A Brief Tutorial for the GALAXY Method

Qi Wang[1]
Dragan A. Savić[2]
Zoran Kapelan[2]

[1]School of Civil and Transportation Engineering, Guangdong University of Technology
[2]Centre for Water Systems, University of Exeter

✉ wangqiguangzhou@163.com
20th January, 2017

Contents

1. Introduction

GALAXY stands for Genetically Adaptive Leaping Algorithm for approXimation and diversitY (GALAXY), which is a hybrid optimiser for dealing with the multi-objective design of Water Distribution Systems (WDSs). The GALAXY method follows the generational framework of many state-of-the-art multi-objective evolutionary algorithms (MOEAs), such as the non-dominated sorting genetic algorithm II (NSGA-II) [*Deb et al.*, 2002], and implements six search operators simultaneously to improve the effectiveness and efficiency. Several important strategies are also included to maintain a better balance between the global and local search.

The GALAXY method includes four key steps, which are *Initialisation*, *Selection*, *Generation* and *Replacement*. Firstly, an initial population is randomly generated. The solutions are then evaluated using the objective functions. Secondly, the parents are selected from the population and offspring are produced. Then, the fitness of the offspring is evaluated using the objective functions. If some of them dominate the members of the current population, the dominated individuals are replaced by these offspring. This procedure is implemented repeatedly until a certain stopping criterion is satisfied.

The multi-objective design of WDSs is a typical combinatorial optimisation problem, which is NP-hard and computationally intensive. Many design problems only consider integer variables, such as the pipe sizing and/or pump scheduling. Therefore, the search space, in terms of both objective and decision spaces, is vast, discrete, multi-modal and normally constrained.

To fit the needs for solving discrete, combinatorial optimisation problems, GALAXY adopts the integer coding scheme and eliminates majority of individual parameters of those search operators initially developed for solving continuous, real-valued optimisation problems. In particular, the turbulence factor (TF), differential evolution (DE), simulated binary crossover for integers (SBXI), uniform mutation (UM), Gaussian mutation (GM) and dither creeping (DC) are employed and tailored in the GALAXY method. These operators are used in a genetically-adaptive way, in which an equal number of individuals (i.e., 1/6 of population size) are generated using each operator concurrently; and afterwards, the number of individuals that is allowed to produce by each search operator is determined based on the reproductive rate (ratio of the children alive to the children created) in the previous generation [*Vrugt and Robinson*, 2007].

When a search operator fails to contribute even a single individual in the current population, the one-child policy is applied in the next round to release maximum reproductive chances. Specifically, it borrows one opportunity from the topmost operator in the previous round. If two of six operators fail, each of them borrows one opportunity from the two topmost operators, and so on. Therefore, the most successful

operator is always favored by getting the highest number of offspring in the reproduction process, and no operator is completely discarded even though it exhibits the worst performance.

In the *Replacement* process, which actually steers the population towards the Pareto-optimal front, a hybrid replacement strategy is implemented. Generally speaking, the Pareto-dominance [*Deb et al.*, 2002] and ε-dominance concepts [*Laumanns et al.*, 2002] are combined to screen the candidates from the solutions with the topmost rank. When the number of solutions in the top rank is less than the population size, the Pareto-dominance based replacement is executed, with a secondary ranking procedure according to the crowding distance. However, if the number of individuals with the top rank exceeds the population size, the ε-replacement is carried out instead, in which the non-dominated solutions in the first front are sorted once again based on the ε-dominance concept. As a result, the ε-non-dominated solutions are copied into the next population. If there are still free spaces left, some ε-dominated solutions are also selected which have smaller distances to the ideal global optima.

Besides the aforementioned hybrid replacement strategy, GALAXY also implements some other ones, including the genetically adaptive strategy (which has been described briefly), the global information sharing [*Vrugt and Robinson*, 2007] and the duplicates handling strategies. They all play an important role in improving the capability of the GALAXY method and maintaining a well converged and distributed population. Main features of GALAXY include:

- ✓ Tailored operators for solving multi-objective WDS design problems;
- ✓ No parameters to be fine-tuned (requiring only *popSize*[1] and *NFEs*[2]);
- ✓ Structured problem formulation, easily extended to other cases;
- ✓ Robust and easy-to-use.

