iECGA: Integer Extended Compact Genetic Algorithm

Ping-Chu Hung
Ying-Ping Chen

NCLab Report No. NCL-TR-2006005
February 2006

Natural Computing Laboratory (NCLab)
Department of Computer Science
National Chiao Tung University
329 Engineering Building C
1001 Ta Hsueh Road
HsinChu City 300, TAIWAN
http://nclab.tw/

iECGA: Integer Extended Compact Genetic Algorithm

Ping-Chu Hung and Ying-Ping Chen
Department of Computer Science
National Chiao Tung University
HsinChu City 300, Taiwan
{banyhong@gmail.com, ypchen@cs.nctu.edu.tw}

February 3, 2006

Abstract

Extended compact genetic algorithm (ECGA) is an algorithm that can solve hard problems in the binary domain. ECGA is reliable and accurate because of the capability of detecting building blocks, but certain difficulties are encountered when we directly apply ECGA to problems in the integer domain. In this paper, we propose a new algorithm that extends ECGA, called integer extended compact genetic algorithm (iECGA). iECGA uses a modified probability model and inherits the capability of detecting building blocks from ECGA. iECGA is specifically designed for problems in the integer domain and can avoid the difficulties that ECGA encounters. In the experimental results, we show the performance comparisons between ECGA, iECGA, and a simple GA, and the results indicate that iECGA has good performances on problems in the integer domain.

1 Introduction

Since the genetic algorithm (GA) was proposed [1, 2], GA has been widely used in solving problems of which the solutions can be formulated as binary strings. For example, GA has been applied in VLSI design [3], clustering [4], and pattern recognition [5]. The traditional genetic operators such as one-point crossover and bitwise mutation cannot appropriately handle the problems that require the utilization of linkage information [6, 7]. The concepts of genetic linkage and building blocks are the essential components of GAs [8, 2]. As a consequence, the processing of genetic linkage, such as detection and utilization, has attracted much attention in the field of evolutionary and genetic computation.

One way to gather linkage information is to collect and process the global, population-wise statistics. Unlike traditional GAs, probabilistic model building genetic algorithms (PMBGAs), also called estimation of distribution algorithms (EDAs) [9, 10], breed offspring according to the statistical information of the whole population. Some PMBGAs assume all genes or variables are independent, such as the population based incremental learning (PBIL) [11], the univariate marginal distribution algorithm (UMDA) [12], and the compact genetic algorithm (cGA) [13]. The others compute the dependencies and/or relations between genes, such as the Bayesian optimization algorithm (BOA) [14], and the extended compact genetic algorithm (ECGA) [15].

ECGA was proposed by Harik in 1999 [15]. The idea of ECGA is to solve hard problems by learning genetic linkage on the fly. As a PMBGA, ECGA employs the marginal product model (MPM) to represent the joint probability distribution of genes or variables and adopts the minimum description length (MDL) as the criterion to determine how good the learned joint

distribution is. Harik's numerical experiments indicate that ECGA has better performance than a simple GA does when solving hard problems [15].

When tackling binary optimization problems, ECGA is reported quick, reliable, and accurate. However, if ECGA is applied to integer optimization problems, can it perform as well as it can on binary ones? This paper reveals the disadvantages for ECGA to solve integer optimization problems and proposes a new algorithm called the integer extended compact genetic algorithm (iECGA) that can efficiently solve hard integer problems. Particularly, in ECGA, we modify the chromosome representation, extend the marginal product model, and adjust the MDL criterion to make the ECGA mechanism working well on integer problems.

In the remainder of this paper, we will first review how ECGA works and its disadvantages in solving integer optimization problems in section 2. The proposed algorithm, iECGA, will be described in detail in section 3. The numerical experiments that compare and contrast iECGA, ECGA, and a simple GA are presented in section 4. Finally, we will summarize this paper by discussing the potential future work, followed by the conclusion in sections 5 and 6.

2 Brief Review of ECGA

Linkage learning can be considered as identifying building blocks. The extended compact genetic algorithm (ECGA) [15, 16] extends the idea of the compact genetic algorithm (cGA) [13] that the population of GAs can be viewed as a probability model and makes use of probability distributions to learn genetic linkage. In this section, as the study subject, we briefly review the concept and procedure of ECGA as well as illustrate the disadvantages of ECGA on integer problems.

2.1 Linkage Learning and Probability Model

There are two important assertions behind the concept of ECGA. Firstly, learning a "good" probability distribution is equivalent to learning genetic linkage. Secondly, the "goodness" of a probability distribution is based on how much computational resource, mainly the space, the computer system needs to store the population and the distribution.

cGA views the GA population as a vector of probability distributions and the crossover as a sampling operation on the distributions. A GA population can be transformed from and into a probability model. Thus, finding the optimal solution in cGA is equivalent to finding the optimal probability distribution. ECGA extends the probability model in cGA from a probability vector to the marginal product model (MPM). MPMs are similar to the models employed by cGA and PBIL, except that they can represent the joint probability distribution over more than one gene at a time.

