Sensor Location Problems As Test Problems Of Nonsmooth Optimization And Test Results Of A Few Nonsmooth Optimization Solvers

Fuchun Huang
School of Engineering and Science, Victoria University
Melbourne, Australia

***Abstract*—In this paper we address and advocate the sensor location problems and advocate them as test problems of nonsmooth optimization. These problems have easy-to-understand practical meaning and importance, easy to be even randomly generated, and the solutions can be displayed visually on a 2-dimensional plane. For testing some nonsmooth optimization solvers, we present a very simple sensor location problem of two sensors for four objects with the optimal solutions known by theoretical analysis. We tested several immediately ready-to-use optimization solvers on this problem and found that optimization solvers MATLAB's ga() and VicSolver's UNsolver can solve the problem, while some other optimization solvers like Excel solver, Dr Frank Vanden Berghen's CONDOR, R's optim(), and MATLAB's fminunc() cannot solve the problem.**

Keywords-sensor location problems; mathematical programming; nonsmooth optimization solver; test problems.

I. INTRODUCTION

Nonsmooth optimization is an important research field of optimization and has wide applications in real life. Although there are many test problems of nonsmooth optimization [1], some of them are too academic and lack practical backgrounds and importance, while some others are not so flexible in generating random problems of various sizes for testing purposes. Hence it is still good to have more test problems, in particular if the problems have easily understandable practical meanings and importance, and more preferably visual displays. In this paper we address sensor location problems and advocate them as a new group of test problems of nonsmooth optimization solvers. The problems are generally nonsmooth and difficult to solve. We present test results of a simple sensor location problem solved by some nonsmooth optimization solvers, which are: Excel solver developed by FrontLine Solvers [2], CONDOR developed by Frank Vanden Berghen [3], R's optim() function [4], MATLAB's fminunc() function and some other solvers [5], and VicSolver's UNsolver [6]. All these solvers have directly ready-to-use (that is, no need to compile or link by using a compiler) evaluation versions available to anyone, hence the test results reported in this paper can be repeated by anybody.

This paper is organized as following. Section II addresses the sensor location problem from different practical backgrounds, section III explains the abovementioned ready-to-use solvers and their test results and section IV summarizes the main points and results of the paper and points out some future work.

A. Sensor location problems

We use Figure 1 to help us illustrate the sensor location problems. Suppose we want to use three sensors to sense n objects in an area. The locations of the n objects are known, as shown in Figure 1. We want to determine the "best" locations of the three sensors. There could be different criteria for determining the "best" locations. One of them is to minimize the largest squared distance from an object to the nearest sensor, that is,

1) *Sensor Location Problem:* For sensing n objects at locations: (ox(i),oy(i)): i=1,2,…,n, find locations of s sensors: (sx(j),sy(j)): j=1,2,…,s, such that the largest squared distance from an object to the nearest sensor

max(min((ox(i)-sx(j))2+(oy(i)-sy(j))2,j=1,2,…s),i=1,2,…,n)

is minimized.

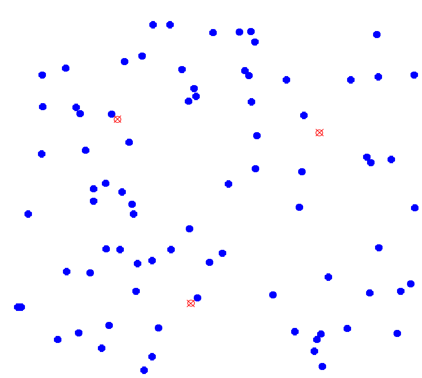

Figure 1. Locations of n objects and 3 sensors.

The same type of location problems could come from different real life backgrounds. The following are just two versions of them among many others, which might be good for teaching purposes.

2) Well Location Problem: For serving n houses at locations: (ox(i),oy(i)): i=1,2,…,n, find locations of s wells:

3) (sx(j),sy(j)): j=1,2,…,s, such that the largest squared distance from a house to the nearest well max(min((ox(i)-sx(j))2+(oy(i)-sy(j))2,j=1,2,…s),i=1,2,…,n) is minimized.

4) *Light Location Problem:* For lighting n target locations: (ox(i),oy(i)): i=1,2,…,n, find locations of s lights: (sx(j),sy(j)): j=1,2,…,s, such that the largest squared distance from a target location to the nearest light max(min((ox(i)-sx(j))2+(oy(i)-sy(j))2,j=1,2,…s),i=1,2,…,n) is minimized.

