Using a GA Adaptor in Multi-applications

Cheng Wei Hang and Yen Cheung

School of Business Systems,
Monash University, VIC 3800, Australia

Abstract. The topic discussed in this paper is a Genetic Algorithm (GA) for multi-applications without any user involvement in the process of the problem design. This user-friendly GA can be used as part of an intelligent tool for problem solving. The main idea of the design is to build a GA based on an adaptor, which encodes problems into a standardized format. This standardized format is then presented to the GA. Using this method; we can separate the GA from any particular application and ensures that it becomes a reusable tool. An extension of this concept is that we can build a number of format adaptors so that it can tackle a wider range of problems. We tested the GA using a number of combinatorial optimisation problems, particularly the Traveling Salesman Problem (TSP). The results indicated that the adaptor helped to improve the overall performance of the GA.

1 Introduction

Research in GA (genetic algorithm) theory and its applications are still being explored in recent years. Efforts in improving the performance of GAs and tackling problems in new applications have been found in numerous publications, especially applications in solving optimization problems. In applying a genetic algorithm (GA) for a particular application, it involves: identification of the object(s), representation of the problem, design of a GA, and interpretation of the search result(s) to the solution. In this process, the design of a GA strongly relies on the particular problem. Different GAs will use different data structures and hence different input formats. The research work discussed in this paper is an attempt to separate the design of a GA from any particular application. A reusable adaptor to the GA is needed. The pre-processor is able to adapt different types of problems before they are presented to the GA.

In achieving this purpose, the idea is to build a GA that associates with a predefined representation format(s). For a given application, the work is to encode the problem into the adaptor(s), and the GA is then able to return search result(s). The overall concept of this approach is as shown in Figure 1.

Fig. 1. Adaptors and the GA

Various forms of GA tools have been developed, but they are mostly for specific applications. Very little work can be found in building GA tools for general-purpose use. Similar recent work of this attempt can be found in (Reddy, Jain, and Pius, 2002). The authors use Borland C++ 5.0 to build a GA tool-package. In this package, it contains a set of tools that use various forms of genetic algorithms including both forms of “Binary Genetic Algorithms” and “Continuous Parameter Genetic Algorithms” (Gerez, 1999). Through a Graphical User Interface (GUI), it is able to provide the solutions for two different types of problems: Traveling Salesman Problem (TSP) and locating an Emergency Response Unit (ERU). Another report in (Walsh, 1999) discussed the development of level GA programs in functional style.

2 A GA Tool Design

2.1 A Form of Representation

We define a sequential symbol as:

$$S_1 S_2 S_3 \dots S_j, \quad j \text{ is a finite number};$$

S_i ($i = 1, 2, 3, \dots, j$) can be a kind of number (a binary number, a decimal number, ...); a kind of symbol (an alphabet, an image, ...); even it can be a data structure.

We define K as a knowledge that is contained in a sequential symbol of $S_1 S_2 S_3 \dots S_j$ (j is a finite number). It can be represented as follows:

$$S_1 S_2 S_3 \dots S_j \sim K$$

For examples, in the traveling salesman problem (TSP), assume that there are ten cities from 0 to 9 and the routing $0 \rightarrow 1 \rightarrow 2 \rightarrow \dots \rightarrow 9 \rightarrow 0$ is represented in the sequential number of 012...9 and its distance is assumed to be 100. We can say the knowledge of 100 is contained in the sequential symbol of 012...9. In a stock portfolio, $Stock_1 Stock_2 \dots Stock_j$ represent stocks holding; $EV(r)$ represent total return value expected in a certain time period with risk r . The relationship can represent as: $Stock_1 Stock_2 \dots Stock_j \sim EV(r)$.

2.2 A GA Design

Domain Space

Base on the notation above, we define a domain space $\mathbf{P}$ that is formed by the elements that have following format:

$$S_1^{(i)} S_2^{(i)} S_3^{(i)} \dots S_j^{(i)} \quad (1)$$

j is a finite number; and i ($= 1, 2, \dots$) is the number of elements in $\mathbf{P}$, and define a solution space $\mathbf{S}$ which is formed by the elements that have following format:

$$S_1^{(i)} S_2^{(i)} S_3^{(i)} \dots S_j^{(i)} \sim K^{(i)} \quad (2)$$

$K^{(i)}$ represents that i th element contains knowledge. If an element contains no knowledge, we consider it as having no value since it does not present any possible solution and there is no learning opportunity. The relationship between $\mathbf{P}$ and $\mathbf{S}$ has:

