Jurnal Ilmiah KURSOR Menuju Solusi Teknologi Informasi

Vol. 7, No. 4, Desember 2014 **ISSN 0216 – 0544**

DESIGN AND DEVELOPMENT OF COMPONENT LIBRARY GENETIC ALGORITHM BY USING OBJECT-ORIENTED DESIGN AND PROGRAMMING

[a]Hadi Suyono, [b]Adharul Muttaqin, and [c]Eka Prakarsa Mandyartha
[a,b,c]Department of Electrical Engineering, Faculty of Engineering, University of Brawijaya
E-Mail: hadis@ub.ac.id

Abstrak

Makalah ini menyajikan desain dan pembuatan komponen *library* Algoritma Genetik dengan menggunakan pendekatan *object-oriented designand programming* (OODP) dan *Component-based Develepment* (CBD). KomponenAlgoritma Genetika (AG) merupakan komponen *software engine*dibuat sendiri yang digunakan untuk membantu menyelesaikan persoalan optimisasi dengan menggunakan struktur Algoritma Genetika yang disebut dengan *Library* Algoritma Genetika (LibAGen). Metodologi OODP dan CBD meliputi analisis kebutuhan, diagram *use-case*, diagram kelas dan diagram sekuensial. *Library* Algoritma Genetika (LibAGen) ini terdiri dari 22 kelas yang dikelompokkan dalam *namespace* berdasarkan struktur desain AG yang diperlukan meliputi representasi populasi, fungsi evaluasi, operator genetika (*crossover* dan mutasi) dan seleksi. Untuk mengukur performansi dari *engine* LibAGen validasi telah dilakukan dengan menggunakan persamaan fungsi sinusoidal dua parameter. Waktu eksekusi dan nilai optimum parameter dengan beberapa pengujian dengan variasi jumlah generasi (iterasi) juga dilakukan pada makalah ini. Parameter AG yang digunakan adalah probabilitas *crossover* 25% dan probabilitas mutasi 1%. Hasil uji validasi menunjukkan bahwa nilai *fitness* terbaik adalah 388,501 dengan nilai parameter x_1 = 11,6256 dan x_2 = 5,7249. Terdapat perbedaan tidak signifikan antara nilai *fitness* terbaik dibandingkan dengan hasil Michalewicz (1999) yaitu sebesar 0,08%.

Kata kunci:Algoritma Genetika, component library, object-oriented design and programming (OODP)

Abstract

This paper presents the design and development of Genetic Algorithm (GA) library components by using object-oriented design and programming (OODP) and Component-based development(CBD). Genetic Algorithm component is an engine software component which is developed by own development for solving the optimization problem by using a structure of Genetic Algorithm (GA) called as Genetic Algorithm Library (LibAGen). OODP and CBD methodologies include requirement analysis, use-case diagrams, and class diagrams. Genetic Algorithm Library (LibAGen) consists of 22 classes which is grouped into namespaces based on GA design structure that include population representation, evaluation function, genetic operators (crossover and mutation) and selection. To measure the performance of the LibAGen engine, a validation has been carried outby using a sinusoidal function with two-parameters. Optimal parameter with some testing through variations of the number generations (iterations) have been performed in this paper. The GA parameters selected are crossover probability of 25% and mutation probability of 5%. Validation test results indicate that the best fitness and parameters are 388,501, x_1 = 11,6256 and x_2 = 5,7249. There is no significant result in term of the best fitness compared with Michalewicz (1999) i.e. 0.08%

Key words:Genetic Algorithm, component library, object-oriented design and programming (OODP)

INTRODUCTION

In solving the optimization problem, there are two widely used approaches, namely the deterministic methods and non deterministic approaches [1]. The deterministic method is an optimization solution by using mathematical and numerical approaches, whereas the non-deterministic method is by using a heuristic probability approach and artificial intelligent (AI). The heuristic probability approach and the AI, Genetic Algorithm (GA) are widely applied for solving The optimization as in the data mining process[2,3], the power system problem [4,5], control systems[6] and other applications.

Genetic Algorithms (GA) is a computational approach to solve an optimization problem by modeling the problem in a process as if biological evolution [7]. In general, the stages in the GA is starting with establishment of a set of candidates for a potential solution that satisfies all constraints required in the initialization process.

