T. Bouktir
L. Slimani

J. Electrical Systems 1-2 (2005): 19-34

JES
Journal of Electrical Systems

Regular paper

Object-Oriented Economic Power Dispatch of Electrical Power System with minimum pollution using a Genetic Algorithm

This paper presents solution of optimal power flow (OPF) problem of electrical power system via a genetic algorithm of real type. The objective is to minimize the total fuel cost of generation and environmental pollution caused by fossil based thermal generating units and also maintain an acceptable system performance in terms of limits on generator real and reactive power outputs, bus voltages, shunt capacitors/reactors, transformers tap-setting and power flow of transmission lines. CPU times can be reduced by decomposing the optimization constraints to active constraints that affect directly the cost function manipulated directly the GA, and passive constraints such as generator bus voltages and transformer tap setting maintained in their soft limits using a conventional constraint load flow. The algorithm was developed in an Object Oriented fashion, in the C++ programming language. This option satisfies the requirements of flexibility, extensibility, maintainability and data integrity. The economic power dispatch is applied to IEEE 30-bus model system (6-generator, 41-line and 20-load). The numerical results have demonstrate the effectiveness of the stochastic search algorithms because its can provide accurate dispatch solutions with reasonable time. Further analyses indicate that this method is effective for large-scale power systems.

Keywords: Power flow, Optimal Power Flow, Environmental Pollution, Genetic Algorithm.

1. INTRODUCTION

The planning, design and operation of electrical power system require simulation analyses to determine the performance and the reliability of the current and future system. During the last three decades, many power system computer programs have been described in the open literature [1-3]. Some of these softwares were designed to provide various engineering analysis ranging from load flow to transient stability. Others were developed for planning and controlling power system in real-time [4,5]. However, these programs have proved their limitation to specified power configuration. For different applications, the data structures are different depending on the programming languages. With the conventional programming languages, the data structures and the algorithm procedures are very strong. So that any change even minor

Corresponding author: tbouktir@yahoo.fr.
Department of Electrical Engineering
University of Larbi Ben M'hidi
Oum El Bouaghi, 04000 Algeria.

may propagate through a whole developed modules of the program. This requires tremendous efforts to debug and time period is proportional to the size of the program source rather than the magnitude of the change. For a large-scale software system, this could conduct to a catastrophic consequence so that the system becomes unmanageable and requires to be redesigned. The limitations of the software in turn restrict the potential use of modern electronic equipment to help manage and protect the grid, because the software cannot manage the increased data such equipment can deliver. The New York Times of August 18, 2003 wrote (after the August 14, 2003 Northeast Blackout), regarding transmission lines in Ohio and the Midwest, "Problems on the lines were becoming more frequent, and a series of reports, by the industry's own quality-control offices and government agencies, described the risks and urged utilities to be more aware of the physical limits of the Midwest system. The complexity and magnitude of the power flows, one report by the Federal Energy Regulatory Commission said, could 'overwhelm the electronic and software tools used to model and manage power flows on the grid'".

To overcome these disadvantages of traditional softwares, one promising approach to achieve software reliability, maintainability and extensibility is to use Object-Oriented Methodology (OOM) [6].

Object-Oriented Technology (OOT) has proved to be an effective tool for complex systems modeling. It allows expansions and modifications over a long period and supports a wide range of applications. With OOP, problems are modeled based on physical real-world concepts. Its greatest benefits come from helping developers express abstract concepts (objects) clearly and communicate them to each other. Object-oriented approaches use the concept of objects as the unit of the organization. Objects are simple entities which are specified *what* they can do rather *how* they are done.

During the few last years, the application of the OOP to electric power system has gained widespread acceptance. The object-oriented modeling scheme is not unique and affects directly the performance of the developed software. Many authors have developed various packages and proposed interesting object-oriented power system model using different OOP languages like C++, Smalltalk, Eiffel [7,8]. However, most of the developed packages are limited to specific subjects among the various fields of power systems engineering.

In a previous article [9], the authors have proposed an object model of power system which has been tested successfully for load flow application on Sonelgaz network. This paper describes the integration of an object Optimal Power Flow (OPF) application as a module in our developed object-oriented software called Object Oriented Electrical Network Simulator (OOENS). This one has been implemented using Borland C++ Builder 5 and run under Windows XP environment.

The objective of the OPF is to minimize the total fuel cost of generation and environmental pollution caused by fossil based thermal generating units and also maintain an acceptable system performance in terms of limits on generator real and reactive power outputs, bus voltages, shunt capacitors/reactors, transformers tap-setting and power flow of transmission lines.