As an example, a simple MPM is shown in Table 1. MPM divides the genes or variables into several groups. In Table 1, four genes are divided into three groups [gene 0, gene 3], [gene 2], and [gene 1]. For each group, we count the occurrence of different patterns in the whole population and store it in the table. For gene 2, the number of 1's and the number of 0's are the same. We choose the MPM for two reasons: 1) they make the exposition simpler; and 2) the structure of the model can be directly translated into a linkage map.

The object of ECGA is to find "good" distributions. How do we define the criterion to judge the goodness of different probability distributions? The idea is to adopt the concept of Occam's Razor long recognized in the domain of machine learning [17]:

> By reliance on Occam's Razor, good distributions are those under which the representation of the distribution using the current encoding, along with the representation

group 1 [0 3]		group 2 [1]		group 3 [2]	
allele	prob.	allele	prob.	allele	prob.
00	0.1	0	0.5	0	0.6
01	0.3	1	0.5	1	0.4
10	0.2				
11	0.4				

Table 1: An example MPM for four genes

Algorithm 1 The procedure of ECGA

```
Generate individuals at random
Generation ← 1
while Generation < maxGen do
  Calculate fitness values of individuals
  Perform tournament selection
  Use MPM to build a joint probability distribution
  Use the generated MPM to perform crossover
  Generation ← Generation + 1
end while
Report the result
```

of the population compressed under that distribution, is minimal.

One way to realize this concept is the minimum description length (MDL) [18]. Following the definition, we can use the MDL model on MPMs and define the model complexity and the compressed population complexity of a probability distribution as

$$Model\ Complexity = \log_2 N \sum_{i=1}^{m} 2^{s_i} \tag{1}$$

and

$$Compressed\ Population\ Complexity = N \sum_{i=1}^{m} \sum_{p} -p \log_2 p \ , \tag{2}$$

where m is the number of groups, s_i is the size of ith group, p is the probability of an allele pattern in ith group, and N is the population size. The combined complexity is the summation of the model complexity and the compressed population complexity.

2.2 Extended Compact Genetic Algorithm

The procedure of ECGA is similar to that of a simple GA: initialization, evaluation, parent selection, and crossover. The difference between ECGA and a simple GA is that ECGA models the probability distribution of the population before crossover and utilizes the distribution information to perform building-block-wise crossover. Since the modeling process is guided and judged by MDL, we expect that the partition of the minimized complexity is close to the optimal partition of genes. Thus, the building blocks will not be destroyed by crossover. Algorithm 1 shows the procedure of ECGA.

Let us take a look on the step for MPM. In each generation, the greedy MPM search assumes that all genes are independent and each gene forms an individual group, that is, the MPM

Gen.	Marginal Product Model
1	[0 1 2 3][32 33 34 35][39][16 18 20 22] [21 23][4 5 6 7][12 13 14 15][36 37 38] [19 27][8 9 10 11][17 28 29 30 31][24 25 26]
2	[0 1 2 3][32 33 34 35]**[36 37 38 39]**[4 5 6 7] **[24 25 26 27]**[12 13 14 15][28 29 30 31] [16 17 18 19][8 9 10 11]**[20 21 22 23]**
3	[0 1 2 3][28 29 30 31][20 21 22 23][4 5 6 7] [36 37 38 39][32 33 34 35][12 13 14 15] [8 9 10 11][16 17 18 19][24 25 26 27]
4	[0 1 2 3][28 29 30 31][24 25 26 27][4 5 6 7] [36 37 38 39][32 33 34 35][12 13 14 15] [8 9 10 11][16 17 18 19][20 21 22 23]

Table 2: The output of ECGA

$[0][1]\dots[L-2][L-1]$ is the starting model for the building process. Each pair of the current groups is merged, and the algorithm calculates the combined complexity of the new partition. If the new partition has a lower measure of complexity than the original partition does, the original partition will be replaced and the number of groups becomes $L-1$. This process is repeated until it is impossible for any improvement in this way. After this process of searching for the linkage configuration, we have a MPM representing the linkage between genes and can use the configuration to perform crossover.

2.3 ECGA on Trap Problems

In this section, we will show the performance of ECGA to solve a trap function [19, 20]. Trap functions are considered fundamental components of GA-hard problems and are usually chosen to test the functionality of learning genetic linkage [15, 7]. A 4-bit trap function can be defined as $g_1 : \mathbb{Z}_2^4 \to \mathbb{Z}$ by

$$g_1(x_1x_2x_3x_4) = \begin{cases} 5, & \text{if } x_i = 0 \text{ for all } i \\ x_1 + x_2 + x_3 + x_4, & \text{otherwise.} \end{cases},$$

Given a 40-bit binary string, there are ten 4-bit strings. The evaluation is to put each substring into g_1 and to sum all values. For example, the fitness of the string "0000 0110 1110" is $5 + 2 + 3 = 10$.

