Optimate Tutorial.

How to use Optimate, the simplest cases.

S. Ivvan Valdez

Center for Research in Mathematics, CIMAT A.C.

Mineral de Valenciana, Guanajuato, Gto., Mexico. C.P. 36000

ivvan@cimat.mx

January, 2014

Abstract

Optimate is an optimization tool delivered as a ready-to-compile C++ program. Hence, the user could insert its own evaluation function, codify tasks that must be performed after or before running the optimization process, and the program can be compiled in order to run in almost any platform or hardware. On the other hand, re-compilation is not always necessary, the compiled executable of Optimate can be connected and communicated to an external evaluator by using text files. Optimate is intended to perform global maximization/minimization of one or more objective functions, defined in continuous, binary or integer search space, with or without constraints. Currently there are openMP and MPI versions, which take advantage of multicore computers and clusters. This document is a small tutorial to use the tool. We present an introduction and examples for using Optimate. We recommend to the reader to look carefully at example 1, which is used as a basis for the others.

Contents

1 What an optimization problem is.

Many scientific and real world problems can be modelled as an optimization problem, it is to say as minimizing or maximizing a function. For example, consider the following real world problems:

- To find the best shape for given load conditions of a mechanical structure. We can modelled this problem as minimizing the structure weight, constrained by the maximum stresses, and displacements in the structure.
- To find a path for a unmanned aerial vehicle, from point a to b. It can be modelled as maximizing the distance between the path and the obstacles, and minimizing the distance between points a and b (as a two).
- To find the adequate profiles for a building structure. This can be modelled as minimizing the structure cost constrained to fulfil a given safety factor.

These are only a few examples of optimization problems, but as can be noticed possibly any real world problem can be described as an optimization problem, we only need to think about it as a maximum or a minimum; e.g. maximizing the performance of an engine, minimizing the cost of a car, maximizing the fuel a plane can load with certain efficiency, etc.

1.1 Representation of a solution

In order to model an optimization problem we need a representation of a solution. For example, consider the problem of designing combinatorial logic circuits. A possible solution for this problem is a proposal of a circuit design, hence, we need to find an adequate representation of a circuit. Figure 1 exemplify the representation of a possible solution, an array given by: $x_i = (1, 2, 2, 2, 0, 0, 1, 0, 3, 1, 2, 2, 2, 0, 1, 1, 0, 0)$ can be used to represent the gates and connections of the circuit. As shown in the left side of Figure 1, the first step is to arrange the numbers in a matrix form, where each cell contains three numbers, the first two numbers in a cell (white squares) are the inputs where the gate is connected to, and the third number (blue square) is a number which represents the gate type, as follows: 0= WIRE, 1=OR, 2=AND, and 3=XOR. the assembled circuit is shown in the right side of Figure 1.

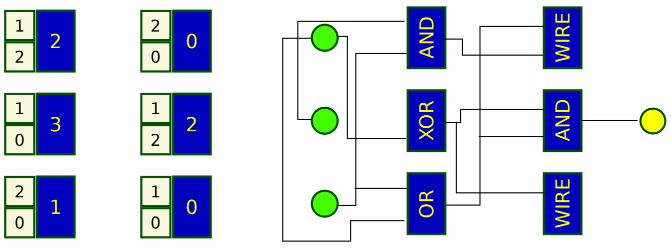

Figure 1: Example of a solution representation for combinatorial circuit design. In this case an array given by: $x_i = (1, 2, 2, 2, 0, 0, 1, 0, 3, 1, 2, 2, 2, 0, 1, 1, 0, 0)$ can be used for represent the gates and connections for the circuit in the figure.

Hence, usually, a solution to a real world optimization problem can be represented by a vector of integer or continuous variables, there are many problems which can be represented by binary variables which are a special case of integer variables. Optimate can propose solutions of continuous, integer or binary variables and its combinations.

1.2 Evaluation of a solution

After defining a representation we need an evaluator of the function, in order to evaluate possible solutions proposed by Optimate.

The evaluation of the objective function can be done by:

- A C/C++ function coded by the user before re-compiling Optimate.
- An external program or evaluator, given by the user as an input parameter.

2 Example 1. Optimizing a continuous function with a C/C++ user defined evaluation.

In this first example, we are going to explain how to use Optimate for the following problem:

$$minimize \quad f(x) = \sum_{i=1}^{n} x_i^2, \tag{1}$$

$$subject \quad to$$

$$-10(i) < x_i < 10(i)$$

The steps to do this are the following:

1. Codify the objective function in the **custom_problem.cpp** file.
2. Create the input file with the necessary Optimate parameters.
3. Compile and execute Optimate.