In summary, GALAXY is a new and efficient hybrid MOEA for solving the discrete, multi-objective WDS design problems. The practical value of GALAXY lies in the fact that it alleviates the parameterisation problem of MOEAs to great extent. Therefore, it is envisaged that this hybrid optimiser will benefit the researchers and practitioners in the water community.

[1] *popSize*: size of population.
[2] *NFEs*: number of function evaluations

2. Pseudo-Code of GALAXY

GALAXY Method

Inputs: population size (*N*), number of function evaluations (*NFEs*)

Outputs: Pareto approximation set (*AS*), Pareto approximation front (*AF*)

Initialisation:

Generate the initial population of *N* individuals randomly in the specified variable domains.

Initialise the quotas* of six search operators equally such that ($\sum_{J=1}^{6} N_J = N$).

Evaluation:

Evaluate the objective function values of the initial population (by using hydraulic simulations).

Rank the population using the non-dominated sorting procedure [*Deb et al.*, 2002].

Update the current number of function evaluations (i.e., set $I = N$).

While $I <= NFEs$

Selection:

Choose all the members in the current population for the ***Generation*** step.

Generation:

For J = 1 to 6

Produce *N* candidate solutions from the current population using operator *J*.

Select N_J offspring randomly from candidate solutions and save them to the offspring set.

End

Check whether the solutions in the offspring set are within the specified variable ranges.

Evaluation:

Evaluate the objective function values of the solutions in the offspring set.

Replacement:

Combine the current population and the offspring set as an intermediate population of size *2N*.

Implement the duplicates handling strategy.

Rank the intermediate population using the non-dominated sorting procedure.

If the number of individuals in the top rank <= *N*

Implement the normal replacement via the crowded-comparison operator [*Deb et al.*, 2002].

Else

Implement the ε-replacement strategy.

End

Form the next population of size *N*.

Update the quotas* of search operators according to their contributions to the next population.

Update the current number of function evaluations (i.e., set $I = I + N$).

End

Set the current population as *AS*.

Set the objective function values of the current population as *AF*.

Note: *A quota of a search operator refers to the number of offspring it is allowed to produce for the next generation.

3. How to Use GALAXY

To apply the GALAXY method to your own design problems, or other combinatorial optimisation problems with discrete decision variables, it is suggested to follow the three steps specified below.

- Step 1: define the optimisation problem in a structured way;
- Step 2: determine the parameters of GALAXY, which are *N* and *NFEs*;
- Step 3: specify the options of GALAXY or use the default settings.

In Step 1, the problem to be solved is formulated to a standard objective function with two inputs and two outputs in a separate M-file. The inputs include a vector of decision variables and a scalar of *NFEs*. The outputs contain a vector of the objective values and a scalar of the amount of constraint violation (see Figure 1). Then, this problem formulation should be registered (linked) in the M-file named *objectiveFunction*. This function has the same inputs and outputs as shown in Figure 1, and includes an additional input of structure array, called *problemDef*, containing the information of problem definition. The *problemDef* is the only output of another M-file named *problemArchive*, in which the meta-data of the problem should be registered as well. The meta-data specify the name and index of the problem, the number of objectives and decision variables, whether the problem is constrained or not, the lower and upper bounds of each decision variable, and the directory of the best-known or true Pareto-optimal front, if available. By following this structured problem formulation, the GALAXY method is able to identify the problem definition and solve it accordingly. If the efficiency of execution is of particular interest, please refer to *Section 5: Efficiency Issues* for a resolution. Note that the procedures used to open and close the hydraulic solver can be put either inside or outside the problem formulation. The former style is adopted for the example shown in *Section 4*.

```
[objs,constrs]=ProblemName(vars,NFEs)
```

Figure 1. Prototype of Problem Formulation

In Step 2, the user should determine the appropriate settings for *N* and *NFEs*, which are the only required parameters for the GALAXY method. There is no universally accepted rule for selecting these two parameters. However, generally speaking, a larger population size (i.e., *N*) and computational budget (i.e., *NFEs*) should be applied to the problems with more decision variables and broader ranges, which increase the search space exponentially.

