International Journal of Digital Information and Wireless Communications (IJDIWC) 4(1): 124-142
The Society of Digital Information and Wireless Communications, 2014 (ISSN: 2225-658X)

A Genetic Algorithm Analysis towards Optimization solutions

Mujahid Tabassum and Kuruvilla Mathew
Swinburne University of Technology, Sarawak Campus, Jalan Simpang Tiga, 93350, Kuching, Sarawak, Malaysia
mtabassum@swinburne.edu.my and kmathew@swinburne.edu.my

ABSTRACT

In today's world, an optimal and intelligent problem solving approaches are required in every field, regardless of simple or complex problems. Researches and developers are trying to make machines and software's more efficient and intelligent. This is where the Artificial Intelligence plays its role in developing efficient and optimal searching algorithm solutions. Genetic algorithm is one of most pervasive and advanced developed heuristic search technique in AI. Genetic algorithm (GA) is developed to find the most optimized solution for a given problem based on inheritance, mutation, selection and some other techniques. It was proved that genetic algorithms are the most powerful unbiased optimization techniques for sampling a large solution space. In this paper, we have used GA for the image optimization and Knapsack Problems, which are commonly found in a real world scenario. Furthermore, a research based on a tool that uses Genetic Algorithm, called the GA Playground is done to demonstrate the capability of solving the Knapsack Problem with the fitness function and a case study on how images can be reproduced using the optimal parameters. Lastly, a few methods such as the Hash Table and the Taguchi Method are suggested to improve the performance of the Genetic Algorithm.

KEYWORDS

Artificial Intelligence, Genetic Algorithms; Optimization; GA operators; Metaheuristics; JNetic; GA Playground; Knapsack Problems; Travelling Salesman Problem; Taguchi Method;

1 INTRODUCTION

A genetic algorithm (GA) is an artificial intelligence search metaheuristic that is derived from the process of biological organism evolution. Genetic algorithms were described in University of Michigan 1960s and 1970s by John Holland [1].

Genetic algorithms are a subset of the evolutionary algorithm set. The design of evolutionary algorithms is based on processes which occur in living organisms in nature, for example, inheritance, mutation, selection and crossover.

GA's are uniquely distinguished by having a parallel-population based search with stochastic selection of many individual solutions, stochastic crossover and mutation. Many other search methods have some of these elements, but only GA's have this particular combination [2]. Genetic algorithms are used various fields of biology, biotechnology, computer science, engineering, economics, chemistry, manufacturing, mathematics, medicine and pharmacology and other fields and have numerous advantages and disadvantages.

In the event the reader finds difficulty understanding biological concepts, an appendix relating to the explanation of those concepts has been included. However, the reader is expected to be familiar with main concepts and keywords used in the field of artificial intelligence.

The GA requires a fitness statistics or function since it emulates the concept of solution evolution by stochastically developing generations of solution populations. This algorithm is applicable to search for the solution of high degree of complexity that often involves attributes that are large, non-linear and discrete in nature. However, GA does not guarantee that optimality can be found but the results are usually close to the global optimum. Since the solutions are considered to be of probabilistic nature, they are not contained by local optima [3].

In this paper real world problems are been included to evaluate the application and performance of genetic algorithm. The purpose of these case studies is to find the parametric values of the GA that provides the most optimum solution (maximum generation, optimum population, optimum crossover rate, optimum mutation rate) and helps to understand the effectiveness of this algorithm and the methods we could use to improve the performance if searching for the optimum solution.

2 RELATED WORKS

Genetic algorithm have quite a number of advantages which makes it as one of the most preferable and widespread search algorithms in Artificial Intelligence. One of the advantages is the capability of solving any optimization problem based on chromosome approach, another remarkably important feature of this algorithm is its capability to handle multiple solution search spaces and solve the given problem in such an environment. Moreover, genetic algorithms are less complex and more straightforward compared to other algorithms. In addition Genetic algorithms are easier to be transferred and applied in different platforms, thereby increasing its flexibility. Many algorithms in AI are complete and optimal in finding a solution; however, these algorithms are only efficient for single objective solutions which mean there is only a single criterion for the solution state. Whereas, looking for solutions with multiple objectives are much more complex. In such an environment each solution will have more than one criteria and every change in one of these criteria will directly affect the others. Genetic algorithm has this capability to generate efficient solutions in such a complex and mathematically sophisticated environments. This process is performed by using the described implementation operators such as crossover and mutation where the new solutions will inherit from more than one previous chromosome. As a result, as the search space is expanded down the hierarchy each chromosome will be actually the outcome of merging different genes of the previous most optimal solutions. This will subsequently result in finding the final optimal solution which is derived from merging all of the previous generations' optimal solutions (at their own generation time) which is the final desirable outcome in a multiple objective environment [4].

The performance and popularity of GA has brought it as rational choice for industries. The key factors to lead the genetic algorithm applications to success are to have a meaningful fitness evaluation and effective genetic algorithm representation. The elegance and simplicity of the genetic algorithms and its high performance to find the solutions for difficult high-dimensional problems are the reasons that make genetic algorithm become so attractive and widely used. Under some situations, the usefulness and efficiency of genetic algorithms will be enlarged. For example, when mathematical analysis is not available, the failures of traditional search methods, the complexity of the search space and the difficulty to encode the domain knowledge to narrow the search space. During these situations, the use of genetic algorithm will provide efficient and reliable helps to solve the problems. The ability to handle arbitrary kinds of objectives and constraints is one of the advantages of genetic algorithm approach. These things are handled as weighted components of the fitness function and to adapt the genetic algorithm scheduler to particular requirements easily. As a result, the genetic algorithms have been widely used in the world for modeling and problem solving. It can be applied to many technology problem, scientific, engineering problems, entertainment, business and etc. There are a lot of applications that used the genetic algorithm which are robotics, engineering design, automotive design and etc. These three applications will be elaborated to show that how genetic algorithm works on it [5].

2.1 Genetic Algorithm Applications

The flexibility of genetic algorithm has resulted in manipulation of this field in various applications over different industries. Some of these applications are discussed as follows [6]:

2.1.1 Evolvable hardware applications

This field is based on manipulation of GA to produce electronic. The Genetic algorithm models exploit stochastic operators to automatically derive new configurations based on the old configurations. As the model keeps evolving while running in its environment context, finally the desirable configuration which is required by the designer will be reached. An example for such a reconfigurable model can be a robot with capability of manipulating built-in GA to regenerate its configuration after some breakdown due to environmental issues such electromagnetic wave which can cause malfunction in its normal configuration. Moreover, such a robot will be able to alter its configuration to a newer version if it meets a situation where it is requires more functionalities to perform its tasks.

2.1.2 Robotics

As it is clear robotics necessitate designers and engineers to experiment and figure out all the required aspects of a robot such as hardware infrastructure and corresponding software architecture to develop a

comprehensive and efficient robot. For any new task all the mentioned activities are required to be performed again to design a new robot which suits the new objectives. By manipulating Genetic algorithm many of these extra designs requirements can be eliminated. Genetic algorithm will provide this capability to automatically generate a collection of optimal designs which can be used for specific tasks and activities. This approach can even be expanded and result in the generation of robots which can perform several tasks and process more comprehensive applications. Another aspect of using GA in robotics is in navigation process. GA provides optimized solutions for navigation process by which the robot can reach to its required destination without being lost or hitting other objects in the environment. In navigation algorithm each chromosome represents a series of path nodes where each node is a gene. Each gene has an x and y value and a Boolean value which represents whether the next node reachable form the current node or no. By using this approach robot will not only be able to always find a way to the target without hitting the objects but also it will be able to find the most optimal path.