2.1 Example 1. Editing the custom_problem.cpp

In the `source` directory, there is a file called: `custom_problem.cpp`, inside there is a function definition that you can edit. The custom arguments are the following:

```
void custom_problem (double *xreal, double *xbin,
     int **gene, double *obj, double *constr, int nreal,
     int nbin, int ngene, int *lgenes)
```

The arguments of this function are linked with the parameters given in the input file, then make sure to set the input file as will be show below. For the example in this section we will only use the following arguments: `xreal, obj, nreal`. All the remaining arguments have an unpredictable value. The arguments have the following meaning:

`xreal`	Vector array with the values of a possible solution.
`obj`	Vector array where the objective(s) values must be stored.
`nreal`	Number of continuous variables, actually floating point variables.

Finally, for our example, the single objective function to evaluate can be coded as follows:

```
void custom_problem (double *xreal, double *xbin, int **gene, double *obj, double *
    constr, int nreal, int nbin, int ngene, int *lgenes)
{
 3      int OnInit=On_Init(), OnInitRun=On_InitRun(), OnFinalize=On_Finalize(),
        OnFinalizeRun=On_FinalizeRun();
        ///Do not evaluate the function if any of the functions above return 1 (a true
        value).
 5      if (OnInit || OnInitRun || OnFinalize || OnFinalizeRun)
            return;
 7
        ///Do the evaluation
 9          obj[0]=0.0;
            for (int i=0; i<nreal; i++)
11              obj[0]+= (xreal[i]*xreal[i]);
   }
```

As can be noticed line 3 calls several functions, the returned values from these functions are used to perform actions at the beginning or end of the execution. The functions have the following meaning:

`On_Init()`	This function only returns 1 when a execution of an optimization process starts.
`On_InitRun()`	Optimate is capable of performing several independent runs in the same execution, this function only returns true when a set of runs starts.
`On_Finalize()`	Returns true when a run finalizes.
`On_InitRun()`	Returns true when a set of runs finalizes.

The user can use this functions to load data from files, or to do anything he/she wants at the execution beginning or end, for example:

```
 1 void custom_problem (double *xreal, double *xbin, int **gene, double *obj, double *
        constr, int nreal, int nbin, int ngene, int *lgenes)
   {
 3      int OnInit=On_Init(), OnInitRun=On_InitRun(), OnFinalize=On_Finalize(),
        OnFinalizeRun=On_FinalizeRun();

 5      if (OnInit){
            ///Do something when a run starts
 7      }
        if (OnInitRun){
 9          ///Do something when a set of runs starts
        }
11      if (OnFinalize){
            ///Do something when a run finalizes
13      }

15      if (OnFinalizeRun){
            ///Do something when a set of runs finalizes
17      }
        ///Do not evaluate the function if any of  flags are turned on (saves the
        computational cost of evaluating).
19      if (OnInit || OnInitRun || OnFinalize || OnFinalizeRun)
            return;
21
        ///Do the evaluation
23          obj[0]=0.0;
            for (int i=0; i<nreal; i++)
25              obj[0]+= (xreal[i]*xreal[i]);
   }
```

2.2 Example 1. Create the input file with the necessary Optimate parameters

There are two ways of passing arguments to Optimate: by using an input file, by command line. The input file have the advantage of being editable, reusable and that it allows comments. When editing the input file consider the following:

- The # symbol starts a comment line. Comments can not be written before arguments in the same line.
- The arguments which require a numerical value are always follow by such numerical value
- Blank lines are automatically skipped.
- Variable limits are always set at the end of the input file.

An example of input file:

```
 #Setting that the problem to evaluate is the custom
-problem custom
#Where to write the output files, output=local directory, custom=filename
-filename output/custom
#Indicate that the selected set (best solutions), must be printed
-printSelec
#How often the solutions must be printed
-printingFreq 49
#Number of objective functions
-nobj 1
#Number of real variables
-nreal 2
#Indicate that the limits will be given, if not default limits are applied
-limits
#Variable limits
-10 10
-20 20
```

Another useful arguments could be:

- `-printProblemData`, without any numerical value, it indicates that the problem data of the run must be reported.
- `-nite 100`, the `-nite` is for number of iterations (generations), in this case it is set to 100.
- `-nparent 200 -nchildren 200 -nbest 200`, these three arguments most of the times must be set to an equal value, only some advance configurations could be set to different values. They are: the number of parents, children and elite individuals in an evolutionary algorithm.
- **-pmutind 1.0**, probability of mutating an individual.
- **-pmutvar 0.2**, probavility of mutating a variable in an individual.
- **-pcroind 1.0**, probability of crossing two individuals.
- **-pcrovar 0.2**, probavility of crossing a variable of two individuals (useful for continuous variables).