2. OVERVIEW OF THE OBJECT-ORIENTED MODELING

The central theme in OO modeling is the process of discovering classes, object, operations as well as the relationships among classes. In this section, we shall give a brief overview of these concepts.

2. 1 Classes and objects

Somewhat predictably, the idea of an object is central to object-oriented programming. An object is defined as an item of data, very much like a variable (or constant) in a conventional programming language. Every object belongs to an object class, which is analogous to a data type in a conventional language. However, one of the most important things about object classes is that new classes can be defined by the programmer, based on existing classes. Every object is an instance of a class. A class defines the methods and attributes that each instance of the class will possess (intentional view). It can also be seen as defining the set of objects which are instances of the class (extensional view). Using the Rumbaugh's notations [11], a class in object models is represented as a rectangle with three parts as shown in Figure 1; at the Top is indicated the name of the class, the middle part contains all data attributes of the class and at the lower part the class functions (methods) are defined. The c*onstructor* "Line (){}" is a function which initializes all the class variables. The d*estructor* "~Line (){}" is a special function for freeing the allocated memory.

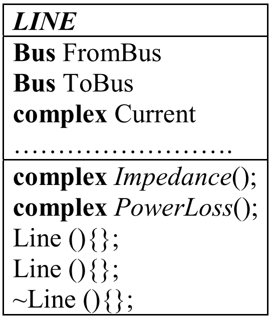


Figure 1: Class representation.

2. 2 Principles of OOM

Four major principles underlie object-oriented modeling [6,11,12]. These are:

i. **Abstraction** which denotes the capability to capture the essential properties of an entity without undue details. It consists of filtering of details not immediately needed.

ii. **Encapsulation** this defines the hiding of implementation. It requires the packaging of the entity into one impermeable unit.

iii. **Polymorphism** means that an entity can assume many forms.

iv. **Inheritance (generalization)** which allows new object classes to be defined in terms of existing object classes, inheriting both data structure and behavior (definition of methods) from the defining parent class or superclass. A class which inherits from another class is said to be a subclass of its parent. It is possible to define a method in a subclass with the same name as a method in its parent class, and this new method will **override** the method from the parent class. Complete class hierarchies can be built up through inheritance, perhaps representing hierarchies in the real-world.

2. 3 Stages of object modeling

In Object Modeling **Technique** (OMT), building a model of an application domain is achieved following three major stages [11,13]:

1 Object-Oriented **Analysis** (OOA): during this stage, the real-world system is modeled as an aggregate of simple linked objects. Their relationships are identifying using the application-domain concepts.

2 Object-Oriented **Design** (OOD): During this phase, decisions are made about how the problem will be solved. This includes two design stages:

   i. System design: in this phase, the overall system architecture is decided.

   ii. Object design: which involves the building of the model by defining the classes and their associations used in the implementation and algorithms of the methods used to implement operations.

3 **Implementation and testing**: during this phase, the developed objects and classes are finally translated in a particular OOP language (e.g. C++Builder) to become complete software.

Object-oriented modeling uses the same conceptual model across analysis, design and implementation. Analysis and design stages are intertwined in some OO methodologies.

3. ANATOMY OF THE DEVELOPED SOFTWARE

The main idea presented in this project software is based upon the object concept. So, all parts of the software are designed separately as objects using OOT in order to facilitate modifications or enhancements.

It consists of five major objects:

- A user-friendly Graphical User Interface (GUI)
- A central Object-Oriented Database (OODB)
- A number of power system applications
- An Object Mathematical Library.
- A network container.

The electrical network is modeled using two main object concepts: the classification which groups similar objects into classes and the specialization which refines classes into subclasses.

According to this approach, an object-oriented system is viewed as a collection of classes and instances ordered by two relations: instantiation and inheritance. The inheritance is a powerful abstraction for sharing simulates among classes while preserving their differences. It is the relationship between a class and one or more refined versions of it. The facilities of the base class are automatically available to all subclasses. The electrical network model is structured in the same way as the physical network. In the present case, the plant components that make up the power system for the static analysis consist of elements such as generators, transformers, transmission lines, loads and capacitor banks.

```
class     Device                  //     class     name
{
private:    //    Private    part    of    the    class
AnsiString       FNames;      //      device       name
int   FNum;                         //   device   number
AnsiString   GetName();     //    getting    the    name
void     SetName(AnsiString);     //     setting     name
int         GetNum();//         getting            number
void    SetNum(const    int);    //     setting    number
public:               //              Public         part
__property     AnsiString     Names=     {read=GetName,
write=SetName};
__property   int   Num   =   {read=GetNum,   write=SetNum};
Device(){}                //                  Constructor
~Device(){} // Destructor } // {Device}
```

Figure 2: The base class Device of the electrical network

Figure 2 illustrates the implementation of the class *Device* that has been defined as the base class for all the electrical network classes. The class *Device* has a private part consisting of parameters that are normally used for describing the electrical behavior of the component and a public part consisting of data accessing methods used to coordinate the class with the other classes and to communicate with the external environment of the class.