The sensor location problem may have constraints. For example, we may want to determine the best locations of three sensors on two roads only, as shown below in Figure 2.

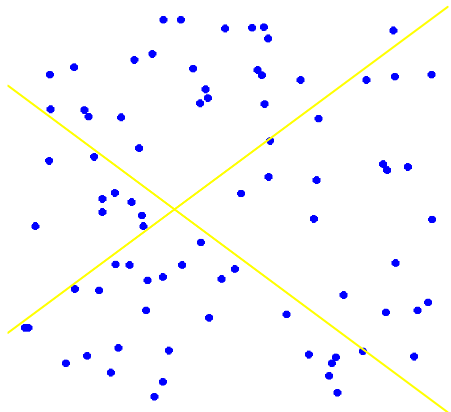

Figure 2. n objects and two roads.

So, in general, the sensor location problem is stated as in the following:

Sensor Location Problem: For sensing n objects at locations: O(i): i=1,2,…,n, find locations of s sensors: S(j): j=1,2,…,s, such that the largest squared distance from an object to the nearest sensor

max(min(distance(O(i),S(j)),j=1,2,…s),i=1,2,…,n)

is minimized.

The largest squared distance from an object to the nearest sensor, as a function of the locations of the sensors, is a continuous but nonsmooth function. The function of a very simple situation yields the nonsmooth surface as shown in Figure 3.

The sensor location problems are like the facility location problems explained in [7] hence in general very difficult to solve. As test problems of nonsmooth optimization, however, they have the merits that they have easily understandable practical backgrounds and importance, the object locations can be randomly generated and there can be a 2-dimensional visual display of the solutions. When the number of objects is small, the optimal solution can be obtained by examining different mappings of objects to different sensors. However, when the number of objects increases, the number of different mappings quickly becomes so huge that checking all different mappings becomes impossible. For example, if there are 100 objects and 5 sensors, then the number of mappings is 5100. Hence smarter algorithms of nonsmooth optimization are necessary for solving medium to large scale sensor location problems.

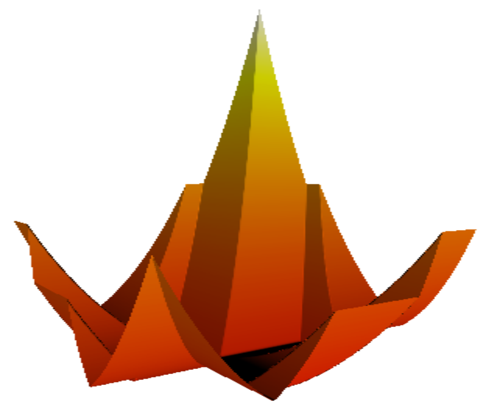

Figure 3. The 3-dimensional surface of a 2-dimensional function of a simple sensor location problem.

II. A SIMPLE SENSOR LOCATION PROBLEM AND TEST RESULTS OF SOME SOLVERS

For testing different solvers of nonsmooth optimization, we have this simple sensor location problem: the four objects are in blue at the corners of a square, that is, (0,0), (0,1), (1,1), (1,0), as shown in Figure 4 below. We want to determine the best locations of two sensors, and apparently we can see there are two optimal solutions: {(0, 0.5), (1, 0.5)} and {(0.5, 0), (0.5, 1)}, as shown in red in Figure 4 below.

Figure 4. One optimal solution of the two sensor location problem

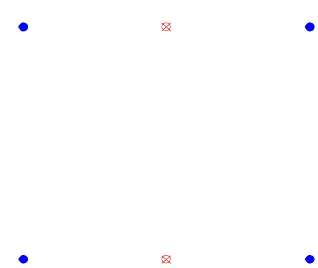

Figure 5. Another optimal solution of the two sensor location problem

There are many published computer programs for nonsmooth optimization, and an incomplete list can be found at http://napsu.karmitsa.fi/nsosoftware/. Here in this paper we focus on only some ready-to-use programs, not those programs in source codes or in a binary library which needs a compiler or linker to compile or link the program to a user's main program. The ready-to-use programs tested in this paper are: Excel solver developed by FrontLine Solvers [2], CONDOR developed by Dr Frank Vanden Berghen [3], R's optim() function [4], MATLAB's fminunc() function and some other solvers [5], and VicSolver's UNsolver developed by Dr Fuchun Huang [6]. In testing these solvers, we use the initial sensor locations {(0.5, 0.5), (0.5, 0.5)}, which is not a local minima as the objective function would decrease if one sensor moves to the left (or up) a little bit and the other moves to the right (or down) a little bit.