The set of potential solutions is defined at the beginning called as chromosomes. Chromosomes are formed randomly and can be as an array numbers in binary or decimal form that is generated and then selected according to the required constraints. The entire set of chromosomes represents a population. Furthermore, these chromosomes will be evolved in several iterations, called generations. The new generation (*offspring*) generated through the process of crossover and mutation. Offspring chromosomes is evolved by a suitability fitness that will be selected as the best results while others are discounted [7,8].

To simplify the analysis and implementation of GA in solving the optimization problem, software with several approaches is needed. The software should be designed and developed thus the application could be updated and maintenance easily. In general, the design and development methodologies are widely applied software is modular or procedural approach in which a problem is divided in the form of the function/procedure/subroutine that represents sub-problem required in the whole problem [9].Another design and implementation approach is design and object-oriented programming (OODP) and component-based development (CBD) [10,11]. CBD and reusability techniques are a new approach that have many benefits including reducing the time, resources, costs, and increase productivity in software development [11,12].

CBD can be developed via two programming approaches, structural-based design or object-oriented design (OOD). However, the OOD approach is widely applied in the development of software applications where a complex system can be maintained more easily. The design is based on the object-oriented approach allows complex problems to be divided into small parts. OODP and CBD implementation can be found in several references [13,14].

Genetic algorithm library is a component software that provides a problem solution by using genetic algorithm structure such as chromosome representation, evaluation function, and genetic operators such as crossover, mutation, and selection**.**

STRUCTURE AND LIBRARY DESIGN

Library Structure

The Genetic Algorithm (GA) library structure is given in Figure 1. Library structure consists of three main layers. The first layer consists of units that are not directly related to the GA, but the implementation of these units is important. GA library implements number random generator to provide the randomly number with different data types such as integer, float, and Boolean. The summary result of chromosome and population operation is provided by statistic observer. Such features provide common functions used by other units which are at the higher layers in the library.

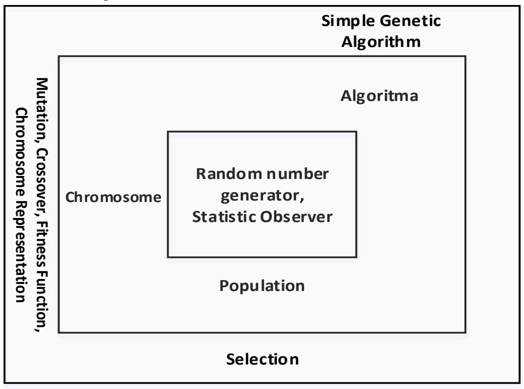
Simple Genetic Algorithm
Algoritma
Mutation, Crossover, Fitness Function, Chromosome Representation
Chromosome
Random number generator, Statistic Observer
Population
Selection

Figure 1.Genetic Algorithm Structure Library

The middle layer consists of three units i.e. Chromosome, Population, and Algorithm. The main features of the library are implemented in this layer. The chromosomes unit represents the generating of chromosomes and population that define the behavior in the system. The Algorithm provides the chromosome process.

The highest layer includes units that deal with the genetic operations such as crossover, mutation, and fitness operation. The population unit is a unit that controls a set of chromosomes (population). The selection operations such as roulette wheel and rank ordering of the fitness are included. The last unit is the Simple GA unit that implements the problem solutions process by using the GA structure.

Use-Case Diagram

Use case model system describes the interaction between the actors who acquired the library with the library feature and environment. The components required to build the library are represented as use-cases. In the LibAgen library design, there are seven components required i.e. selection of chromosome representation, definition of fitness function, crossover method selection, mutation method selection, performing the algorithm, and acquired the solution statistic. The specific actor in this design is the library's user. The interaction actor and use-cases is given in Figure 2.

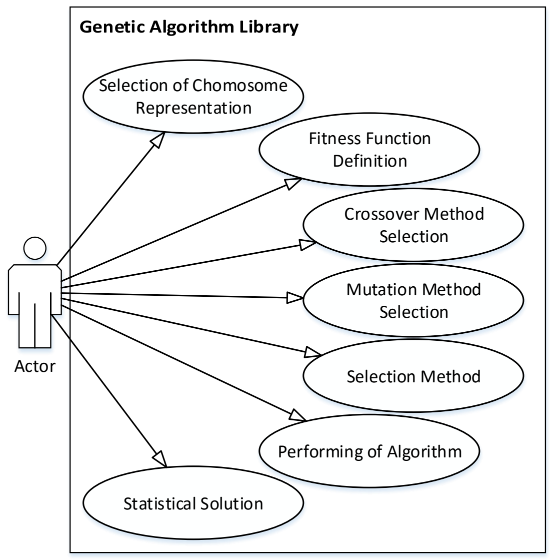
Genetic Algorithm Library
Selection of Chomosome Representation
Fitness Function Definition
Crossover Method Selection
Mutation Method Selection
Selection Method
Performing of Algorithm
Statistical Solution
Actor

