Design and Analysis of Improvised Genetic Algorithm with Particle Swarm Optimization for Code Smell Detection

James Benedict Felix S, Viji Vinod

Abstract: ***Software development phase is very important in the Software Development Life Cycle. Software maintenance is a difficult process if code smells exist in the code. The poor design of code development is called code smells. The code smells are identified by various tools using various approaches. Many code smell approaches are rule based. The rule based approaches are based on trial and error method. Genetic Algorithm is a heuristic Algorithm by Darwin's Theory. This paper presents a metric based code smell detection approach by Genetic Algorithm with particle swarm optimization based on Euclidean data distance. The Euclidean data distance gives best proximity value between two points. Our approach is evaluated on the three open source projects like JFreeChart v1.0.9, Log4J v1.2.1 and Xerces-J for identifying the eight types of code smells namely Functional Decomposition, Feature Envy, Blob, Long Parameter List, Spaghetti Code, Data Class, Lazy Class, Shotgun Surgery.***

Keywords: Code smell, Genetic Algorithm, Improvised Genetic Algorithm,

I. INTRODUCTION

The quality of the software is improved by well designed of code. Detecting code smell is a very difficult by manually. The objective of our work is detecting code smell accurately using advanced
Genetic Algorithm based on Euclidean distance. For this approach, we evaluated code metrics from three open source project. Two of the projects are called initial example and is called base example. Metrics are calculated from class of initial model and class of base example. Finally this approach compared with Parallel Evolutionary Algorithm, Genetic Algorithm and Random Search approach. Section 2 describes, code smell detection techniques. Section 3 explains analysis and design of genetic algorithm key points. Section 4 shows the explanation of Improvised Genetic Algorithm. At last, future work and conclusion is discussed in section 5.

II. A BACKGROUND OF RELATED WORK

In this section describes, background for finding code smell and code smell detection techniques.

Revised Manuscript Received on November 08, 2019.

James Benedict Felix S, Research Scholar, Bharathiar University, Coimbatore, India. Email: x.brothers@live.com

Viji Vinod, Professor & Head, Department of Computer Applications, Dr.M.G.R. Educational and Research Institute, University, Chennai, India. Email: vijivino@gmail.com

A. Code Smell

Code smell is nothing but the bad design of code. Fowler et al. defined nearly 22 type of code smells. In our work, eight types of code smells are focusing. They are,

Table- I: Example of Code Smells

S.No	Code Smell	Description
1	Functional Decomposition	It is a large or complex function in a class.
2	Feature Envy	Feature Envy means methods called many times accessor methods of another class.
3	Blob	The larger blob class monopolizes the behavior of the system.
4	Long parameter list	A method comes with several algorithms parameter.
5	Spaghetti Code	Spaghetti code means complex code comes with small software design.
6	Data class	A data class comes with all attributes and without behavior.
7	Lazy class	It means a class does not do enough actions.
8	Shotgun Surgery	Need small changes in many different classes.

B. Code smell detection technique

Code smell detection techniques can be classified into their level of automation and the number of detected code smells. The most common code smell detection techniques are

i) Manual detection techniques

The early code smell detection technique is manually inspects the code to identify the code smells. This technique improves software quality and software development process. Software reading technique [1] has been implemented in this method to detect the code smell. The time consuming for detection of code smell is very high.

ii) Automatic detection techniques

The Automatic code smell detection technique is proposed by Moonen and Eva Van Emden. The major advantage of this technique is the bad smell detection time consuming is less. The automatic detection tools are recommended refactoring suggestions. Several approaches are implemented in automatic code smell detection. For example, PMD is an Eclipse plug-in tool, which is find three kind of code smells and JDeodorant plug-in can find five types of code smells.

iii) Metric based detection techniques

The Metric based detection technique is introduced by Marinescu [2]. The code smells are identified manually based on software metrics. The characteristics of code smell described by software metrics. In generally metrics are applying in software maintenance and testing etc. Metric based detection technique finding the bad

smells in accurate threshold values. The object oriented software metrics measurements are size of the class, number of attributes and methods in class etc. It is also used to detecting the code smell similarities between the software systems.

III. ANALYSIS AND DESIGN OF GENETIC ALGORITHM

A. Genetic Algorithm

The Genetic Algorithms [3] can be applied in many research areas. The GAs is generally used in search techniques based on genetic and evolution methodology. There are six important phases in Genetic Algorithm.

