), AIP Conference Proceedings, vol. 953, pp. 106-121, Springer, 2008.

- We formulate the drug-reaction relationship problem as a continuous global optimization problem

C-GRASP

M.J. Hirsch, P.M. Pardalos, and M.G.C. Resende, "Solving systems of nonlinear equations with continuous GRASP," Nonlinear Analysis: Real World Applications, vol. 10, pp. 2000-2006, 2009.

- We formulate a system of nonlinear equations as nonlinear function which has min value zero. After finding a root, we add a barrier around the root and resolve to find the next root.

C-GRASP

E.G. Birgin, E.M. Gozzi, M.G.C. Resende, and R.M.A. Silva, "Continuous GRASP with a local active-set method for bound-constrained global optimization," J. of Global Optimization, vol. 48, pp. 289-310, 2010.

- We adapt C-GRASP for global optimization of functions for which gradients can be computed. To to this, we use GENCAN (Birgin and Martínez, 2002), an active-set method for bound-constrained local minimization as the local improvement procedure.

C-GRASP

M.J. Hirsch, P.M. Pardalos, and M.G.C. Resende, "Correspondence of projected 3D points and lines using a continuous GRASP," International Transactions in Operational Research, vol. 18, pp. 493-511, 2011.

- Computer vision application

C-GRASP

D.G. Macharet, A. Alves Neto, V.F. da Camara Neto, and M.F.M. Campos, "Nonholonomic path planning optimization for Dubins' vehicles," 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 2011.

- Path planning for robotic vehicles

C-GRASP

A. L. Guedes, F.D. Moura Neto, and G.M. Platt, "Double azeotropy: Calculations with Newton-like and continuous GRASP (C-GRASP)," International J. of Math. Modelling and Numerical Opt., vol. 2, pp. 387-404, 2011.

- Chemical engineering application

The libcgrpp library

libcgrpp library: features

- libcgrpp is a GNU-style dynamic shared Python/C library of C-GRASP
- Implemented as embedded Python-in-C to take advantage of the simplicity offered by the Python language in implementing complex multi-modal functions

libcgrpp library: features

- Functions can be implemented using the extensive standard library of Python and any non-standard module or library, such as SymPy.
- Functions implemented in Python are loaded automatically without the need to recompile any code.

libcgrpp library: dependencies

The libcgrpp library requires that the following packages be installed:

- Python programming language package (version $\geq$ 2.7).
- GNU Libtool library.

libcgrpp library: downloads

Full distribution of the libcgrpp library is available at

http://www2.research.att.com/~mgcr/src/cgrasp

as the gzipped tar file cgraspp-0.0.1.tar.gz containing the following directory structure:

Files	Description
cgrasp.c	Embedded Python-in-C code of C-GRASP
cgrasp.h	Header file
mt19937ar.c	C version of Mersenne Twister random # gen
mt19937ar.h	Header file
simclist.c	Library for handling lists
simclist.h	Header file
cgrasparser.py	Parser for parameter input file
AUTHORS	Author names and email addresses
ChangeLog	Changes made to package
configure	Script to configure package
COPYING	GNU General Public License
INSTALL	Installation instructions
Makefile	Makefile to build executable
NEWS	User visible changes to package
README	Purpose and instructions
THANKS	Thanks to contributors

libcgrpp library: function module implementation

- Objective function is implemented in Python.
- Consider the Ackley function:

$$A_n(x) = -20e^{-0.2[(1/n)(\sum(i=1..n) x\_i^2)]^{1/2}} - e^{(1/n)\sum(i=1..n)\cos(2\pi x\_i)} + 20 + e$$

- A Python implementation of this function is:

```
from math import *
def f(x):
    sum1 = sum( x[i]**2 for i in range(len(x)) )
    sum2 = sum( cos(2*pi*x[i]) for i in range(len(x)) )
    r = 1.0/len(x)
    return -20.0*exp(-0.2*sqrt(r*sum1))-exp(r*sum2)+20.0+e
```

libcgrpp library: input file formats

The input file (parsing by an embbeded pyparsing parser) must contain the entries:

- -md <module-name> : defines the name of the python module containing the multimodal function(s) to be minimized.
- -ft <function-name> : defines the name of the Python function that implements the multimodal function to be minimized.
- -ds <n> : sets the function dimension to the positive integer <n>.
- -ov <d> or -it <n> or -fe <n> : sets the target optimal objective function value to the real number <d> or sets the number of iterations to the positive integer value <n> or sets the number of function evaluations to the positive integer value <n>.
- -ep <d> : sets to the positive real number <d> the parameter ϵ of optimality gap equation: $GAP = |f(x) - f(x^*)| \leq \epsilon$, if $f(x^*)=0$, otherwise, $GAP \leq \epsilon |f(x^*)|$

  OBS: Note that this entry can only be used in conjunction with -ov <d>.
- -sd <n> : sets the seed of the pseudo-random generator to the positive integer <n>.
- -hs <d> : sets the starting grid discretization density hs to the positive real no. <d>.
- -he <d> : sets the ending grid discretization density he to the positive real.
- -ro <d> : sets the local improv. parameter ρ_{lo} to the positive real # <d> such that $0 < <d> \leq 1$.

libcgrpp library: input file formats (cont.)

- -ls <n> : turns local improvement procedure on (off) if <n> is equal to 1(0);
- -mp <n> : sets the parameter MaxPointsToExamine in the local improvement procedure to the positive integer <n>;
- -of <file-name> : defines the name of the output file to which the solution is written.
- -dm <l> <u> [list-of-exceptions] : sets bounds of the hyper-rectangle $S = \{x = (x_1, \ldots, x_n) \in R^n : l \leq x \leq u\}$, such that $l_i = <l>$ and $u_i = <u>$ for all $(i = 1, \ldots, n)$ dimensions.
  - Ex: -dm −10 10 sets the lower and upper bounds for all dimensions to −10 and 10;
  - Exceptions are used to specify bounds for dimensions for which bounds are different from <l> or <u>. They are expressed as follows:
    - <i> <lo> <up>, with 1 ≤ <i> ≤ n and <lo> ≤ <up>: sets the lower l_i and upper u_i bounds of i-th dimension to <lo> and <up>, respectively. Ex: the exception 3 -12 20 sets the lower and upper bounds of the third dimension to −12 and 20, respectively.
    - <i>:<j> <lo> <up>, with 1 ≤ <i> ≤ <j> ≤ n and <lo> ≤ <up>: sets the lower bounds k to <lo>, and the upper bounds uk to <up>, for all dimensions k = i, . . . , j. Ex: the exception 7:10 -13 17 sets the lower and upper bounds of 7th to the 10th dimensions to −13 and 17, respectively;
    - combinations between formats above described.

