Volume 6, Issue 9, September 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Model of System Software Components Using Genetic Algorithm and Techniques

Ramu Vankudoth*
Department of Computer Science
Kakatiya University, Warangal,
Telangana, India

Dr. P. Shireesha
Department of CSE, KITS
Kakatiya University, Warangal,
Telangana, India

T. Rajani Devi
Department of Computer Science
Kakatiya University, Warangal,
Telangana, India

Abstract— The selection of system software component is an important decision of design stage, and has a significant impact on various system quality attributes. To determine system software component based on architectural style selection, the software functionalities have to be distributed among the components of software component. In this paper, a method based on the Genetic Algorithm of use cases the concept and design procedure of Genetic Algorithm as techniques is proposed to identify software components and their responsibilities. To select a proper Genetic Algorithm method, first the proposed method is performed on a number of software systems using different Genetic Algorithm methods, and the results are verified by expert opinion, and the best method is recommended. By sensitivity analysis, the effect of features on accuracy of Genetic Algorithm is evaluated. Finally, to determine the appropriate number of Genetic Algorithm (i.e. the number of software components), metrics of the interior cohesion of Genetic Algorithm and the coupling among them are used.

Keywords— Software components identification, Genetic Algorithms, selection, crossover and mutation

I. INTRODUCTION

Software architecture is a fundamental artifact in the software life cycle with an essential role in supporting quality attributes of the final software product. Making use of architecture styles is one of the ways to design software systems and guarantee the satisfaction of their quality attributes [1]. After architectural style selection, only type of software architecture organization is specified. Then software components and their responsibilities need to be identified. On the other hand, component-based development (CBD) is nowadays an effective solution for the subject of development and maintenance of information systems [2]. A component is a basic block that can be designed and if necessary be combined with other components [3]. Partitioning software system to components, while effective on later software development stages, has a central role in defining the system architecture. Component identification is one of the most difficult tasks in the software development process [4]. Indeed a few systematic components identification methods have been presented, and there are no automatic tools to help experts for identifying the components, and components identification is usually made based on expert experience.

Starting with a randomly generated population of chromosomes, a GA carries out a process of fitness based selection and recombination to produce a successor population, the next generation. During recombination, parent chromosomes are selected and their genetic material is recombined to produce child chromosomes. These then pass into the successor population. As this process is iterated, a sequence of successive generations evolves and the average fitness of the chromosomes tends to increase until some stopping criterion is reached. In this way, a GA "evolves" a best solution to a given problem. GAs was first proposed by John Holland [7] as a means to find good solutions to problems that were otherwise computationally intractable. Holland's Schema Theorem [7], and the related building block hypothesis [5], provided a theoretical and conceptual basis for the design of efficient GAs. It also proved straightforward to implement GAs due to their highly modular nature. As a consequence, the field grew quickly and the technique was successfully applied to a wide range of practical problems in science, engineering and industry. GA theory is an active and growing area, with a range of approaches being used to describe and explain phenomena not anticipated by earlier theory [8]. In tandem with this, more sophisticated approaches to directing the evolution of a GA population are aimed at improving performance on classes of problem known to be difficult for GAs [9, 6, 10].

II. A GENETIC ALGORITHM

Genetic algorithms are adaptive heuristic search algorithm premised on the Darwin's evolutionary ideas of natural selection and genetic. The basic concept of genetic algorithms is designed to simulate processes in natural system necessary for evolution. As such they represent an intelligent exploitation of a random search within a defined search space to solve a problem. First pioneered by John Holland in the 60's, GAs has been widely studied, experimented and applied in many fields in engineering world. Not only does genetic algorithm provide an alternative method to solving problem, it consistently outperforms other traditional methods in most of the problems link. Many of the real world problems which involve finding optimal parameters might prove difficult for traditional methods but are ideal for genetic

algorithms. This paper starts with the description of various GA operators in section 2. Section 3 gives the outline of the genetic algorithm. In section 4, we introduce global optimization and discuss how genetic algorithm can be used to achieve global optimization and illustrate the concept with the help of Rastrigin's function. In section 5, we explore the reasons why GA is a good optimization tool. The discussion ends with a conclusion and future trend.