$$\mathbf{P} \supseteq \mathbf{S}$$

For examples, in TSP, we have $\mathbf{P} = \mathbf{S}$ (Since all combinations represent possible solution.). With a stock portfolio, if any stock has its expected return value with risk in a certain time period, we may obtain the total expected return value for that portfolio; so in this case, we have $\mathbf{P} = \mathbf{S}$. Otherwise, we have $\mathbf{P} \supset \mathbf{S}$. In strategic pricing problem (Cheung and Hang, 2002), we have $\mathbf{P} \supset \mathbf{S}$.

We use the format in expression (2) as an adaptor of the GA tool that we will discuss. So the GA tool should be able to solve the problems that are encoded in this format. In other word, the possible solution(s) of these problems can be represented in form of expression (2) or can be interpreted from it.

Obviously, the GA tool only searches solution(s) from $\mathbf{S}$. The solution space $\mathbf{S}$ can be associated with an intelligent database or case-base reasoning system that it is able to provide the information required by the GA tool. The meaning of an intelligent database or case-base reasoning system is not a simple data system. For example, in TSP, the database may only store the cities and the distances between each city. If using the GA tool to solve the problem, the database must be able to provide the length of any particular routing.

Chromosomes (Set of Population)

The design of individuals in the population is structured as in (Cheung and Hinde, 2000):

Gene [n]	Phenocell
----------	-----------

The Gene¹ [n] and Phenocell² have following structures:

Gene [n]:

Operator	gene value	gene value	...
----------	------------	------------	-----

Phenocell:

$S_1^{(i)}$	$S_2^{(i)}$	...	$S_j^{(i)}$
-------------	-------------	-----	-------------

The gene value takes a random from the phenocell. n is the gene value size. This will vary between different systems. Crossover is operated over the gene value between individuals which be discussed in the following section. In the GA tool, the value of n cannot be an option since we do not know what problem the tool will be used for. It also cannot be chosen as a constant, because this will reduce the randomness in search process, particularly if the value of j is much larger than the value of n. We choose n as:

$$n = \text{rand}(\max(1, (j*10^{-1}/2)), \max(1, j*10^{-1}))$$

In this way we can separate n from a particular problem; at same time it is able to keep sufficient randomness in the search process.

The operator in the gene structure is a specially designed genetic operator named as the parasitic operator (Cheung and Hinde, 2000) in which along with above structure of individuals, it adds further randomness in the search process to increase the search space.

Genetic Operators

Crossover occurs when the gene values have been exchanged between individuals. For example, if there are two individuals as follow:

Individual1:

operator	gene value	...	$S_1^{(i)}$	$S_2^{(i)}$	...	$S_i^{(i)}$
----------	------------	-----	-------------	-------------	-----	-------------

Individual2:

operator	gene value	...	$S_1^{(n)}$	$S_2^{(n)}$	...	$S_m^{(n)}$
----------	------------	-----	-------------	-------------	-----	-------------

the exchange position can be as follows:

$$\text{position} = \text{rand}(1, \min(j, m)).$$

¹ A gene is a stretch of DNA being located in a cell's nucleus.

² Phenocell is the collective physical characteristics and properties that an organism develops through the expression of its genes.

This guarantees that an exchange position will be found between two individuals. Once the crossover is completed, the gene value will be inserted back into the phenocell. Thus two offspring are formed. Before they are inserted into the population, the mutation operator will check if any individual has the opportunity to be mutated. The population uses an overlap method over the generations, and the individual is allowed to have different lengths. This method provides an opportunity to adapt the GA to a wider range of problems.

As mentioned the parasitic is special genetic operator. It uses six pre-defined values which affects the crossover and mutation operations on the phenocell. The purpose of this is also to increase randomness in the search space.

Fitness Function

Fitness function is used to measure each individual during the evolution process. It often works together with object function. In a GA tool, the fitness is designed as an input function as defined by the user. The method described in (Walsh, 1999) can be used in the tool where the fitness evaluation scheme assigns fitness value through building a population based on input arguments (such as $K^{(i)}$ as the input arguments).

Parameters

The parameters in GA work in a complex way. Their settings affect the performance in various problems. We attempt to build a GA tool with a high capability of adaptation for problems that is at the same time, user-friendly.