libcgrpp library: input file format example

The example input file:

-hs 0.5 -he 0.0001 -ro 0.01 -ls 1 -mp 100 -of output.file -sd 270001

-md ackley -ft f -ds 5 -ov 0 -ep 0.001 -dm -10 10 1 -5 3 4:5 -13 7

specifies that C-GRASP will try to find a solution

$$x \in S = \{x = (x1, \ldots, x5) \in \mathbb{R}^5$$

$$\text{with } (-5, -10, -10, -13, -13) \leq x \leq (3, 10, 10, 7, 7)\},$$

such that function f of python module (ackley.py) that implements the Ackley function will be such that

$$GAP = |A_5(x) - 0| \leq 0.001,$$

using the following parameters:

$$hs = 0.5,\ he = 0.0001,\ \rho_{lo} = 0.01,\ seed = 270001,$$

and MaxPointsToExamine = 100.

Using the libcgrpp library in a C/C++ program

To use the function double cgrasp(int, **char) of the libcgrpp library in a C program (which we shall call userprog.c):

1. Put #include <cgrasp.h> in the source code of the C program userprog.c:

```
#include <cgrasp.h>

...

double x;

...

void main(int argc, char **argv){

    ...

    x = cgrasp(argc,argv);

    ...

}
```

Using the libcgrpp library in a C/C++ program (cont.)

To run the program, type:

<program_name> <input_file_name>

(Example: ./userprog input)

libcgrpp library: output file formats

The program produces two kinds of output:

- STDERR (terminal): occasional error messages.
- STDOUT (terminal, unless redirected to a file with >) and FILE (file name specified by the "-of" option in the input file):
  1. Summary of the execution, including information about the instance itself as well as the execution parameters;
  2. For each objective function improvement, a line is printed with the following format: h=<value>, h_e=<value>

     o-iteration: <value> // outer loop

     i-iteration: <value> // internal loop

libcgrpp library: output file formats

The program produces two kinds of output:

- STDERR (terminal): occasional error messages.
- STDOUT (terminal, unless redirected to a file with >) and FILE (file name specified by the "-of" option in the input file):

3. For each objective function improvement, a line is printed with the following format: <responsible> :

<keyword> <value>

  - The procedure responsible for the improvement can be the construction procedure or the local search procedure or random
  - Keywords are self-descriptive: CPU time (in seconds) of improvement or best value or improved solution

libcgrpp library: output file formats (cont.)

The program produces two kinds of output:

- STDERR (terminal): occasional error messages.
- STDOUT (terminal, unless redirected to a file with >) and FILE (file name specified by the "-of" option in the input file):

4. Total CPU time (in seconds) in the following format:

   time: <value>

5. Total function evaluations in the following format:

   evaluations: <value>

6. Value of the overall best solution found in the following format:

   optimum: <value>

libcgrpp library: output file example (cont.)

Consider as an example the output generated by the algorithm to find a solution $x \in [-10, 10]^2$, such that the Booth function:

$$BO(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \leq \epsilon = 0.001$$

using the following parameters:

$$hs = 0.5,\ he = 0.0001,\ \rho_{lo} = 0.01,\ seed = 270001,$$

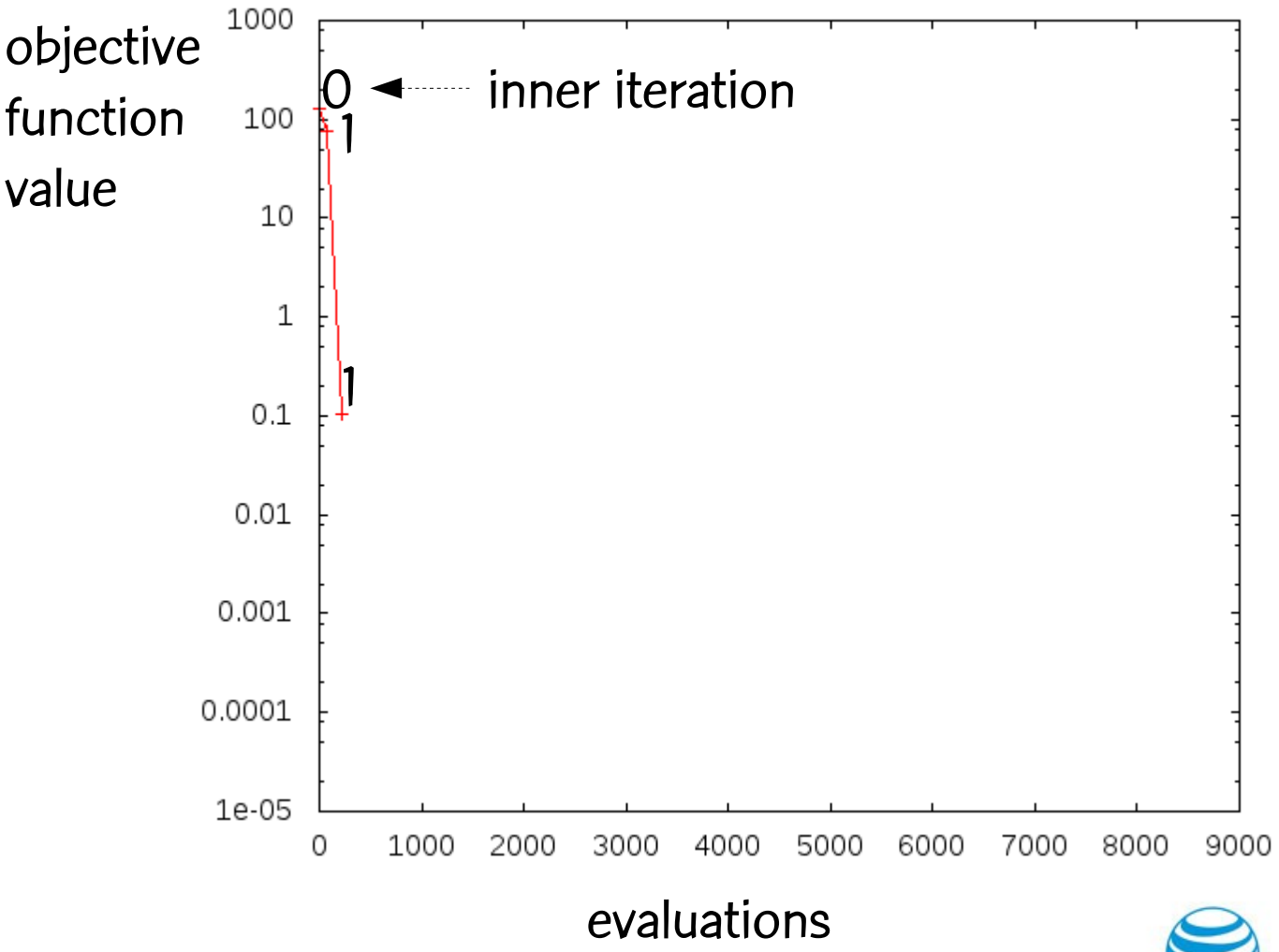
and MaxPointsToExamine = 100.