A. *Frontline Solvers*

Frontline's Excel solver has three methods or algorithms: GRG nonlinear for solving smooth nonlinear optimization problems; Simplex LP for solving linear problems; and Evolutionary for nonsmooth problems, as shown below in the solvers application interface wizard.

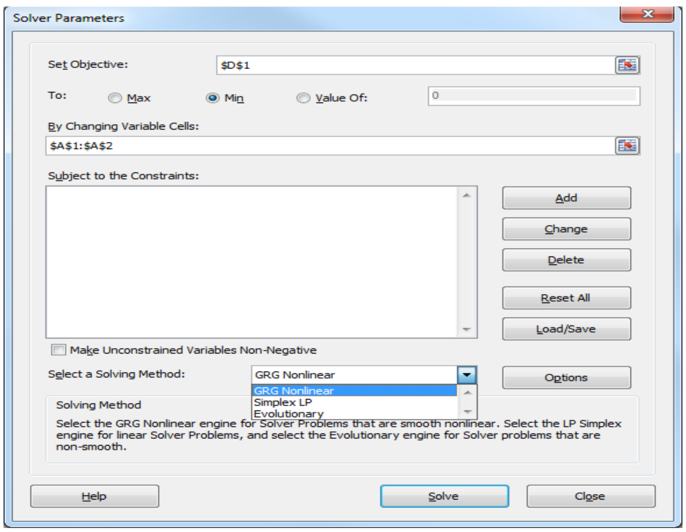

Figure 6. Excel solver's three methods.

For solving the sensor location problem stated at the beginning of this section, we put initial values (x1=0.5, y1=0.5, x2=0.5, y2=0.5) of the locations of the two sensors in cells B1:B4, and the largest squared distance from an object to the nearest sensor is computed by the following formula in cell E1:

E1=MAX(MIN(B1^2+B2^2, B3^2+B4^2), MIN((B1-1)^2+(B2-0)^2, (B3-1)^2+(B4-0)^2), MIN((B1-0)^2+(B2-1)^2, (B3-0)^2+(B4-1)^2), MIN((B1-1)^2+(B2-1)^2, (B3-1)^2+(B4-1)^2)), as shown in Figure 7.

When solve the problem by 'Evolutionary' method as shown in Figure 6, it ends up with the message that 'Solver cannot improve the current solution'.

When solve the problem by 'GRG nonlinear' method, it also ends up with the message that 'Solver cannot improve the current solution'.

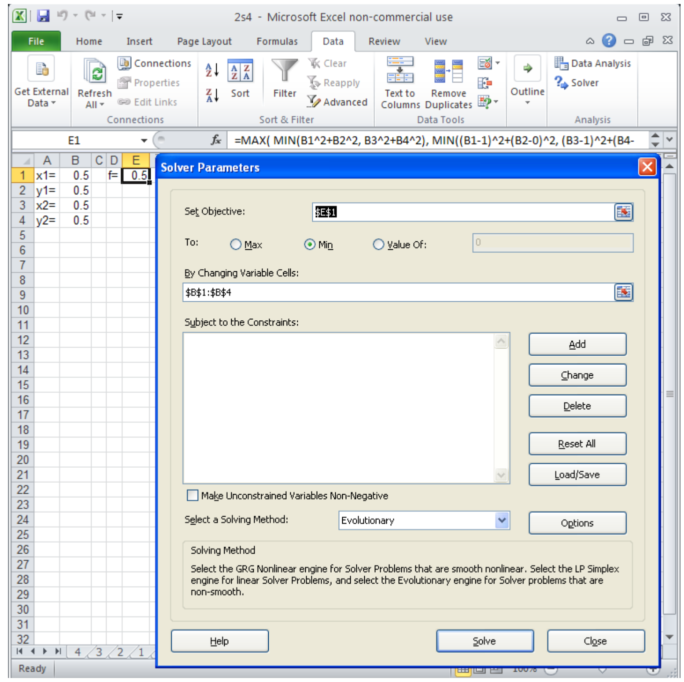

Figure 7. Excel Solver's Evolutionary method to solve the sensor location problem of two sensors.

