A Simple C++ Template Program for Simulating Operation Research Problems for Students' Learning

Zahra Asgari Rizi

***Abstract*— Operation research is the representation of real-world systems by mathematical models together with the use of quantitative methods (algorithms) for solving such models, with a view to optimizing. As we know, C++ language is a middle level language and a computer program with this language is executed faster than application softwares of the operation research area. In this paper, a simple template program with simple C++ instructions is presented for simulating many types of optimization problems. This template is very easy to learn and understand for students or anyone who works in the operation research area. They can quickly and easily simulate many types of optimization problems with this template.**

***Index Terms*—computer programming, C++ language, operation research, optimization, simulation**

I. INTRODUCTION

Operation Research (OR) is the study of mathematical models for complex organizational systems. Optimization is a branch of OR which uses mathematical techniques such as linear and nonlinear programming to derive values for system variables that will optimize performance [1]. Another definition of OR (Operational Research in Europe) is the study of mathematical models and tools with the goal of providing solutions and insights for complex decision problems.

We can also define a mathematical model as consisting of:

- Decision variables or alternatives, which are the unknowns to be determined by the solution to the model.
- Constraints to represent the physical limitations of the system
- An objective function which is an appropriate objective criterion for evaluating the decision variables or alternatives
- An optimal solution to the model is the identification of a set of variable values which are feasible (satisfy all the constraints) and which lead to the optimal value of the objective function.

In general terms we can regard OR as being the application of scientific methods / thinking to decision making. Underlying OR is the philosophy that:

- Decisions have to be made; and
- Using a quantitative (explicit, articulated) approach will lead to better decisions than using non-quantitative (implicit, unarticulated) approaches.

Indeed it can be argued that although OR is imperfect, it offers the best available approach to making a particular decision in many instances (which is not to say that using OR will produce the right decision) [2].

The principal phases for implementing OR in practice include:

1) Definition of the problem
2) Construction of the model (Formulate the Problem)
3) Solution of the model
4) Validation of the model
5) Implementation of the solution.

We suppose that the industrial engineering students or anyone who works in the operation research area can construct the mathematical programming model from an operation research problem. After that, they can solve the problem with using the presented template program in this paper. The general form of a mathematical programming model is:

$$\begin{aligned} & \min \text{ or } \max f(x_1, \ldots, x_n) \\ & s.t: \quad g_i(x_1, \ldots, x_n) \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b_i \qquad (1) \\ & x \in X \end{aligned}$$

Linear program (LP): all functions f and g_i are linear and X is continuous.

Integer program (IP): X is discrete [3].

LP's are important. Because:

- Many practical problems can be formulated as LP's
- There exists an algorithm which enables us to solve LP's numerically relatively easily.

Computer development during the past fifty years have led to the improvement of the optimization methods, so that several methods such as enumeration method, calculus-based method, heuristic and metaheuristic (Random method) and combinational optimization have developed during this period. The growth of OR since it began (especially in the last 30 years) is, to a large extent, the result of the increasing power and widespread availability of computers. Most (though not all) OR involves carrying out a large number of numeric calculations. Without computers this would simply not be possible.

As the enumeration method is the simplest algorithm for implementation and the programs with this algorithm have a

Manuscript received January 5, 2012; revised January 17, 2012. This work was supported in part by the Islamic Azad University, Falavarjan Branch, Isfahan, Iran.

Author is a member of department of Computer Engineering, Islamic Azad University of Falavarjan, Isfahan, Iran (e-mail: Z_asgari@iaufala.ac.ir and asgarir2000@yahoo.com)

structure without complexity and high comprehensibility for the students, the implementation of optimization problems is presented with enumeration method for the learning of the students.

Because in the enumeration method, just one point of domain space of objective function is studied at each iteration, this method is much simpler than other methods for implementation, but it needs considerable calculations. In this method, there is no mechanism to reduce domain space, and searchable domain space is very vast. Execution complexity or execution order of an enumeration algorithm depends on the number of decision variables and also the vastness of the acceptable range for each variable in the optimization problem. As the number of decision variables increases and the acceptable range for each variable expands, computers have to spend more time to execute optimization problems which are implemented with enumeration algorithm.

As mentioned, we suppose that the industrial engineering students or anyone who works in the operation research area can extract all the required information from the optimization problems and construct the mathematical programming model for them. In other word, the students must be able to define the relations in equation 1 from an optimization problem. In the next section, the way of implementation of optimization problems with simple program with C++ language will be illustrated.