libcgrpp library: output file example (cont.)

```
h=0.500000, h_e=0.000100
o-iteration: 0
i-iteration: 1912224224
random:
time: 0.000000
evaluations: 1
best value: 127.067622
solution: 6.046783 -5.067851

h=0.500000, h_e=0.000100
h=0.500000, h_e=0.000100
o-iteration: 0
i-iteration: 1
construction:
time: 0.000000
evaluations: 80
best value: 77.292449
solution: 7.546783 -2.067851

h=0.500000, h_e=0.000100
o-iteration: 0
i-iteration: 1
local search:
time: 0.000000
evaluations: 226
best value: 0.103908
solution: 1.042637 2.824020
```



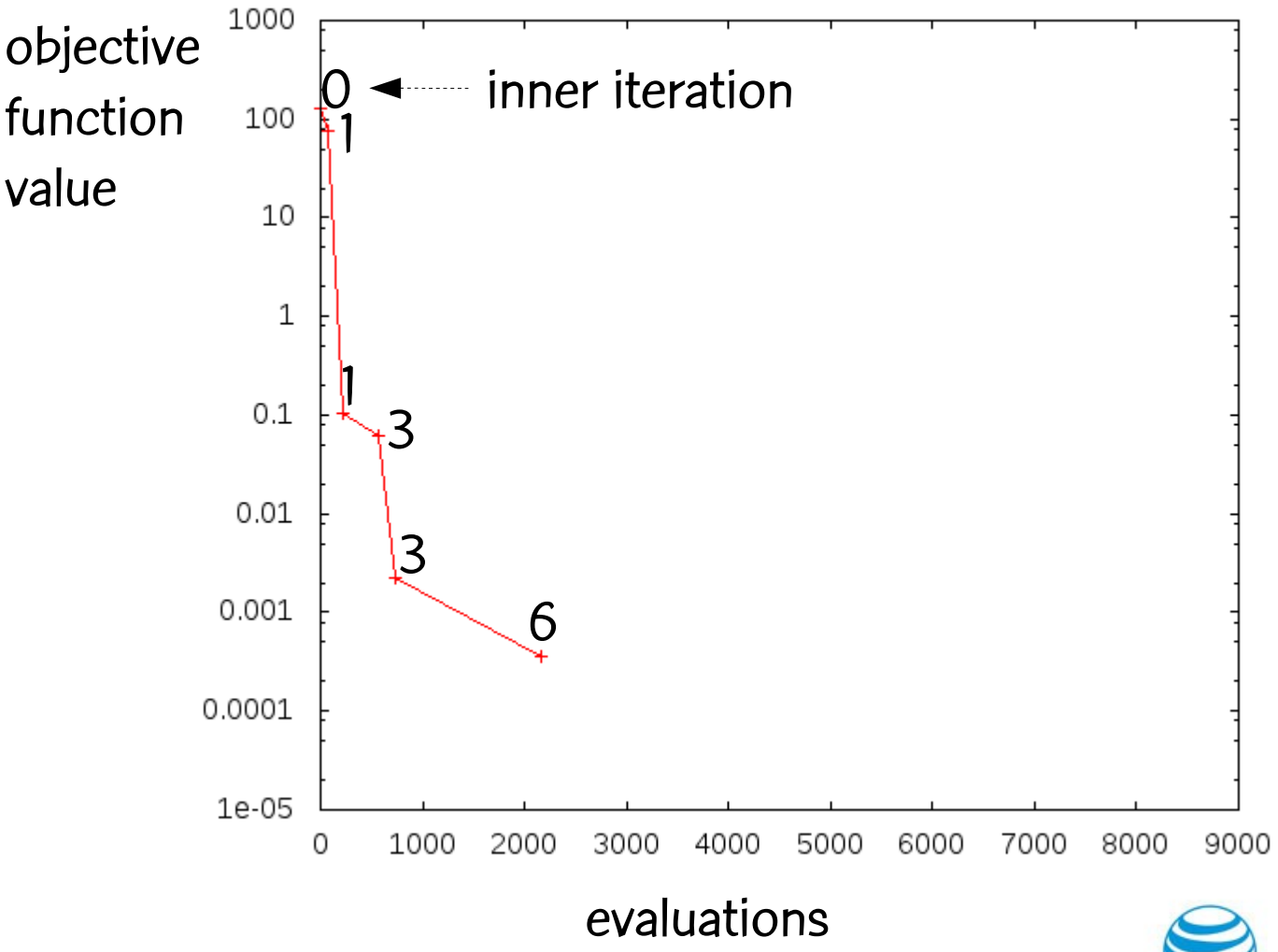
at&t
Your world. Delivered.

libcgrpp library: output file example (cont.)

```
h=0.250000, h_e=0.000100
o-iteration: 0
i-iteration: 3
construction:
time: 0.000000
evaluations: 567
best value: 0.061733
solution: 1.042637 3.074020

h=0.250000, h_e=0.000100
o-iteration: 0
i-iteration: 3
local search:
time: 0.000000
evaluations: 736
best value: 0.002220
solution: 0.965009 3.026204

h=0.062500, h_e=0.000100
o-iteration: 0
i-iteration: 6
local search:
time: 0.010000
evaluations: 2170
best value: 0.000361
solution: 1.013926 2.987302
```