2.3 Example 1. Compiling and running

The Optimate is delivered with a Makefile, hence in Linux/Unix systems (with make installed), in order to compile Optimate one only must type:

```
make
```

The Optimate directory includes the following subdirectories: `source, build, bin, output`. The `source` directory contains the source code of Optimate, the `build` directory is used to stored the .o files

For executing Optimate using the input file one must type in the term where the executable is:

```
./optimate -inputfile input_file_name.txt
```

Where the input file has the name `input_file_name.txt`.

2.4 Example 1. Output

According to the input file, Optimate write into the `output` directory a set of files with a name similar to `custom_selec_20.txt`. `custom` refers to the base name given in the input file with the option `-filename output/custom`, the `selec` part is to make reference to the `selected set` which are the best solutions, and finally the 20 is the iteration or generation.
The file looks similar to the following:

```
1.390268e-03 3.500136e-02 1.285195e-02
1.452163e-03 2.329872e-02 3.015513e-02
1.710894e-03 3.379626e-02 2.384757e-02
1.743504e-03 3.427531e-02 2.384757e-02
1.760703e-03 3.650891e-02 2.068339e-02
1.867401e-03 1.196029e-02 4.152532e-02
1.867401e-03 1.196029e-02 4.152532e-02
1.965634e-03 4.131876e-02 1.607465e-02
1.972256e-03 4.195471e-02 1.456221e-02
...
```

The first column is the objective function value, the remaining two columns are the variable values. The best solution is in the first row, the other are solutions in the selected set of the same iteration.

3 Example 2. Binary variables

In this example we will use as basis the Example 1. Hence, we start editing the input file, then coding the objective function, and finally, executing Optimate.

3.1 Example 2. Editing the input file

First we edit the input file in order to set the problem data as follows:

```
#Setting that the problem to evaluate is the custom
-problem custom
#Where to write the output files
-filename output/custom
#Indicate that the selected set (best solutions), must be printed
-printSelec
#How often the solutions must be printed
-printingFreq 49
#Number of objective functions
-nobj 1
#Number of real variables
-nreal 0
#Number of binary strings
-nbin 2
#Indicate that the limits will be given
-limits
#Binary strings length (instead of limits)
10
20
```

Notice the following changes:

- `-nreal 0`, the number of real variables is set to 0.
- `-nbin 2`, this argument is added and set to 2. It is the number of binary strings which represent a single solution, some times it is convenient to have several strings in order to applied the operators in a different manner in each of them, and for advanced processes that are not discussed here.
- `10`
  `20`
  At the end of the file instead of the limits for real variables, we set the length of each of the 2 binary strings set with the `-nbin` option.

Another useful parameter is: `-pswap` which is the probability of swapping a bit in a binary string.

3.2 Example 2. Coding the objective function

In this case we will use as example the following minimization problem:

$$minimize \;\; f(x) = \sum_{i=1}^{n} x_i \tag{2}$$

$$where \;\; x_i \in \{0, 1\} \tag{3}$$

The code for this function is as follows:

```
void custom_problem (double *xreal, double *xbin, int **gene, double *obj, double *
    constr, int nreal, int nbin, int ngene, int *lgenes)
{
    int OnInit=On_Init(), OnInitRun=On_InitRun(), OnFinalize=On_Finalize(),
    OnFinalizeRun=On_FinalizeRun();

    if (OnInit){
        ///Do something when a run starts
    }
    if (OnInitRun){
        ///Do something when a set of runs starts
    }
    if (OnFinalize){
        ///Do something when a run finalizes
    }
    if (OnFinalizeRun){
        ///Do something when a set of runs finalizes
    }
    ///Do not evaluate the function if any of the if before are used.
    if (OnInit || OnInitRun || OnFinalize || OnFinalizeRun)
        return;

    ///Do the evaluation
        obj[0]=0.0;
        for (int i=0; i<ngenes; i++)
            for (int j=0; j<lgenes[i];j++)
                obj[0]+= gene[i][j];
}
```

Notice that the number of strings is in `ngene` and the length of each string is in `lgene`, and finally, each string is a row of the matrix `gene`.