For the example, from the literature, we know that a simple GA without the capability of learning genetic linkage cannot find the optimal solution to g_1 if the chromosome encoding is not appropriate [21]. Only *competent* GAs [8] that can regard related four bits as one building block are able to find the optimal solution. That is why we take trap functions to verify the ability of ECGA.

In this simple test, the population size is 1000, and tournament size is 32. The result is shown in Table 2. In the first generation, ECGA does not detect all building blocks correctly, but it soon finds all building blocks in second generation. The optimal solution "0000 ... 0000" is found in 4th generation. Detailed experiments and analysis are provided in [15, 16].

2.4 Problems in Integer Domain

When we directly apply an algorithm that solves problems in binary domain, such as ECGA, to problems in integer domain, certain difficulties will be encountered. The first difficulty is the gap between the genotype and the phenotype. Clearly, an integer ranging from 0 to 15 needs 4 bits to represent. How many bits are needed to represent an integer ranging from 0 to 10? We still need 4 bits to represent such an integer. Hence, to solve the representation gap, we can limit the chromosome in a given range or map two binary strings onto the same integer. When the population evolves in the genotypic space, it seems impractical to constrain a binary string from 0000(0) to 1010(10). Furthermore, mapping two binary strings onto the same integer is not fair for other alleles, because some individuals have more representatives in the genotypic space than others do.

The second difficulty comes from the linkage learning ability of ECGA. The bits that belong to the same integer have linkage, and the integers that belong to the same building block also have linkage at a higher level. In order to correctly find all building blocks, ECGA needs to discover genetic linkage at two different levels. The extra computational cost makes ECGA inaccurate and unreliable. Moreover, the linkage ECGA finds at the bit level may not be the actual linkage at the integer level at which we are solving the optimization problem.

One simple way to overcome these difficulties is to adopt the integer representation. By using an integer vector to represent integers, there is no gap between the phenotype and the genotype, and the linkage between the bits of the same integer, which is obvious in integer optimization problems, is implicitly recognized. Therefore, an integer version of ECGA is in order.

3 iECGA: Integer Extended Compact Genetic Algorithm

In this section, we propose a modified version of ECGA, integer extended compact genetic algorithm (iECGA). We will show what of the representation, MPM, and MDL criterion to modify of the original ECGA in the following paragraphs.

3.1 Representation

An individual in iECGA is an integer vector, instead of a bit vector in the original ECGA. All the integers are assumed to share the same upper bound u and lower bound l, and the cardinality is $u - l + 1$ at the integer level. In this study, to make fair comparisons between iECGA and ECGA, we choose the cardinality at the integer to be a power of 2 to avoid the gap between the phenotype and the genotype. We let $l = 0$ and $u = 2^k - 1$, which simplify the difficulties to represent an integer as a binary string.

3.2 Marginal Product Model

In ECGA, the implementation of MPM is a counting process. Assume that $s = [1\ 3\ 4]$ is a group of genes and $|s| = 3$ is the size of s. The example is shown in Table 3. We count the number of the eight possible patterns in the population (* means "don't care") and record the occurrence, which is equivalent to the probability of the corresponding pattern.

In iECGA, we also have to generate all the possible patterns. Given the upper bound u and the lower bound l, the cardinality of the domain is $d = u - l + 1$. There are $d^{|s|}$ patterns for a group of size $|s|$. If we want to build the MPM, we have to count all $d^{|s|}$ patterns. For example, in Table 4, the upper bound $u = 7$ and the lower bound $l = 0$. The cardinality $d = 8$. If the group of genes is [1 3], we have to count the occurrence of 8^2 patterns in the population.

Current Population
00110
01010
01110
01100
00010
10001

Pattern	Count
0*00*	0
0*01*	2
0*10*	1
0*11*	2
1*00*	1
1*01*	0
1*10*	0
1*11*	0

Table 3: An MPM example in ECGA

Current Population
3472
1624
0314
6715
4360
7164

Pattern	Count
0*0*	0
0*1*	1
0*2*	0
⋮	⋮
7*6*	1
7*7*	0

Table 4: An MPM example in iECGA

3.3 MDL Model

Since the cardinality of the integer domain is quite different from that of the binary domain, we have to modify the complexity measurement to appropriately handle the condition.

$$Model\ Complexity = \log_2 N \sum_{i=1}^{m} d^{|s_i|} ,$$

where d is the cardinality, and s_i is the ith group. The compressed population model is identical to that in ECGA.