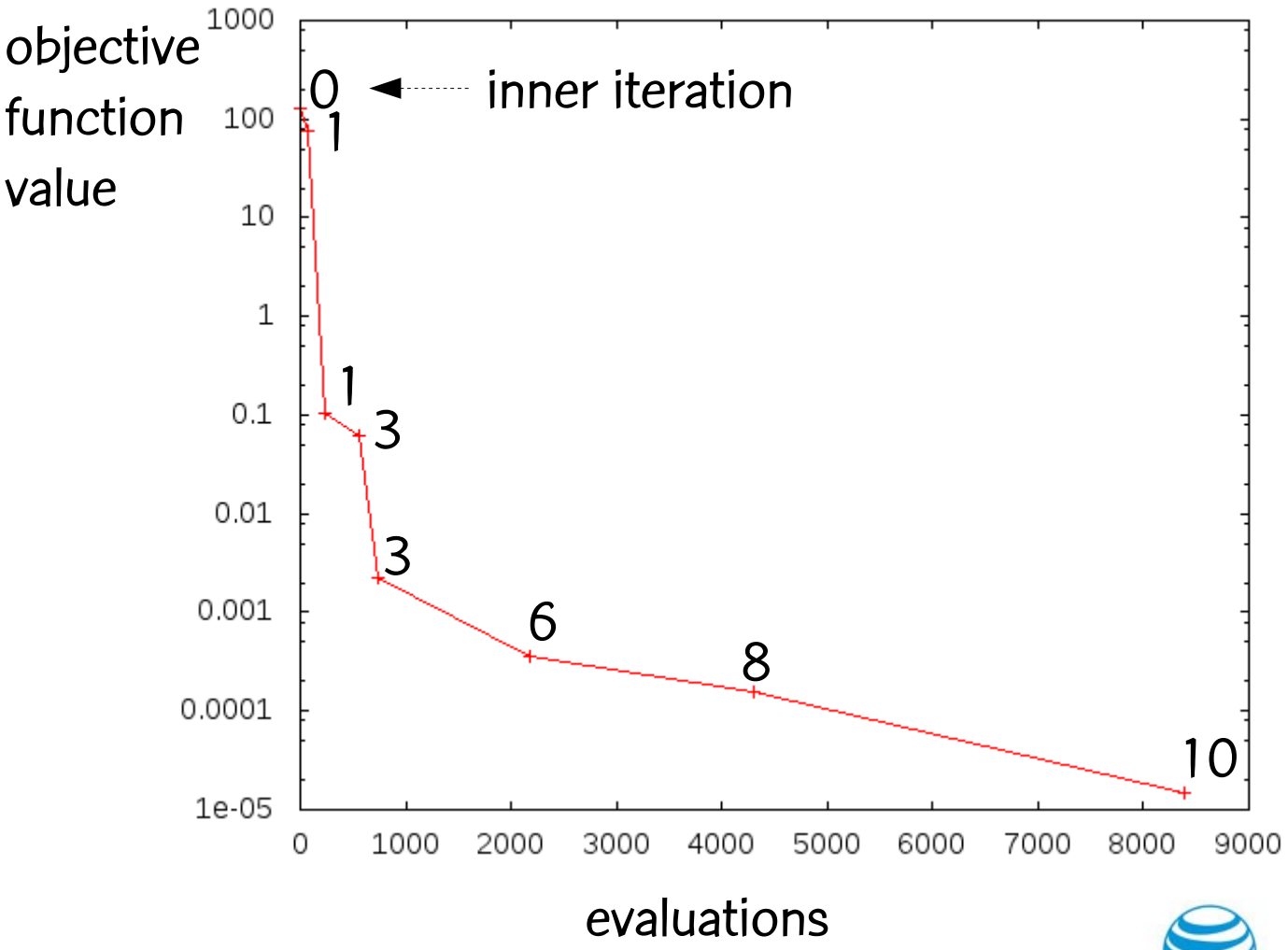


libcgrpp library: output file example (cont.)

```
h=0.031250, h_e=0.000100
o-iteration: 0
i-iteration: 8
local search:
time: 0.020000
evaluations: 4309
best value: 0.000159
solution: 0.990671 3.008178

h=0.015625, h_e=0.000100
o-iteration: 0
i-iteration: 10
local search:
time: 0.020000
evaluations: 8382
best value: 0.000015
solution: 1.001818 2.997228

time: 0.020000
dimension: 2
epsilon: 0.000100
seed: 270001
h_s: 0.500000
h_e: 0.000100
ro: 0.010000
LS option: 1
LS max points: 100.000000
output file: booth.out
```



An example

Application of of C-GRASP for this example was first described in

M.J. Hirsch, P.M. Pardalos, and M.G.C.R, "Solving systems of nonlinear equations with continuous GRASP," Nonlinear Analysis: Real World Applications, vol. 10, pp. 2000-2006, 2009.

Computational environment

Computer with a 1.66GHz Intel Core 2 processor with 1 GB of Memory

Ubuntu version 4.3.2-1ubuntu11

C language, gcc compiler version 4.3.2

Random-number generator: Mersenne Twister algorithm (Matsumoto and Nishimura, 1998)

Robot kinematics application

Robot kinematics application

- First described by Tsai and Morgan (1985).
- Given a 6-revolute manipulator (rigid-bodies, or links, connected together by joints), with the first link designated the base, and the last link designated the hand of the robot: Determine the possible positions of the hand, given that the joints are movable.
- Problem is reduced to solving a system of eight nonlinear equations in eight unknowns.
- Considered a "challenging problem" in Floudas et al. (1999).

Robot kinematics application

Find $\mathbf{x} = (x_1, x_2, \ldots, x_8)$ such that:

- $f_1(\mathbf{x}) = 4.731 \cdot 10^{-3} x_1 x_3 - 0.3578 x_2 x_3 - 0.1238 x_1 + x_7 - 1.637 \cdot 10^{-3} x_2 - 0.9338 x_4 - 0.3571 = 0$
- $f_2(\mathbf{x}) = 0.2238 x_1 x_3 + 0.7623 x_2 x_3 + 0.2638 x_1 - x_7 - 0.07745 x_2 - 0.6734 x_4 - 0.6022 = 0$
- $f_3(\mathbf{x}) = x_6 x_8 + 0.3578 x_1 + 4.731 \cdot 10^{-3} x_2 = 0$
- $f_4(\mathbf{x}) = -0.7623 x_1 + 0.2238 x_2 + 0.3461 = 0$
- $f_5(\mathbf{x}) = x_1^2 + x_2^2 - 1 = 0$
- $f_6(\mathbf{x}) = x_3^2 + x_4^2 - 1 = 0$
- $f_7(\mathbf{x}) = x_5^2 + x_6^2 - 1 = 0$
- $f_8(\mathbf{x}) = x_7^2 + x_8^2 - 1 = 0$

Robot kinematics application

We form the optimization problem:

$$\text{Find } \mathbf{x}^* = \text{argmin}\{F(\mathbf{x}) = \sum_{i=1..8} f_i^2(\mathbf{x}) \mid \mathbf{x} \in [-1,1]^8\}.$$

- Since $F(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in [-1,1]^8$, then $F(\mathbf{x}) = 0 \Leftrightarrow f_i(\mathbf{x}) = 0$ for all $i \in \{1, \ldots, 8\}$.
- Hence $\exists\, \mathbf{x}^* \in [-1,1]^8 \ni F(\mathbf{x}^*) = 0 \Rightarrow \mathbf{x}^*$ is a global minimizer of problem and $\mathbf{x}^*$ is a root of the system of equations:

  $f_1(\mathbf{x}), \ldots, f_8(\mathbf{x})$.
- There are 16 known roots to this system. Solving problem 16 times using C-GRASP with different starting solutions gives no guarantee of finding all 16 roots.