Figure 2. Diagram *use-case library*

Selection of chromosomes representation use-case is to select or define the appropriate chromosome representation such as binary encoding, float or real value, or permutation approaches. Fitness function use-case defines the fitness function, in which the fitness function allows the user library to provide any specific function and later the fitness value will be provided. Crossover and mutation use-cases are used to select the crossover and mutation methods provided by the library or the user can define by their own method.

Selection method and perform the algorithm use-cases are used to define the algorithm selection methods that not provided by the library and to solve the problems that have been defined previously. Population statistics use-case is conducted to obtain data on the best chromosome, the average fitness, and maximum or minimum fitness values.

Library Design

Units contained in the GA library structure consists of group classes and interfaces. Grouping of some classes and interfaces in a single component called a package - further in this paper is called the namespace. Table 1 presents a list compiled library namespace of genetic algorithm with their description.

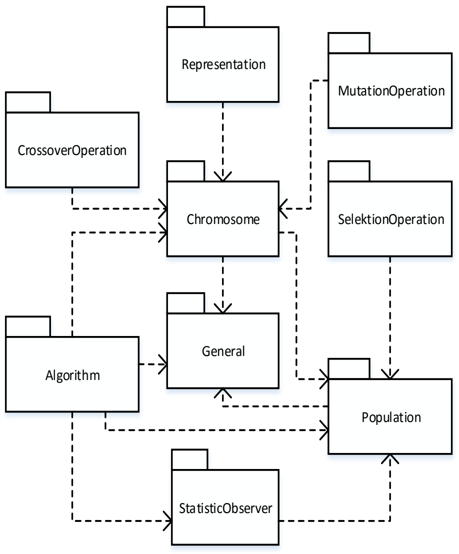

Fig. 3. Namespace relation

Table 1. List of namespace and description

Namespace	Description
Algorithm	Consist of classes required to implement the genetic algorithm structure
Chromosome	Consist of interfaces and classes needed to implement the behavior and representation of chromosomes and genetic operations
Chromosome. CrossoverOperation	Consist of the implementation of crossover operation
Chromosome. MutationOperation	Consist of the implementation of mutation operation
Chromosome. Representation	Consist of the implementation of chromosome representation
General	Contains common classes used by the library
Population	Consist of the classes used to create a Population of chromosome and genetic operation
Population.SelectioOperation	Consist of the implementation of the operation selection
StatisticObserver	Consist of the classes used for providing the Population statistic information

Genetic algorithm library architecture design is described as a relation between namespace to represent the overall system modeling. Each namespace consists of the class members. The relationships among namespace can be seen in Figure 3. The relationships are depicted with dashed arrows indicate that the namespace end used by the connecting arrows (aggregation), for example, the General namespace used by namespace Chromosome, Population, algorithms, and also by other namespaces.

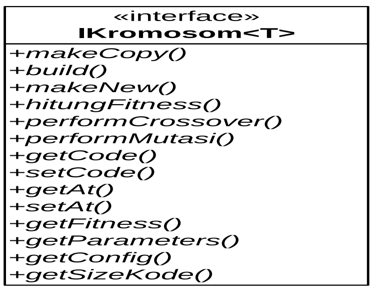

Figure 4. Class Diagram of `IKromosom` Interface

CLASSES DESIGN

Chromosome

Chromosome is the main object in the GA structure library, which is defined by **`IKromosom`**class. **`IKromosom`** class is the actual implementation of the interface for devoping the chromosome. **`IKromosom`** class is shown in Fig. 4, in which there are many functions associated to the operation of the establishment and operation of chromosomes, such as **`makeNew`**() function to create a new chromosome, **`hitungFitness`**() function is the process of calculating fitness value of each chromosome.