Figure 3 shows the global design of the base classes and their subclasses. Four base classes are identified and derived from the base class *Device*.

The class *Bus* represents a main base class because all the electrical components are connected to one or two buses. *DeviceWithTwoConnections* represents the electrical components which are linked to two buses whereas those linked to a single bus are grouped in *DeviceWithOneConnection class*. The fourth class is grouping all protection devices. Using inheritance, new classes representing the remaining network elements are derived from these classes. For example, the class *DeviceWithTwoConnections* is the base class of the network lines and transformers. Attributes and operations attached to the base class *DeviceWithTwoConnections* such as the sending bus and the receiving bus numbers, and also the resistance, reactance and susceptance are inherited by the line and transformer subclasses.

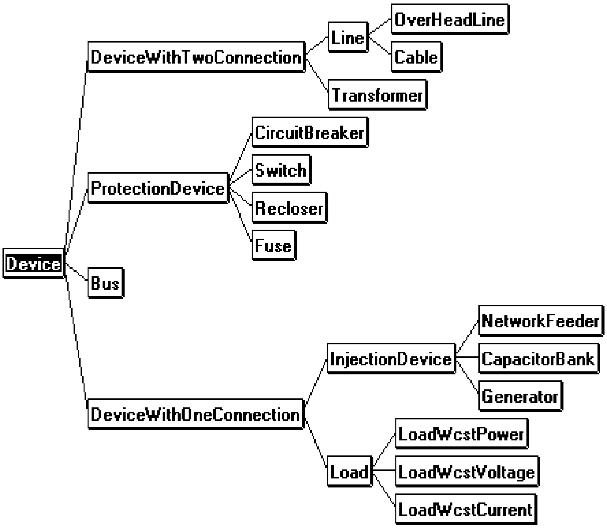


Figure 3. Class diagram of the electrical network object model

4. NETWORK CONTAINER CLASS

The network container is a dynamic vector instance inclosing the different components of the network model. The topology of the electrical system is defined by the admittance matrix which is declared in the public part. This can be used by other applications having an instance of the network container. Figure 4 illustrates the network container class with its different parameters and necessary methods providing access to manage the component objects.

```
class          Network:          public          Device{
private:
vector<Bus>                                            Bus;
vector<Generator>                                Generator;
vector<Line>                                          Line;
vector<Load>                                          Load;
vector<Transformer>                            Transformer;
void                                           Admittance();
public:
__property   nit   NB   =   {read=GetTotalBusesNumber,
write=SetTotalBusesNumber};
__property  int  NL  =  {read=GetTotalBranchesNumber,
write=SetTotalBranchesNumber};
__property    int    NSLACK         ={read=GetBusBar,
write=SetBusBar};
__property     double     BASE      ={read=GetBasePU,
write=SetBasePU};
..................................
Network(){};//                                 Constructor
~Network(){};// Destructor};
```

Figure 4: Network container class with its attributes and Operations

5. DATABASE MODULE AND DATA DICTIONARY

5.1 Database Module

The data associated with any electrical device can be stored using windows-based database system, and can be invoked by any analysis application. The advantages of database structure are: Flexibility of data transfer between the environment and other applications and data source. Data can be imported and exported easily via the SQL (structured Query Language) Security of data from loss or corruption is correctly ensured. Extension to the database can be made without difficulties.

5.2. The Data Dictionary

Attributes of all electrical objects are defined using Power System Application Data Dictionary (PSADD).

