ACO-C: A C-language Implementation of Ant Colony Optimization Algorithms

by

Samdani Saurabh Arun

September 2, 2005

Department of Civil Engineering,
Indian Institute of Technology Guwahati,
Guwahati -781039, Assam, India.
E-mail: samdani@iitg.ernet.in

ACO-C: A C-language Implementation of Ant Colony Optimization Algorithms

Samdani Saurabh Arun
Department of Civil Engineering,
Indian Institute of Technology Guwahati,
Guwahati -781039, Assam, India.
E-mail: samdani@iitg.ernet.in

1 Introduction

ACO-C[1] is a C-language implementation of various Ant Colony Optimization (ACO) algorithms for a general n-variable minimization (optimization) problem. This report is included as a concise introduction to the ACO-C distribution. It is presented with the assumptions that the reader has a general understanding of Ant Colony Optimization algorithms and a good working knowledge of the C programming language. The report begins with a discussion about the assumptions made regarding the problem structure. Outline of the files included in the ACO-C distribution, and the routines they contain is described next. The system requirements are given in the outline. In the next section , discussion of guideline steps that could be followed to implement one's own application in ACO-C is taken up. Based on these steps , two example applications are shown. Some of the lacunae of this package are also discussed. Report concludes with the final comments and future plans.

For an introduction to Ant Colony Optimization reader is referred to [7, 6, 5]. Ant System, Elitist Ant System are the earliest members of the class of ACO algorithms and were introduced in [7]. Ant Colony System was introduced next in [6]. Rank-based version of Ant System was presented in [2]. The ACO algorithms implemented are in this package are Ant System, Elitist Ant System, Rank-based version of Ant System and Ant Colony System.

2 Assumptions for the problem

ACO-C was developed for following assumptions regarding the problem:

- Problem has been formulated as a *non-linear programming problem* as follows [3] :
  - Vector of decision variables $\mathcal{X} = (\mathcal{X}_1, \mathcal{X}_2, \cdots, \mathcal{X}_i, \cdots \mathcal{X}_n)$
  - Objective function
    minimize $f(\mathcal{X})$
  - Subject to Constraints
    $g_j(\mathcal{X}) \leq 0 \quad j = 1, 2, \cdots, J$
    $h_k(\mathcal{X}) = 0 \quad k = 1, 2, \cdots, K$
  - Variable bounds
    $\mathcal{X}_i^l \leq \mathcal{X}_i \leq \mathcal{X}_i^u \quad i = 1, 2, \cdots, n$
- Objective function $f(\mathcal{X})$ is to be minimized and $f(\mathcal{X}) > 0$. If $f(\mathcal{X})$ is negative you should add a suitable positive number to it , so that $f(\mathcal{X}) > 0 \ \ \forall \ \ \mathcal{X}$.

[1] **Disclaimer:** ACO-C is distributed under the terms described in the file gpl.txt.
The software package is freely available subject to the GNU General Public Licence. This means that ACO-C has no warranty implied or given, and that the authors assume no liability for damage resulting from its use or misuse.

- If your objective function $f(\mathcal{X})$ is to be maximized then you can return

$$objfn = \frac{1}{1 + f(\mathcal{X})} \tag{1}$$

  as the objective function. So, based on suitable transformations you should transform your objective function for minimization with positive values.

- Variables are discrete or can be discretized between a lower and upper bound using suitable increment. For non–uniform increment map your variable to a dummy variable with lower limit 1 and upper limit as (maximum number of values) for that variable and set increment to 1

- Constraints will be handled using penalty function method.

- NO heuristic value is used i.e η_{ij} is not defined. ACO-C was developed with precisely these kind of problems in mind , where it is difficult to associate heuristic value with a solution component. Simple analytical functions to be minimized belong to this category.