4 Results and Discussion

In this section, we first define our test functions. Then, we compare iECGA, ECGA, and a simple GA on these functions. Finally, we will discuss the performance and properties of iECGA.

4.1 Test Functions

The test functions required in this study should have certain trap structure in the fitness landscape, so an algorithm cannot find the optimal solution without learning genetic linkage. A *deceptive* function is one of such a function in which the low-order schema fitness averages favor a particular local optimum, but the global optimum is located at the complement of that local optimum's position [21, 22]. To solve a deceptive function, GA must have the ability to learn linkage. Therefore, we choose deceptive functions as the basic components of our test functions.

The purpose of the experiments is to show whether iECGA outperforms ECGA in the integer domain. Here we define the following four test functions in the integer domain. Each test function is composed of several smaller deceptive functions. For example, if the input length

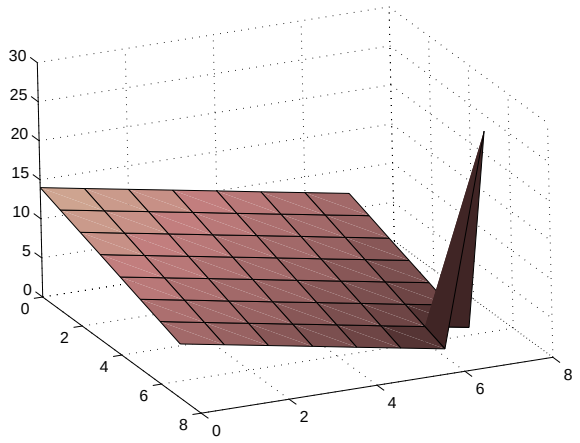


Figure 1: The suboptimum of $f_2(x_1x_2)$ is at (0,0), but the optimum of $f_2(x_1x_2)$ is at $(7,7)$.

of a deceptive function is 2 integers, a test function with input length 10 is composed of five deceptive functions. In the following function definitions, we assume u is the upper bound of a integer and the lower bound is 0.

$$f_1(x) = \begin{cases} 5, & \text{if } x = 0 \\ 1, & \text{if } x = 1, 2, 4, 8 \\ 2, & \text{if } x = 3, 5, 6, 9, 10, 12 \\ 3, & \text{if } x = 7, 11, 13, 14 \\ 4, & \text{if } x = 15 \end{cases}$$

The input of f_1 is a 4-bit integer. There is no building block at the integer level, so f_1 is easy for GAs to solve. But if we see it at the bit level, we will find that the fitness is the number of 1's. Since the local optimum and the global optimum are located in two different directions, this is a deceptive function at the bit level.

$$f_2(x_1x_2) = \begin{cases} 4u, & \text{if } x_i = u \text{ for } i = 1, 2 \\ 2u - x_1 - x_2, & \text{otherwise} \end{cases}$$

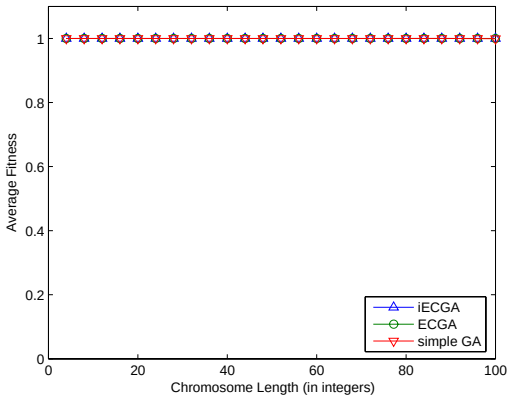
$$f_3(x_1x_2x_3) = \begin{cases} 6u, & \text{if } x_i = u \text{ for } i = 1, 2, 3 \\ 3u - x_1 - x_2 - x_3, & \text{otherwise} \end{cases}$$

$$f_4(x_1x_2x_3x_4) = \begin{cases} 8u, \text{ if } x_i = u \text{ for } i = 1, 2, 3, 4 \\ 4u - x_1 - x_2 - x_3 - x_4, \text{ otherwise} \end{cases}$$

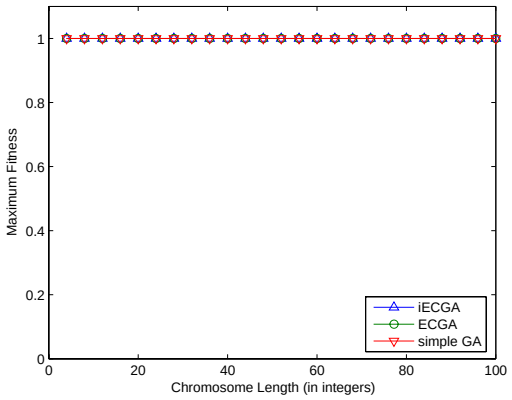
f_2, f_3, and f_4 are specifically designed for the integer domain. The global optimum is located at $(u, u, \ldots, u)$, and the local optimum is located at $(0, 0, \ldots, 0)$. To visualize the landscape of such a function, Figure 1 shows the landscape of f_2. To solve these functions, general GAs are not enough. Regarding related integers as a building block is a necessity to accomplish the task.