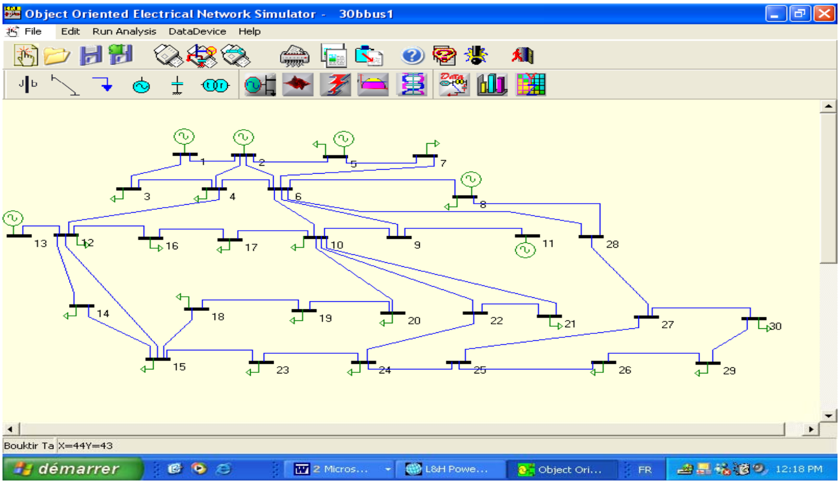
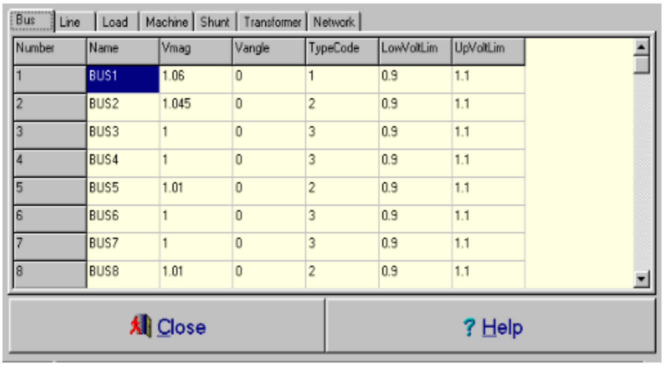


Figure 5: The main form of the simulator



Number	Name	Vmag	Vangle	TypeCode	LowVoltLim	UpVoltLim
1	BUS1	1.06	0	1	0.9	1.1
2	BUS2	1.045	0	2	0.9	1.1
3	BUS3	1	0	3	0.9	1.1
4	BUS4	1	0	3	0.9	1.1
5	BUS5	1.01	0	2	0.9	1.1
6	BUS6	1	0	3	0.9	1.1
7	BUS7	1	0	3	0.9	1.1
8	BUS8	1.01	0	2	0.9	1.1

Figure 6: Attributes of electrical devices stored using database system

Its can be stored using windows-based database system in order to build a flexible general-purpose power flow environment that have great expendability and flexibility (Figure 6). The PSADD is an attempt by IEEE to extend and renovate the old standby of the power industry, the IEEE Common Format, which served the industry well for many years. The Data Dictionary is meant primarily as a definition of terms used for analytical applications within power system models. It is organized around the notion of objects. In this paper, we us this Data dictionary to define attributes of all electrical objects. The PSADD integrates two viewpoints: a bus-oriented viewpoint common to analytical studies, and an equipment-oriented viewpoint common to measured values and system operation. The communication between the graphics, the database, network classes and power system applications are presented in the figure 7.

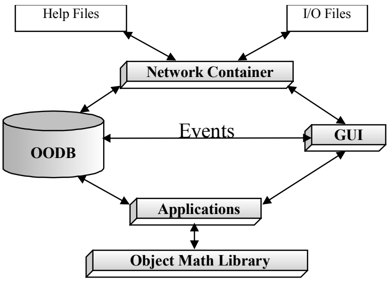


Figure 7. Architecture of the OOP environment.

6. OBJECT MATH LIBRARY MODULE

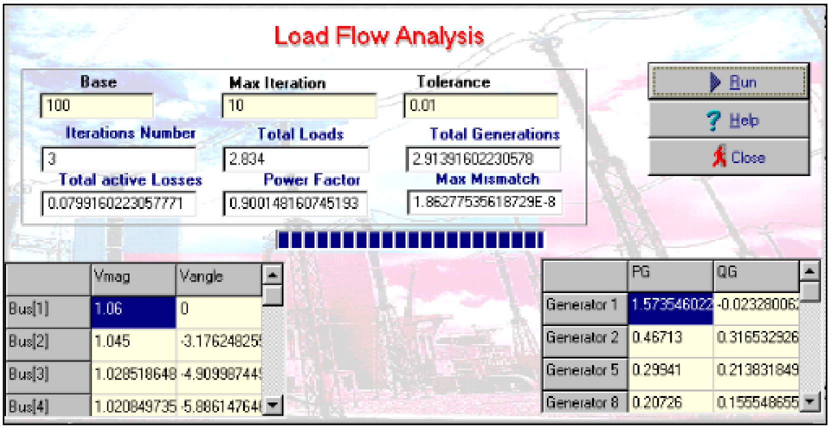
The power system analysis requires powerful numerical methods to obtain accurate results. Using OOP concepts and operators overloading inline functions, a flexible mathematical library which includes the commonly used numerical methods has been implemented. The main idea is to include numerical algorithms and methods as conceptual objects. The heart of this module is arrays objects (vector and matrix) which are called in solving iterative and non-iterative equations. These arrays have been implemented as templates objects in order to support all types of data declaration (integer, complex and float). The memory is dynamically managed using the ***new*** and ***delete*** C++ keywords.