- Individual

An individual is defined by set of parameters. It consists of three main parts. It has class of initial model, class of base example and code smell which is detected from base example.

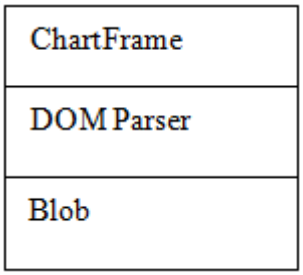

Fig1. Individual

In the above figure, ChartFrame is a class which is derived from JFreeChart open source project and DOM Parser is a class which is derived from Xerces-J software system. Blob is a code smell which is detected from base example.

- Initial population

Set of individuals is called initial population. Figure 2 is an example of population.

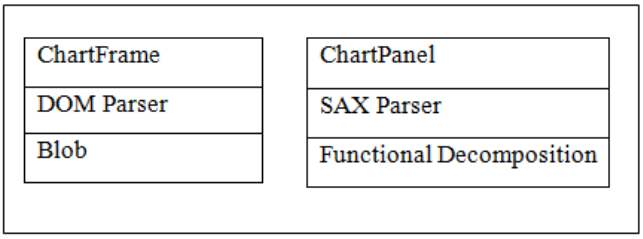

Fig2. Initial populations

- Fitness Function

Fitness function determines ability of an individual to compete with other individuals. It calculates fitness score to individuals and gives quality of individuals. It performs the similarity between initial model and base model.

- Selection

The selection phase selects the best pair of individuals (Parents) from population. In each iteration the GA selects individuals based on fitness value.

- Crossover

Crossover phase is significant one in genetic algorithm. For each pair of individuals to be mated, a crossover point is chosen randomly.

- Mutation

In mutation phase, randomly replace a variable or delete a variable to the genotype. User can change the parameter of the Genetic Algorithm.

```
Step 1: START
Step 2: Generate the initial population
Step 3: Compute fitness
Step 4: REPEAT
        Select Individuals
        Crossover the individuals
        Mutation
        Compute fitness
    UNTIL population has converged
Step 5: STOP
```

B. Software Metrics

Software metrics gives useful information to access the software and it can be used to the metrics similarities between the software systems. In this work, eight metrics are considered to implement the Advanced Genetic Algorithm. The table 2 describes software metrics. All the attributes are calculated in a single class which is used to in the Improvised GA.

Table-II: Software Metrics

S.No	*Software Metrics*	*Description*
1	NPrA	Number of Private Attributes
2	NPbA	Number of Public Attributes
3	NProA	Number of Protected Attributes
4	NPrM	Number of Private Methods
5	NPbM	Number of Public Methods
6	NProM	Number of Protected Methods
7	NoA	Number of Associations
8	NoG	Number of Generalization Relationships

IV. IMPROVISED GENETIC ALGORITHM

A. Genetic Algorithm with Particle Swarm Optimization

The Particle Swarm Optimization method is used to solve population based global optimization problems. Each particle is considered as Individual and group of particle is considered as swarm. The PSO algorithm [5] selects a particle according to the selection process. In each iteration, particle moves according to velocity. The position of particle after the velocity is calculated by the below formulas.

$$x_i(t+1) \leftarrow x_i(t) + v_i \text{ ---------- Eq (1)}$$
$$v_i(t+1) \leftarrow \omega v_i(t) + c_1 r_1(pbest_i(t) - x_i(t) + c_2 r_2(gbest_i(t) - x_i(t) \text{ ------- Eq (2)}$$

$x_i(t)$ and $x_i(t+1)$ means places of particles in time t and t+1. pbset(t) means best personal position of the particle and gbest(t) means best global position of the particle. r_1 and r_2 is random variables. c_1 and c_2 are acceleration variables. ω means the inertia weight of the particle.

$$(pbest_i(t+1) = \begin{cases} (pbest_i(t) & \text{if } f(pbest_i(t)) <= f(x_i(t+1)) \\ x_i(t+1) & \text{if } f(pbest_i(t)) > f(x_i(t+1)) \end{cases} \quad \text{------ Eq (3)}$$

$$gbest(t+1) = \min\{ f(y), f(gbest(t))\} \text{ where } y \in \{ pbest_0(t), pbest_0(t), \ldots pbest_s(t)\} \text{---Eq (4)}$$

The equations (3) and (4) determine how the pbest and gbest values are updated in time t.

```
Step 1: Initialize particle

Step 2: Do
          For each particle
            Calculate fitness value
              If the fitness value is better than the best
fitness value (pbest)
                    set current value as the new pbest
          End

   Choose the particle with the best fitness value of all the
particles as the gbest
   For each particle
      Calculate particle velocity
      Update particle position
   End
```