4.2 Experiments and Parameters

We use iECGA, ECGA, and a simple GA to solve test functions f_1, f_2, f_3, and f_4. There are 12 set of experiments. Each set of experiments is conducted in 30 independent runs, and the mean fitness values are reported as experimental results.



(a) The average fitness of iECGA, ECGA, and GA.



(b) The best fitness of iECGA, ECGA, and GA.

Figure 2: The (a) average and (b) best fitness of three algorithms in f_1. X-axis is the length of a chromosome in the number of integers. Y-axis is the proportion to maximum fitness.

Because both ECGA and iECGA use tournament selection, we also use tournament selection in the simple GA. The tournament size is 32 in all the experiments. Reported in many empirical studies, GA with uniform crossover has the best performance, so we use uniform crossover in the simple GA. Because of the memory limitation, the cardinality is 16 for f_1 and f_2, 8 for f_3, and 4 for f_4.

The maximum detectable length of building blocks in ECGA is $\log_2 \mu$, where μ is the population size. The length in bit of building blocks in our test problems are 8 or 9. In all the experiments, the population size is 70,000, so the maximum detectable length is 16 bits. Three algorithms are executed up to 15 generations equivalent to 1,050,000 function evaluations.

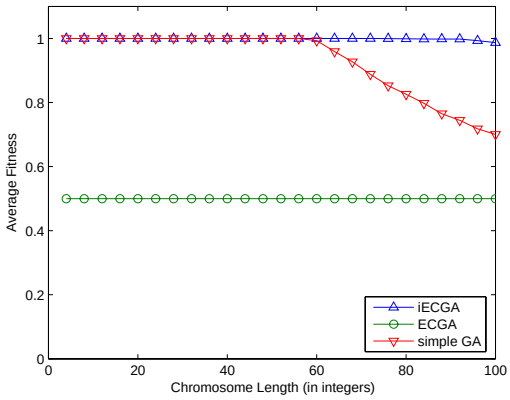
4.3 Results

In all of the test functions, the second best solution has a fitness value half of that of the best solution. If the fitness value of the optimal solution is 20, the fitness value of the second best solution is 10. We present the results of experiments as the proportion to the maximum fitness, that is, we normalize the fitness to the range from 0 to 1. The scaled fitness value of the optimal solution is 1, and the scaled fitness value of the second best solution is 0.5.

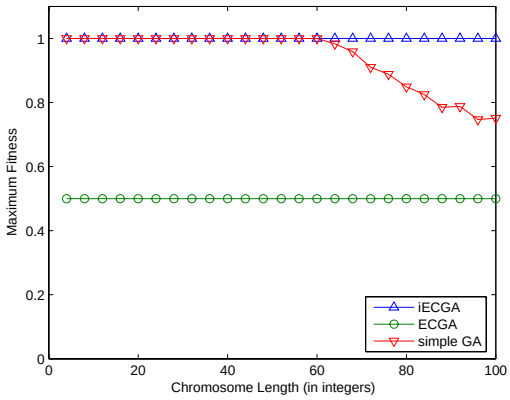
The result of experiment one was shown in Figure 2. For function f_1, three algorithms perform perfectly and all have fitness 1.0 for chromosomes of all lengths.

In Figure 3, we can see that ECGA has problems finding the optimal solution. For all chromosome lengths, ECGA can only find the second best solutions. The best and average fitness values are all 0.5. When the chromosome length is smaller than 60 integers (240 bits), iECGA and GA perform perfectly, but the performance of the simple GA decays quickly when the chromosome gets longer. When the chromosome length comes to 100 integers (400 bits), iECGA can find the optimal solution for most of the runs, but GA cannot.

The result of experiment four was shown in Figure 5. The result is almost the same as that of experiment two, except the performance of GA decays earlier. When the chromosome length is larger than 44 integers (88 bits), GA cannot find the optimal solution. The performance of

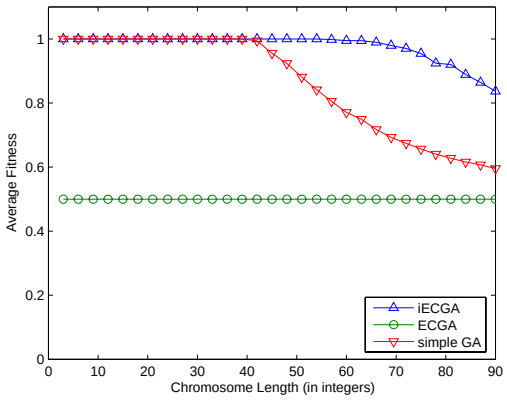


(a) The average fitness of iECGA, ECGA, and GA.