3 Files distributed with ACO-C

ACO-C is distributed as a gzipped tar file. To install the code, first obtain the file ACO-C.tar.gz. Put the file in a directory. Extract the contents of the file by typing
`$ tar -xvzf ACO-C.tar.gz` Latest verion of this package can be obtained by sending an email to the author. Some functions of this code were not written by the author. Some utility functions, statistical routines and random number generators were taken from various sources like [9], [4].

3.1 Code

The package was developed in C under Linux, using the GNU gcc compiler and extensively tested in this environment.

System specifications:

- gcc version was (GCC) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)

- GNUPLOT version used was: gnuplot 4.0 patchlevel 0. You can run code without gnuplot 4.0 also but you wont be able to have a look at the graphs drawn automatically by `./ants.sh`

- If you dont have gnuplot 4.0 patchlevel 0, then you can use shell scripts given in `shellscripts-for-gnuplot-3.7.tar` which *may* work with gnuplot 3.7.

- LaTeX version:– TeX (Web2C 7.4.5) 3.14159 kpathsea version 3.4.5

To compile it under Linux just type 'make' and the executable 'aco' is produced.
NO guarantee that the code will run under a different environment

3.2 C source files

The main control routines, main: `aco.c`

Parameters of the algorithm `par.h parameters.h`

Declaration and definition of variables and data structures: `dec.h dec.c func.h` Procedures to implement the ants behaviour: `ants.c ants.h trail.c`

Input / output / statistics routines: `input.c output.c init.c stats.c rand.c sort.c`

Procedures for objective function management: `analysis.c`

Procedures specific to the evaluation of objective function of problem: `obj.c`

Procedures for objective function testing and checking values returned: `test.cpp`

Local search procedures: `ls.c ls.h`

Time measurement: `timer.c timer.h`

3.3 Makefiles

`makefile:` Main makefile , compiles the code to give `aco` as executable. `aco` can be executed as
`$ ./aco <inputfile>`

`obj.make:` Makefile to compile objective function in `obj.c` with variable values in `test.c`. Produces executable `test` with command
`$ make -f obj.make`
`test` can be executed as
`$ ./test`

3.4 Shell scripts

`ants.sh:` Shell script for running the program. Following syntax to be followed to use it
`./ants.sh <nants> <iterations> <runs> <rho> <input-file>`

`bkp.sh:` Script to create a backup of the code with date in the archive name.

`cln.sh:` Script to clean output files for a particular inputfile. Following syntax to be followed to use it
`./cln.sh <filename whose output files are to be removed>`

`test.sh:` Script to test the objective function from variable values in `test.c`. Calls `make -f obj.make`.

`gplot.sh:` This script is called with the inputfile name. This script in turn calls all graph plotting scripts with inputfilename as argument. Finally the graphs are processed using `gnuplot <gnuplot-command-file>.gnu`.

`constructiongraph.sh:` Script to output gnuplot code to plot the solution construction graph.

`path.sh:` Script to process arrows in gnuplot commands for construction graph.

`varconv.sh:` Script to output gnuplot code to plot the convergence of variable values versus iteration number for all runs.

`eval.sh:` Script to output gnuplot code to plot the evaluations required to find the objective function versus run number.

`iter.sh:` Script to output gnuplot code to plot the iterations required to find the objective function versus run number.

`objconv.sh:` Script to output gnuplot code to plot the convergence of objective function versus iteration number for all runs.

`runstats.sh:` Script to output gnuplot code to plot the best objective function found in various runs.

`stddevplot.sh:` Script to output gnuplot code to plot the various run time statistics with iteration number.

`time.sh:` Script to output gnuplot code to plot the CPU time required to find the objective function versus run number.