7. POWER SYSTEM APPLICATIONS

7.1 Load flow calculation

Performing a load flow solution in a distribution network is required after any change in loads. This will provide updated voltages, angles and transformer taps and points out generators having exceeded reactive limits. to determining all active and reactive power of all generators and to determine power that it should be given by the slack generator after any change in load. The load flow is also necessary to perform other studies such as fault analysis, transient stability, and optimal power flow. All these require a fast and robust load flow program with best convergence properties. The developed load flow module is based upon the full Newton-Raphson algorithm.

By clicking the appropriate push button on the main window, an interactive and graphic interface for the Load Flow Application is invoked (see figure 8).


Load Flow Analysis

Base	Max Iteration	Tolerance
100	10	0.01
Iterations Number	**Total Loads**	**Total Generations**
3	2.834	2.91391602230578
Total active Losses	**Power Factor**	**Max Mismatch**
0.0799160223057771	0.900148160745193	1.86277535618729E-8

	Vmag	Vangle
Bus[1]	1.06	0
Bus[2]	1.045	-3.17624825
Bus[3]	1.028518648	-4.90998744
Bus[4]	1.020849735	-5.88614764

	PG	QG
Generator 1	1.573546022	-0.02328006
Generator 2	0.46713	0.316532926
Generator 5	0.29941	0.213831849
Generator 8	0.20726	0.155548655

Figure 8: Power Flow Window

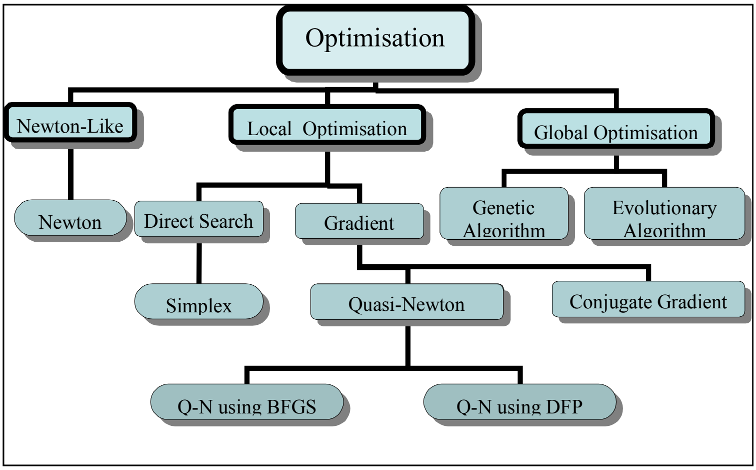


Figure 9: Optimization method hierarchy

To perform the PF calculation, the user clicks the run button then the load flow program is executed, and the results are displayed on the load flow window.

7.2 Object Oriented Optimization library

Using Object Oriented Programming, the optimisation methods used to compute the optimal power flow are designed in a hiearchical structure. In this structure, low-level objects are relatively abstract or general, while higher-level ones are more problem-specific. We will devide the methods into three classes:

- Local optimization methods class

- Global optimization methods class
- Newton type methods class.

In the local optimization methods, the current iterates in the optimization process are derived from the update of previous iterates that lie in a nearby neighborhood. For example the nonlinear conjugate gradient method, the Nelder-Mead simplex method and the Quasi-Newton methods fall into the local optimization methods class.