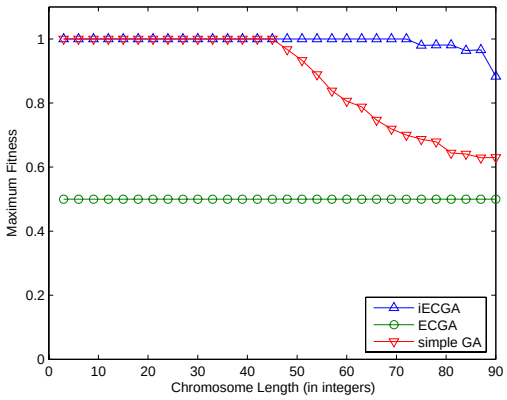


(b) The best fitness of iECGA, ECGA, and GA.

Figure 3: The (a) average and (b) best fitness of three algorithms in f_2. X-axis is the length of a chromosome in the number of integers. Y-axis is the proportion to maximum fitness.



(a) The average fitness of iECGA, ECGA, and GA.



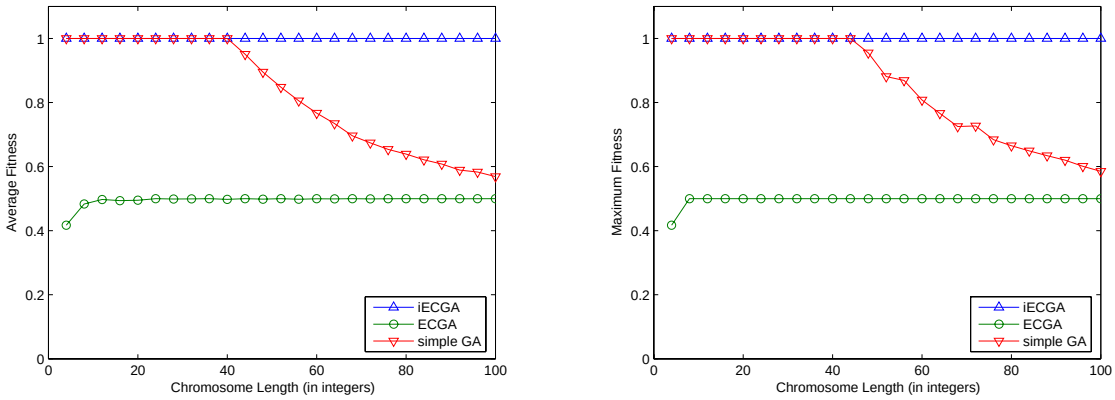
(b) The best fitness of iECGA, ECGA, and GA.

Figure 4: The (a) average and (b) best fitness of three algorithms in f_3. X-axis is the length of a chromosome in the number of integers. Y-axis is the proportion to maximum fitness.

ECGA is still much worse than that of the other two algorithms.

The result of experiment three was shown in Figure 4. The performance of iECGA is not as good as in the other three experiments but still outperform the other two algorithms.

The convergence analysis was shown in Figure 6. The speed of convergence of iECGA is much faster than that of GA. For f_1, iECGA converges after 700,000 function evaluations, but GA cannot converge even after 1,050,000 function evaluations.



(a) The average fitness of iECGA, ECGA, and GA.

(b) The best fitness of iECGA, ECGA, and GA.

Figure 5: The (a) average and (b) best fitness of three algorithms in f_4. X-axis is the length of a chromosome in the number of integers. Y-axis is the proportion to maximum fitness.

4.4 Discussion

iECGA and GA are both operate in that integer domain, but why iECGA performs better than GA? The concepts of genetic linkage and building blocks are important components for GAs to solve problems. The main difference between iECGA and GA is the capability of detecting building blocks and genetic linkage. If the linkage configuration we find is correct, we may expect "good" building blocks will be preserved and "bad" building blocks will be weeded. Hence, iECGA performing better than GA is not unexpected.