2.1.3 Engineering design

Designing a new engineering model is a complex and time consuming process, but designing an optimal model which uses the minimum resources to deliver the maximum output is even much complex. Such a task requires great deal of effort and experience to be completed perfectly. This is where one more time the functionality of Genetic algorithm comes into action. GA can be integrated into computer based engineering design applications. By following such a strategy the application will be able to analyze different aspect of engineering design principles when generating a new design for a given problem. This approach in addition to providing the required design will also assist the designers to identify the frailties and possible failure points of the design. Such an approach is currently being used in many engineering industries such as aerospace, civil, automotive, robotics, electronics, mechatronics etc. These are a small subset of the fields where Genetic algorithm is currently being used to improve the outcome to the fullest. There are many more fields such as telecommunication routing, trip and shipment routing, gaming, encryption and code breaking, chemical analysis, finance strategies, marketing strategies, which also uses the GA. In summary, Genetic algorithm has obtained a great role in modern world's technological and scientific fields and this is on the increase.

2.1.4 Data Encryption

In the field of cryptology, Genetic Algorithm is being used to produce a new advance encryption by using the operations of Crossover and Mutation. Cryptosystem is a group of algorithm that contains secret keys for the encoding of information or messages into cipher text and decodes them back into their original state. The above figure shows the model proposed by Shannon for a secret key system. A new symmetrical block ciphering system called ICIGA (Improved Cryptology Inspired by Genetic Algorithms) enables the generation of a unique session key in a random process. The users are able to determine the size of the blocks and key lengths during the start of ciphering. The ICIGA is in fact created from an enhancement made upon the system GIC (Genetic algorithms Inspired Cryptography). The ICIGA operates based on the length of secret key being used by the user by splitting the plaintext into parts of the same size. In the ciphering process, the first part is split into block of equal size and being used for the generation of the secret key. The secret key being created will then be used for ciphering other parts of the message. The ciphering process also involves the genetic operations of crossover and mutation by doing a left shit to every block that is processed [7].

2.1.5 Computer Gaming

In the field of gaming, the opponent of the human player is often a form of advance artificial intelligence (AI) that incorporates Genetic Algorithm. Strategies that are being used in the past are being programmed using GA to ensure that the AI is able to learn and improve from the past experience. The learning technique allows the AI to avoid repeating past mistakes, which therefore increase the playability of the game. This enables a more realistic experience to the human player as they are required to change their strategies from time to time. It also helps to avoid the scenario of a situation whereby the human player found a sequence of steps which ultimately leads to success, which means that the game does not pose any challenge anymore. For the GA to work, it requires a method to represent the challenge in terms of the solution and a fitness function for it to decide the quality of an instance. The fitness function first receives a mutated instantiation of an entity and

proceeds to determine its quality. Next, this particular function is customized to fit the problem domain. The fitness function can just be a system timing functions in most cases especially the ones with code optimization. As soon as a genetic representation and fitness function are defined, the GA will instantiate the initial candidates that will then apply the repetitive range of operators made up of selection, crossover and selection to improve the fitness value of the candidates [8].

3 TRADITIONAL METHODS VS GENETIC ALGORITHM

After analyzing the above genetic algorithm examples, we can understand that the genetic algorithm have much more differences from most of the traditional optimization search methods. Instead of working with the variables, genetic algorithm will rather work with the string-coding of variables. Its advantage is that even though the function might be continuous, the coding will still discretizes the search space. Besides that, a discrete or discontinuous function can be controlled with no extra burden because genetic algorithms only require function values at various discrete points. Therefore, genetic algorithm can be applied to solve many problems. Genetic algorithms operators make use of the resemblances in string-structures to make an effective search are also one of the advantages. The genetic algorithm work with a population of points unlike the traditional method which work with single point, it is the most striking differences between them. The expected genetic algorithm solution might be the global solution because more than one string had been processed. Genetic algorithm emphasizes the good information that are found previously by using reproductive operator and propagated adaptively through mutation operator and crossover operator while the traditional method doesn't utilize the obtained information efficiently. By using population based search algorithm, genetic algorithm will reduce the workload to apply the same algorithm many times because multiple optimal solutions can be captured easily in the population. Besides that, there are still other advantages which are GAs use probabilistic transition rules instead of using deterministic rules, work with coding of solution instead of the solution themselves, work with population of solutions instead of single solution and so on. All these advantages can be used to maximize the function and can only obtained by applying genetic algorithm to solve real life problem which the traditional optimization methods cannot [5].

4 CLASSIFICATIONS OF GENETIC ALGORITHMS

A metaheuristic is a procedure which aims to find an acceptable solution in very complex optimization and search problems. [2]

In comparison to other heuristics, metaheuristics make usage of low-level heuristic or search algorithms. Therefore, metaheuristics use concrete heuristics or algorithms which are more abstract.

In comparison to optimization algorithms and iterative methods, retrieved solution is dependent on the set of random variables generated.

Compared to optimization algorithms and iterative methods, metaheuristics do not guarantee that a globally optimal solution can be found on some class of problems. Many metaheuristics implement some form of stochastic optimization, so that the solution found is dependent on the set of random variables generated.

Metaheuristics such as genetic algorithms are characterized by the following attributes [4]:

- Metaheuristics guide search process with defined strategies. Although randomized, metaheuristic algorithms such as GA are not random searches. They exploit prior information to direct the search into a region of better performance within the search space.

- Their goal is to efficiently search the state space to find near-optimal solutions. Thus metaheuristic searches include simple local search algorithms and complex learning processes.

- Metaheuristic algorithms are approximate and usually non-deterministic.

- Metaheuristic algorithms are not problem-specific. They make few assumptions about the optimization or search problem being solved, and therefore they can be used for a variety of problems.

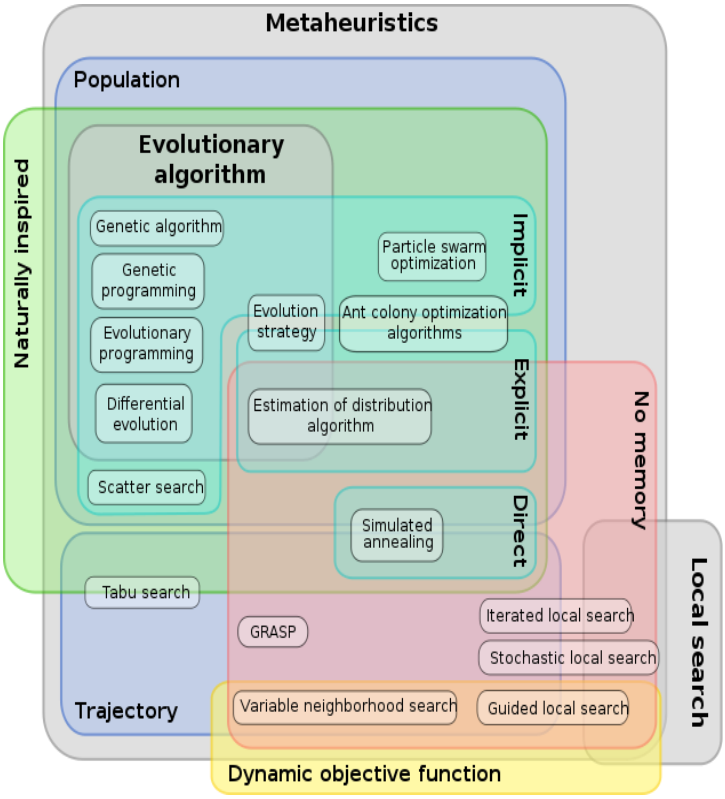


Figure. 1. Metaheuristics classification [9]

GA's are parallel, population-based searches which have a learning component. GA uses stochastic solutions of individual solutions, stochastic crossover and mutation.