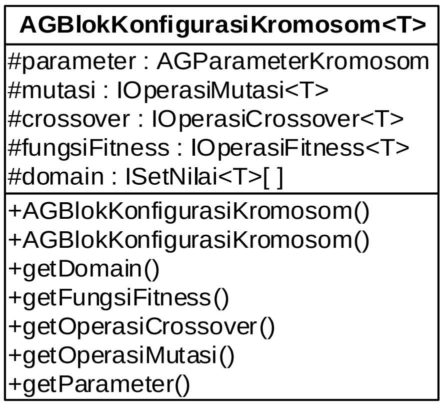
AGBlokKonfigurasiKromosom<T>
#parameter : AGParameterKromosom
#mutasi : IOperasiMutasi<T>
#crossover : IOperasiCrossover<T>
#fungsiFitness : IOperasiFitness<T>
#domain : ISetNilai<T>[ ]
+AGBlokKonfigurasiKromosom()
+AGBlokKonfigurasiKromosom()
+getDomain()
+getFungsiFitness()
+getOperasiCrossover()
+getOperasiMutasi()
+getParameter()

Figure 5. Class Diagram of `AGBlokKonfigKrom`

Chromosome configuration data includes parameters of chromosomes, mutation and crossover methods, fitness function to be used, and chromosome representation. The chromosome configuration is defined in the AGBlokKonfigKrom class as given in Figure. 5.

Crossover and mutation operations, and also the fitness function is defined by the interface of each class such as IOperasiCrossover, IOperasiMutasi, and IoperasiFitness respectively. IOperasiFitness is an interface for the fitness function that can be defined by users. Those interface and classes are clustered into the **`Chromosome`** namespace.

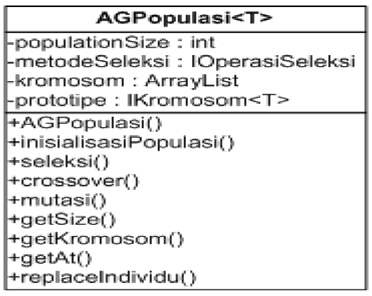

Figure 6. Class Diagram of `AGPopulasi`

Population

Population object contains of chromosome and population configuration objects that is given in Figure 6. Configuration population includes the population parameter such as the population size and the proposed selection method. **`AGPopulasi`**is assembled into Population namespace.

Representation ofChromosome

`AGKromosomRealValue` class can be used for the representation of the chromosome that represents a solution to the real number coding, where the details of the definition of the class diagram shown in Figure 7. **`AGKromosomRealValue`** class is a subclass of class **`IKromosom`**.

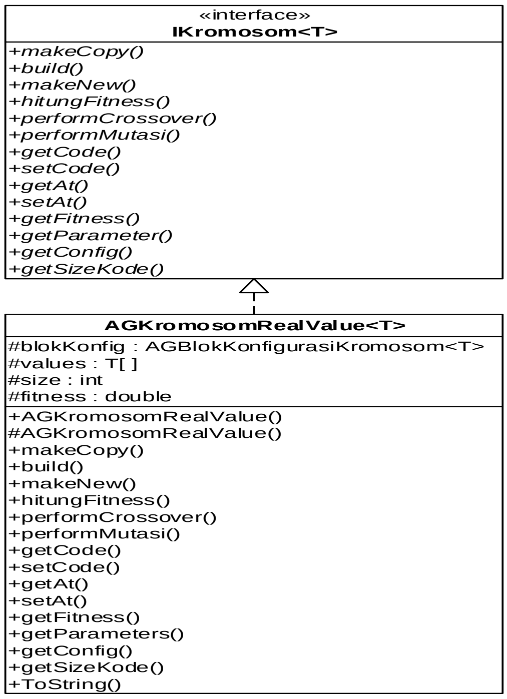

Figure 7. Class Diagram of `AGKromosomRealValue`

Genetic operations, namely crossover and mutation, can be defined and implemented in separate classes. Furthermore, object classes are defined in **`AGBlokKonfigKrom`** as given in Figure 8.