The optimized setting of parameters is unlikely to be found for a particular application; since a group setting of parameters must be made before applying a GA. Normally we use many different settings to test the results and choose the best one. This method can be automated in the GA tool without needing to input parameters' setting. The recent work in the design of parameter-less GA can be found in (Harik & Lobo, 1999). The authors described a method to build a user-friendly GA without asking the user to choose parameters setting and suggested doubling the size of the population in each test based on its previous test.

The optimized setting of parameters is unlikely to be found for a particular application; since a group setting of parameters must be made before applying a GA. Normally we use many different settings to test the results and choose the best one. This method can be automated in the GA tool without needing to input parameters' setting. The recent work in the design of parameter-less GA can be found in (Harik & Lobo, 1999). The authors described a method to build a user-friendly GA without asking the user to choose parameters setting and suggested doubling the size of the population in each test based on its previous test.

Normally, increasing the population size will improve the GA performance. We use the population size as a major parameter in testing results. The tool starts with a small population size:

$$p^{(1)} = j * 10^{-1}$$

j is the length of phonecell, also $p^{(1)} \geq 4$ (otherwise just select $p^{(1)} = 4$ to start.). The fitness value of $p^{(1)}$ is $f^{(1)}$ ($f^{(1)}$ is the value of last remaining population when the algorithm is terminated.). The second test is doubling the population size:

$$p^{(2)} = 2 * p^{(1)},$$

and the fitness value is $f^{(2)}$. For the i th test, we use population size:

$$p^{(i)} = \left(1 + \frac{|f^{(i-1)} - f^{(i-2)}|}{|f^{(2)} - f^{(1)}|} \right) * p^{(i-1)}.$$

From the above expression, when $|f^{(i-1)} - f^{(i-2)}| \rightarrow 0$, the population size used in the each test becomes stabilized. This means that: when the GA performance is getting better, it is not necessary to increase population size any further for improved performance.

For the genetic operators' probability rates, with a certain population size, we do not know what rates will be best for its performance. However too low probability rates may make the algorithm to converge in local maxima. It also may cause premature convergence when the population size is relatively small. Since the tool uses the population size as a major parameter in improving its performance and for simplicity reasons, the operations rates is randomly chosen from the range of 0.5 to 0.9.

Condition of Terminators

The GA tool is terminated by the condition of convergence that is specified by the user. In many cases, the users are not able to provide such a condition because they need to see the test results before making any judgments. So two terminators are needed in the tool. Firstly, how many times do we need to test? Secondly, for a given population size, how many generations are needed to terminate the algorithm?

The only way to obtain the judgment from users is to output the results each time after the test and get feedback from users to determine if further tests are needed. In the second question, for a given population size, it is unknown how many generations should run to reach the convergence. Since the tool uses a multi-test method to find a better solution, we can choose the number of generations as:

$$5 * p^{(i)}$$

where $p^{(i)}$ is population size of i th test. From our experiments, a factor of 5 seems to be reasonable.

3 Discussions

3.1 Search Space

The maximum population space of problems, which can be tackled by the tool, is j^j (j is the length of phonecell.). For some problems, their solution space may differ from population space as mentioned, since some individuals may not represent any possible solution. Even the initial population is chosen from solution space, because the method used in the evolution process reproduces new individuals that may be located outside the solution space as shown in Figure 2.

Fig. 2. The Relationship of Population and Solution Space

There are two methods that can solve this problem. Firstly, we can build a function to select an individual from the solution space and replace that individual that is located outside the solution space. So the evolution process can be guaranteed to converge at solution space. Secondly, we can assign a special value (for example, the value represents the worst fitness.) to the individual that is out of the solution space and use the selection process to eliminate it from the population in every generation.

3.2 Constraints

Constraints of a problem set the limit of possible solutions. Assuming constraints are a set of $\{A_\alpha\}$, α is a constraints number; solution space is S . After imposing constraints to the solution space, we have a new solution space S' as:

$$S' = S - (\bigcup_{\alpha} A_{\alpha})$$

Actually, the constraints of a problem reduce the solution space size. The tool handles constraints during the formation of the solution space, and the GA does not need to deal with. It only needs to associate with the solution space of S' .