A. Optimization problems

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. [11]

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

A typical genetic algorithm requires:

1. A genetic representation of the solution domain,
2. A fitness function to evaluate the solution domain.

A standard representation of each candidate solution is as an array of bits.[11] Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming.

Initialization

The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Often, the initial population is generated randomly, allowing the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

Selection

During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as the former process may be very time-consuming.

The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The fitness of the solution is the sum of values of all objects in the knapsack if the representation is valid or 0 otherwise.

In some problems, it is hard or even impossible to define the fitness expression; in these cases, a simulation may be used to determine the fitness function value of a phenotype (e.g. computational fluid dynamics is used to determine the air resistance of a vehicle whose shape is encoded as the phenotype), or even interactive genetic algorithms are used.

Genetic operators

The next step is to generate a second generation population of solutions from those selected through a combination of genetic operators: crossover (also called recombination), and mutation. For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. Although reproduction methods that are based on the use of two parents are more some research [12] [13] suggests that more than two "parents" generate higher quality chromosomes.

These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions. These less fit solutions ensure genetic diversity within the genetic pool of the parents and therefore ensure the genetic diversity of the subsequent generation of children.

Termination

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above

B. Genetic algorithm Limitations

There are limitations of the use of a genetic algorithm compared to alternative optimization algorithms:

- Repeated fitness function evaluation for complex problems are often the most prohibitive and limiting segment of artificial evolutionary algorithms. Finding the optimal solution to complex high dimensional, multimodal problems often requires very expensive fitness function evaluations
- Genetic algorithms do not scale well with complexity. That is, where the number of elements which are exposed to mutation is large there is often an exponential increase in search space size. This makes it extremely difficult to use the technique on problems such as designing an engine, a house or plane.
- The "better" solution is only in comparison to other solutions. As a result, the stop criterion is not clear in every problem.
- In many problems, GAs may have a tendency to converge towards local optima or even arbitrary points rather than the global optimum of the problem. This means that it does not "know how" to sacrifice short term fitness to gain longer-term fitness. The likelihood of this occurring depends on the shape of the fitness landscape: certain problems may provide an easy ascent towards a global optimum; others may make it easier for the function to find the local optima. This problem may be alleviated by using a different fitness function, increasing the rate of mutation, or by using selection techniques that maintain a diverse population of solutions, [14] although the No Free Lunch theorem [15] proves that there is no general solution to this problem.
- Operating on dynamic data sets is difficult, as genomes begin to converge early on towards solutions which may no longer be valid for later data. Several methods have been proposed to remedy this by increasing genetic diversity somehow and preventing early convergence, either by increasing the probability of mutation when the solution quality drops (called triggered hypermutation), or by occasionally introducing entirely new, randomly generated elements into the gene pool (called random immigrants).
- For specific optimization problems and problem instances, other optimization algorithms may be more efficient than genetic algorithms in terms of speed of convergence.

III. WORKING PRINCIPLE OF GENETIC ALGORITHMS (GAS)

Genetic algorithms (GAs) may contain a chromosome, a gene, set of population, fitness, fitness function, breeding, mutation and selection. Genetic algorithms (GAs) begin with a set of solutions represented by chromosomes, called population. Solutions from one population are taken and used to form a new population, which is motivated by the possibility that the new population will be better than the old one. Further, solutions are selected according to their fitness to form new solutions, that is, off springs. The above process is repeated until some condition is satisfied. Algorithmically, the basic genetic algorithm (GAs) is outlined as below:

- Step 1: Represent the problem variable domain as a chromosome of a fixed length; choose the size of a chromosome population N, the crossover probability pc and the mutation probability pm.
- Step 2: Define a fitness function to measure the performance, or fitness, of an individual chromosome in the problem domain. The fitness function establishes the basis for selecting chromosomes that will be mated during reproduction.
- Step 3: Randomly generate an initial population of chromosomes of size N: x1, x2 , . . . , xN
- Step 4: Calculate the fitness of each individual chromosome: f (x1), f (x2), . . . , f (xN)
- Step 5: Select a pair of chromosomes for mating from the current population. Parent chromosomes are selected with a probability related to their fitness.
- Step 6: Create a pair of offspring chromosomes by applying the genetic operators - crossover and mutation.
- Step 7: Place the created offspring chromosomes in the new population.
- Step 8: Repeat Step 5 until the size of the new chromosome population becomes equal to the size of the initial population, N.
- Step 9: Replace the initial (parent) chromosome population with the new (offspring) population.
- Step 10: Go to Step 4, and repeat the process until the termination criterion is satisfied.
- The genetic algorithms performance is largely influenced by crossover and mutation operators. The block diagram representation of genetic algorithms (GAs) is shown in Fig.1

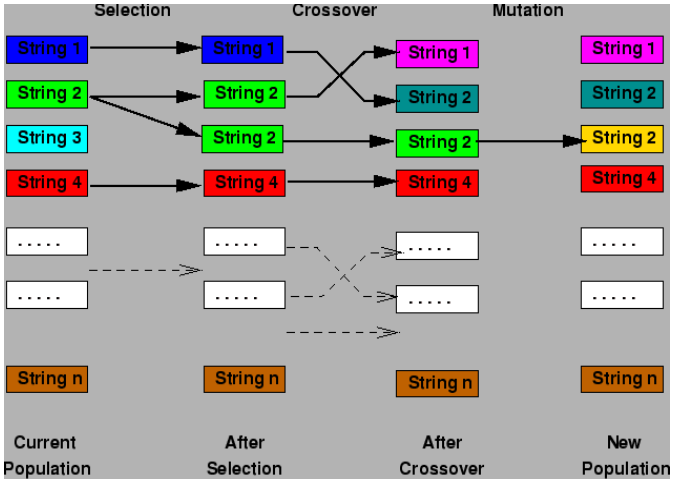


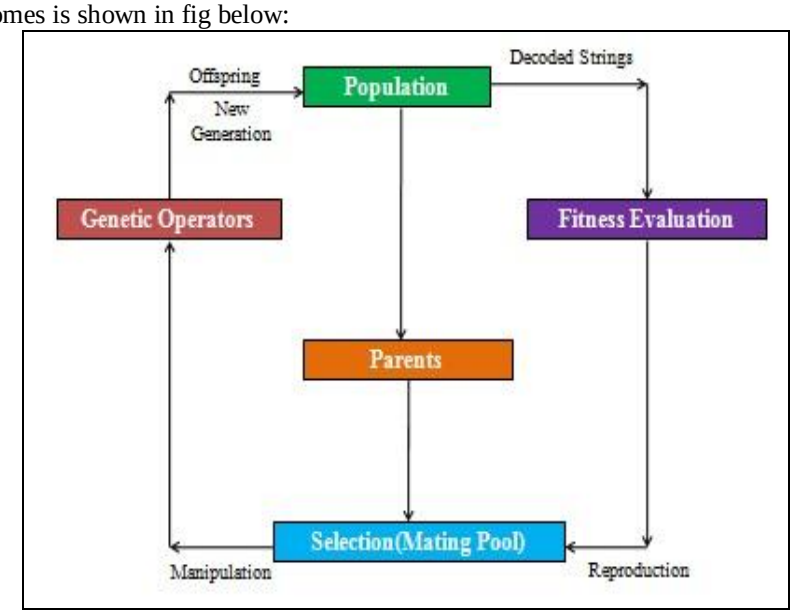
Figure.1 Representation of genetic algorithms (GAs)

A. Evaluation of the Fitness

Each string in initial population or subsequent population is assigned a fitness value which is related to the objective function. For maximizing a function the fitness can be equal to string's objective function value. To find the optimal minimum the fitness will be equal to 1/(1+f(x)). The beauty of the binary coding is shielding between actual problem and the working of GA. The GA processes only the strings of bits which may represent any number of variables depending on the problem. We have to change only the definition of the coding.