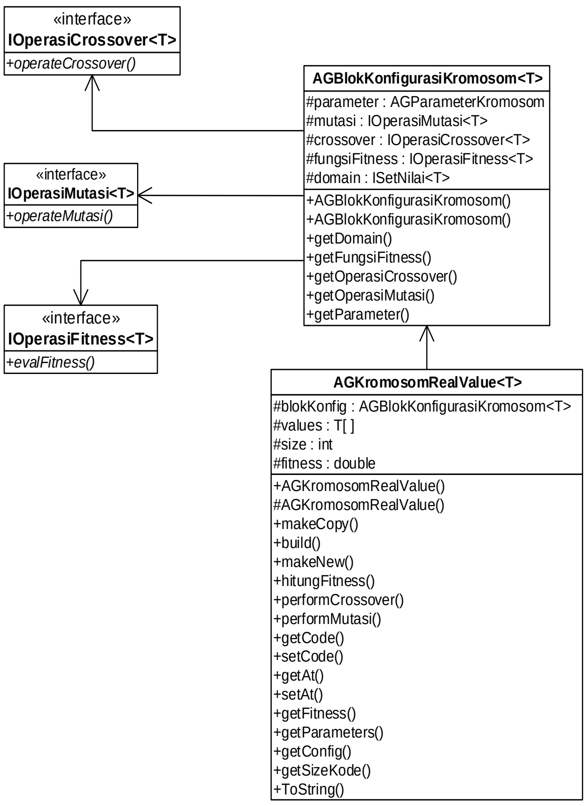

Figure 8. Relationship of class diagram between `AGKromosomRealValue`class and GA operation interfaces

In most optimization problem by using the GA solution, the variables which are represented as chromosome have restrictions called as constraints. These constraints on the library object is realized through **`ISetNilai`** class interface. **`ISetNilai`**class interface from the object set with the value of the class definition is given in Figure 9. The relation of the three classes **`AGKromRealValue`**, **`AGBlokKonfKro`** and **`ISetNilai`**interface are given in Figure 10.

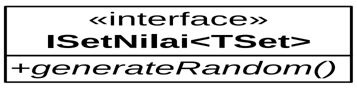

Figure 9. Class Diagram of `ISetNilai`Interface

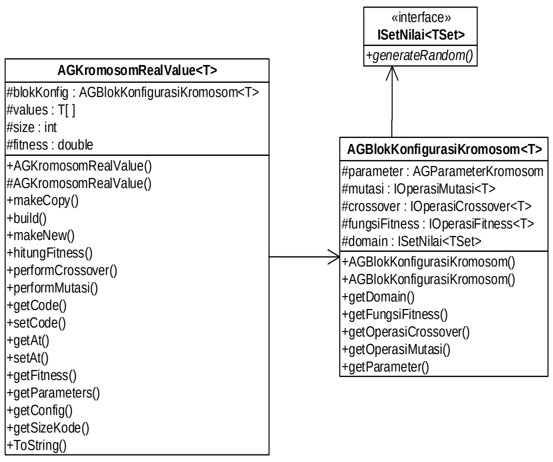

Figure 10. Relation Class Diagram `AGKromRealValue`, `AGBlokKonfKro` and interface `ISetNilai`

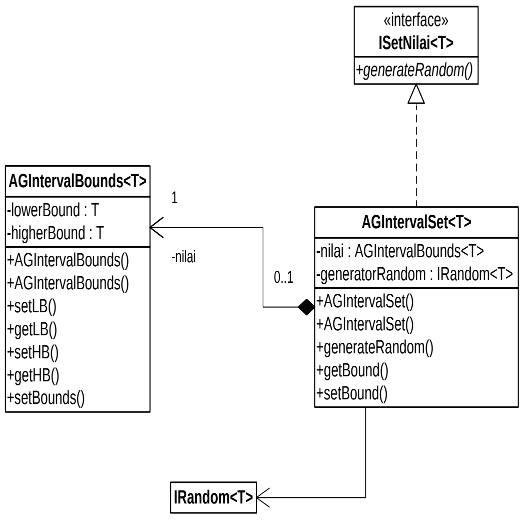

Figure 11. Class Diagram of AGIntervalSet and AGIntervalBounds

AGIntervalSet class, which is an implementation of the ISetNilaiinterface, represents the set of values that will generate numbers randomly according to pre-defined intervals. The boundaries of the interval defined by the class that is composed AGIntervalBounds on AGIntervalSet class. The relationships design between the classes is given in Figure 11.

Random Number Generator

The random number generator object has task to produce random numbers that can be an integer, float and double data types. This object is represented as AGRandGen class that is given in Figure 12. The generated random numbers could be any values between 0 and 1 for float and double data types, whereas for integer data types between 0 and the maximum value of 32-bit.