In Step 3, the user needs to specify some options when working with the GALAXY method. A list of such options and the corresponding effects are summarised in Table 1.

Table 1. Options of the GALAXY method

Options	Settings	Effect
InitialisationMethod	{'LHS' | 'EXT' | 'INT'}	choose the initialisation method among Latin hypercube sampling, uniformly distributed random integers, or loading the initial population from an external file
SearchOperators	{'TF','DE','SBXI','UM','GM','DC'}	decide the combination of search operators; note that the names of the selected operators should be included in the curly brackets
RangeOfMutation	(0,1)	default is 0.7; this option turns out to have minor influence on GALAXY's performance
UseMex	{'Yes' | 'No'}	choose 'Yes' if an MEX-file has been compiled to enjoy a substantial speedup; otherwise use the generic functions by setting 'No'
RecordPerformance	{'Yes' | 'No'}	whether to record GALAXY's performance during optimisation; in case that a reference front is not available, set this option to 'No'
PerformanceIndicators	{'GD','HV','EI','EP'}	choose the performance indicator(s) to monitor; suggest to keep all of them
PlotDuringOptimisation	{'Yes' | 'No'}	choose 'Yes' if you want to see the real-time progress of GALAXY in a scatter plot; choose 'No' to improve the efficiency
ScatterSize	area of each marker (unit: points^2)	set the marker size, default is 10
ScatterColor	short or long name of color specification	set the marker color, default is 'blue'

Before calling the main function *GALAXY* within the command window of Matlab, please make sure that all the items in the following checklist have been ticked.

☑ The EPANET input file (*.inp) has been placed in the current folder where GALAXY exists.
☑ The objective function developed (*.m) has been placed in the current folder.
☑ The dynamic link library (DLL) of EPANET Programmer's Toolkit (*.dll) and the associated header file (*.h) have been placed in the current folder. If the MEX-file is used, copy the compiled program (*.mexw32 or *.mexw64) to the current folder.

4. Examples

Here, a very simple example, known as the bi-objective design of Two-loop network, is demonstrated to facilitate the applications of GALAXY to other user-specified problems. The objectives are to minimise the total cost of pipes and to maximise the network resilience, which is a surrogate indicator of network reliability [*Prasad and Park*, 2004]. Please refer to *Wang et al.* [2015] for details of the problem formulation.

```
function [objs,constrs]=TLN2obj_Mat(vars,NFEs)
% This is an example of the bi-objective design of Two-loop network.

% Input Vars:
%   (1) vars: decision variables (a 1-by-8 vector);
%   (2) NFEs: number of function evaluations (scalar);
% Output Vars:
%   (1) objs: objective values (a 1-by-2 vector);
%   (2) constrs: corresponding constraint violation (scalar);
------------------------------------------------------------------------------
global numOfEvaluation
if isempty(numOfEvaluation)
    numOfEvaluation=1;
else
    numOfEvaluation=numOfEvaluation+1;
end
% open the hydraulic solver
if numOfEvaluation==1
    if ~libisloaded('epanet2')
        loadlibrary('epanet2.dll','epanet2.h');
    end
    calllib('epanet2','ENopen','TLN.inp','reportFile.rpt','');
    calllib('epanet2','ENopenH');
    calllib('epanet2','ENresetreport');
    calllib('epanet2','ENsetreport','MESSAGES NO');
    disp('The INP file has been opened.');
end
% convert the decision variables to integers
vars=floor(vars+1);
------------------------------------------------------------------------------
% close the hydraulic solver
if numOfEvaluation==NFEs
    calllib('epanet2','ENcloseH');
    calllib('epanet2','ENclose');
    disp('The INP file has been closed.');
    numOfEvaluation=0;
    if libisloaded('epanet2')
        unloadlibrary('epanet2');
    end
end
```

Figure 2. Sample Sections of the Problem Formulation of Two-loop Network

The sample sections shown above include two major subroutines besides the header part of the objective function. The first subroutine opens the hydraulic solver just once during optimisation, and the second subroutine closes the solver eventually. Note that the variables are added by 1 to comply with the 1-based index rule in Matlab.