4 Steps to solve a problem with ACO.

1. Identify the number of variables in your problem. Then go to file `parameters.h` and put the number of variables in front of `#define nov`
   e.g `#define nov 10`

2. For each variable , following data should be written in an input file

   - variablename (without white space)

- lower limit
- upper limit
- increment

e.g `var1 5 18 0.5`

3. If uniform increment for variable cannot be kept for e.g say domain of variable y is $\{2.5, 3.5, 5.0, 10.2\}$ then this variable should be defined as
   $dummy - y$
   lowerlimit = 1
   upperlimit = 4
   increment = 1
   i.e. like this in inputfile
   `dummy-y 1 4 1`
   then in objective function you can assign value to y based on $dummy - y$

4. If your problem is constrained put
   `#define constrained 1`
   otherwise put
   `#define constrained 0`
   in `parameters.h`
   All constraints should be normalized and should be of type
   $g_j(\mathcal{X}) \leq 0$.

5. Code your objective function in `obj.c`, at proper place depending on your problem being constrained or not.
   Function `objective_function(float * varvalues)`
   or
   `objective_function(float * varvalues, float *antpenalty)`
   should return `(float) obj-fn`,

6. For constrained problems ,sum of cumulative constraint violation should be calculated and assigned to `*antpenalty`

7. For testing your objective function , go to test.c, and put appropriate values for
   `float values[nov]`
   e.g `float values[nov] = 1 , 2, 2,1,6 ,3,2,1,0,5;` for `nov 10.`

8. Compile your code by typing
   `$ test.sh` which runs
   `$ make -f obj.make .`

9. If your objective function is working fine then type
   `$ make`
   at the prompt ($) and check if compilation is ok. Executable aco will be produced.

10. If compilation is successful run your code like this
    `$ ./ants.sh <nants> <num-iterations> <number-runs> <evap> <input-file>`
    Or define these parameters in `par.h` and run like this
    `$ ./aco < input-file>`

5 Examples for solving problems with ACO-C

5.1 Example 1

Suppose we wish to solve the constrained problem `ex1` with ACO-C.

$$\text{minimize} \quad f(x_1, x_2) = 1 + (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \tag{2}$$

subject to

$$0 \leq x_1 \leq 6.0 \tag{3}$$

$$0 \leq x_2 \leq 6.0 \tag{4}$$

$$g_1(x_1, x_2) = (x_1 - 5)^2 + x_2^2 - 26 \geq 0 \tag{5}$$

$$g_2(x_1, x_2) = 4x_1 + x_2 - 20 \geq 0 \tag{6}$$

Step-wise we can use ACO-C as follows :

1. The problem at hand has two variables x_1 and x_2. So go to file `parameters.h` and put the number of variables as 2 i.e.,
   `#define nov 2`.

2. Now variable bounds are to be written in a file say `exmpl1.inp`. Assume increment is kept to 0.01 (i.e the precision with which variable value is calculated). Contents of `exmpl1.inp` will be
   ```
   $ cat exmpl1.inp
   x1 0 6.0 0.01
   x2 0 6.0 0.01
   $
   ```

3. Problem is constrained so put
   `#define constrained 1`
   in `parameters.h`. Normalize the constraints like this

$$g_1(x_1, x_2) = 1 - \frac{(x_1 - 5)^2 + x_2^2}{26} \leq 0 \tag{7}$$

$$g_2(x_1, x_2) = 1 - \frac{4x_1 + x_2}{20} \leq 0 \tag{8}$$

   So, $g_1(x_1, x_2) > 0$ or $g_2(x_1, x_2) > 0$ implies violation of constraint.

4. Go to `obj.c`. Now just above
   `#if(constrained==1)`
   write
   ```
   #define var10 0
   #define ex1 1
   ```
   since our problem is named as `ex1`.
   Set
   ```
   float objfn, x1,x2 ;
   x1 = varvalues[0];
   x2 = varvalues[1];
   ```
   Now write the code for calculating objective function as
   ```
   # if(ex1)
   objfn = 1 +( (x1*x1 +x2 -11) *(x1*x1 +x2 -11) ) + ( (x1 +x2*x2 - 7) * (x1 + x2*x2 -7));
   g1 = 1.0 - (( pow((x1 -5.0),2) + pow((x2),2) ) / 26.0 ) ;
   ```

```
g2 = 1.0 - ( ( 4 * x1 + x2 ) / 20.0) ;


float sumconstr = 0.0, penaltyconst = 1;
if(g1 >=0)
{
sumconstr += g1;
}
if(g2 >=0)
{
sumconstr += g1;
}
*antpenalty = penaltyconst * sumconstr;

return objfn * ( 1 + *antpenalty);
# endif
```