B. Reproduction and Selection

Reproduction selects good strings from the population and puts them in mating pool. There are number of reproduction operators. The idea is to pick up the strings with above average fitness from current population and apply genetic operators to new strings for the successive population. One of the important techniques is the fitness proportionate selection. The chances of a string Si being selected to participate in reproduction are proportional to its fitness. This is performed by roulette wheel selection .Here; all the chromosomes are placed on an imaginary roulette wheel where each chromosome in the population gets a place big on the wheel proportionate to its fitness. A roulette wheel for five chromosomes is shown in fig below:



IV. SELECTION TECHNIQUES IN GENETIC ALGORITHMS (GAS)

Selection is an important function in genetic algorithms (GAs), based on an evaluation criterion that returns a measurement of worth for any chromosome in the context of the problem. It is the stage of genetic algorithm in which individual genomes are chosen from the string of chromosomes. The commonly used techniques for selection of chromosomes are Roulette wheel, rank selection and steady state selection.

A. Roulette wheel selection

In this method the parents are selected according to their fitness. Better chromosomes, are having more chances to be selected as parents. It is the most common method for implementing fitness proportionate selection. Each individual is assigned a slice of circular Roulette wheel, and the size of slice is proportional to the individual fitness of chromosomes, that is, bigger the value, larger the size of slice is. The functioning of Roulette wheel algorithm is described below:

Step 1[Sum] Find the sum of all chromosomes fitness in the population.

Step 2[Select] Generate random number from the given population interval

Step 3[Loop] Go through the entire population and sum the fitness. When this sum is more than a fitness criteria value, stop and return this chromosome figure 3 (a) shows Roulette wheel for six individuals having different fitness values. The Sixth individual has a higher fitness than any other, it is expected that the Roulette wheel selection will choose the sixth individual more than any other individual.

Figure 3 (a) Roulette wheel method

B. Rank selection method

The application of Roulette wheel selection method is not satisfactory in genetic algorithms (GAs), when the fitness value of chromosomes differs very much. It is a slower convergence technique, which ranks the population by certain criteria and then every chromosome receives fitness value determined by this ranking. This method prevents quick convergence and the individuals in a population are ranked according to the fitness and the expected value of each individual depends on its rank rather than its absolute fitness.

The rank selection method is shown in Figure 3 (b). For example, if the best chromosome fitness is 80 percent, its circumference occupies 80 percent of the roulette wheel and then other chromosomes will have minimum chances to be selected. On the other hand, the rank selection first ranks the population according to their fitness and then every chromosome receives ranking. The worst will have fitness 1, the second worst will have a fitness of 2, and the best one will have a fitness value n, where n is the number of chromosomes in the population.

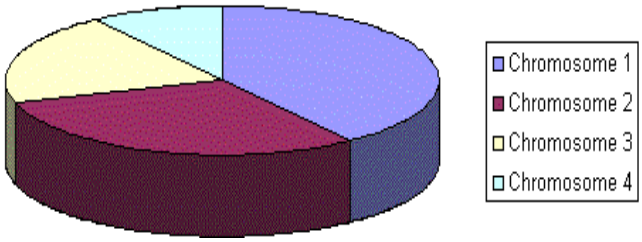


Figure 3(b). Rank selection methods

C. Steady-state selection

This method replaces few individuals in each generation, and is not a particular method for selecting the parents. Only a small number of newly created off springs are put in place of least fit individual. The main idea of steady-state selection is that bigger part of chromosome should retain to successive population.