When the problem has been coded and tested, register it within the M-files named *objectiveFunction* and *problemArchive*, respectively (as shown in Figures 3-4).

```
elseif strcmpi(options.UseMex,'No') % using the generic functions
    if problemIndex==2
        for i=1:popSize
            [objs(i,:),constrs(i)]=TLN2obj_Mat(vars(i,:),NFEs);
        end
    end
end
```

Figure 3. Register the Problem within *objectiveFunction.m*

```
elseif problemIndex==2 || strcmpi(problemName,'TLN')
    % -----Two-Loop Network-----
    problemDef.Index=2;
    problemDef.Name='TLN';
    problemDef.NumOfObj=2;
    problemDef.NumOfVar=8;
    problemDef.IsConstrained=1;
    problemDef.VarLowerBound=0*ones(1,8);
    problemDef.VarUpperBound=13*ones(1,8);
    load([pwd '/benchmark/best_known_PF/truePF_TLN'],'ParetoFront');
    problemDef.PF=ParetoFront;
```

Figure 4. Register the Problem within *problemArchive.m*

Now, it is ready to run GALAXY by typing the following command after the prompt.

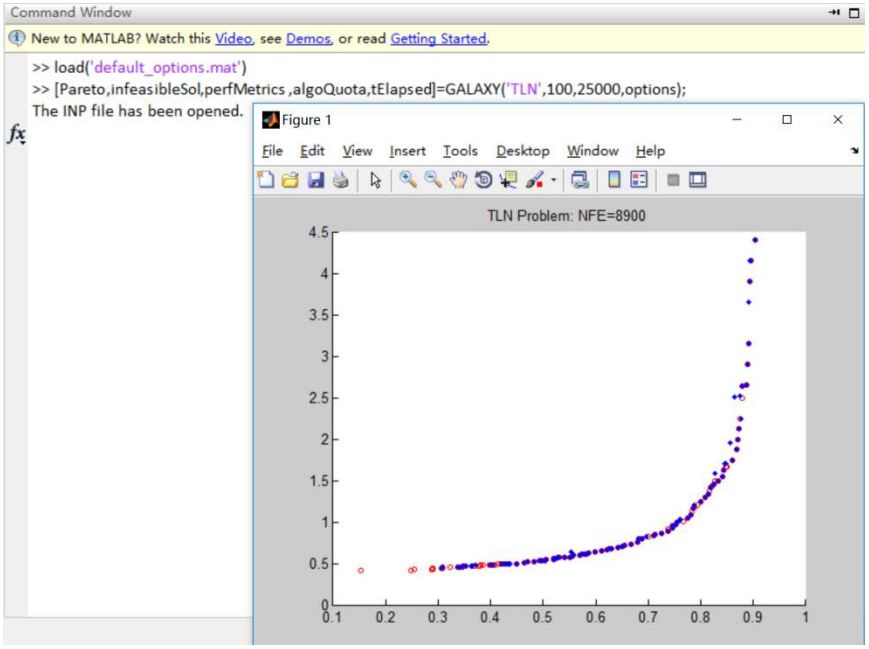

Figure 5. Screenshot of the Execution of GALAXY

5. Efficiency Issues

If the problem to be solved is very complicated, for instance, many decision variables involved, or time-consuming hydraulic simulations required, it is probably worth considering speeding up the execution of optimisation via the Matlab executable (MEX) file (or function). MEX files enable the invocation of subroutines created in *C*, *C++*, or *Fortran* from the MATLAB command line as built-in functions. These programs, called binary MEX-files, are dynamically linked with the MATLAB interpreter. Therefore, one can take advantages of both languages, making the time-consuming routines be implemented outside Matlab and constructing other routines quickly within Matlab.

Based on the preliminary analyses, it is found that the hydraulic simulation accounts for the majority of the time elapsed for solving the multi-objective WDS design problems. Therefore, the computational overhead when dealing with such kind of problems (especially for larger problems) can be effectively reduced by calling the MEX-files, which encapsulate objective functions involving hydraulic simulations written in *C* language.

The sample code for compiling such a MEX-file for solving the Two-loop Network problem is presented in Figure 6, which includes the MEX-file interface as well as part of the problem formulation (minimise total cost vs. maximise network resilience).