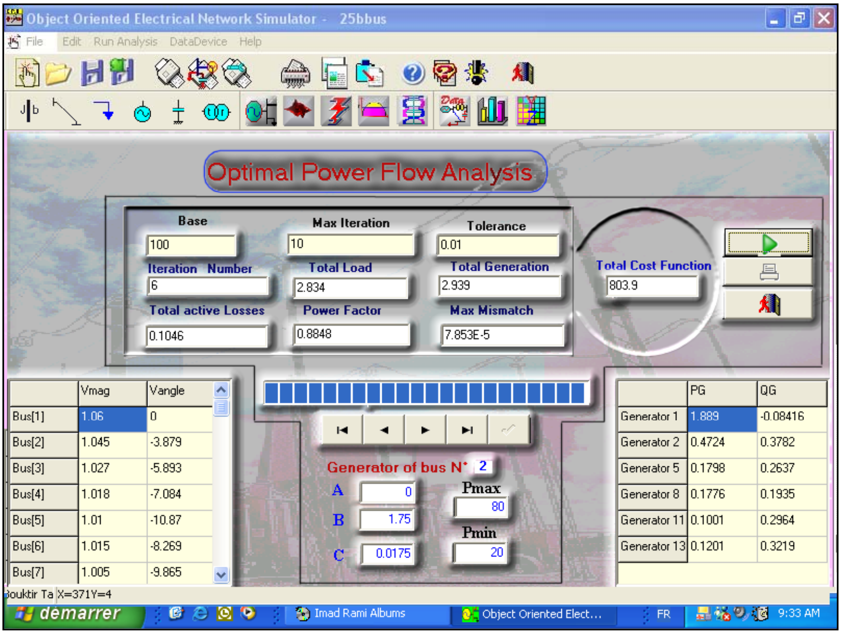
If the objective function has a high degree of complexity, then optimization might require the use of stochastic algorithms. The genetic algorithm and evolutionary algorithm fall into this global optimization methods class.

Due to their evolutionary nature, genetic algorithms will search for solutions without regard for the specific inner structure of the problem. GAs can handle any kind of objective functions and any kind of constraints, linear or nonlinear, defined on discrete, continuous, or mixed search spaces. In The Newton type methods, we need the availability of the objective function and analytic first and the second derivatives. In Figure 9, we present the optimization method hierarchy.

7.3 Optimal Power Flow Class

The Optimal Power Flow (OPF) problem is modelled as a class multi-inherited from a Load Flow (LF) class and a Genetic Algorithm (GA) class and has a direct access to a main power system object class. The necessary data for performing the optimization are the upper and lower constraint vectors and the coefficient vector of the cost function. Several computing methods are available in this class such as the cost function method, the optimal line search of the minimum, the integration of the inequality constraints and the method for updating the design vectors (Figure 10).

The OPF module is treated using a genetic algorithm and can be used in large power distribution systems. To make the dispatch more practical, our OPF takes into account the network losses. To accelerate the processes of the optimization, the controllable variables are decomposed to active constraints and passive constraints. The active constraints which effect directly the cost function are included in the GA process. The passive constraints which affect indirectly this function are maintained in their soft limits using a conventional constraint load flow, only, one time after the convergence of GA. The search of the optimal parameters set is performed taking into the account that the power losses are 2% of the power demand. The slack bus parameter will be recalculated in the load flow process to take the effect of the passive constraints and the exact power losses.


Object Oriented Electrical Network Simulator - 25bbus
File Edit Run Analysis DataDevice Help
Optimal Power Flow Analysis
demarrer
Imad Rami Albums
Object Oriented Elect…
FR
9:33 AM

Base	Max Iteration	Tolerance	
100	10	0.01	
Iteration Number	Total Load	Total Generation	Total Cost Function
6	2.834	2.939	803.9
Total active Losses	Power Factor	Max Mismatch	
0.1046	0.8848	7.853E-5	

	Vmag	Vangle
Bus[1]	1.06	0
Bus[2]	1.045	-3.879
Bus[3]	1.027	-5.893
Bus[4]	1.018	-7.084
Bus[5]	1.01	-10.87
Bus[6]	1.015	-8.269
Bus[7]	1.005	-9.865

Generator of bus N° 2

A	0	Pmax	80
B	1.75	Pmin	20
C	0.0175		

	PG	QG
Generator 1	1.889	-0.08416
Generator 2	0.4724	0.3782
Generator 5	0.1798	0.2637
Generator 8	0.1776	0.1935
Generator 11	0.1001	0.2964
Generator 13	0.1201	0.3219

Bouktir Ta X=371Y=4

Figure 10. Optimal Power Flow Window

Therefore one can say that a more natural representation of the problem offers more efficient solutions. Then one of the greater improvements consists in the use of real numbers directly. Genetic Algorithms with real code in Economic Power Dispatch provide an edge over common GA mainly because they do not require any special coding of individuals. In this case, since the desired outcome is the operating point of each of the dispatched generators (a real number), each of the individuals can be directly presented as a set of real numbers, each one being the produced power of the generator it concerns.