4 Experiments

Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is most focused in combinational optimization. We use the tool to tackle this problem. In our TSP, there are 318 cities; assume the distance d_{ij} is from city(i) to city(j), we have $d_{ij} = d_{ji}$. Object function is:

$$\text{Object} = \min(\sum_{\text{all individuals}} d_{ij}),$$

and the fitness function is:

$$\text{fitness} = (\text{Number of individuals}) * \max(\sum_{\text{all individuals}} d_{ij}) - \sum d_{ij}$$

We can easily represent the TSP in the adaptor as:

$$\text{City}(\text{start}) > \text{City}(\text{next}) > \dots > \text{City}(\text{end}) \sim \sum d_{ij}$$

Genetic operators' rates are randomly selected from 0.5 to 0.9.

Test 1

Parameters setting output:

Mutation rate = [0.8]
Crossover probability = [0.7]
Parasite rate = [0.6]
Population = [32]
Number generations = [160]

Object function return: 48174

Fitness value in last remaining
population: 1109306

Test 3

Parameters setting output:

Mutation rate = [0.7]
Crossover probability = [0.9]
Parasite rate = [0.7]
Population = [128]
Number generations = [640]

Object function return: 48174

Fitness value in last remaining
population: 1104046

Test 2

Parameters setting output:

Mutation rate = [0.6]
Crossover probability = [0.9]
Parasite rate = [0.7]
Population = [64]
Number generations = [320]

Object function return: 48174

Fitness value in last remaining
population: 1247119

Test 4

Parameters setting output:

Mutation rate = [0.5]
Crossover probability = [0.7]
Parasite rate = [0.9]
Population = [133]
Number generations = [665]

Object function return: 48174

Fitness value in last remaining
population: 1180242

Test 5

Parameters setting output:

Mutation rate = [0.6]
Crossover probability = [0.9]
Parasite rate = [0.9]
Population = [207]
Number generations = [1035]

Object function return: 48174

Fitness value in last remaining
population: 1124406

Fig. 3. Test 5

Test 6

Parameters setting output:

Mutation rate = [0.8]
Crossover probability = [0.8]
Parasite rate = [0.7]
Population = [290]
Number generations = [1450]

Object function return: 48124

Fitness value in last remaining
population: 1099882

Fig. 4. Test 6

From Figure 4, we can see that convergence of the tool is very stable and the result has improved from previous tests. So the tool will stop further tests from this point.

5 Conclusion and Recommendations for Further Work

An attempt to build a GA tool for different types of problems is achievable through the use of an adaptor. The end-user or AI agent can use the tool in their applications. Initial experiments have demonstrated that the tool can find solutions for problems such as the TSP. At the same time, other types of problems can be encoded using the adaptor described in this paper. We can use the tool in many applications as long as the problems can be encoded into the adaptor. Other format adaptors can be developed to tackle a wider range of problems. We can also develop a technique-set that associates with a GA such as search with bias. An advantage of this design is that it is easy to include new techniques without changing the GA design. Test results obtained have shown that the adaptor helps to improve the overall performance of the GA. Further research work involve testing the tool, in particular the adaptor with a wider set of problems as well as investigations into other format adaptors.

References

- Reddy G.N., Jain M., Pius L. (2002) *The Genetic Algorithms – A Tutorial*, Artificial and Computational Intelligence, Proceedings of the IASTED International Conference, September 25-27, 2002, Tokyo, Japan. P. 90–95.
- Reddy G.N., Jain M., Pius L. (2002) *The Genetic Algorithms – A Tutorial*, Artificial and Computational Intelligence, Proceedings of the IASTED International Conference, September 25-27, 2002, Tokyo, Japan. P. 90–95.
- Cheung, Y., Hang, C. (2002) *Strategic Pricing using Genetic Algorithm*, Artificial and Computational Intelligence, Proceedings of the IASTED International Conference, September 25-27, 2002, Tokyo, Japan. P. 280–285.
- Cheung Y., Hind C. (2000) *Economic Modelling with A General Purpose GA*, Proc. Int. Conference Advances in Intelligent Systems: Theory and Applications, AISTA '2000, 2-4 Feb 2000, IEEE ACT Section, Canberra, Australia, p. 96–101.
- Walsh P. (1999) *Afunctional Style and Fitness Evaluation Scheme for Inducting High Level Programs*, GECCO-99, July 13–17, 1999, San Francisco, California, Volume 2, p 1211–1216.
- Harik G., Lobo F. (1999) *A Parameter-Less Genetic Algorithm*, GECCO-99, July 13-17, 1999, San Francisco, California, Volume 1, p. 258–265.