To compile the MEX-files on your computer, please take the following steps assuming that the 64-bit Windows OS is used:

1. Copy the library file [*epanet2.lib*] to the directory like [*C:\Program Files\Microsoft Visual Studio 10.0\VC\lib*];
2. Call [*mex –setup*] in Matlab to configure the complier;
3. Choose [*Microsoft Visual C++ 2010 Express*] or similar complier rather than the default one [*LCC*] (you may need to download the free compiler from the official site of Microsoft);
4. Call [*mex -v TLN2obj.c -L"C:\Program Files\Microsoft Visual Studio 10.0\VC\lib" -lepanet2*] to build the MEX-file;
5. The DLL file [*epanet2.dll*] should be placed in the same folder where the MEX-file exists to make it work;
6. Pay attention to the format of function usage, it should be called like "[objs, constrs]=TLN2obj(vars, NFE)" (without double quotes) in which "vars" represents a 1-by-M vector (M denoting the number of decision variables) and "NFE" represents the total number of function evaluations.

```
#include <math.h>
#include "mex.h"
#include "epanet2.h"

void TLN2obj(double *vars, double *objs, double *constrs, int NFE) {
    ****************************************************************
    * HERE IS THE CODE FOR COMPUTING OBJECTIVE VALUES.             *
    * HYDRAULIC SIMULATION IS ALSO IMPLEMENTED IN THIS FUNCTION.   *
    ****************************************************************
    objs[0] = totalCost/1000000.0; // $ => $ MM
    objs[1] = -networkResilience;
    constrs[0] = -(pressureViolation+errorCode);
    return;
}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    double *vars,*objs,*constrs;
    int NFE;
    /* Check for proper number of arguments. */
    if(nrhs!=2)
    {
        mexErrMsgIdAndTxt( "MATLAB:TLN2obj:invalidNumInputs", "Two input required.");
    }
    else if(nlhs>2)
    {
        mexErrMsgIdAndTxt( "MATLAB:TLN2obj:maxlhs", "Too many output arguments.");
    }
    /* Create matrix for the return argument. */
    plhs[0] = mxCreateDoubleMatrix((mwSize)1, (mwSize)2, mxREAL);
    plhs[1] = mxCreateDoubleMatrix((mwSize)1, (mwSize)1, mxREAL);
    /* Assign pointers to each input and output. */
    vars = mxGetPr(prhs[0]);
    NFE = mxGetScalar(prhs[1]);
    objs = mxGetPr(plhs[0]);
    constrs = mxGetPr(plhs[1]);
    /* Call the TLN2obj subroutine. */
    TLN2obj(vars, objs, constrs, NFE);
}
```

Figure 6. Sample code for compiling the MEX-file for the Two-loop Network

Major References

Deb, K., A. Pratap, S. Agarwal, and T. Meyarivan (2002), A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II, *IEEE T. Evolut. Comput.*, *6*(2), 182-197, doi: 10.1109/4235.996017.

Laumanns, M., L. Thiele, K. Deb, and E. Zitzler (2002), Combining convergence and diversity in evolutionary multiobjective optimization, *Evol. Comput.*, *10*(3), 263-282, doi:10.1162/106365602760234108.

Prasad, T. D., and N.-S. Park (2004), Multiobjective Genetic Algorithms for Design of Water Distribution Networks, *J. Water Res. Pl-ASCE*, *130*(1), 73-82, doi:10.1061/(ASCE)0733-9496(2004)130:1(73).

Vrugt, J. A., and B. A. Robinson (2007), Improved Evolutionary Optimization from Genetically Adaptive Multimethod Search, *P. Natl. Acad. Sci. USA*, *104*(3), 708-711, doi:10.1073/pnas.0610471104.

Wang, Q., M. Guidolin, D. Savić, and Z. Kapelan (2015), Two-Objective Design of Benchmark Problems of Water Distribution System via MOEAs: Towards the Best-Known Approximation to the True Pareto Front, *J. Water Res. Pl-ASCE*, *141*(3), doi:10.1061/(ASCE)WR.1943-5452.0000460.