4.1 Representation of Genetic Algorithms Using Bit Strings

To represent chromosomes in a genetic algorithm population, strings of bits are normally used, although there are many other existing methods currently in their infancy and under development such as fix-point binary representation, order-based representation, embedded list, variable element list and LISP S-expression.

Every position (locus) in the string (chromosome) has two possible values (alleles): 0 and 1. A solution (chromosome) is composed of several genes (variables). A chromosome represents a solution in the search space of potential solutions.

A population contains a set of chromosomes which are composed of genes. In other words, the solution space contains a set of strings which are composed of bits.

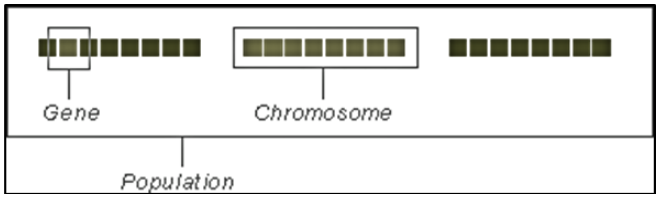


Figure. 2. Population, Chromosomes and Genes [10]

4.2 Implementation Overview

The initial population is first generated at random. Then, the GA goes through the following operators:

4.2.1 Selection operator

The principle here is "survival of the fittest". This operator chooses chromosomes in the population for reproduction. Based on an evaluation function, the 'better' chromosomes are identified and will be more likely to be selected for reproduction. This fitness operator's implementation is dependent on the specific problem at hand.

The 'better' chromosomes may be determined by an objective function which applies uniformly to all chromosomes or by a subjective function, where some rules may not be applied uniformly.

Generally, the probability of selection is proportional to the fitness. It is possible that the best chromosome does not get selected in one run of the GA. However, if the GA is run many times, the probability of selection will converge towards its mathematical expected value.

4.2.2 Crossover operator

This process is closely related to the reproduction of haploid (single-chromosome) organisms or by the fusion of a sperm cell with an ovum to produce a diploid zygote. This operator chooses a position in the string at random. Then, the subsequences before and after that position are exchanged between the strings. The result is the creation of two children strings, i.e. chromosomes.

Given two strings which represent chromosomes, e.g. 11000 and 00111, a crossover operator will randomly select a position, e.g. second position. The resulting children of the crossover will be 11111 and 00000.

Parents: **11|000** and **00|111**
Children: **11111** and **00000**

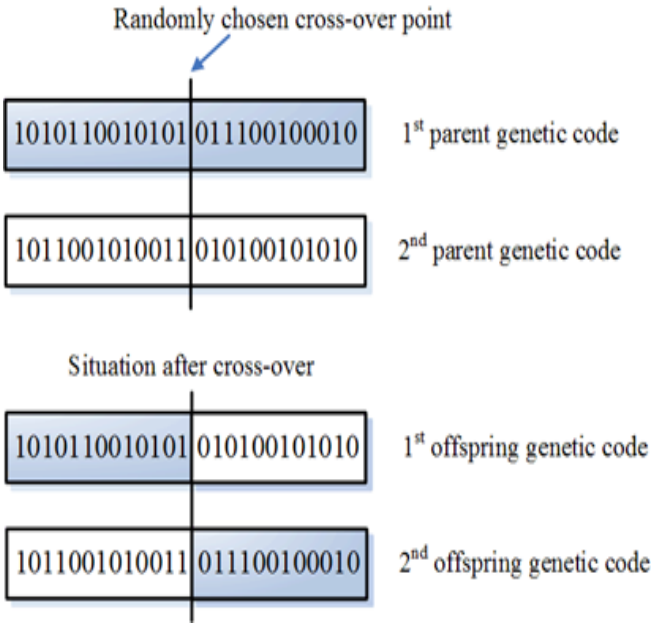


Figure. 3. Genetic Code of the parents and offspring before and after the crossover [11]

Multi-point crossovers are simply crossovers with more than 1 position where crossover will occur.

4.2.3 Mutation operator

This process mimics diversity within a population and helps to prevent premature convergence. This is because mutation and selection alone create hill-climbing algorithms. This operator randomly changes one or some bits in the string representation of a chromosome. For example, a string 10101010 could be mutated in its first position to give 00101010. The probability of mutation and the position of the mutation are controlled by the mutator operator.

Before mutation: 10101010
After mutation: 00101010

4.2.4 Inversion operator

This operator changes the order of genes between two randomly selected points in a chromosome. This fourth operator is rarely used. Its advantages, if any, are not well established. It has been suggested by some artificial intelligence developers that an improvement of search is achieved by such an operator. However, we have noted that all references used for this report indicate that there is no mathematical and empirical evidence of such improvement.

4.3 Effect of Genetic Operators

Using selection alone will have the tendency to generate a population with the fittest chromosomes. Without crossover and mutation, the solution found will be much less optimal.

Using selection and crossover operators only, the GA will converge on a local maxima, which is less optimal than what would otherwise be achieved using all three operators. Using mutation alone makes the GA go on a randomized search through the search space. Using selection and mutation alone will result in the creation of a hill climbing algorithm.

4.4 Pseudo Code for Simple Genetic Algorithm

Assuming that the problem that been clearly defined and candidate solutions have been generated and are represented using X bits, the following steps describe what a genetic algorithm will do, in the following order:

1) Randomly generate n X-bit strings
2) Evaluate fitness of each string in the population using evaluation function f(x)
3) While n offspring have not been created, do
   (a) Select two parent chromosomes based on probability of selection. The higher the fitness, the higher the selection probability.
   (b) Given a crossover and multi-point crossover probability, the crossover operator creates two offspring. In the event no crossover takes place, then the two offspring are identical replicas of their respective parents.
   (c) The two offspring are mutated at each locus using the mutator operator, given a defined mutation probability. The resulting children chromosomes are inserted in the population
   (d) If n is odd, a discard function will have a higher probability of discarding a less fit chromosome
4) The current population is replaced with the new one.

5) Go back to step 2 until fit enough solution is found or until X number of iterations have occurred.

Please note that every time the algorithm goes back to step 2, a new generation is created.

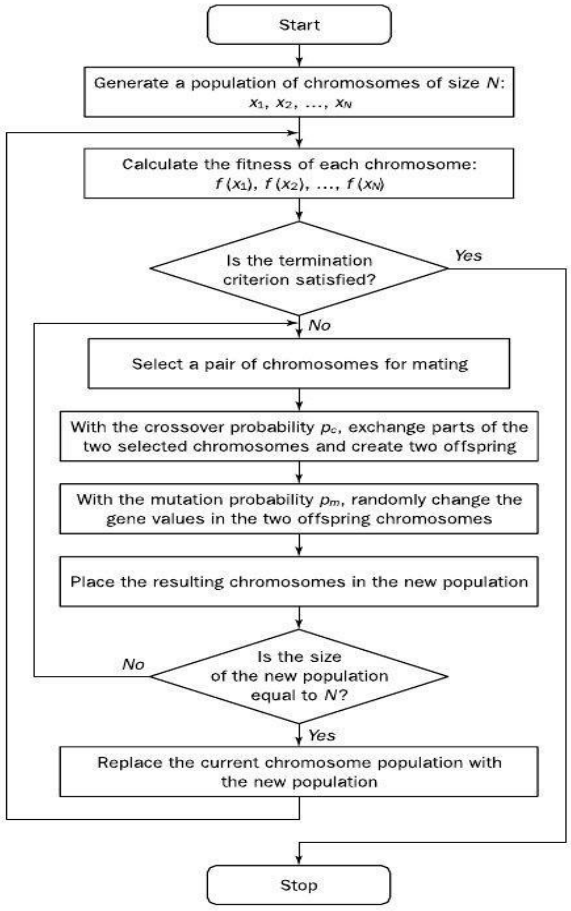


Figure. 4. A basic genetic algorithm [12]

4.5 Genetic Algorithm Parameters

The fundamental parameters of any genetic algorithm are discussed in the following sub-sections.