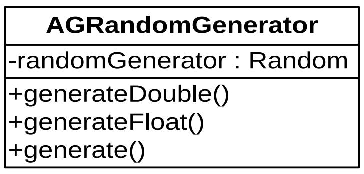

Figure 12. Class Diagram `AGRandGen`

In this GA library has also generating random value objects are more specific that can be defined by user. This feature is defined in the AGRandGenobject. There are three classes for the random values generatorsnamely AGRandDoubleclass for random value with double data type, AGRandomIntegerclass for random value with an integer data type, and AGRandomBoolfor random value with Booleandata type. The relationships design between classes on AGRandGenis given in Figure 13. The class AGRandDouble, AGRandomInteger, and AGRandomBool is composed in the AGRandGen class.

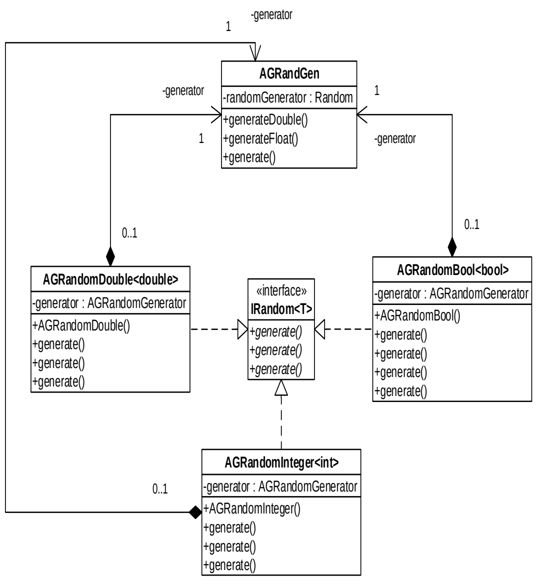

Figure 13. Random Generator Class Diagrams of `AGRandGen`

Crossover, Mutationand Selection Operations

Crossover, mutation and selection methods are provided as a single or multiple cross-points, random uniform of mutation, and roulette-wheel selection method respectively. The crossover, mutation and selection operations are given in Figure 14, Figure 15, and Figure 16 respectively.

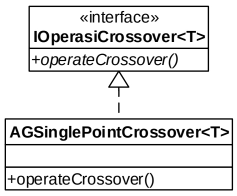

Figure 14. Class Diagram of *Crossover*

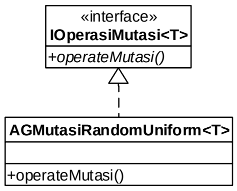

Figure 15. Class Diagram of Mutation

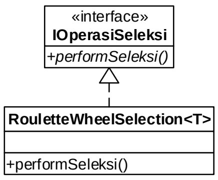

Figure 16. Class Diagram of *RouletteWheel*

Algorithm

The **GASimple** class implements a simple genetic algorithm without any overlapping of population. The**GASimple** class is composed into Algorithm Namespace as shown in Figure 17. The overall stage process in the GA to be done by executing the GA solution functions that has been defined in the class.

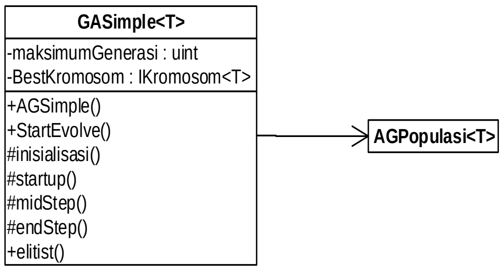

Figure 17. GASimple Class Diagram

Statistic Observer

The AGObserver class is designed to give a summary about the best gens (chromosomes) and fitness, number iteration required, mutation and crossovers probabilities, and population statistics. The design is given in Fig. 18. This object provides the information about chromosome population statistics such as the best and the worst fitness values achieved, the average fitness, and total fitness in a population. The AGObserver class is grouped into StatisticObserver namespace.

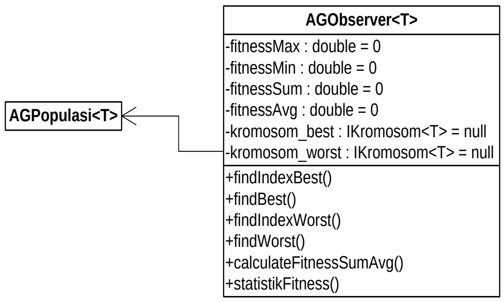

Figure 18. `AGObserver`Class Diagram

Implementation of *Component Library*

The class and namespaces designed to developer the GA library are then wrapped in the component library. The implementation of the library in the form of dynamic link library (*.dll) file is done by using the object-oriented programming language that uses C # programming language.