B. *CONDOR's result*

CONDOR [3] is a constrained, non-linear, derivative-free parallel optimizer for continuous, high computing load, noisy objective functions developed by Dr Frank Vanden Berghen. CONDOR is also available via NEOS Server [8].

The following file is the AMPL [9] code for solving the sensor location problem at the beginning of the section:

```
var x{i in 1..4};

minimize f:
max(
min(x[1]^2+x[2]^2, x[3]^2+x[4]^2),
min((x[1]-1.0)^2+(x[2]-0.0)^2, (x[3]-1.0)^2+(x[4]-
0.0)^2),
min((x[1]-0.0)^2+(x[2]-1.0)^2, (x[3]-0.0)^2+(x[4]-
1.0)^2),
min((x[1]-1.0)^2+(x[2]-1.0)^2, (x[3]-1.0)^2+(x[4]-
1.0)^2)
);

let x[1] := 0.5;
let x[2] := 0.5;
let x[3] := 0.5;
let x[4] := 0.5;

display x;
display f;
```

When the file is submitted to NEOS server to be solved by CONDOR, the following 'optimal' solution is returned:

```
Best Value Objective=2.575139e-01 (nfe=324)
rho=1.000000e-04; fo=2.575139e-01; NF=325
rho=1.000000e-04; fo=2.575139e-01; NF=325
CONDOR 1.06 finished successfully.
325 ( 312) function evaluations
Final obj. funct. Value=0.25751389
_svar [*] :=
1  0.971055
2  0.50182
3  0.0866737
4  0.500002
```

We see the optimal solution and the minimum value

"Final obj. funct. Value=0.25751389"

are not so close to the truly optimal solution and minimum value 0.25.

C. *R's optim() function*

R [4] is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. The version of R we used is 2.14.1. R's optim() function has five methods for multi-dimensional optimization: "Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN".

The following are R codes of the sensor location problem with optimal value (from theoretical analysis but tested by the code):

```
> # number of sensors:
> ns=2
>
> sxy=rep(0,2*ns);
>
> fr <- function(sxy)
+ {
+ sx=sxy[1:ns];
+ sy=sxy[(ns+1):(ns+ns)];
+
+ dmax=0.0;
+ for(i in 1:no){
+     j=1;    dmin=(sx[j]-ox[i])^2+(sy[j]-oy[i])^2;
+ for(j in 2:ns){d=(sx[j]-ox[i])^2+(sy[j]-oy[i])^2;
if(d<dmin)dmin=d;};
+ if(dmax<dmin)dmax=dmin;
+ }
+ return(dmax);
+ };
>
> # optimal
> fr(c(0,1,0.5,0.5))
[1] 0.25
```

The following are R codes of solving the problem by Nelder-Mead method:

```
>   sxy=c(0.5,0.5,0.5,0.5)

>   optres=optim(sxy, fr, NULL, method = "Nelder-
Mead", control=list(maxit=999999));

control=list(maxit=999999));

> cat(optres$value, fill=T);

0.5

> cat(optres$par[1:(ns+ns)], fill=T);

0.5 0.5 0.5 0.5
```

We see the solver cannot improve the initial values. The other methods, "BFGS", "CG", "L-BFGS-B", "SANN" all have the same results, that is, none of them can improve the initial values.

D. *MATLAB's fminunc() function and other solvers*

MATLAB [5] is a numerical computing environment and fourth-generation programming language developed by MathWorks. The version of MATLAB we used is 7.11.1. The following shows the MATLAB m-file of the two-sensor four-object problem stated at the beginning of the section, and running results of the optimization function fminunc() with default option settings:

```
function f=s2o4(x)

f=0.0;

f=max(f,min((x(1)-0.0)^2+(x(2)-0.0)^2,(x(3)-
0.0)^2+(x(4)-0.0)^2));

f=max(f,min((x(1)-1.0)^2+(x(2)-0.0)^2,(x(3)-
1.0)^2+(x(4)-0.0)^2));

f=max(f,min((x(1)-0.0)^2+(x(2)-1.0)^2,(x(3)-
0.0)^2+(x(4)-1.0)^2));

f=max(f,min((x(1)-1.0)^2+(x(2)-1.0)^2,(x(3)-
1.0)^2+(x(4)-1.0)^2));

end

>> s2o4([0.5;0.5;0.5;0.5])

ans =

    0.5000

>>

  x = fminunc(@s2o4,[0.5;0.5;0.5;0.5])

Warning: Gradient must be provided for trust-region
algorithm;

  using line-search algorithm instead.

> In fminunc at 347

Initial point is a local minimum.

Optimization completed because the size of the
gradient at the initial point

is less than the default value of the function
tolerance.

<stopping criteria details>

x =

    0.5000

    0.5000

    0.5000

    0.5000
```

We see the solver cannot improve the initial values, and wrongly claims the initial point is a local minimum. Other option settings of the solver yield the same results.