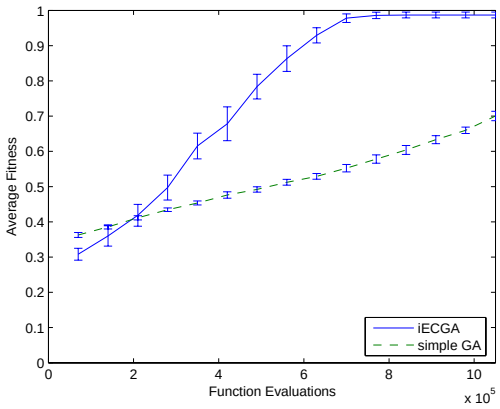
ECGA is reliable and efficient in the binary domain, but why ECGA fails in the integer domain? If ECGA wants to find the linkage between integers, it has to consider several bits as one integer, and then consider several integers as one building block. That is, ECGA has to find building blocks of different hierarchies. It is the first difficulty.

The second difficulty is the selection of coding schemes. Most of GA users employ two's complement to represent an integer, but there are many other kinds of representation, like the gray code. If the linkage between integers can be detected at the bit level, we call that the linkage "propagates" to the bit level. Different representations have different linkage propagations. The linkage between integers may or may not be detected at the bit level. Thus, how to choose an appropriate chromosome representation is an essential issue for GA to succeed.

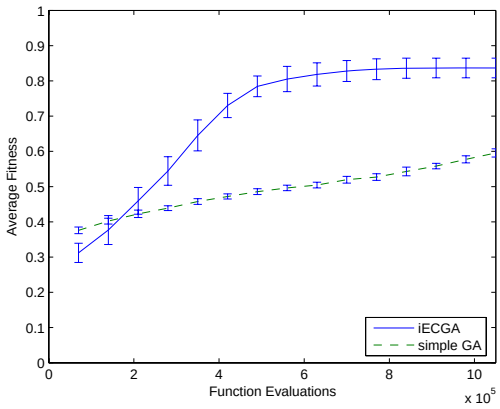
Because of these difficulties, using ECGA to solve integer problems oftentimes cannot satisfy GA users. When we have to solve integer problems, we should use a specialized algorithm. Merely encoding the solutions as binary strings might not be a good choice.

The convergence speed is an interesting property of ECGA and iECGA. Because they exchange building blocks but not genes, they avoid exchanging genes blindly. They converge more quickly than the simple GA does.

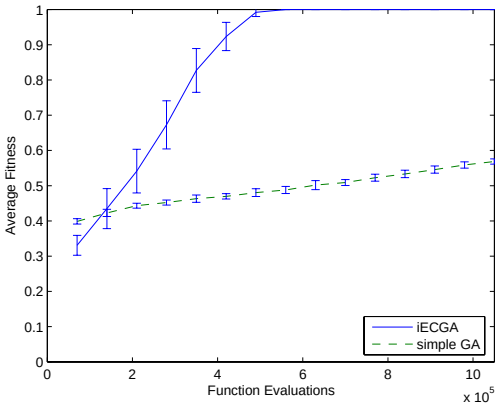
Another interesting observation is that ECGA and iECGA need sufficient individuals to start the MPM step [16]. The population size has a direct ratio to d^{BB}, where d is the cardinality of an integer, and BB is the order of building blocks. In Table 5, we can see that the required number of individuals of f_3 is twice as large as that of f_2 and f_4. Therefore, because the population size



(a) The convergence speed for f_2.



(b) The convergence speed for f_3.



(c) The convergence speed for f_4.

Figure 6: The convergence speed for (a) f_2, (b) f_3, and (c) f_4.

is not enough to start MPM, the performance of iECGA in f_3 suddenly goes down.

5 Future Work

Even the performance of iECGA achieves our expectation, there are several directions we can pursue in the future. iECGA avoid the problem of hierarchical linkage by encoding several bits as one integer but does not really solve the problem. How to find the hierarchical linkage is still a good question waiting to be answered.

Kumara and Goldberg have integrated ECGA with a mutation operator [23], iECGA can also be integrated with a similar mutation operator. Currently, iECGA can handle only the building blocks without overlap. The ability to handle overlapping building blocks can be developed in iECGA [24].

Some problems in other domains, such as the traveling salesman problem (TSP), need special representations. We believe that iECGA can perform well on TSP when we use the adjacency

function	d	BB	d^{BB}
f_2	16	2	256
f_3	8	3	512
f_4	4	4	256

Table 5: The cardinalities (d) and the length of building blocks (BB) of f_2, f_3, and f_4

representation, but such a claim needs to be proven.

Several items of the future work mentioned in [15] can be implemented or developed for iECGA. Readers who are interested in this topic can refer to that article for further information.

6 Conclusions

In this paper, we briefly reviewed the extended compact genetic algorithm (ECGA) and proposed iECGA, the integer extension of ECGA. The main difference between iECGA and ECGA is that they work in the different problem domains. iECGA can detect building blocks at the integer level but cannot find linkage at the bit level. In contrast, ECGA can successfully find linkage at the bit level, but fail to find hierarchical linkage in integer problems. For different types of problems, the appropriate algorithm should be selected to apply. Moreover, GA and iECGA both work in the integer domain. According to the experimental results, iECGA outperforms GA when the problems have linkage of high order. But if the problem has no linkage between genes, GA and iECGA have the similar performance.