4.5.1 Cross-over probability

Cross-over points are randomly selected within the length of the chromosomes.

4.5.2 Number of cross-over points

In general, if the chromosome is long, then multiple cross-over points are preferable.

4.5.3 Mutation probability

This refers to the probability of any individual gene which a chromosome to change value from 0 to 1 or vice versa.

4.5.4 The maximum number of iterations

Acknowledging the fact that computational resources are limited, genetic algorithms must have a defined maximum number of generations or a point to indicate when to stop iterating. A maximum number of iterations is especially useful if we have poor knowledge of the ending criteria to indicate the end of iterations.

4.5.5 The value of the termination criterion

If we know how the end criteria should approximately be like, then this parameter is preferred over defining a maximum number of iterations. Examples of termination criteria are:

1. Time Elapsed since fitness increase remains under a certain value

2. If the population diversity falls below a specific value.

4.5.6 The population (generation) size

The greater the number of local maxima, the greater the population size should be. In some cases, the number of local maxima is not known. Therefore the ideal approach would be to have a larger population sample.

5 CONCEPTUAL ADVANTAGE OF GA

Using program evolution to solve computational problems has numerous conceptual applications [13]:

1. To allow a program to adapt and perform according to minimum standards in a changing environment, e.g. robot control in a changing environment or computer interfaces that need to adapt to the idiosyncrasies of an individual user.

2. To allow a program to create innovations on their own, for example, to create a new algorithm to solve a problem. This approach is especially relevant given the fact that intelligent systems are much too complex to program from a ‘top-down’

approach. Therefore an apparently better route to artificial intelligence is through a 'bottom-up' approach where programmers encode the simple rules, and complex behaviors (artificial intelligence) are derived from those rules.

3. Other typical search optimization techniques like linear programming, iterations, simple heuristic functions, depth first search and breadth first searches would involve too much time and space complexity for the solution to be economically viable. By searching over a large set of feasible solutions, metaheuristics can often find good solutions with less computational effort. Therefore metaheuristics like GA are useful for optimization problems.

6 TYPES OF INITIALIZATION OF PROCEDURES

Initialization is actually a very crucial part of Genetic Algorithm, because bad initialization can cause the time it takes to achieve the goal to increase and in the worst case scenario bad initialization can even prevent the algorithm from achieving the best possible solution to the goal. Therefore there are many types of initialization procedures being developed to give developers alternatives and to prevent bad initialization [14].

6.1 Bit String-Uniform Procedure

Firstly is the most commonly used procedure which is the Bit String-Uniform Procedure. Almost all Genetic Algorithm uses the Bit string-Uniform procedure which works by assigning value 0 or 1 with probability of 0.5 to every bit. The argument favoring that procedure is its uniformity in the binary space: using the Bit String-Uniform procedure, all points of 0 and 1 have equal probability of 0.5 to be drawn. Moreover, the bit-wise diversity is maximal, too: at every bit position, there will be as many 0's as 1's. Consider now the point of view of the number of 1's in each bit string. Denote #1(b) the number of 1's in bit string b. The probability distribution of the sum of independent random Boolean variables having probability p to draw 1 follows the well-known binomial law: P [1 = k] = C k n p k (1 p) n.

For large values of n (e.g. n < 30), the binomial law can be approximated by a Gaussian law of mean np and variance np(1- p). Hence for the Bit string-Uniform initialization procedure (p = 0:5), #1 lies in a very restricted range: For instance, for n = 900, 99%.Of all bit strings generated by the standard Uniform procedure have a number of 1's in [411; 489]. Hence, from the point of view of the number of 1's in the bit- strings, the Bit String-Uniform initialization procedure is certainly not uniform.

6.2 Uniform Covering Procedure

Another initialization procedure is theUniform Covering Procedure. The Uniform Covering procedure is designed for each bit string, select a density of 1's, uniformly in 0's and 1's then choose each bit to be one with that specific probability. The Uniform Covering procedure of course leads to a uniform probability distribution of #1. But, on the other hand, the probability for a bit at a given position l in a random bit string to be 1 is 0.5 and the expectation of the total number of 1's in the population is exactly 1/2 of the total number of bits, the Uniform Covering procedure still achieves maximal bit-wise diversity. But let us now consider another alternative point of view, considering the size of sequences of identical bits that are likely to occur in a bit string. When using the Bit string Uniform procedure, sequences of k 1's happen at a given locus with probability 1 1/2 k. With the Uniform Covering procedure, this probability is 1= (k + 1) (= R 1 0 r k dr). Depending on the pre-supposed regularity of the solution (in term of sequences of identical bits), it might be useful to further bias the Uniform Covering procedure to favor the emergence of homogeneous sequences.

6.3 Homogeneous Block Procedure

Besides that, there is another procedure called Homogeneous Block Procedure. The basic idea of the Homogeneous Block initialization procedure is to start from a bit string initialized to a default value for example 0, and to gradually add homogeneous blocks of the other value 1. The critical issue then becomes the number of such blocks, and the way their characteristics are randomly drawn. In the same vein as for the Homogeneous Block procedure, the number of blocks is adjusted depending on a uniformly drawn target density of 1's. Blocks are added to an initial bit string of all 1's, until that target density value is reached. The block characteristics are chosen as follows: A local length L is randomly chosen, uniformly in [0; L max], where L max is user-defined (typically n=10). The center c and the length l of each

block are chosen uniformly, respectively in [0; n] and [0; L]. Finally, to avoid too long loops in the case of high target density and small local length L, a maximum number of blocks is authorized (typically n). Due to the tight control over the density of 1's, the bit-wise diversity should be close to optimal. However, because of the limit on the number of blocks, that procedure is slightly biased toward bit strings having more 0's than 1's. A way to counter-balance that bias is to draw half of the bit strings using 0 as the default value, and the other half by reversing 0 and 1.

6.4 Case-base Initialization

Unlike the above procedures the case-based initialization initializes genetic algorithms in changing environments which is part of an ongoing investigation of machine learning techniques for sequential decision problems. This procedure is a novel combination of two methods of machine learning (genetic algorithms and case-based approaches) to perform learning in a changing environment in which we can monitor and record the changes. Anytime learning with case-based initialization shows a consistent improvement over anytime learning without case-based initialization. Case-based initialization allows the system to automatically bias the search of the genetic algorithm toward relevant areas of the search space. Little time is lost attaining a similar level of learning as in the previous same cases, and then improving on that performance.

One advantage of the method is that the population of the genetic algorithm is seeded without user intervention. Another advantage is that by using good individuals from several different past cases as well as exploratory strategies, default strategies and members of the previous population, our method tends to provide more diversity, guarding against premature convergence.

7 TYPES OF TERMINATION METHODS

Termination is the process where the genetic algorithm decides whether to continue searching or to stop. Each of the enabled termination criteria is checked after each generation to see if it is time to stop. Different types of termination methods include the following [15]:

7.1 Generation Number

The first termination method here is called Generation Number; it works by stopping the evolution the algorithm reached the maximum number of evolution specified by the user. Furthermore the Generation Number is a constantly active termination method is.