Robot kinematics application

- Suppose the k-th root (roots are denoted $\mathbf{x}_1, \ldots, \mathbf{x}_k$) has been found.
- Then C-GRASP will restart, with the modified objective function given by:
  - $F(\mathbf{x}) = \sum_{i=1..8} f_i^2(\mathbf{x}) + \beta \sum_{j=1..k} e^{-\|\mathbf{x}-\mathbf{x}(j)\|} \chi_\rho(\|\mathbf{x}-\mathbf{x}_j\|)$

  where
  - $\chi_\rho(\delta) = 1$ if $\delta \leq \rho$; 0, otherwise

  β is a large constant, and ρ is a small constant.
- This has the effect of creating an area of repulsion near solutions that have already been found by the heuristic.

Robot kinematics application: C source code

```
#include <cgrasp.h>
double main(int argc, char **argv){
        double res;
        res = cgrasp(argc,argv);
        return res;
}
```

Robot kinematics application: function module implementation in python (kinematics.py)

```
from math import *

def g(x):
        beta = 10**10
        ro = 1
        roots = []
        f = [0,0,0,0,0,0,0,0]
        f[0] = 4.731*0.001*x[0]*x[2] - 0.3578*x[1]*x[2] - 0.1238*x[0] + x[6] - 1.637*0.001*x[1] - 0.9338*x[3] - 0.3571
        f[1] = 0.2238*x[0]*x[2] + 0.7623*x[1]*x[2] + 0.2638*x[0] - x[6] - 0.07745*x[1] - 0.6734*x[3] – 0.6022
        ...
        f[7] = x[6]**2 + x[7]**2 – 1
        sum1 = sum( f[i]**2 for i in range(8) )
        sum2=0
        if len(roots)>0:
                for k in range(len(roots)):
                        dist = sqrt( sum( (x[j]-roots[k][j])**2 for j in range(len(x)) ) )
                        if dist <= ro:
                                sum2 = sum2 + exp(-dist)
        return sum1 + beta*sum2
```

Robot kinematics application: input file

- The input file:

  -hs 0.5 -he 0.001 -ro 0.01 -ls 1 -mp 1000 -of kinematics.out -sd 270001
  -md kinematics -ft g -ds 8 -ov 0 -ep 0.001 -dm -1 1

  specifies that C-GRASP will try to find a solution

$$\mathbf{x} \in S = \{\mathbf{x} = (x1, \ldots, x8) \in [-1,1]^8\}$$

  such that function g of python module (kinematics.py) that implements the robot kinematics problem will be such that

$$GAP = |g(\mathbf{x}) - 0| \leq \epsilon = 0.001,$$

  using the following parameters:

  hs = 0.5, he = 0.001, $\rho_{lo} = 0.01$, seed = 270001,

  and MaxPointsToExamine = 1000.

- As roots are found, they are included in the roots list of kinematics.py module.

Robot kinematics application

- We made ten independent runs of C-GRASP with $\rho = 1$, $\beta = 10^{10}$

- In each case, the heuristic was able to find all 16 known roots.

Robot kinematics: output file of 1st run

h=0.500000, h_e=0.010000

o-iteration: 0

i-iteration: -980710944

random:

time: 0.000000

evaluations: 1

best value: 2.593753

solution: 0.604678 -0.506785 -0.168359 -0.190572 0.569999 0.764520 0.610387 0.812561

h=0.500000, h_e=0.010000

h=0.500000, h_e=0.010000

o-iteration: 0

i-iteration: 1

construction:

time: 0.000000

evaluations: 31

best value: 0.527528

solution: 0.604678 -0.506785 -0.668359 -0.690572 0.569999 -0.735480 0.110387 0.812561

h=0.250000, h_e=0.010000

o-iteration: 0

i-iteration: 2

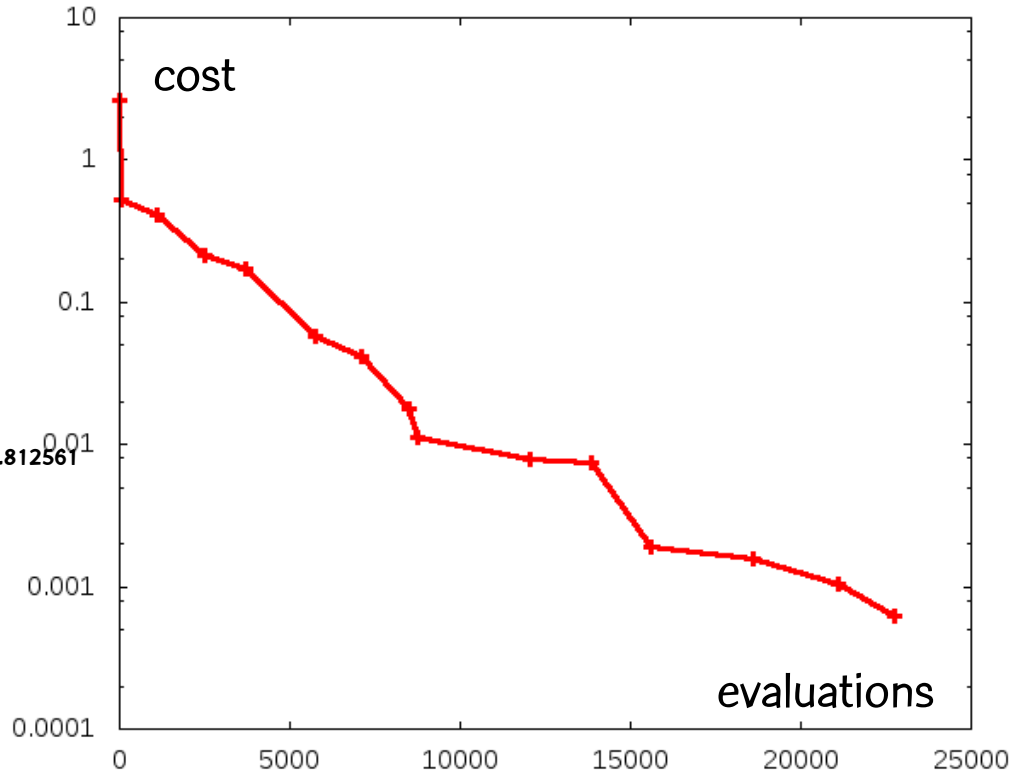
construction:

time: 0.010000

evaluations: 1092

best value: 0.412249

solution: 0.604678 -0.756785 -0.668359 -0.690572 -0.680001 -0.485480 0.360387 0.812561