V. CONCLUSIONS

In this paper, a method was proposed to automatically determine system software components based on genetic algorithm of use cases features. First, the system use cases features were extracted and the components were determined based on the proposed method using different genetic algorithms methods. Then, the appropriate genetic algorithm method was selected by comparison of genetic algorithm techniques results with expert opinion. To determine the appropriate number of GA's of the analysis, the effect of each feature on accuracy of genetic algorithm was determined and finally the closest to optimum set of features providing the required accuracy in genetic algorithm. Genetic algorithms are very effective techniques of quickly finding a reasonable solution to a complex problem which shows the superiority of Genetic Algorithm than standalone soft computing controllers and conventional controllers. In future work, these techniques may be implemented for other process controllers.

REFERENCES

[1] M. Shaw، and D. Garlan. Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall، 1996

[2] L. Peng, Z. Tong, and Y. Zhang, "Design of Business Component Identification Method with Graph Segmentation", 3rd Int. Conf. on Intelligent System and Knowledge Engineering, pp. 296-301, 2008

[3] R. Wu,"Componentization and Semantic Mediation", 33th Annual Conf. of the IEEE Industrial Electronics Society, Taiwan, pp. 111-116, 2007

[4] M. Fan-Chao, Z. Den-Chen, and X. Xiao-Fei, "Business Component Identification of Enterprise Information System: A hierarchical Genetic Algorithm method", Proc. Of the 2005 IEEE Int. Conf. on e-Business Engineering, pp. 473-480, 2005

[5] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[6] G.R. Harik, F.G. Lobo, D.E. Goldberg, The compact genetic algorithm, in: D.B. Fogel (Ed.), Proceedings of the IEEE Conference on Evolutionary Computation 1998 (ICEG'98) IEEE Service Centre, Piscataway, NJ, 1998, pp. 523–528.

[7] J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, MI, 1975.

[8] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, Cambridge, MA, 1998.

[9] H. Mühlenbein, G. Paaß, From recombination of genes to the estimation of distributions I. Binary parameters, Parallel problem solving from nature, in: H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefeleds (Eds.), Lecture Notes in Computer Science, vol. 1141, Springer, Berlin, 1996, pp. 178–187.

[10] M. Pelikan, D.E. Goldberg, F.G. Lobo, A Survey of Optimization by Building and using Probabilistic Models, University of Illinois Genetic Algorithms Laboratory, Urbana, IL, IlliGAL Report No. 99018, 1999.

[11] Whitley 1994, p. 66.

[12] Eiben, A. E. et al (1994). "Genetic algorithms with multiparent recombination". PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: 78–87. ISBN 3-540-58484-6

[13] Ting, Chuan-Kang (2005). "On the Mean Convergence Time of Multi-parent Genetic Algorithms Without Selection". Advances in Artificial Life: 403–412. ISBN 978- 3-540-28848-0

[14] Taherdangkoo, Mohammad; Paziresh, Mahsa; Yazdi, Mehran; Bagheri, Mohammad Hadi (19 November 2012). "An efficient algorithm for function optimization: modified stem cells algorithm". Central European Journal of Engineering. 3 (1): 36–50. doi:10.2478/s13531-012- 0047-8.

[15] Wolpert, D.H., Macready, W.G., 1995. No Free Lunch Theorems for Optimisation. Santa Fe Institute, SFI-TR-05-010, Santa Fe.

ABOUT AUTHOR

Ramu Vankudoth was born on June 1986 at Khammam, Telagana, India, received B.Sc degree in Computer Science from SSRJ Arts & Science College, Khammam in 2008 and also recived MCA degree from Kakatiya University, Warangal. He is currently doing Full Time Research Scholar (Ph.D), Department of Computer Science, Kakatiaya University, Warangal. He has presented one Paper Pakistan Journal of Biotechnology and one Paper IEEE conferences published. His area of interests is Software reusable Components, Cloud Computing, Genetic Algorithm, Retrieval Components and Software Engineering.

Shireesha Pakala received the M.Sc.Computer Science from Kakatiya University in 2001 and awarded her Ph.D. (Computer Science) from Kakatiya University, Warangal in 2012. She is working as Assistant Professor in the Department of MCA in KITS, Warangal, since 2006.She published 6 papers in International Journals and 1 paper in International Conference. She is the member of the ISTE.