4 Example 3. Mixing real and binary variables

The following example will minimize the sum of the functions in example 1 and 2, as follows:

$$\begin{aligned} minimize \;\; f(x) = x_1^2 + x_2^2 + \sum_{i=3}^{33} x_i \qquad (4) \\ where \\ -10i < x_i < 10i, \;\; for \;\; i = 1,2 \\ and \\ x_i \in \{0,1\} \;\; for \;\; i = 3..33 \end{aligned} \qquad (5)$$

In the objective functions coded in the next subsection, just remember that the indices start in 0 in C code.

4.1 Example 3. Coding the objective function

Mixing the examples above we get the following code for the objective function.

```c
void custom_problem (double *xreal, double *xbin, int **gene, double *obj, double
    *constr, int nreal, int nbin, int ngene, int *lgenes)
{
    int OnInit=On_Init(), OnInitRun=On_InitRun(), OnFinalize=On_Finalize(),
    OnFinalizeRun=On_FinalizeRun();

    if (OnInit){
        ///Do something when a run starts
    }
    if (OnInitRun){
        ///Do something when a set of runs starts
    }
    if (OnFinalize){
        ///Do something when a run finalizes
    }
    if (OnFinalizeRun){
        ///Do something when a set of runs finalizes
    }
    ///Do not evaluate the function if any of the if before are used.
    if (OnInit || OnInitRun || OnFinalize || OnFinalizeRun)
        return;

    ///Do the evaluation
        obj[0]=0.0;
    ///Binary part
        for (int i=0; i<ngene; i++)
            for (int j=0; j<lgenes[i];j++)
            obj[0]+= gene[i][j];
    ///Real part
        for (int i=0; i<2; i++)
            obj[0]+=(xreal[i]*xreal[i]);
}
```

4.2 Example 3. Input file for mixed variables

The input file for this example is similar to the following:

```
#Setting that the problem to evaluate is the custom
-problem custom
#Where to write the output files
-filename output/custom
#Indicate that the selected set (best solutions), must be printed
-printSelec
#How often the solutions must be printed
-printingFreq 5
#Number of objective functions
-nobj 1
#Number of real variables
-nreal 2
#Number of binary strings
-nbin 2
-nite 100
#Indicate that the limits will be given
-limits
#Real-variable limits
-10 10
-20 20
#Binary strings length
10
20
```

5 Example 4. Multiobjective Optimization

In the following example assume that we intent to optimize at the same time the functions:

$$minimize \quad f_1(x) = \sum_{i=1}^{2} x_i^2, \tag{6}$$

$$and$$

$$minimize \quad f_2(x) = \sum_{i=1}^{2} (x_i - 2)^2, \tag{7}$$

$$where$$

$$-10i \geq x_i \leq 10i \tag{8}$$

5.1 Example 4. Coding the objective function

The objective function looks as follows:

```c
void custom_problem (double *xreal, double *xbin, int **gene, double *obj, double
    *constr, int nreal, int nbin, int ngene, int *lgenes)
{
    int OnInit=On_Init(), OnInitRun=On_InitRun(), OnFinalize=On_Finalize(),
    OnFinalizeRun=On_FinalizeRun();

    if (OnInit){
        ///Do something when a run starts
    }
    if (OnInitRun){
        ///Do something when a set of runs starts
    }
    if (OnFinalize){
        ///Do something when a run finalizes
    }
    if (OnFinalizeRun){
        ///Do something when a set of runs finalizes
    }
    ///Do not evaluate the function if any of the if before are used.
    if (OnInit || OnInitRun || OnFinalize || OnFinalizeRun)
        return;

    ///Do the evaluation
        obj[0]=0.0;
    ///Binary part
        for (int i=0; i<2; i++)
            obj[0]+=(xreal[i])*(xreal[i]);
    ///Real part
        obj[1]=0.0;
        for (int i=0; i<2; i++)
            obj[1]+=(xreal[i]-2.0)*(xreal[i]-2.0);
}
```

5.2 Example 4. Input file

```
#Setting that the problem to evaluate is the custom
-problem custom
#Where to write the output files
-filename output/custom
#Indicate that the selected set (best solutions), must be printed
-printSelec
#How often the solutions must be printed
-printingFreq 5
#Number of objective functions
-nobj 2
#Number of real variables
-nreal 2
#Number of binary strings
-nite 100
#Indicate that the limits will be given
-limits
#Real-variable limits
-10 10
-20 20
```

5.3 Example 4. Output

The output of a multiobjective problem is Pareto set. The output is similar to example 1, but the first two columns (because there are 2 objective function) of the file are the two objective values. The objective function values can be plotted in order to present a graphical representation of the Pareto front, which show the compromise between the two objective functions. For this example the Pareto front is shown in Figure 2.