Our Genetic Algorithm is based on the completed Genocop III system [15], developed by Michalewicz, Z. and Nazhiyath, G. Genecop III for constrained numerical optimization (nonlinear constraints) is based on repair algorithms. Genocop III incorporates the original Genocop system [16] (which handles linear constraints only), but also extends it by maintaining two separate populations, where a development in one population influences evaluations of individuals in the other population. The first population Ps consists of so-called search points, which satisfy linear constraints of the problem; the feasibility (in the sense of linear constraints) of these points is maintained by specialized operators (as in Genocop). The second population, Pr, consists of fully feasible reference points. These reference points, being feasible, are evaluated directly by the objective function, whereas search points are "repaired"' for evaluation.

8. APPLICATION STUDY

The OOOPF using Genetic Algorithm (GA) has been developed by the use of Borland C++ Builder version 5 . More than 6 small-sized test cases were used to demonstrate the performance of the proposed algorithm. Consistently acceptable results were observed. The IEEE 30-bus system with 6 generators is presented here. The total load was 283.4 MW. We propose to apply a genetic algorithm of real type to present active powers of the 6 generators directly. The parameters of the developed GA are: the number of maximal iteration is 5000, the size of population is 70, the crossover used is of heuristic type, the mutation of "non-uniform" type, the operator of selection remains identical as the one of the roulette wheel, the probability of replacement is 0.25.

Upper and lower active power generating limits and the unit costs of all generators of the IEEE 30-bus test system are presented in Table 1. The NOx emission characteristics of generators are grouped in Table 2.

Table 1: Power generation limits and cost coefficients for IEEE 30-bus system.

Bus	Pg_{min} *(MW)*	Pg_{max} *(MW)*	*a* (\$/hr)	*b* (\$/MW.hr)	$c.10^{-4}$ (\$/MW²$.hr)
1	50	200	0	2.00	37.5
2	20	80	0	1.75	175.0
5	15	50	0	1.00	625.0
8	10	35	0	3.25	83.0
11	10	30	0	3.00	250.0
13	12	40	0	3.00	250.0

Table 2: Pollution coefficients for IEEE 30-bus system

Bus	$a.10^{-2}$	$b.10^{-4}$	$c.10^{-6}$	$d.10^{-4}$	$e.10^{-2}$
1	4.091	-5.554	6.49	2.0	2.857
2	2.543	-6.047	5.638	5.0	3.333
5	4.258	-5.094	4.586	0.01	8.0
8	5.326	-3.55	3.38	20.0	2.0
11	4.258	-5.094	4.586	0.01	8.0
13	6.131	-5.555	5.151	10.00	6.667

Table 3. Results of minimum total cost for IEEE 30-bus system in three cases (α=1, α=0.5 and α=0)

Variable	Initial state	Generation cost minimum	Generation cost + Emission minimum	Emission minimum
Pg_1(MW)	99.211	180.818	128.2124	66.6384
Pg_2(MW)	80	48.9381	65.59	65.7536
Pg_5(MW)	50	18.9533	23.7065	49.9997
Pg_8(MW)	20	20.5136	26.4992	34.9999
Pg_{11}(MW)	20	10.3941	24.6192	29.9395
Pg_{13}(MW)	20	13.7138	21.7388	39.9998
Generation cost ($/hr)	901.9180	803.1060	824.9884	943.1008
Power Loss (MW)	5.8120	9.9308	6.9661	3.9309
Emission (ton/h)	0.2391	0.3771	0.2659	0.2051
Total cost ($/h)	1033.6	1010.7	971.4	1056.1

The results including the generation cost, the emission level and power losses are shown in Table 3. This table gives the optimum generations for minimum total cost in three cases: minimum generation cost without using into account the emission level as the objective function (α=1), an equal influence of generation cost and pollution control in this function and at last a total minimum emission is taken as the objective of main concern (α=0). The active powers of the 6 generators as shown in this table are all in their allowable limits. We can observe that the total cost of generation and pollution control is the highest at the minimum emission level (α=0) with the lowest real power loss (3.931 MW). As seen by the optimal results shown in the table 3, there is a trade-off between the fuel cost minimum and emission level minimum. The difference in generation cost between these two cases (803.1060 $/hr compared to 943.1008 $/hr), in real power loss (9.9308 MW compared to 3.9309 MW) and in emission level (0.3771 Ton/hr compared to 0.2051 Ton/hr) clearly shows this trade-off. To decrease the generation cost, one has to sacrifice some of environmental constraint. The minimum total cost is at α=0.5. The security constraints are also checked for voltage magnitudes, angles and branch flows. The voltage magnitudes and the angles are between their minimum and the maximum values. No load bus was at the lower limit of the voltage magnitude.

The branch MW/MVAR/MVA flows do not exceed their upper and lower limits. These results are not included in this paper.

