iECGA: Integer Extended Compact Genetic Algorithm

Ping-Chu Hung
Department of Computer Science
National Chiao Tung University
HsunChu City 300, Taiwan
bjhong@cs.nctu.edu.tw

Ying-Ping Chen
Department of Computer Science
National Chiao Tung University
HsunChu City 300, Taiwan
ypchen@cs.nctu.edu.tw

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. With the experimental results, we show the performance comparisons between ECGA, iECGA, and a simple GA. The results indicate that iECGA has good performance on problems in the integer domain.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms

Keywords

Extended compact genetic algorithms, integer representations, genetic linkage, building blocks

1. INTRODUCTION

ECGA was proposed by Harik in 1999 [1]. 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 [1].

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.

2. PROBLEMS IN INTEGER DOMAIN

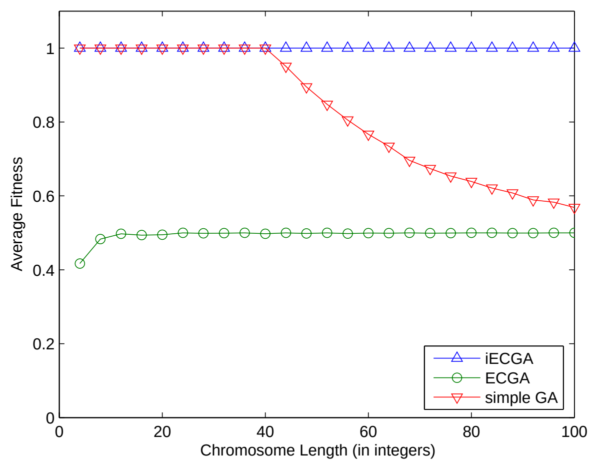
When we directly apply an algorithm that solves problems in binary domain, such as ECGA, to problems in integer domain, certain difficulties will be encountered. The first difficulty is the gap between the genotype and the phenotype. Clearly, an integer ranging from 0 to 15 needs 4 bits to represent. 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 problem.

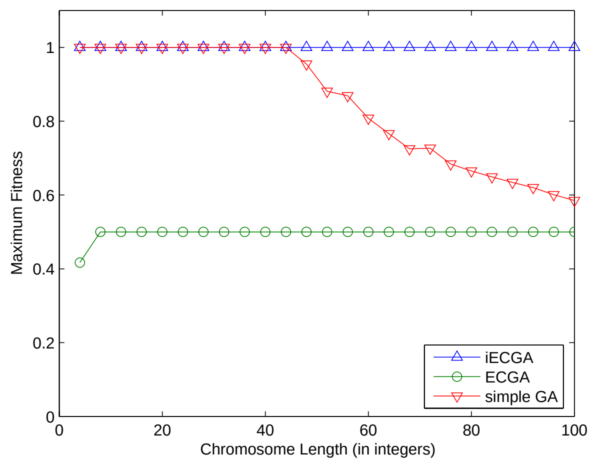
One simple way to overcome these difficulties is to adopt the integer representation. By using an integer vector to represent integers, there is no gap between the phenotype and the genotype, and the linkage between the bits of the same integer, which is obvious in integer optimization problems, is implicitly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

3. TEST FUNCTIONS

In this work, we define four test functions f_1, f_2, f_3, and f_4, but here we only show f_4 as an example due to the space limitation. The global optimum of f_4 is located at $(u, u, \ldots, u)$, and the suboptimum is located at $(0, 0, \ldots, 0)$. To solve this function, traditional GAs are not enough. Re-



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



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

Figure 1: The average(a) and best(b) fitness of three algorithms in f_4. Y-axis is the proportion to maximum fitness.

garding related integers as one building block is a necessity to accomplish the task.

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

4. 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 sets of experiments. Each set of experiments is conducted in 30 independent runs, and the mean fitness values are reported.

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.

In all the experiments, the population size is 70,000. Three algorithms are executed up to 15 generations equivalent to 1,050,000 function evaluations. The results for f_4 are shown in Figure 1.

5. CONSLUSIONS

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 at different levels. It is the first difficulty.

The second difficulty is the selection of coding schemes. Most GA users employ two's complement to represent an integer, but there are many other kinds of representation, such as the gray code. If the linkage between integers can be detected at the bit level, we call the phenomenon *linkage propagation*, which describes that the linkage "propagate" from one level to another. 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.

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.

6. ACKNOWLEDGMENTS

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

7. REFERENCES

[1] G. Harik. Linkage learning via probabilistic modeling in the ECGA. Technical Report 99010, UIUC, Illinois Genetic Algorithms Laboratory, Urbana, IL 61801, USA, 1999.