5. For testing objective function , go to test.cpp, and put appropriate values for
   `float values[nov]`
   i.e `float values[nov] = {1 , 3};`
   and set
   `cost = objective_function( values, &penalty);`
   `printf(" ofn = %15.15f , sumconstr = %6.8f ", cost,penalty);`

6. We are now ready to test our objective function , so type
   `$ ./test.sh`

7. Objective function is working fine and gives
   ```
   g++ -g -c test.cpp
   g++ -g -c obj.c
   g++ -g -o objfn test.o obj.o -lm
   ofn = 63.538463592529297 , sumconstr = 0.07692308
   ```

8. We are now ready to test the problem with ACO-C by typing
   `$ make`
   at the prompt ($) and check if compilation is ok. Executable `aco` will be produced.

9. Compilation is successful ,run your code like this
   e.g. `$ ./ants.sh 75 50 5 0.1 exmpl1.inp`
   Check the output file as given.

5.2 Example 2

Suppose we wish to solve the problem `ex2` with ACO-C.

$$\text{minimize} \quad f(x_1, x_2) = 1 + (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \tag{9}$$

subject to

$$0 \leq x_1 \leq 6.0 \tag{10}$$

$$0 \leq x_2 \leq 6.0 \tag{11}$$

Step-wise we can use ACO-C as follows :

1. The problem at hand has two variables x_1 and x_2. So go to file `parameters.h` and put the number of variables as 2 i.e.,
   `#define nov 2`.

2. Now variable bounds are to be written in a file say `exmpl2.inp`. Assume increment is kept to 0.01 (i.e the precision with which variable value is calculated). Contents of `exmpl2.inp` will be
   ```
   $ cat exmpl2.inp
   x1 0 6.0 0.01
   x2 0 6.0 0.01
   $
   ```

3. Problem is unconstrained so put
   `#define constrained 0`
   in `parameters.h`.

4. Go to `obj.c`. Now just above
   `#if(constrained==0)`
   write
   ```
   #define f1 0
   #define f2 0
   #define f3 0
   #define ex2 1
   ```
   since our problem is named as `ex2`.
   Set `x1 = varvalues[0];`
   `x2 = varvalues[1];`
   Now write the code for calculating objective function as
   `objfn = 1 +( (x1*x1 +x2 -11) *(x1*x1 +x2 -11) ) + ( (x1 +x2*x2 - 7) * (x1 + x2*x2 -7));`

5. For testing objective function , go to test.c, and put appropriate values for
   `float values[nov]`
   i.e `float values[nov] = {3 , 3};`
   and set
   ```
   cost = objective_function( values);
   printf(" ofn = % 15.15f ", cost);
   ```

6. We are now ready to test our objective function , so type
   `$ ./test.sh`

7. Objective function is working fine and gives
   `ofn = 27.000000000000000`

8. We are now ready to test the problem with ACO-C by typing
   `$ make`
   at the prompt ($) and check if compilation is ok. Executable `aco` will be produced.

9. Compilation is successful ,run your code like this
   e.g. `$ ./ants.sh 25 20 3 0.1 exmpl2.inp`
   Check the output file as given.