The fitness function and other interfaces to execute the GA library was done by using the console window/ command-line interfaces as follows:

1. Object instantiationfor Fitness function:
   fungsiFitness fungsiObjektif = newfungsiFitness();
   AGBlokKonfigKrom<double> konfigKromos = newAGBlokKonfigKromo<double>(setNilai, 2, parameter_kromosom, newAGSinglePointCross<double>(), newAGMutasiRandomUniform<double>(), fungsiObjektif);
2. Chromosomeprototype instantiation:
   AGKromosomRealValue<double> prototipe = newAGKromosomRealValue<double>(konfigurasiKromosom,2);

3. Instantiation of Population object, chromosome prototype, and selection:
   AGPopulasi<double> populasi = newAGPopulasi<double>(prototipe, newRouletteWheelSelection<double>(),10);
4. Executing the Simple GA:
   AGSimple<double> algoritma = newAGSimple<double>(ref populasi, 1000);
5. The summary result of the population:
   AGObserver<double> observer = newAGObserver<double>(populasi);
   AGKromoRealValue<double>bestChromosome =(AGKromoRealValue<double>)observer.getBestKromosom();
6. The best chomosome :
   Console.WriteLine("Best Kromosom: x= "+bestChromosome.getAt(0)+"; y= "+bestChromosome.getAt(1)+" nilai fitness = "+bestChromosome.getFitness());

VALIDATION

LibAGen library validation is done by comparing the calculation results with other references that have known the results by using the same data. In the validation test is done by comparing the library LibAGen with the results given by Michalewicz (1999) [15], with a case as follows:

$f(x_1, x_2) = 21{,}5 + x_1 \sin(4\pi x_1) + x_2 \sin(20\pi x_2)$

Constraints: $-3{,}0 \le x_1 \le 12{,}1$ and $4{,}1 \le x_2 \le 5{,}8$

The proposed parameter is given as follows: population size of 10, mutation probability of 5% and crossover probability of 25%. The testing is performed with a different generation number of 10.000, 50.000, and 100.000 generations. Detailed results of each generations is given in Table 2.

With the different generationstested, the best fitness achieved was 388.501 with x_1= 11.6256 and x_2 = 5.7249. Results LibAGen validation is done by comparing the results obtained by Michalewicz (1999) [15] as given in Table 3. The result shows that there is no significant difference in the results LibAGen 0.08% against Michalewicz (1999).

Since the validation case is the maximization problem, based on Table 3 also shows that the best fitness obtained by LibAGen more better than Michalewiczwith the best fitness is about 38,85010.

Table 2. Result of the LibAGen Validation Test

Generation	**The Best Variable Chromosome**		**The Best *Fitness***
	x_1	x_2	
10.000	12,0877	5,7247	38,0096
	11,6039	5,7253	38,4243
	11,6424	5,6239	38,4756
50.000	11,6253	5,7254	38,8491
	11,6256	5,7249	38,8501
	11,6261	5,7247	38,8490
100.000	11,6241	5,7250	38,8485
	11,6241	5,7248	38,8476
	11,6239	5,7251	38,8479

Table 3. Comparison of the Best Fitness Results LibAGen and Michalewicz (1999) [15]

Result Testing	**The Best *Fitness***	**Percentage Comparison with (1)**
Michalewicz	38,818208	-
LibAGen	38,85010	0,08%

TEST CASE

The following test case is given to be solved for maximization problem:

MAX [f(x_1, x_2)= $2x_1 - x_2$], with constraints:

$0 \le x_1 \le 5$; $2 \le x_2 \le 7$;

The solution by using the mathematical approach, theobtained solution parameter are $x_1 = 5$ and $x_2 = 2$; with the maximum objective function is f(5,2) =8. On the other hand, the solutions with the LibAGen library, the maximum function can be obtained as follows: x1 = 4.986 and x2 = 2.033; with the best fitness is f(4.986, 2,033) = 7.938.