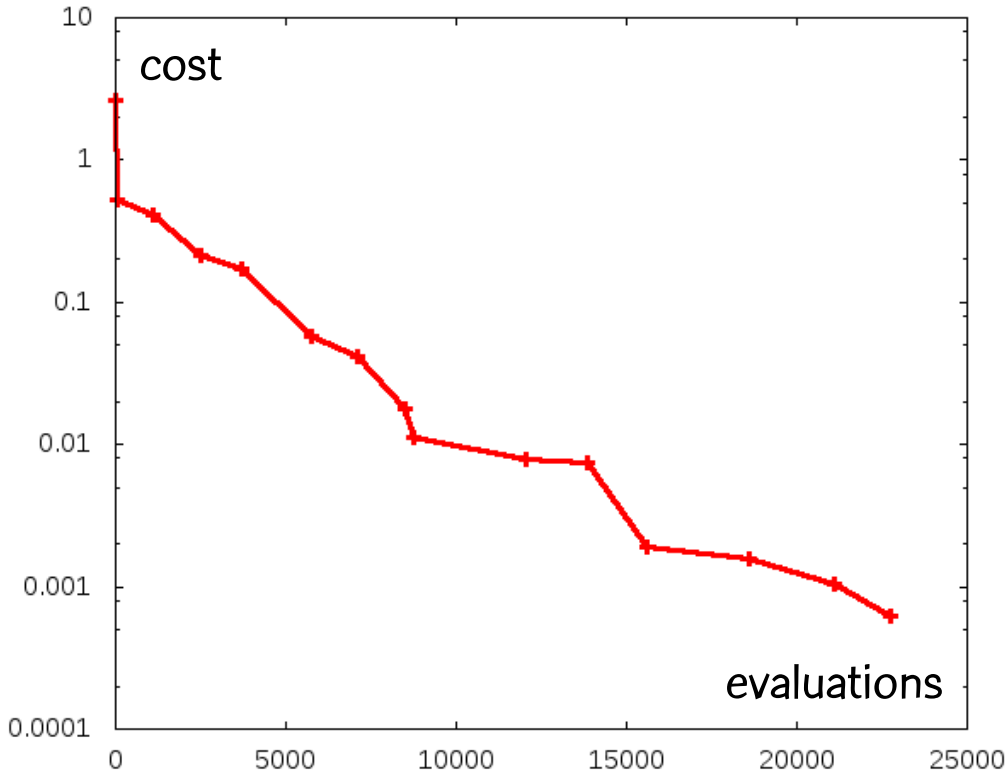


Robot kinematics: output file of 1st run (cont'd)

```
h=0.250000, h_e=0.010000
o-iteration: 0
i-iteration: 2
local search:
time: 0.020000
evaluations: 2507
best value: 0.212264
solution: 0.485155 -0.828485 -0.752117 -0.583786 -0.858302 -0.347513 0.234342 0.812561

h=0.125000, h_e=0.010000
o-iteration: 0
i-iteration: 4
construction:
time: 0.040000
evaluations: 3689
best value: 0.173348
solution: 0.485155 -0.828485 -0.752117 -0.583786 -0.983302 -0.222513 0.3

h=0.125000, h_e=0.010000
o-iteration: 0
i-iteration: 4
local search:
time: 0.060000
evaluations: 5739
best value: 0.057780
solution: 0.321092 -0.875156 -0.875766 -0.426199 -0.925468 -0.200632 0.354109 0.888619
```



Robot kinematics: output file of 1st run (cont'd)

h=0.062500, h_e=0.010000

o-iteration: 0

i-iteration: 6

construction:

time: 0.080000

evaluations: 7113

best value: 0.040999

solution: 0.383592 -0.937656 -0.875766 -0.426199 -0.987968 -0.138132 0.354109 0.888619

h=0.062500, h_e=0.010000

o-iteration: 0

i-iteration: 6

local search:

time: 0.090000

evaluations: 8498

best value: 0.017629

solution: 0.242929 -0.937656 -0.891891 -0.383381 -0.987968 -0.085540 0.3

h=0.062500, h_e=0.010000

o-iteration: 0

i-iteration: 7

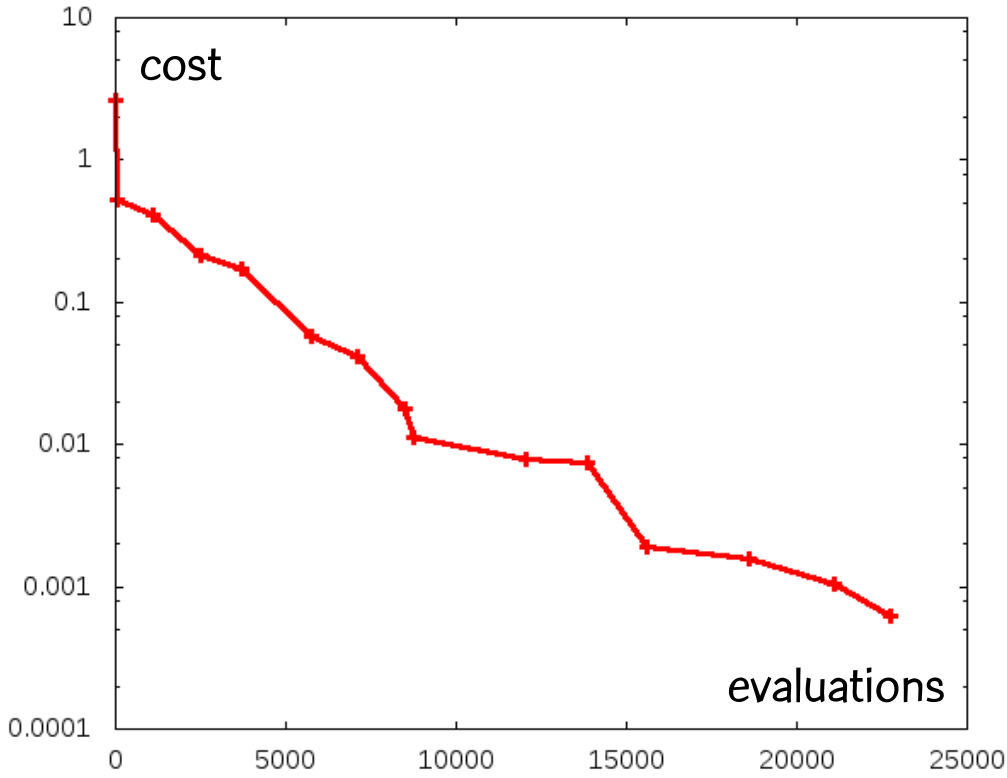
construction:

time: 0.100000

evaluations: 8747

best value: 0.011326

solution: 0.242929 -0.937656 -0.954391 -0.320881 -0.987968 -0.085540 0.387694 0.886060