5.3 Output files

Many outpufiles are produced for an input file. Check output files for previous example as
`$ ls exmpl2.inp.*`
You will see a long list , something like this: `exmpl2.inp.var.tex`
`exmpl2.inp.best.tex`

```
exmpl2.inp.constructiongraph.eps
exmpl2.inp.constructiongraph.gnu
exmpl2.inp.dev.eps
exmpl2.inp.dev.gnu
exmpl2.inp.eval.eps
exmpl2.inp.eval.gnu
exmpl2.inp.graph.out
exmpl2.inp.iter.eps
exmpl2.inp.iter.gnu
exmpl2.inp.lsvar.out
exmpl2.inp.obj.eps
exmpl2.inp.obj.gnu
exmpl2.inp.path.out
exmpl2.inp.plot.out
exmpl2.inp.run.eps
exmpl2.inp.run.gnu
exmpl2.inp.run.out
exmpl2.inp.stat.out
exmpl2.inp.table.aux
exmpl2.inp.table.dvi
exmpl2.inp.table.log
exmpl2.inp.table.pdf
exmpl2.inp.table.tex
exmpl2.inp.time.eps
exmpl2.inp.time.gnu
exmpl2.inp.var.eps
exmpl2.inp.var.gnu
```

Each file with extension `.gnu` contains gnuplot commands for plotting graph with corresponding `.eps` file as output file. You can check the details of the graph in the caption and title in `exmpl2.inp.table.pdf` . LaTeX commands for compiling all the graphs and tables are in `exmpl2.inp.table.tex` . By running
`latex exmpl2.inp.table.tex`
and then
`dvipdf exmpl2.inp.table.dvi`
you will obtain
`exmpl2.inp.table.pdf` . Please check the order of commands followed in `ants.sh` .

6 Lacunae

There are many lacunae in this code. Declarations have not been separated as all of them have been declared in `dec.h` Memory for all the structures is not dynamically allocated. I have used the shell script `ants.sh` to `# define` these parameters and then compile.

7 Final Comments

ACO-C is not intended to be definitive in terms of its efficiency or the grace of its implementation. The author is interested in the comments, criticisms, and bug reports from ACO-C users, so that the code can be refined for easier and accurate use in subsequent versions. Please e-mail your comments to `samdani@iitg.ernet.in`. In future, implementation of other ACO algorithms like $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MMAS}$ as suggested in [10]) and implmentation in Hyper-Cube framework for ACO [1] will be considered. Another interesting task would be to implement ACO-C for mixed variable problems, as outlined in [8]. The present implementation considers, trail parameter τ_{ij} to be associated with $\mathcal{X}_{ij}$ i.e j th variable value at i th variable. Effect

of associating trail parameters $\mathcal{T}_{ij}$ with connections between solution components could be studied and implemented.

Acknowledgments

Many friends encouraged me to write this user-manual, I would like to thank them. I thank Suresh Kataria for providing help in honing the output from the program. Various people have put their source codes for free on the net, I thank all such people.

References

[1] C. Blum and Marco Dorigo. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 34(2):1161–1172, 2004.

[2] B. Bullnheimer, R. Hartl, and C. Strauss. A new rank based version of the ant system — a computational study, 1997.

[3] Kalyanmoy Deb. *Optimization for engineering design: Algorithms and examples.* New Delhi: Prentice-Hall., 1995.

[4] Kalyanmoy Deb. *GA-in-C:A GA implementation in C using binary and real coded variables.* `http://www.iitk.ac.in/kangal/soft.htm, Accessed:June,2004.`, 2001.

[5] Marco Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In David Corne, Marco Dorigo, and F. Glover, editors, *New Ideas in Optimization*. McGraw-Hill, New York, NY, USA, 1999.

[6] Marco Dorigo and Luca M. Gambardella. Ant colony system: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions On Evolutionary Computation*, 1(1):53–66, April 1997.

[7] Marco Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 26:29–41, 1996.

[8] Krzysztof Socha. ACO for Continuous and Mixed-Variable Optimization. In Marco Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004*, volume 3172 of *LNCS*, pages 25–36. Springer-Verlag, Berlin, Germany, 2004.

[9] Thomas Stuetzle. *ACOTSP, Version 1.0. Ant Colony Optimization code for Travelling Salesman Problem (TSP) in C.* `http://www.aco-metaheuristic.org/aco-code,Accessed:August 2004.`, 2004.

[10] Thomas Stützle and H. H. Hoos. $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.