The difference between themathematicalcalculations and LibAGen library solution is

$$\frac{8 - 7.938}{8} \times 100\% = 0.77\%.$$

The difference is very small is about (0.77% < 5%), such that the LibAGen solution is valid. The difference between two approaches is due to the genetic algorithm is the stochastic solution with considering the random approach.

CONCLUSION

Based on the design and testing the conclusion can be drawn as follows:

Library genetic algorithm consists of interfaces and classes that contain attributes and methods. Classes and interfaces are grouped into namespace in accordance with their respective functions. There are seven main namespace to establish the GA library are: Algorithm, Chromosome, Crom.OperCross, Crom.OperMutation, Crom.Representation, General, Population,Pop.Selection, and StatisticObserver. These namespace are designed by using object-oriented design (OOD) such as inheritance, composition, and generalization.

Validation library LibAGen comparing the calculation results with the calculated standard [15] obtained the fitness difference of 0.08%. On the other hand, another case study has also been tested. Based on the result, there is no significance difference between mathematical and LibAGen library solution. The indication that the LibAGen can solve the problem in accordance with a predetermined function and correct calculation.

In addition, based on the validation test shown that the calculation results with GA structures will be consistent with the increasing number of generations.

REFERENCES

[1] E.I. Amoiralis, M.A. Tsili, A.G. Kladas, "Global transformer design optimization using deterministic and non-deterministicalgorithms," *in Proceedings of International Conference on Electrical Machines (ICEM) XXth*, pp. 2323 – 2331, 2012.

[2] S. Mitra, S.K. Pal, P. Mitra, , "Data mining in soft computing framework: a survey," *IEEE Transactions on Neural Networks*, vol. 13, no. 1, pp. 3-14, 2002.

[3] Chun-Hao Chen, V.S. Tseng, T.P. Hong, "Cluster-Based Evaluation in Fuzzy-Genetic Data Mining," *IEEE Transactions on Fuzzy Systems*, vol.16, no. 1, pp. 249–262, 2008.

[4] M.W. Mustafa, M.H. Sulaiman, H. Shareef, S.N.A. Khalid, "Reactive power tracing in pool-based power system utilising the hybrid genetic algorithm and least squares support vector Machines," *IET Generation, Transmission & Distribution*, vol. 6, no. 2, pp. 133–141, 2012.

[5] S. Gerbex, R. Cherkaoui, A.J. Germond, "Optimal location of multi-type FACTS devices in a power system by means of genetic algorithms," *IEEE Transactions on Power Systems*, vol.16 , no.3, pp. 537-544, 2001.

[6] M. Reformat, E. Kuffel, D. Woodford, W. Pedrycz, "Application of genetic algorithms for control design in power Systems," *IEE Proceedings-Generation, Transmission and Distribution*, vol. 145 , Issue: 4, 1998 , Page(s): 345 – 354

[7] S.N. Sivanandam and S.N. Deepa, Introduction to Genetic Algorithms, New York : Springer-Verlag Berlin Heidelberg, 2008.

[8] Chambers, Lance, The Practical Handbook of Genetic Algorithms Applications, Second Edition. Boca Raton: Chapman & Hall / CRC, 2001.

[9] A. Ahmad, M. Talha, "A measurement based comparative evaluation of effectiveness of object-oriented versus conventional procedural programming techniques and languages," *in Proceedings of Software Engineering Conference*, Ninth Asia-Pacific, pp. 517 – 526, 2002.

[10] B. Stroustrup, "What is object-oriented programming?" *IEEE Software*, vol. 5, no. 3, pp. 10-20, 1988.

[11] I. Sommerville, Software Engineering, Eighth Edition. Harlow: Addison-Wesley, 2004.

[12] D. Gorter, “1997: The Year of the Component”, *In Proceedings of Conference WESCON/97*, pp. 320-3224-6 Nov.1997.

[13] K.M. Nor, H. Mokhlis, H. Suyono, M. Abdel-Akher, A.H.A. Rashid, T.A. Gani, “Development of Power System Analysis Software Using Object Components,” *In Proceedings of TENCON 2005, IEEE Region* 10, pp. 1-6, 2005.

[14] H. Suyono, K.M. Nor, S. Yusof, “Transient Stability Program Using Component-Based Software Engineering, “ *IEEE Region 10 TENCON 2005*, pp. 1 – 6, 2005.

[15] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag Berlin Heidelberg New York, ISBN 3-540-60676-9, 1999.