MATLAB has several other solvers and their results are presented and explained below and in Figure 8.

```
>> fminsearch(@s2o4,[0.5 0.5 0.5 0.5])
ans =
    0.5000    0.5000    0.5000    0.5000
Optimization running.
Objective function value: 0.5
Optimization terminated:
 the current x satisfies the termination criteria
using OPTIONS.TolX of 1.000000e-004

 and F(X) satisfies the convergence criteria using
OPTIONS.TolFun of 1.000000e-004

>> simulannealbnd(@s2o4,[0.5 0.5 0.5 0.5])

Optimization terminated: change in best function
value less than options.TolFun.
ans =
    0.5000    0.5000    0.5000    0.5000
>>
```

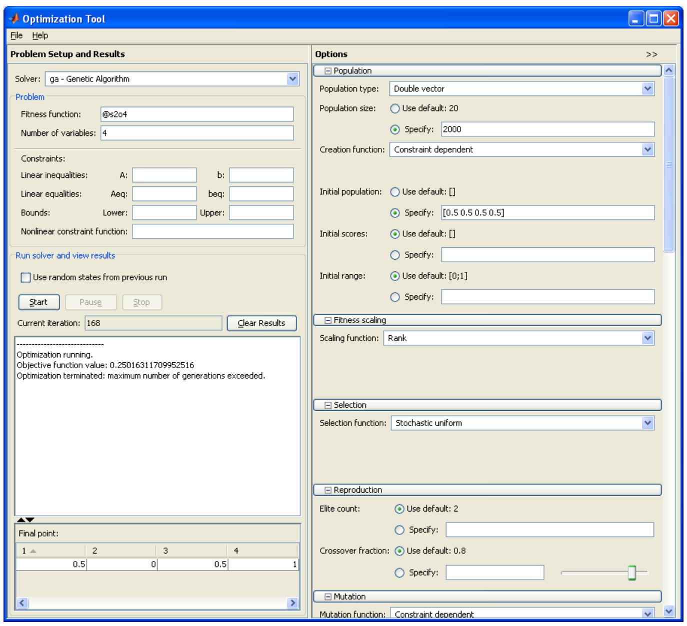

Figure 8. MATLAB's optimization tool wizard.

We see MATLAB's fminsearch() and simulannealbnd() cannot solve the problem, and the solver ga() using genetic algorithm returns the objective function value 0.250163 by using population size 2000 and iteration number 168. As the population size and iteration number increases, the returned objective values become closer and closer to the true minimum, which is 0.25.

E. Unsolver

VicSolver's UNsolver [6] is an unconstrained derivative-free nonsmooth optimization solver developed by Dr Fuchun Huang of Victoria University, Australia. UNsolver uses FEFAR [10] as an application programming user's interface for users to specify an unconstrained smooth or nonsmooth optimization problem. The following is the LEFAR [10] code for the sensor location problem at the beginning of the section.

```
!! optimal 2 sensors for 4 points

function frf(if123,fmin,b)
real                 :: fmin
real, dimension(4)   :: b
integer              :: if123

if(if123==1)
      b(1)=0.5
      b(2)=0.5
      b(3)=0.5
      b(4)=0.5
end if

fmin=0.0
if(if123<=2)
fmin=max(fmin,min(b(1)^2+b(2)^2, b(3)^2+b(4)^2));
fmin=max(fmin,min((b(1)-1)^2+(b(2)-0)^2, (b(3)-
1)^2+(b(4)-0)^2));
fmin=max(fmin,min((b(1)-0)^2+(b(2)-1)^2, (b(3)-
0)^2+(b(4)-1)^2));
fmin=max(fmin,min((b(1)-1)^2+(b(2)-1)^2, (b(3)-
1)^2+(b(4)-1)^2));
end if

if(if123==3)
   print,fmin
   print,b(1)
   print,b(2)
   print,b(3)
   print,b(4)
end if

end function
```