II. How can we implement optimization problems with a simple program with C++ language

To implement optimization problems, at first we should obtain the exact range of all decision variables from constraint relations. In other word, we should determine the minimum value and maximum value for each decision variable. For example, if we have three constraints as following:

Subject to:

$$\begin{cases} 10 \times X_1 + 4 \times X_2 <= 100 \\ 5 \times X_1 + 1 \times X_2 <= 40 \\ X_1 , X_2 >= 0 \end{cases} \quad (2)$$

In this example, the minimum values for two decision variables (X_1 and X_2) are zero. With substituting the minimum value of X_1 in two first constraints, we have:

$$\begin{cases} 10 \times 0 + 4 \times X_2 <= 100 \\ 5 \times 0 + 1 \times X_2 <= 40 \end{cases} \quad (3)$$

Since both of the constraints should be correct, so X_2<=25. In the same way, with substituting the minimum value of X_2 in two first constraints, we have:

$$\begin{cases} 10 \times X_1 + 4 \times 0 <= 100 \\ 5 \times X_1 + 1 \times 0 <= 40 \end{cases} \quad (4)$$

And therefore X_1<=8.

In optimization problems with the aim of maximizing the objective function, we need minimum value of the objective function. In this case, the minimum value of the objective function could be calculated with substituting the minimum values of the decision variables in the objective function.

In optimization problems with the aim of minimizing objective function, we need maximum value of the objective function. In this case, the maximum value of the objective function could be calculated with substituting the maximum values of the decision variables in the objective function.

Before describing the template program, we want to explain three basic C++ instructions which are used in this template program: the *for* instruction, the *if* instruction and the *cout* instruction.

In C++ programming language the *for* instruction is used to make a loop. In the other word, we use *for* instruction for a block of instructions which should be executed several times. In this template program, the *for* instruction is used to make loops for the decision variables.

Another simple instruction that we need in this template program is *if* statement. *If* statement is used for evaluating some comparing relations or conditions. In this template program, the *if* statement is used to check the correctness of constraints with certain values for the decision variables.

To display the results of the optimization problems, we use *cout* instruction before the end of the program.

In this program, at first we should define all decision variables and objective function symbols in two categories, after that we should initialize the objective function variables (MaxF) with its minimum value or MinF with its maximum value, depending on the aim of the problem. One category of variables' definition is for enumerating all possible values in acceptable range, and another category is for saving just those values of acceptable range which are satisfied with all constraints and lead to better optimization. The following three statements of the template program are two categories of variables' definition and then initializing the MaxF:

```
long   X1 , X2,..., f;                          (5)
long   X1result,X2result,...,MaxF;
MaxF= initialize with the minimum possible value of f;
```

After definition and initializing MaxF or MinF variables, we should use one loop for each decision variable to enumerate all possible values in the related acceptable range for that decision variable, from the minimum value to the maximum value of it. Because in many types of optimization problems, there are at least two decision variables, so the program has nested loops (a loop which has another loop in its block).

In the innermost loop, the constraints should be evaluated. It's done with the following statement:

```
if (Constraint1  && Constraint2  &&  ...)        (6)
```

After that, the objective function should be calculated with those values of the decision variables which are satisfied with all constraints. This operation is done with the following statement in this template program:

```
f=Objective function with substituting current values
        of  X1 ,X2, ...  ;                       (7)
```

ISBN: 978-988-19251-9-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

After that, the current value of the objective function should be compared with the last optimized value that was calculated from previous iterations. If the current value of the objective function is better than the last optimized value, then all the current values of decision variables should be saved in the related result variables and current value of objective function should be saved in the MaxF or the MinF. These operations are done with the following statement in this template program:

```
If (f> MaxF)
        {
          X1result = X1;
          X2result = X2;
          ...                                    (8)
          MaxF= f;
        }
```

After all iterations, when all values in the acceptable range, for all decision variables, have been enumerated, the result values of the decision variables and the optimized value of the objective function must be displayed. The complete template program is shown in figure 1

III. STUDYING OF DIFFERENT OPTIMIZATION PROBLEMS AND DIFFERENCES IN TEMPLATE PROGRAM

As there are various types of optimization problems with different needs, there are some differences in their implementation with C++ language. In this section, these differences and their effects in the template program are presented as follow:

- If the problem is integer linear programming, so in terms of the range that the decision variables should enumerate, the decision variables should be defined as *int* or *long* data type, else they should be defined as *float* or *double* data type.
- In an optimization problem, for any decision variable, the program has one loop.
- If from all constraints of the problem, at least one of them should be correct, in the *if* instruction (to check the correctness of constraints), we must use || operation symbol (Logical OR) between the constraints. Such as:

  if (Constraint1 || Constraint2 || ...) (9)