7.2 Evolution Time

Secondly Evolution Time is another termination method; it works by stopping the evolution when maximum evolution time specified by the user is exceeded. One flaw of this method is that by default, it will not stop the evolution until the evolution of the current generation is completed even if the maximum evolution time has been exceeded. However this is not a huge flaw as it can be easily solve by allowing the evolution to be stopped during a generation.

7.3 Fitness Threshold

Thirdly Fitness Threshold is another termination method. This method works for both maximize fitness objective and minimize fitness objective. For maximize fitness this termination method will stop the evolution when the fitness of the current population exceeds the fitness threshold specified by the user, while stopping the evolution when the fitness of the current population is lower than the user specified fitness threshold if the objective is to minimize the fitness.

7.4 Fitness Convergence

The fourth termination method is called Fitness Convergence, which stops the evolution when the fitness is considered converged. To determine if the fitness is converged, two filters of different length are used to smooth over the generation obtaining the best fitness and if the best fitness obtained by the long filter and the best fitness obtained by the short filter is less apart than a user specified percentage the fitness is considered converged and the evolution will be stopped.

7.5 Population Convergence

The fifth termination method is called the Population Convergence, which terminates the evolution when the population is considered converged. The population is converged when the average fitness of the population and the best fitness of the population is less apart than a user specified percentage.

7.6 Gene Convergence

The sixth termination method is Gene Convergence, which terminates the evolution when a percentage of genes specified by the user in a chromosome are

converged. A gene is considered as converged when the average value of that gene across all of the chromosomes in the current population is less than a user-specified percentage away from the maximum gene value across the chromosomes.

7.7 Hyper Volume Convergence

Another termination method would be Hyper Volume Convergence, which is a termination method for Multi-Objective Optimization in which the optimal solution set is measured by Hyper Volume of the Pareto-ptimal Front. Evolution will be terminated when the Hyper Volume is stabilized.

7.8 Greedy Search & Back Elimination

Last but not least would be the Greedy Search and Black Elimination termination method, which is for Attribute Selection. When using Greedy Search algorithm, the evolution stops when the fitness does not improve when a single input is added to the previous input set. However when Back Elimination is used, the evolution stops when the fitness becomes worse when a single input is removed from the previous input collection. A variation of the Greedy Search and Back Elimination is also implemented that keeps an elite pool with the size defined as Elite Size. When every solution in the elite pool are checked using above criteria, the search is stopped.

8 TYPES OF OPTIMIZATION PROBLEMS

8.1 Knapsack Problems

The situations; when there are number of items have to pick up into a knapsack. The goal is to optimized item picked be most valuable, which calculate from total value of the item with maximum weight per item to be picked such as problems occur in logistic industrial for efficient transport of product [16].

8.2 Facility Problems

The situations; when a series of facility needed to be placed at the most strategic placement. It is to optimized areas include minimal cost for user, place multi-facilities where existing different type of facilities to minimize the total cost and most strategic area for attraction in order to maximize market opportunity. It is mostly used for shopping complex or city public facilities built [16].

8.3 Traveling Salesman Problem (TSP)

One of most common GA approach could be apply to solve this problem. It is optimized when there are number of nodes which needed for the salesman to travel with minimum cost of distance. These kinds of problem occur in circuit design, routing for transportation and others [17].

8.4 Bin Packing Problem

This problem is derived from terminology of Knapsack problem. It is to optimize when a set of object with different characteristic (volume, weight and other measurement) packed into to numbers of bins, GA helps to minimize the number of bins used. Problem like data packed into a server has to be as minimal as possible; if first bin of space is not fully packed yet, it would be efficient to be able store in data into the remaining [118].

8.5 Multi-modal (global) Optimization Problems

It is to optimize the task where a minimal parameter is received (constraints) but still able to obtain multiple (global or local) best results. It can ease problem if one solution is not useful even through choosing the rest of the results is still at its best estimated result [16].

9 CASE STUDIES USING GENTIC ALGORITM TOOLS PLAYGROUND AND JNETIC

Genetic Algorithm is a method to solve problem and optimize problem at the same time. However, it is impossible to implement without any help from software engineer thus there are existing tools to make this happen. Open source application usually come with limited features while paid application can be custom made depend on what outcome and whether it can be achieve via using Genetic Algorithm.

In this paper we have used two tools such as Genetic Algorithm Playground, which programmed to solve minor problems only, all the problem are listed above types of optimization problem section. Those problems are commonly experienced in real world situation. The application is running cross-platform localized and internet; the application can be run by using applet also known as java plug-in for in computing to make it

convenient to run along the internet, while the application is program using Java programming language, thus it can also be run as a standalone application too if proper installation is adapt. For the testing purpose, due to constraint of sources found, applet is used to run the application through Internet [18].

JNetic is software designed by Steve Bergen. It is used to create stylized images from existing ones, by means of a GA [9]. JNetic makes use of a base image, and tries to replicate it as best it can using a specified number of shapes and colors, all supplied by the user. We already know how the 'fittest' image looks like, all that needs to be done is to evolve a population to get as close as it can to that image. The user can specify the basic GA parameters before running the program. This also includes other settings such as shapes, colors and sizes. However, those other settings will not be varied in this Case study because they are not the focus of this case study is only on GA. The only settings of interest are how changes in GA parameters affect the reproduction of the original image.

9.1 Genetic Algorithm Playground

9.1.1 Problem Description

A knapsack traveller has pick up to valuable item that sufficient needs item for the traveller to travel. The problem the traveller faced is there are only 50 objects can be store with a single knapsack, while these objects contain different variable of weights and values. The goal is to take the greatest valuable object as conceivable [18].

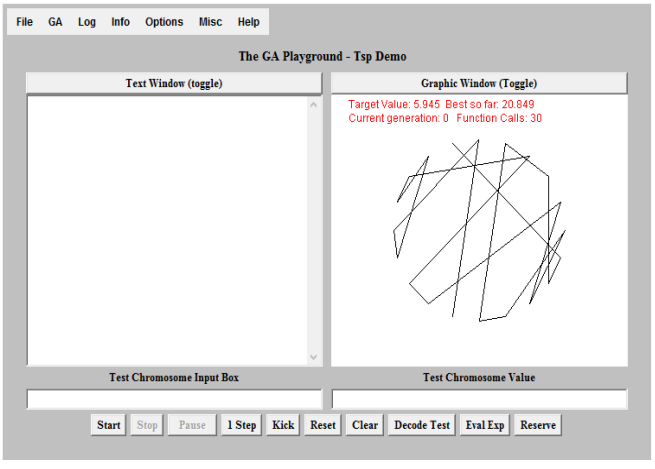


Figure. 5. GA Playground – TSP Demo

9.1.2 Basic Operation of Genetic Algorithm

Reproduction: Duplicate the potential solution of chromosome

Fitness: Calculate each Fitness of all the chromosome

Repeating the following steps

- **Selection:** Define fitness function by choosing two parent
- **Crossover:** Swapping parent to create new children
- **Mutation:** Alter the value of the potential solution

Exit when satisfactory solution is found

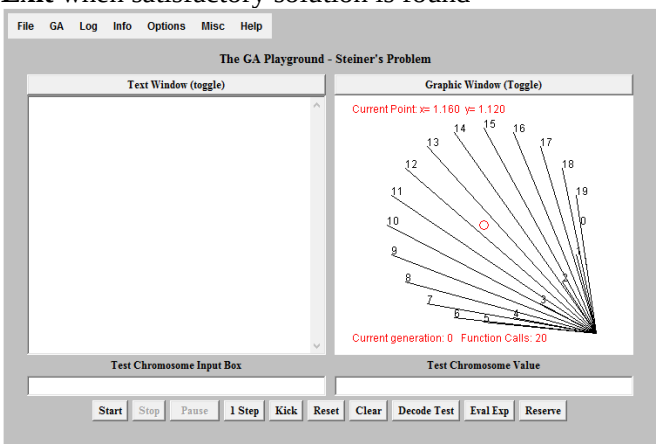


Figure. 6. GA Playground – Steiner's Problem

9.1.3 Optimization and Planning

Those object will be group into set, each set contains a total value and weight cost. The total value of the item can be accumulate from each item included in a set. Item will be compare with different item combination to obtain the highest value. At the same time, total value will be subjected to the total weight, weight is determine where total amount of weight should not exceed maximum weight supported of the knapsack. Thus the outcome must be total value as higher as possible and not exceeding the maximum weight of knapsack.