Fig 4. The PSO Algorithm

B. Euclidean Distance

Euclidean distance is used to compute the distance between two points. After getting pbest and gbest, Euclidean distance is calculated between pbest and gbest values.

$$\text{Euclidean distance } (j,i) = \alpha . f(p_j) - f(p_i) / \|p_j - p_i\| \text{ ------Eq.(5)}$$

$$\alpha = \|s\| / f(p_g) - f(p_w) \text{ ----------Eq.(6)}$$

$$S = \sqrt{\sum_{i=1}^{p} (x_{i_1} - x_{i_2})^2} \text{ ---------- Eq.(7)}$$

Fig.5 Euclidean distance equations

In equation (5) $f(p_j)$ means the pbest value of j^{th} particle and $f(p_i)$ means the pbest value of i^{th} particle.

α – scaling factor

$f(p_g)$ – global best of the particle

$f(p_w)$ – global worst of the particle

S – Size of the search space

p – size of the swarm

x_1 and x_2 - position of the particle.

Based on the Euclidean distance, the gbest are selected for single pair crossover and mutation. After mutation process, new offsprings will get. This process is repeated until the solution will get.

C.Calculate Fitness function

Table – III: Metrics values from initial model and base model

S.No	Classes	NPrA	NPbA	NProA	NPrM	NPbM	NProM	NoA	NoG
1	ChartFrame	8	10	5	8	24	7	0	1
2	DOM Parser	6	0	45	0	8	7	0	0

The similarity value is checked between initial model (IM) and base example (BE). Compute the similarity between initial and base model using the below formula.

$$\text{Similarity (IM, BE)} = 1/m \sum_{i=1}^{m} \text{Sim}(IM_i, BE_i) \text{ ---------- (8)}$$

$$\text{Sim}(IM_i, BE_i) = \begin{cases} 1 & \text{if } (IM_i = BE_i) \\ 0 & \text{if } (IM_i = 0 \text{ and } BE_i \neq 0) \text{ OR } (IM_i \neq 0 \text{ and } BE_i = 0) \\ IM_i / BE_i & \text{if } IM_i < BE_i \\ BE_i / IM_i & \text{if } BEi < IM_i \end{cases} \text{ ---------- (9)}$$

V. REVIEW CRITERIA

The fitness function is calculated from the equation 10.

Fitness function =f1 + f2 /2 ---------- (10)

$f1 = 1/n \sum_{j=1}^{n} \text{similarity } (IM_i, BE_i)$ --- (11)

f1=1/8[(6/8+0+5/45+0+8/24+1+1+0)]

=1/8[(3.194)]

f1 =0.3993

f2=Individual size / Maximum Individual Size

---------- (12)

So, Fitness function= (0.3993 + 1) / 2

=0.69

D. ILLUSTRATION OF CROSSOVER

Two parent individual (blocks) are selected for crossover based on minimum Euclidean distance particles.

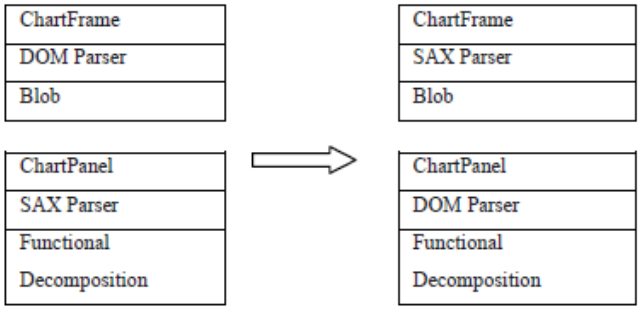

Fig.6 Crossover

Two individuals are selected for crossover. Mainly two base

examples parameters interchanged to new child.

E. Illustration of Mutation

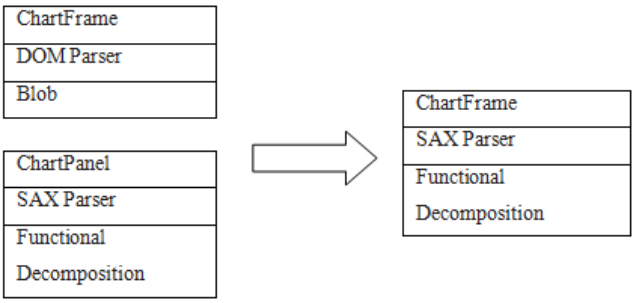

Fig.7 Mutation

Here random mutation operation is implemented. In fig.6, DOM Parser of defect code smell Blob is replaced with the SAX Parser (base example) of defect Functional Decomposition.