The following is the running result of UNsolver:

```
C:\ISMtalk>UNsolver.exe
 Input the file name of the function:
2s4a.far
  fmin=  0.50000000000000000       at
0.50000000000000000       0.50000000000000000
0.50000000000000000       0.50000000000000000

. . . . . .

 fmin=  0.25000000000000000       at
0.0000000000000000        1.0000000000000000
0.50000000000000000       0.50000000000000000
 fmin=  0.25000000000000000       at
0.0000000000000000        1.0000000000000000
0.50000000000000000       0.50000000000000000

. . . . . .

 fmin=  0.25000000000000000       at
0.0000000000000000        1.0000000000000000
0.50000000000000000       0.50000000000000000
 fmin=  0.25000000000000000       at
0.0000000000000000        1.0000000000000000
0.50000000000000000       0.50000000000000000
```

We see UNsolver finds one of the two optimal solutions. An evaluation version and some other test results of UNsolver are available from http://sites.google.com/site/VicSolver.

III. Summary Remarks And Future Scope

We present the sensor location problems and advocate them as test problems of nonsmooth optimization.

These problems have easy-to-understand practical meaning and importance, easy to be even randomly generated, and the solutions can be displayed visually on a 2-dimensional plane. For testing some ready-to-use nonsmooth optimization solvers, we present a very simple sensor location problem of two sensors for four objects with the optimal solutions known by theoretical analysis. We tested several optimization solvers on this problem and found that optimization solvers MATLAB's ga() and VicSolver's UNsolver can solve the problem, while some other optimization solvers like Excel solver, Dr Frank Vanden Berghen's CONDOR, R's optim(), and MATLAB's fminunc() cannot solve the problem. In the near future some medium to large scale "standard" sensor location problems will be generated and put online for researchers testing nonsmooth optimization solvers.

ACKNOWLEDGMENT

The author thanks Professor Yosihiko Ogata, Professor Satoshi Ito, Professor and Director Tomoyuki Higuchi and The Institute of Statistical Mathematics, Tokyo, Japan, for supporting his research visit to the institute from the 5th of January to the 28th of February in 2012.

REFERENCES

[1] L. Luksan and J. Vlcek, "Test Problems for Nonsmooth Unconstrained and Linearly Constrained Optimization," Technical Report, No. 798, Institute of Computer Science, Academy of Sciences of the Czech Republic, 2000.

[2] Frontline Solvers, http://www.solver.com/sdkplatform2.htm, retrived on March 29, 2012.

[3] F. V. Berghen, "CONDOR, a parallel, direct, constrained optimizer for high-computing-load, black box objective functions", Proceeedings of the third *MIT* conference on Computational Fluid and Solid Mechanics, Elsevier, June 14-17, 2005.

[4] R Development Core Team, "R: A language and environment for statistical computing", R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0, URL http://www.R-project.org/, 2011.

[5] Wikipedia, "MATLAB", http://en.wikipedia.org/wiki/MATLAB, retrived on March 29, 2012.

[6] F. Huang, "UNsolver: a new solver of unconstrained nonsmooth optimization problems", http://sites.google.com/site/VicSolver.

[7] F. V. Berghen, J. Cardinal and S. Langerman, "Min-max-min Geometric Facility Location Problems", 22nd European Workshop on Computational Geometry, Delphi, March 27-29, 2006.

[8] J. Czyzyk, M.P. Mesnier, and J.J. More, "The NEOS Server", IEEE Computational Science and Enginnering, Vol. 5, pages 68-75, 1998.

[9] R. Fourer, D.M. Gay, and B.W. Kernighan, AMPL: A Modeling Language for Mathematical Programming. Scientific Press, San Francisco, CA, 1993.

[10] F. Huang, "A New Application Programming Interface And A Fortran-like Modeling Language For Evaluating Functions and Specifying Optimization Problems At Runtime", International Journal of Advanced Computer Science and Applications, vol. 3, issue 4, pp. 1-5, April 2012.

AUTHORS PROFILE

Dr Fuchun Huang is a Senior Lecturer in the School of Engineering and Science at Victoria University, Melbourne, Australia. He was awarded a PhD degree by The Graduate University of Advanced Studies, Tokyo, Japan, and has published papers on computational statistics, in particular Monte Carlo methods, pseudo-likelihood and generalized pseudo-likelihood methods, and developed solver software for solving smooth and nonsmooth optimization problem. He is a member of The Japan Statistical Society.