Let Xi as item, Pi as value and Wi as weight, the collection of cost is than subject to the weight where weight should not be less than maximum weight [18].

$$\text{sum}(x(i)*p(i)) \text{ subject to } \text{sum}(x(i)*w(i)) <= W$$

9.1.4 Application Mode

Parameters can be modified respectively to user's preference, the file must be save locally and it only allow ASCII key. The fitness and function value can be change in population file, this allow the mapped new

chromosome (solution) or when the population type are different. It is optional because the application will generate the default function [18].

In this example, there will not be any use of modification because it must be run locally while program are run through applet.

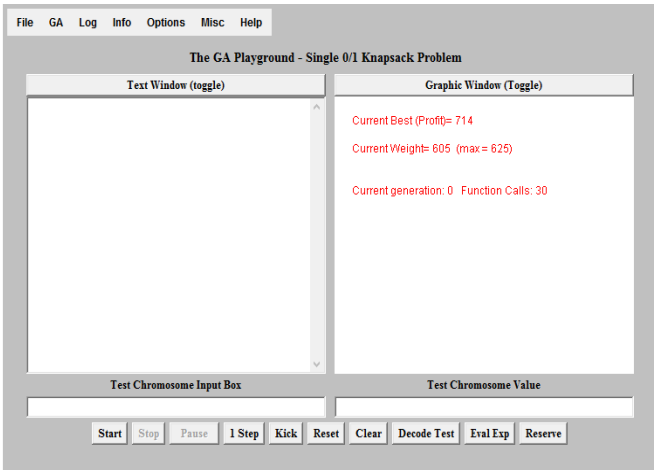


Figure. 7. GA Playground – Application in Process

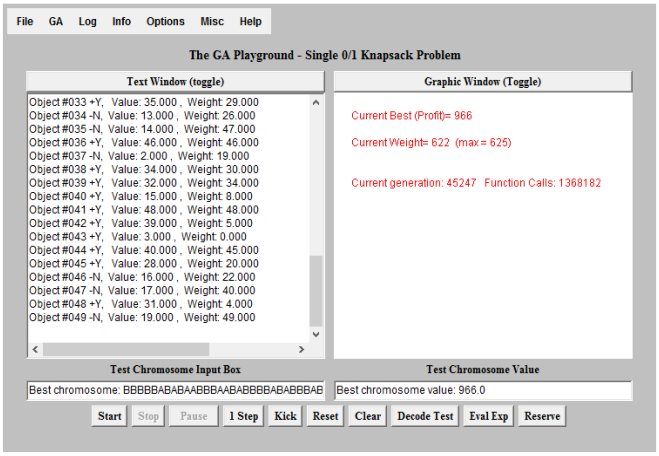


Figure. 8. GA Playground – Application in Process

6.1.4 Results and Finding

Results are attached in Appendix.

The knapsack has a maximum capacity of 625 and needed to obtain as higher as possible but not exceed the maximum weight. We uses 0/1 knapsack solution to maximize the total profit in this area. 0/1 means item is either entirely store or totally leave all item not store any item.

GA is an artificial intelligent uses heuristic search algorithm, the idea is derived from genetic finding. It can find all most usable solution for a set of parameters. Thus, answers are being shown in chromosome, which represent a structure of DNA. The chromosome obtain is: <BBBBBABABAABBBAABABBBBABABBBABB BBBAABABBBBBBBBAABA>. The DNA is then being encoded into readable value which is 966, therefore chromosome is representing the content of the problem and each of the DNA has individual data.

Firstly, the knapsack capacity has been stack into 605 weights and with a profit of 714. After applied Genetic Algorithm (GA) to the problem, surprisingly it had maximized the profit from 605 to 966. In addition, the weight did not exceed the maximum capacity, the weight still have 3 left. The weight is added in exchange to pick up more valuable object while it is still at the same amount of weight.

9.1.5 Limitation of the GA playground

- GA playground can be very slow in process when it is solving problem which is much complex and large number of parameters.
- GA playground need manual observation and configuration because it has the potential of malfunction, never ending load when result and condition coalition. In knapsack problem, weight found is at 3 number away but never stop looping because there are no more parameter to load up 3 number to maximum weight.
- The GA playground is in opening stage, thus the number of problem can be solve is very minimal.
- The knapsack problem can only choose minimum of 50 objects into a single knapsack and cannot be configure when in applet mode.

9.1.6 Advantages

- GA allows to optimized huge variety of problems, for example, TSP can eventually link to circuit design, scheduling, and delivery problems.
- GA is a problem solving mechanism and is able to generate multiple results that are most appropriate.
- Bad parameter initially does not affect the end result, moreover GA will discard those bad parameter.
- GA can solve problem with more complex and finite number of parameter, it uses fitness function.
- GA can relate easily to existing real world situation instead of just genes encoding.

9.1.7 Disadvantages

- GA does not always come with global optimum all the time especially when overall solution has various population.
- A tool is dependent on computers speed and only real time application can produce quick respond time.
- To a non-professional, the return encoded result maybe not be able to understand in application to the user's problem, because GA does not simulation the instructions but rather it shows the encoded chromosome as solution values (not applicable to everyone to use).
- Every time GA provides different results which only allow situation that tolerate with trial and failure results
- GA is complex to be able to apply problem or to be understand.

9.2 JNetic

9.2.1 Selection of Source Image

The image in figure 5 is the SUTS logo, and we will be using this for the experiments.

Figure. 9. SUTS Logo

9.2.2 Choosing Geometric Primitives

The smaller the "pixel" used, the better the final image solution would be. However, the computation resource required to create a solution image which is even grossly similar is extremely high for an average laptop. Therefore, we used large rectangles to represent pixels.

9.2.3 Choosing a Color Scheme

By default the color model of the geometric primitives will follow a full RGB color scheme, which includes all 16 million colors with alpha transparency, i.e., which have varying degrees of transparency and opacity. A range of transparency allows the creation of graphics that have smoother edges.

Transparency has been disabled due to high resource usage. Therefore, all primitives will have 256 of each red, green and blue value, rendered over a white background.

9.2.4 Setting GA Parameters

The success of the final image is extremely dependent on the GA parameters used. The recommended number of generations was set to 500, population to 150, crossover rate to 100%, mutation rate to 0.1% and tournament size (number of individuals needed to fill a tournament during selection) to 4.

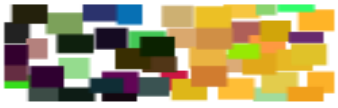
Image correctness is ranges from 0 to 1, 1 being an exact replica of the original image. Image correctness is measured automatically by the built-in fitness monitor by the program.

Important: Because of the randomness of a GA, every run is likely to produce a slightly different final image

9.2.5 Changing the Number of Generations

Population = 200,
Crossover rate = 100%,
Mutation = 1%,
Tournament size = 4

TABLE I. NO OF GENERATIONS

Value	Average Image Correctness (5 d.p)	Image
100	0.45238	
200	0.54667	
500	0.71397	
1000	0.79594	

1500	0.81322	

The GA appeared to reach convergence at about 1500 generations, using current parameters.