This study indicates the importance of using an appropriate algorithm to tackle problems of different types, categories, or domains. Transferring or encoding the solutions may just introduce extra, unexpected difficulties to reduce the applicability and capability of existing good algorithms, instead of making the problem easier to solve. Therefore, we need to understand and investigate the algorithmic components much further in the future to design and develop better evolutionary algorithms.

Acknowledgments

The work was partially sponsored by the National Science Council of Taiwan under grant NSC-94-2213-E-009-120. The authors are grateful to the National Center for High-performance Computing for computer time and facilities.

References

[1] J. H. Holland, *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Co., 1989.

[3] N. Yoshida and T. Yasuoka, "Multi-gap: parallel and distributed genetic algorithms in vlsi," *Systems, Man, and Cybernetics*, vol. 5, pp. 571–576, 1999.

[4] L. Lorena and J. Furtado, "Constructive genetic algorithm for clustering problems," *Evolutionary computation*, vol. 9, no. 3, pp. 309–327, 2001.

[5] R.-Z. Wang, C.-F. Lin, and J.-C. Lin, "Image hiding by optimal lsb substitution and genetic algorithm," *Pattern Recognition*, vol. 34, no. 3, pp. 671–683, 2001.

[6] G. R. Harik, "Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms," Ph.D. dissertation, University of Michigan, 1997, also IlliGAL Report No. 97005.

[7] Y.-P. Chen, *Extending the Scalability of Linkage Learning Genetic Algorithm.* Springer, 2005.

[8] D. E. Goldberg, *The Design of Innovation: Lessons from and for Competent Genetic Algorithms.* Kluwer Academic Publishers, 2002.

[9] H. Mühlenbein and G. Paaß, "From recombination of genes to the estimation of distribution I. binary parameters," in *Proceedings of the Fourth International Conference on Parallel Problem Solving from Nature (PPSN IV)*, 1996, pp. 178–187.

[10] M. Pelikan, D. E. Goldberg, and F. G. Lobo, "A survey of optimization by building and using probabilistic models," *Computational Optimization and Applications*, vol. 21, no. 1, pp. 5–20, 2002.

[11] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," Carnegie Mellon University, Tech. Rep. CMU-CS-94-163, 1994.

[12] M. Pelikan and H. Muehlenbein, "Marginal distribution in evolutionary algorithms," in *4th International Mendel Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks, Rough Sets.* Czech Republic, 1998, pp. 91–95.

[13] G. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," UIUC, Illinois Genetic Algorithms Laboratory, Urbana, IL 61801, USA, Tech. Rep. 97006, 1997.

[14] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "BOA: The bayesian optimization algorithm," in *Proceedings of Genetic and Evolutionary Computation Conference 1999 (GECCO-99)*, 1999, pp. 525–532, also IlliGAL Report No. 99003.

[15] G. Harik, "Linkage learning via probabilistic modeling in the ECGA," UIUC, Illinois Genetic Algorithms Laboratory, Urbana, IL 61801, USA, Tech. Rep. 99010, 1999.

[16] K. Sastry and D. E. Goldberg, "On extended compact genetic algorithm," UIUC, Illinois Genetic Algorithms Laboratory, Urbana, IL 61801, USA, Tech. Rep. 2000026, 2000.

[17] T. Mitchell, *Machine Learning.* McGraw Hill Text, 1997.

[18] J. Rissanen, *Stochastic Complexity in Statistical Inquiry.* World Science, 1989.

[19] D. Ackley, *A connectionist machine for genetic hill climbing.* Boston: Kluwer Academic, 1987.

[20] K. Deb and D. Goldberg, "Analyzing deception in trap functions," in *Foundations of Genetic Algorithms 2*, 1993, pp. 93–108.

[21] D. E. Goldberg, B. Korb, and K. Deb, "Messy genetic algorithm: Motivation, analysis, and first results," *Complex Systems*, vol. 3, pp. 493–530, 1989.

[22] D. E. Goldberg, K. Deb, and B. Korb, "Messy genetic algorithm revisited: Studies in mixed size and scale," *Complex Systems*, vol. 4, pp. 415–444, 1990.

[23] K. Sastry and D. E. Goldberg, "Designing competent mutation operators via probabilistic model building of neighborhoods," in *GECCO 2004*, June 2004, pp. 114–125.

[24] T.-L. Yu, D. E. Goldberg, A. Yassine, and Y.-P. Chen, "Genetic algorithm design inspired by organizational theory: Pilot study of a dependency structure matrix driven genetic algorithm," in *GECCO 2003*, July 2003, pp. 1620–1621.