Robot kinematics: output file of 1st run (cont'd)

h=0.062500, h_e=0.010000

o-iteration: 0

i-iteration: 7

local search:

time: 0.130000

evaluations: 12028

best value: 0.007878

solution: 0.208830 -0.937656 -0.938610 -0.324136 -0.987968 -0.081492 0.387416 0.908990

h=0.062500, h_e=0.010000

o-iteration: 0

i-iteration: 8

local search:

time: 0.150000

evaluations: 13831

best value: 0.007383

solution: 0.211682 -0.937656 -0.938610 -0.331521 -0.987968 -0.075448 0.3

h=0.031250, h_e=0.010000

o-iteration: 0

i-iteration: 10

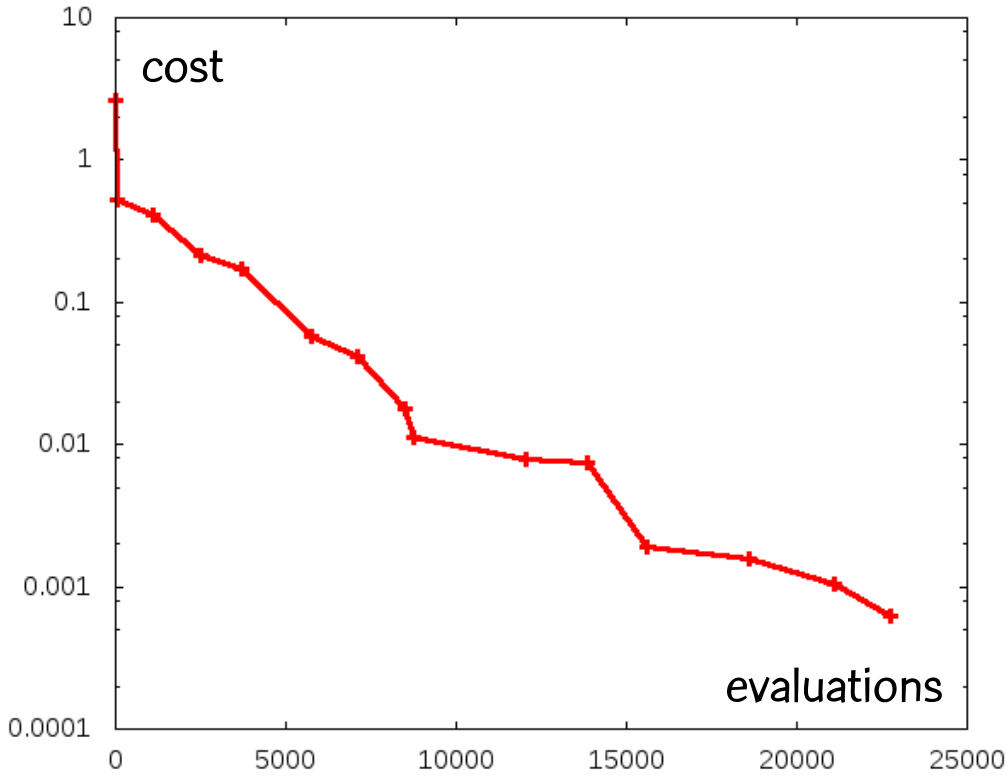
construction:

time: 0.170000

evaluations: 15588

best value: 0.001883

solution: 0.211682 -0.968906 -0.938610 -0.331521 -0.987968 -0.075448 0.408174 0.914949



Robot kinematics: output file of 1st run (cont'd)

h=0.031250, h_e=0.010000

o-iteration: 0

i-iteration: 10

local search:

time: 0.210000

evaluations: 18591

best value: 0.001588

solution: 0.175261 -0.968906 -0.939350 -0.331431 -0.987968 -0.057541 0.400847 0.921011

h=0.015625, h_e=0.010000

o-iteration: 0

i-iteration: 12

construction:

time: 0.240000

evaluations: 21116

best value: 0.001057

solution: 0.190886 -0.984531 -0.939350 -0.331431 -0.987968 -0.073166 0.4

h=0.015625, h_e=0.010000

o-iteration: 0

i-iteration: 12

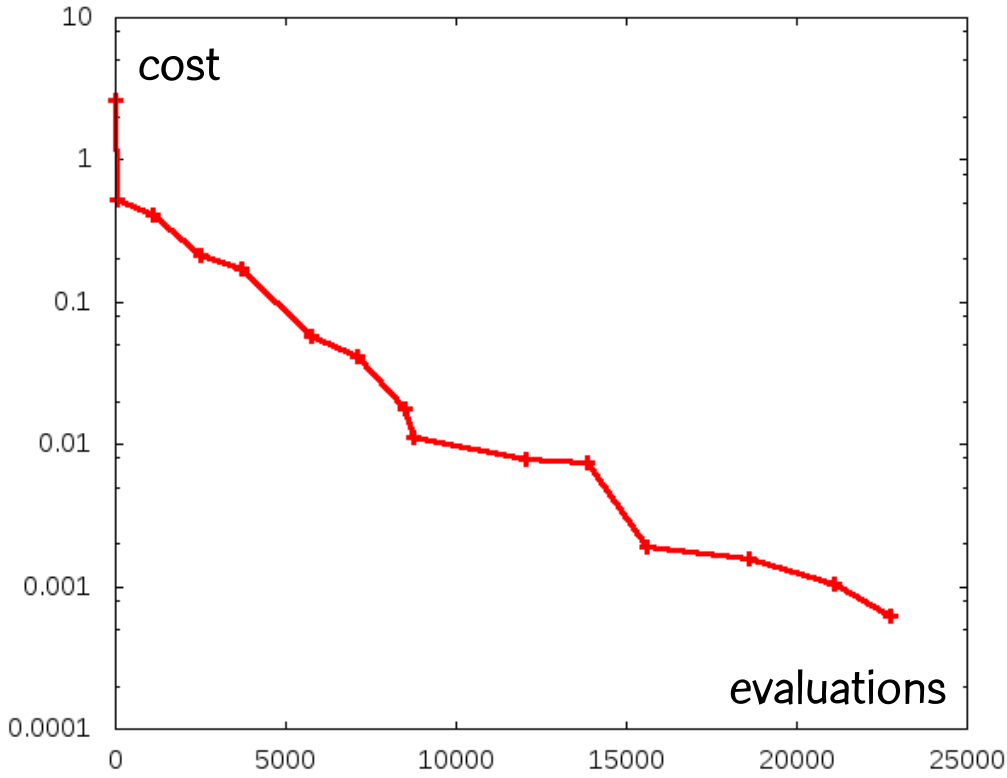
local search:

time: 0.250000

evaluations: 22734

best value: 0.000624

solution: 0.171251 -0.984531 -0.939835 -0.330984 -0.987968 -0.063526 0.411416 0.912903



Robot kinematics: output file of 1st run (cont'd)

time: 0.250000

dimension: 8

epsilon: 0.001000

seed: 270001

h_s: 0.500000

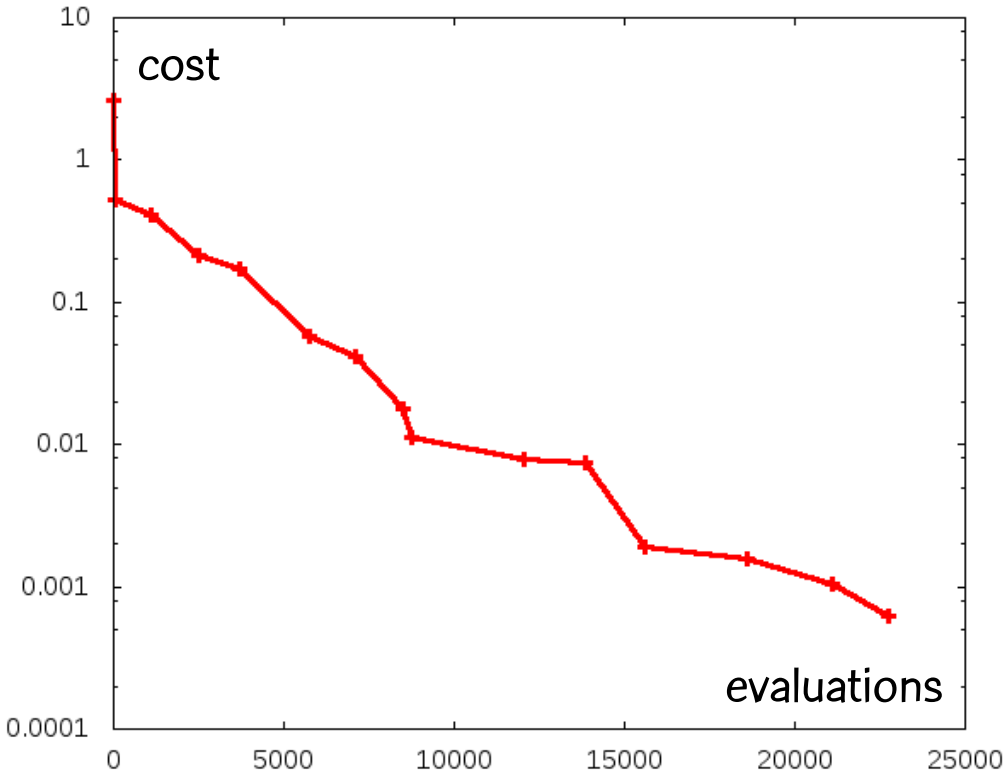
h_e: 0.010000

ro: 10.000000

LS option: 1

LS max points: 1000.00

output file: kinematics.out



Robot kinematics: including a new root in the function module in python (kinematics.py)

```
from math import *

def g(x):
        beta = 10**10
        ro = 1
        roots = [[0.171251, -0.984531, -0.939835, -0.330984, -0.987968, -0.063526, 0.411416, 0.912903]]
        f = [0,0,0,0,0,0,0,0]
        f[0] = 4.731*0.001*x[0]*x[2] - 0.3578*x[1]*x[2] - 0.1238*x[0] + x[6] - 1.637*0.001*x[1] - 0.9338*x[3] - 0.3571
        f[1] = 0.2238*x[0]*x[2] + 0.7623*x[1]*x[2] + 0.2638*x[0] - x[6] - 0.07745*x[1] - 0.6734*x[3] – 0.6022
        ...
        f[7] = x[6]**2 + x[7]**2 – 1
        sum1 = sum( f[i]**2 for i in range(8) )
        sum2=0
        if len(roots)>0:
                for k in range(len(roots)):
                        dist = sqrt( sum( (x[j]-roots[k][j])**2 for j in range(len(x)) ) )
                        if dist <= ro:
                                sum2 = sum2 + exp(-dist)
        return sum1 + beta*sum2
```

Robot kinematics: output file of 2nd run

h=0.500000, h_e=0.010000

o-iteration: 0

i-iteration: 1562925536

random:

time: 0.000000

evaluations: 1

best value: 2.593753

solution: 0.604678 -0.506785 -0.168359 -0.190572 0.569999 0.764520 0.610387 0.812561

...

h=0.062500, h_e=0.010000

o-iteration: 0

i-iteration: 7

construction:

time: 0.190000

evaluations: 10036

best value: 0.052576

solution: 0.361938 -0.938890 -0.948696 -0.340600 0.892794 -0.208991 0.407803 0.879394

...

h=0.500000, h_e=0.010000

h=0.031250, h_e=0.010000

o-iteration: 1

i-iteration: 9

local search:

time: 0.920000

evaluations: 52826

best value: 0.000661

solution: 0.666316 0.744665 -0.644458 -0.756348 -0.960886 -0.265529 -0.436140 0.888684

Robot kinematics: output file of 2nd run

time: 0.920000

dimension: 8

epsilon: 0.001000

seed: 270001

h_s: 0.500000

h_e: 0.010000

ro: 10.000000

LS option: 1

LS max points: 1000.000000

output file: kinematics1.out

Robot kinematics application

- We made ten independent runs of C-GRASP with $\rho = 1$, $\beta = 10^{10}$

- In each case, the heuristic was able to find all 16 known roots.

THE END

Thanks!