9.2.6 Changing Population

Max generation =200, Crossover rate = 100%, Mutation = 1%, Tournament size = 4

A lower population dramatically decreases computational time. The tradeoff however, is a far below optimum solution. The most optimum population was 100. After this point, there was a 'diminishing returns' on the ratio at an increasing rate.

TABLE II. POPULATION

Value	Average Image correctness (5 d.p)	Computation time/s	Ratio (correctness/time)	Image
50	0.58707	0.077	7.624	
100	0.69151	0.139	4.974	
150	0.71944	0.199	3.615	<negligible difference>
200	0.72498	0.245	2.959	<negligible difference>
250	0.72589	0.304	2.388	<negligible difference>

9.2.7 Changing Crossover Rate

Max generation = 200, Population = 200, Mutation = 1%, Tournament size = 4

TABLE III. CHANGING CROSSOVER RATE

Value	Average Image Correctness (5 d.p.)	Image
10	0.69028	
30	0.64933	
50	0.62929	
70	0.65025	
90	0.69936	
100	0.64544	

From the above table, the best crossover rate is about 90%. The value of 100% yielded diminishing returns.

9.2.8 Changing Mutation Rate

Max generation = 200, Population = 200, crossover rate = 100%, tournament size = 4

TABLE IV. CHANGING MUTATION RATE

Value	Average Image Correctness (5 d.p.)	Image
0.1	0.66306	
0.2	0.70309	<negligible difference>
0.4	0.72663	<negligible difference>
0.6	0.73970	<negligible difference>
0.8	0.7399	<negligible difference>
1.0	0.74140	
1.2	0.73669	<negligible difference>
1.5	0.71198	<negligible

		difference>
2.0	0.67763	

From the above table, it is observed that the best mutation rate is about 1.0 %.

After 1200 generations, population size =100, crossover rate = 90%, mutation rate = 1.0% and using large "pixels" of 10 by 30 to represent rectangles, the best possible result is shown in figure 6.

It is possible to achieve near 100% replica if the pixel sizes used were much smaller. However, the population sizes and generations must be increased and this demands days of computational power on a laptop.

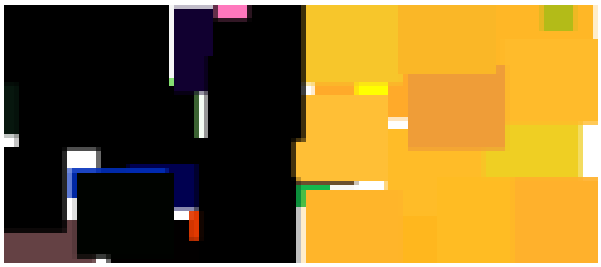


Figure. 10. FINAL SUTS Logo

A progress chart labels the best, worst and average fitness over time. Once the average and best finesses reach near the same value, the GA has converged, and cross-over will be near-useless. This made a compelling case to increase mutation rate to maintain diversity.

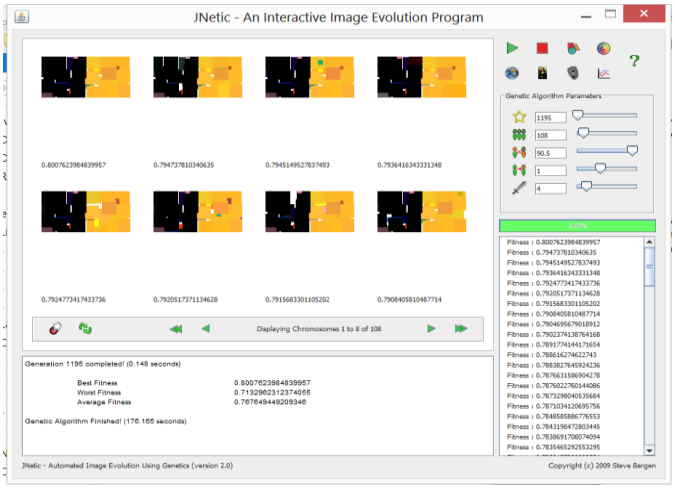


Figure. 11. JNetic Software

10 ACKNOWLEDGEMENT

We are thanks to Sailesh Choytooa, Caleb Lai, Gary Lim and Nelson Ting for their contribution to this work.

11 CONCLUSION

In this paper, we have implemented two case studies to find the parametric values of the GA that provides the most optimum solutions. In conclusion, Genetic Algorithm (GA) is the best application to solve various common problems using fitness function. Genetic Algorithm is an exhaustive approach which can be applied in Artificial intelligence field to find optimal solution in complex search spaces. It is a heuristic search algorithm that will exploit the historical information for best solution. The genetic algorithm is well suited for high complexity problems without any known sophisticated solution techniques like the combinatorial optimization problems. By applying the comprehensive functionalities of Genetic Algorithm in real world scientific and industrial fields, it will not only be ensured that many of the existing problems will be resolved but also there will be a promising future towards development of agents which can perform such a task efficiently and effectively without human intervention. Besides that, the application also optimize problem through generate multiple most appropriate solution. Most of the operation are taken in calculate the fitness by repeating recalculate in order to obtain multiple solution with the most accurate result. Thus, it has taken more time for processing problems. In order to improve the performance, these are the methods should be apply:

- Hash Table

In this method, there are 3 steps should be taken to improve the performance of GA. First, create function that receives single parameter the size of core data structure. Secondly, implement put method to takes two parameters which are the key and the element. Those key are referral for the element. Lastly, get method, which can receive one parameters of key and it can return two values. Those two values include whether the element found and the element [19].

- Taguchi Method(TM)

This method is to select the ones which are useful to select as a result and let go redundancy of calculation;

it helps to reduce the number of conducts calculation. The result selection should covers these 3 priority: (1) Factor of interest, (2) the level of interest and (3) desired cost limitation [20].

- Binary Tree Algorithm

Binary tree algorithm is very similar to Hash table approach where put and get method are implemented. The only difference between Hash algorithm is it uses binary tree method to store data structure while Hash algorithm uses array. Binary tree method is by branching when key is greater than the current value using binary format [19].

- Keep Last Generation Algorithm

This method has been derived from Hash algorithm, the difference in this method is, its stored previous fitness value and able to reuse those values as comparison and such. By doing so it able to reduce the size of hash table than the Hash Table in Hash algorithm where it keeps grows as large as possible [19].

In order to obtain the maximum improvement of GA, these method are recommended to be implement, as from prove it does significantly improve the performance of simple GA approach.

12 REFERENCES

1. E. Eiben (1994). "Genetic algorithms with multi-parent 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.
2. M. Melanie (1999), "An Introduction to Genetic Algorithms", Cambridge, MA: MIT Press. ISBN 9780585030944.
3. Marijke,K 2002, "Genetic Algorithms, An overview", http://www.meteck.org/gaover.html.
4. Prof. Dr. R. Safaric & Dr. A. Rojko, "Genetic code of the parents and the offspring before and after the cross-over", Maribor December 2006, http://www.ro.feri.uni-mb.si/predmeti/int_reg/Predavanja/Eng/3.Genetic%20algorithm/_04.html.
5. Tom, v.M n.d., "Genetic Algorithm", http://www.civil.iitb.ac.in/tvm/2701_dga/2701-ga-notes/gadoc/
6. n, "Real World Uses of Genetic Algorithms", brainz learn something, http://brainz.org/15-real-world-applications-genetic-algorithms/
7. Ankita Agarwal, "Secret Key Encryption algorithm using GA", http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.661&rep=rep1&type=pdf , Vol 2, Issue 4, April 2012. ISSN: 2277128X
8. Lucas, S., and Kendell, G. 2006, "Evolutionary computation and games" In IEEE Comput Intell Mag. February, 10–18. IEEE.
9. T. Wong & H. Wong, "Introduction to Genetic Algorithms", Department of Computing Imperial College of Science and Technology and Medicine UK, http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/hmw/article1.html.
10. Prof. Dr. R. Safaric & Dr. A. Rojko, "Genetic code of the parents and the offspring before and after the cross-over", Maribor December 2006, http://www.ro.feri.uni-mb.si/predmeti/int_reg/Predavanja/Eng/3.Genetic%20algorithm/_04.html.
11. E. Schultz, J. Mellander & C. Endorf (2008), "Intrusion Detection & Prevention - A basic genetic algorithm", [image online], http://my.opera.com/blu3c4t/blog/show.dml/2636486.