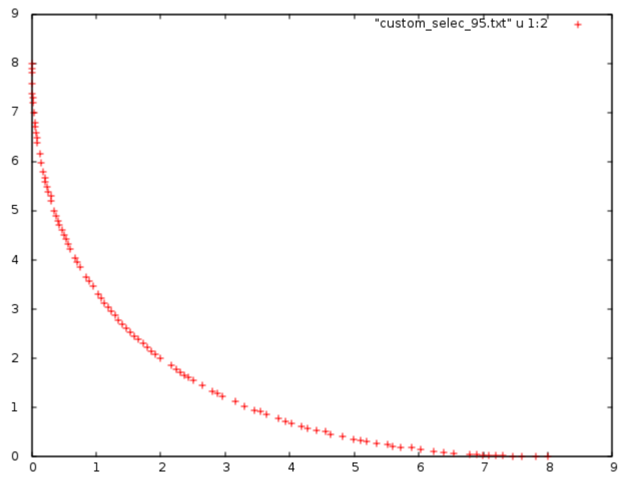

Figure 2: Pareto front from the multiobjective optimization of example 4.

6 Example 5. Using Optimate with an external evaluator

Optimate can perform communication with an external evaluator in order to provide to the user a way of optimizing functions evaluated by other programs (such as commercial ones). There are 3 basic ways of using Optimate with an external evaluator:

1. Executing Optimate and giving it the name of the evaluator program which will be executed every iteration, to evaluate a bunch of candidate solutions.
2. Executing Optimate and giving it the name of the evaluator program which will be executed only at the beginning of the optimization process, the evaluator program is responsible of indicating to Optimate when it had finished the evaluation process by writing a Flag-File.
3. Executing Optimate and executing separately the evaluator, the Flag-File is used as well as in the approach before, but in contrast Optimate will be no responsible of executing the external evaluator, so it must be executed by the user.

We will show how to use Optimate with these approaches, starting from the last one in order to clarify to the user how it works.

6.1 Optimizing by using an external program, Case 1 executing

Assume we are trying to approximate the optimum of:

$$\begin{aligned} \min f(x) &= x_1^2 + x_2^2 \qquad (9)\\ -10 &< x_1 < 100 \\ -15 &< x_2 < 3 \end{aligned} \qquad (10)$$

We will need an external evaluator of this function. For this example we well use a R script to simulate the external evaluator.

The input file must be modified with the following lines:

```
 #Problem to evaluate: evaluator which means an external evaluator
-problem evaluator
```

```
#Number of variables
nvar<-2
#Arguments collected from the shell
args <- commandArgs(trailingOnly = TRUE)
evaluatorFlagFile<-args[[1]] #First argument a flag file
evaluatorInputFile<-args[[2]]
evaluatorOutputFile<-args[[3]]
problemData<-args[[4]]
flag<-0
while(flag!=3)
{
  flag<-2
  while(flag==2){
   if(file.exists(evaluatorFlagFile)){
     flag<-scan(file=evaluatorFlagFile,what=integer(),n=1,quiet=TRUE)
   }
   Sys.sleep(1)
  }
  if (flag==3)
```

```
        break
    #Read input file
    if (flag==1)
        X<-data.matrix(read.table(evaluatorInputFile, skip=1))
    #Evaluate
        Fval<-rowSums((X-1.5)^2)
    #Print the evaluation to ourput file
    write.table(Fval, evaluatorOutputFile,col.names = FALSE,row.names = FALSE)
    #Print the flag file.
    write('2',   evaluatorFlagFile)

}
```

Code 1: optimate.R file

The script above performs the following:

- Notice that the arguments are: the flag-file, the input-file, the output-file, the problemdata-file.
- The script reads the flag file looking for a file with a flag different of 2.
- If the flag is 1, then the evaluator reads the inputfile, with the candidate solutions to evaluate. Each row is a candidate solution.
- The evaluator then, writes back the evaluation to the output file.
- The flag file is rewritten with a 2, in order to notify to Optimate that the candidate solutions have been evaluated.
- Notice that the program has a delay of 1 second each time it check for a new flag file with a 1 in it.