The figure 11 shows the best fitness found for every generation (for α=0.5). We note a fast progress of the value of the best fitness for every generation. The optimum has been obtained after only 1 second for the 5000 generations tested with P4 1.5,GHz,128MO. Ever since the iteration 2968, it converges already, toward the optimum value of the order of 973.05 \$/h. This value does not take into account the exact cost of the total real power losses. We proceed then to a power flow calculation of type Newton-Raphson and readjust slack generation that takes in consideration the exact losses of real powers. The convergence of the method of N-R is achieved after 4 iterations and 0.1 sec.

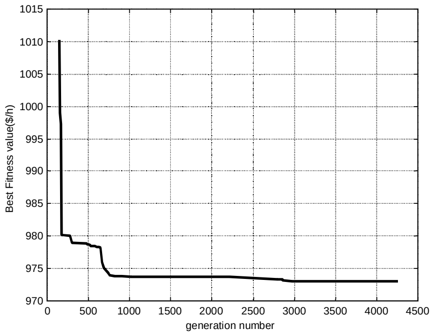


Figure 11: Better Fitness during the EP processes for IEEE 30-bus system

9. CONCLUSION

In this paper, the application of the object-oriented concepts and the Builder C++ programming language to the OPF computation using genetic algorithm have been explored and tested. The implementation of the OPF algorithm takes full advantage of inheritance from the load flow, genetic algorithm and network container. This has provided facilities in terms of easy programming, reusability and easier maintenance. The proposed OOOPF-GA program has been tested and validated on the IEEE 30-bus system. It shows that OPF-GA is a global method since it converges to the same solution from almost any starting control vector and gives a secure control vector. This method would be very useful for power planner and/or operator treat not only cost but also with environmental objective power system. In this approach, only the active constraints are taken to calculate the optimal solution set using evolutionary programming based on GENECOP III system and the passive constraints are taken in an efficient load flow by recalculating active power of the slack bus.

REFERENCES

[1] T. Adielson, SIMPOW- A digital program system for static and dynamic simulation of power systems" OEPSI conference, Bankok, Nov. 1982.

[2] A. F. Neyer, F. F. Wu and K. Imhof, object-oriented programming for flexible software; example of a load flow, IEEE Trans. on Power Systems, Vol. 5, No.3, Aug. 1990.

[3] M. Rochefort, N. De Guise and L. Gingras, Development of a graphical user interface for a real-time power system simulator, Electric Power Systems Research 36, 1996, pp.203-210.

[4] M. Foley and A. Bose, Object-orientated on-line network analysis, IEEE Trans. on Power Systems, Vol. 10, pp.125-132, 1995.

[5] K. Hasan K, B. Ramsay and I. Moyes, Object oriented expert system for real-time power system alarm processing- Part 1. selection of a toolkit, Electric Power System Research 30, pp. 77-82, 1994.

[6] B. Belkhouche, Object-oriented modeling tools, www.eecs.tulane.edu/Belkhouche

[7] E. Z. Zhou, Object-oriented programming, C++ and power system simulation, IEEE Trans. on Power Systems, Vol. 11, No. 1, pp. 206-214, Feb. 1996.

[8] G. Eldridge, Introducing Eiffel: An Object-technology approach to power system analysis, www.progsoc.uts.edu.au/~geldridg/psa-objects/ppr_9601.zip.

[9] T. Bouktir, A. Gherbi, L. Belfarhi and M. Belkacemi, An efficient object-oriented load flow applied to a large-scale power system: Application to sonelgaz network, ICEL2000, Oran, Algeria.

[10] J. Rumbaugh, M. Baha, W. Premerlani, F. Eddy and W. Lorensen, Object-oriented modeling and design, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1991.

[11] K. Reisdorph, Borland C++ Builder 3, Simon & Schuster Macmillan, France, 1998.

[12] G. Booch, Object-oriented design with applications, 2nd Edition, Benjamin Cummings, Coleman Arnold, 1994.

[13] J. Zhu and D. L. Lubkeman, Object-oriented development of software systems, IEEE Trans. on Power systems, Vol. 12, No. 2, pp. 1002-1007, May 1997.

[14] G. W. Stagg and A. H. El-Abiad, Computer methods in power system analysis, McGraw Hill, 1968.

[15] Z. Michalewicz and G. Nazhiyath, Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints, Proceedings of the Second IEEE ICEC, Perth, Australia, 1995.

[16] Z. Michalewicz and C. Janikow, Handling constraints in genetic algorithms, Proceedings of Fourth ICGA, Morgan Kauffmann, pp. 151-157, 1991.