.

```
# include <iostream.h>
void main () {
    long   X1 , X2,…, f;
    long   X1result,X2result,…,MaxF;
    MaxF= initialize with the minimum possible value of f;
    for (X1= minimum value of X1; X1<  =maximum value of   X1; X1 ++)
                for (X2= minimum value of X2; X2<  =maximum value of   X2; X2 ++)
          …
              if (Constraint1  && Constraint2  &&   …)
               {
                  f=Objective function with substituting current values of X1 ,X2, …   ;
                  If (f> MaxF)
                    {
                      X1result = X1;
                      X2result = X2;
                       …
                      MaxF= f;
                    }
               }
      cout <<endl <<"Maximum f ="<<MaxF;
      cout <<endl <<"X1 solution ="<<X1s;
      cout <<endl <<"X2 solution ="<<X2s;
       …
}
```

Replace current values of X1, X2 … and f to related result variables; because the current value of f is bigger than previous value of maximum f

Fig. 1. C++ template program for simulating operation research problems

ISBN: 978-988-19251-9-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

- If the problem is non-integer linear programming or nonlinear programming with negligible error as **ε**, in the for instruction, we use X+= **ε** instead of X++ (for Gradient Search [4]) and so the *for* loop should be written as following:

for(X=MinimumValue;X<=MaximumValue;X+= ε) (10)

- If the problem has just one decision variable, so the program has just one *for* loop.
- If the aim of the problem is minimization, we should define MinF and initialize it with the maximum possible value at the beginning of the program. Also the *if* statement, for comparing f and MinF, should be written as following:

if (f<MinF) (11)

- If the problem is zero-one programming, each decision variable can be zero or one, so the *for* loop should be written as following:

for (X1=0;X1<=1;X1++) (12)

In this case the speed of program execution is very high and it just depends on the number of the decision variables. Execution order or the big O of the zero-one programming with n decision variables in this template program equals to $O(2^n)$ [5].

- If the problem has no constraint and it should start with a certain point (in Unconstraint programming), we should write the assignment instruction for MaxF or MinF with substituting the start point values in the objective function as following. And also in the innermost loop, it's not necessary to use *if* instruction for checking the correctness of conditional statement for the constraints [4].

MaxF= initialize with the start point value;
Or (13)
MinF= initialize with the start point value;

- In some optimization problems, there are multiple objective functions. These problems have known as Multi-objective function. The simplest method for these optimization problems is to make one new objective function as a linear combination of the primary multiple main objective functions. In this combination, the impact of each primary objective function is specified with its coefficient (weight). To implement such these optimization problems, we should define and implement the primary main objective functions as peripheral functions before *main()* function, and then these peripheral functions should be called in the statement which calculates value of f. For example if the linear combination of the primary multiple main objective functions was defined as:

$f= a*g_1(X_1)+b*g_2(X_2)+...$ (14)

We should define and implement the $g_1(X)$ and $g_2(X)$ and the other primary objective functions before *main()* function.

- The big O or the execution order of the presented template program depends on the number of the decision variables and the bigness of their acceptable range. If the problem has n decision variables and the bigness of the acceptable range of each variable on average is m, so the big O of the program equals to $O(m^n)$.

IV. CONCLUSION

The template program which was presented has a structure without complexity and high comprehensibility to learn for students or anyone who works in the operation research area. It's not necessary for students to be professional in programming. They just should be familiar with the basic instructions of C++ language. To run this template program for a certain optimization problem, we just need a C++ compiler such as Turbo C++ version 3.0 or 4.5. These versions of C++ compiler are free, so the students can simulate optimization problems with this template for free for the users or themselves. The students can simulate many types of operation research problems with this template quickly and easily and then run the program and get the desirable results or debug the program and see how it works to get the results.

REFERENCES

[1] M.Galati, "Introduction to Operation Research", pp.5, Available:http://coral.ie.lehigh.edu/~maghpresent/stetson01.pdf

[2] Y. İlker Topcu, "Operation Research", pp.6, Available: http://pdf-ebooks.org/pdf/41997/OPERATIONS-RESEARCH-LECTURE-NOTES-pdf.pdf

[3] F.S.Hillier, G.J.Lieberman, "Introduction to Operation Research", 9nd Ed. New York:McGraw-Hill, 2010, pp.30-60.

[4] M.Moddares, A.Asefvaziri, "Operation Research, Mathematical Programming", 3rd Ed. Tehran:Javan Pub, 2007, pp.184-240.

[5] M.Razavi, "An Introduction to Operation Research", First Ed. Tehran: Iran Industrial Research end Education Center Pub, 2006, pp.256-258.

ISBN: 978-988-19251-9-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)