12. S. Bergen, "JNetic Textures", http://www.cosc.brocku.ca/~bross/JNeticTextures/.
13. Varma & Nathan Erhardt, "Artifical Intelligence from a philosophical and biological perspective", http://biology.kenyon.edu/slonc/bio3/AI/index.html.
14. Leila Kallel & Marc Schoenauer, "Alternative Random Initialization in Genetic Algorithm", ICGA. 1997.
15. Termination Methods n.d., NeuroDimension Inc., http://www.nd.com/genetic/termination.html
16. Silvano Martello & Paolo Toth, "Knapsack Problems: Algorithms and Computer Implementations", DEIS, University of Bologna, John Wiley & Sons, New York.
17. Obitko, M n.d., IV, "Genetic Algorithm", University of Applied Sciences, http://www.obitko.com/tutorials/genetic-algorithms/ga-basic-description.php .
18. Dolan, A n.d., "A single knapsack problem with 50 objects", http://www.aridolan.com/ga/gaa/Knapsack01.aspx
19. Povinelli, RJ n.d., Improving Computational Performance of Genetic Algorithms: "A Comparison of Techniques", Marquette University, http://povinelli.eece.mu.edu/publications/papers/gecco2000b.pdf.
20. Anagun, AS & Sarac, T 2006, "Optimization of Performance of Genetic Algorithm for 0-1 Knapsack Problems Using Taguchi Method", Eskishir Osmangazi University, http://mmf2.ogu.edu.tr/sanagun/Docs/dok4.pdf
21. Mujahid Tabassum, Kuruvilla Mathew and Sailesh Choyooa, "A Genetic Algorithm Approach towards Image Optimization", ICIEIS2013, SDIWC, ISBN: 978-09891305-2-3.
22. Anil,K & Ghose,M.K n.d., "Information Security using Genetic Algorithm and Chaos, http://www.skoch.in/images/stories/security_paper_knowledge/ Information%20Security%20using%20Genetic%20Algorithm%20and%20Chaos.pdf.
23. Baldi, MM, Crainic, TG, Perboli, G & Tadei, R 2011, The Generalized Bin Packing Problem, Cirrelt, https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2011-39.pdf.
24. Dolan, A n.d., *A general GA toolkit implemented in Java, for experimenting with genetic algorithms and handling optimization problems*, http://www.aridolan.com/ga/gaa/gaa.html#DefinitionFiles.
25. Drezner, T, Drezner, A & Salhi, S 2002, *Solving the Multiple competitive facilities location problem*, Elsevier, http://www.ceet.niu.edu/faculty/ghrayeb/IENG576s04/papers/SA/Solving%20the%20multiple%20competitive%20facilities%20location%20problem.pdf.
26. Genetic Algorithm n.d., http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/tcw2/report.html#TSP.
27. Ramsey, LC & Grefenstette, JJ n.d., Case-Based Initialization of Genetic Algorithms, citeseerx edu. , http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.661&rep=rep1&type=pdf.
28. Ronkkonen, J 2009, *Continuos Multimodal Global Optimization with Differential Evolution-based Methods*, Lappenranta University of Technology, https://www.doria.fi/bitstream/handle/10024/50498/isbn%209789522148520.pdf.
29. *Genetic Algorithms To Constraint Satisfaction Problems* n.d., University of Portsmouth, http://www.economicsnetwork.ac.uk/cheer/ch13_1/ch13_1p16.htm.

Appendix: GA Playground – results:

Single 0/1 Knapsack Problem - Solution:		
Chromosome: <BBBBBABABAABBBAABABBBBABABBBABBBBBAABABBBBBBBBBAABA>		
Value: 966.000		
Object#000 +Y	Value: 49.000	Weight: 21.000
Object#001 +Y	Value: 23.000	Weight: 1.000
Object#002 +Y	Value: 21.000	Weight: 4.000
Object#003 +Y	Value: 37.000	Weight: 2.000
Object#004 +Y	Value: 28.000	Weight: 28.000
Object#005 +Y	Value: 27.000	Weight: 47.000
Object#006 +Y	Value: 21.000	Weight: 2.000
Object#007 +Y	Value: 6.000	Weight: 23.000
Object#008 +Y	Value: 38.000	Weight: 12.000
Object#009 +Y	Value: 7.000	Weight: 12.000
Object#010 +Y	Value: 11.000	Weight: 45.000
Object#011 +Y	Value: 0.000	Weight: 0.000
Object#012 +Y	Value: 23.000	Weight: 29.000
Object#013 +Y	Value: 27.000	Weight: 36.000
Object#014 +Y	Value: 15.000	Weight: 43.000
Object#015 +Y	Value: 0.000	Weight: 0.000
Object#016 +Y	Value: 17.000	Weight: 1.000
Object#017 +Y	Value: 17.000	Weight: 38.000
Object#018 +Y	Value: 5.000	Weight: 4.000
Object#019 +Y	Value: 41.000	Weight: 33.000
Object#020 +Y	Value: 32.000	Weight: 18.000
Object#021 +Y	Value: 32.000	Weight: 3.000
Object#022 +Y	Value: 21.000	Weight: 38.000
Object#023 +Y	Value: 28.000	Weight: 11.000
Object#024 +Y	Value: 0.000	Weight: 35.000
Object#025 +Y	Value: 49.000	Weight: 13.000
Object#026 +Y	Value: 30.000	Weight: 29.000
Object#027 +Y	Value: 5.000	Weight: 3.000
Object#028 +Y	Value: 11.000	Weight: 32.000
Object#029 +Y	Value: 41.000	Weight: 26.000
Object#030 +Y	Value: 28.000	Weight: 21.000
Object#031 +Y	Value: 42.000	Weight: 47.000
Object#032 +Y	Value: 22.000	Weight: 9.000
Object#033 +Y	Value: 35.000	Weight: 29.000
Object#034 +Y	Value: 13.000	Weight: 26.000
Object#035 +Y	Value: 14.000	Weight: 47.000

Object#036 +Y	Value: 46.000	Weight: 46.000
Object#037 +Y	Value: 2.000	Weight: 19.000
Object#038 +Y	Value: 34.000	Weight: 30.000
Object#039 +Y	Value: 32.000	Weight: 34.000
Object#040 +Y	Value: 15.000	Weight: 8.000
Object#041 +Y	Value: 48.000	Weight: 48.000
Object#042 +Y	Value: 39.000	Weight: 5.000
Object#043 +Y	Value: 3.000	Weight: 0.000
Object#044 +Y	Value: 40.000	Weight: 45.000
Object#045 +Y	Value: 28.000	Weight: 20.000
Object#046 +Y	Value: 16.000	Weight: 22.000
Object#047 +Y	Value: 17.000	Weight: 40.000
Object#048 +Y	Value: 31.000	Weight: 4.000
Object#049 +Y	Value: 19.000	Weight: 49.000