F. Improvised Genetic Algorithm

```
Ind: = set of (IM, BE, Code smells in Base example) /* Ind
represent Individual */
P: =set of (Ind) /* P represents particle*/
Initial_population (P, Max_Size)
repeat
   for each Ind in P do    /* Compute fitness */
      Calculate fitness f1
      Calculate fitness f2
      fitness_function (Ind): = (f1+f2)/2
   end

   for all Ind  in P do /*finds the best-solution*/
      best_fit_function (Ind):= min (fitness_function (Ind))
         best_solution:= best_fit_function (Ind)
    end for
         while (1)
            Update velocity of the individuals
            Move the individuals
            Calculate the Euclidean distance
            Perform crossover
            Perform random mutation operation
          end
          P := generate_new_ population (P) /*new
      population generation*/
      it:= it + 1
until it:= max_it
    return best_solution
```

Fig.8 Improvised Genetic Algorithm

Set of quality metrics and code smell from base example is an input and code smell from initial model is an output of this algorithm.

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed a new algorithm which is based on particle swarm optimization. Comparing with genetic algorithm, this algorithm detects code smell accurately. The main advantage of this algorithm is we can find code smell using set of metrics in open source project. Eight kind of code smells using this work in three open source projects. We are focusing detection of code smells only. In future, automatically correct code smells using refactoring.

REFERENCES

1. G. Travassos, F. Shull, M. Fredericks, and V. R. Basili, "Detecting defects in object-oriented designs: using reading techniques to increase software quality," in ACM Sigplan Notices, 1999, pp. 47-56.
2. R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," in Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM'04), 2004, pp. 350-359.
3. Ahmed A. A. Esmin and Stan Matwin, "A Hybrid Particle Swarm Optimization Algorithm with genetic mutation", International Journal of Innovative Computing, Information and Control, 2013, pp, 1919-1934
4. N.E. Fenton, S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS Publishing Co., Boston, MA, 1998.
5. D.E. Goldberg, Genetic Algorithms in Search, Addison-Wesley, Optimization and Machine Learning, 1989.
6. G.Saranya, "Model level code smell detection using EGAPSO based on similarity measures", 2017.

AUTHORS PROFILE

JAMES BENEDICT FELIX S is working as an Assistant Professor & Head in the Department of Computer Science in Mar Gregorios College, Chennai, India. He has completed his B.Sc, Computer Science Degree from St.Xavier's College, Tirunelveli, India in 2004 and finished Masters Degree in Computer Science from St.Joseph's College, Trichy, India in 2006. He is pursuing research in Bharathiar University, India. He has published 8 articles in reputed journals and conferences. He has published one E-Book in the title of "Problem Solving through Programming" for DBE Education & HR Consultant Private Limited, Chennai. He has 11 years of teaching experience in colleges.

Born on the mainland of India in1975, Professor **VIJI** received her primary and secondary schooling at N.S.S arts and science co-educational college, India. She furthered her studies, graduating with a B.Sc (with Great Distinction) in Mathematics from Calicut University in 1996. **Dr.Viji Vinod** started her Masters in Computer Applications from Bharathiar University in 1999.

Dr.Viji Vinod started her career in working on real time software engineering. She joined as a software engineer at Pentafour communications ltd, Chennai, India in 1999 where she achieved in developing enterprise resource planning applications in client/server architecture. She has worked as a software developer with Lawrence and Associates ltd, a US based Software Company where she developed various web based applications. She had the passion of doing research and reach great height in academics, influenced her decision to join as the faculty of the Department of Information Technology at Dr.MGR Educational & Research Institute, University. She was the first professor in object oriented analysis and design and component based technology at Dr.MGR University in 2004. From 2004 to 2005, she served as an Assistant Professor of Department of Information Technology, from 2006 to 2010, as Assistant Professor of the Department of Computer Applications, and from 2010 to till now, as Professor& Head of Department of Computer Applications, Dr.MGR Educational & Research Institute, University.

Professor Viji specializes in areas of Information Technology Management, Web Technology, and various software engineering issues with the goal of improving both software engineering practices and processes. She has edited one book and published articles and notes in professional journal and conferences.