Send Orders for Reprints to reprints@benthamscience.ae

Open Access

A Genetic Algorithm Using Infeasible Solutions for Constrained Optimization Problems

Yalong Zhang[1,*], Hisakazu Ogura[2], Xuan Ma[3], Jousuke Kuroiwa[2] and Tomohiro Odaka[2]

[1]*College of Electrical and Information Engineering, Quzhou University, Quzhou 324000, China;*

[2]*Graduate School of Engineering, University of Fukui, Fukui 910-8507, Japan;*

[3]*Faculty of Automation and Information Engineering, Xi'an University of Technology, Xi'an 710048, China*

Abstract: The use of genetic algorithms (GAs) to solve combinatorial optimization problems often produces a population of infeasible solutions because of optimization problem constraints. A solution pool with a large number of infeasible solutions results in poor search performance of a GA, or worse, the algorithm ceases to run. In such cases, the methods of penalty function and multi-objective optimization can help GAs run to some extent. However, these methods prevent infeasible solutions from surviving in the solutions pool. Infeasible solutions, particularly those that are produced after several generations, exhibit some achievements in evolutionary computation. They should serve as a positive function in the process of evolution instead of being abandoned from the solution pool. In this study, we extract excellent gene segment for infeasible solutions with a function operation to increase the search performance of GAs. Simulation results on zero-one knapsack problems demonstrate that applying infeasible solutions can improve the search capability of GAs.

Keywords: Constrained Optimization, Genetic Algorithm, Infeasible Solution, Artificial Immune Operation.

1 INTRODUCTION

Over the last two decades, as a result of high global search performance and robust performance, genetic algorithms (GA) have been widely applied to large-scale combinatorial optimization problems. Several optimization problems have constraint conditions. With respect to the constrained optimization problem, GA searches the feasible solutions that satisfy the constraint conditions with the objective function over the entire genetic space. The solutions that do not satisfy the constraint conditions are referred to as infeasible solution whose encoding referred to as lethal chromosomes (LCs).

In the population of GA, due to crossover and mutation operations, LCs are sometimes produced at high rates, especially in combinatorial optimization problems having severe constraints. The greater the number of LCs in the population, the worse the search performance of the GA, in the worst case, the algorithm ceases to run.

Iima Hitoshi [1] (1995) investigated the effects of LCs on the performance of the GA but did not propose a method for handling these problems. If LCs were found to follow some rules, it would be possible to avoid creating LCs. However, designing a genetic operation to avoid the generation of LCs is generally difficult. Generally, LCs are eliminated from the population. However, after evolution through several generations, the LCs may contain useful traits. Therefore, Z. Michalewicz [2] (1995) concluded that "Do not kill unfeasible individuals". If a GA uses the LCs instead of abandoning them, then the search performance of the algorithm may be improved. Mengchun Xie [3] (1996) proposed an algorithm model called the double islands model to revive the LCs by random crossover and mutation operations. Due to its randomness, and without using characteristic information, the efficiency of the double islands model algorithm must be improved.

Research focusing on problems associated with infeasible solutions has advanced in recent years. Yu and Zhou [4] (2008) theoretically showed that the use of infeasible solutions could change the “hardness” of a task. Lyndon While and Philip Hingston [5] (2013) proposed new empirical and mathematical analyses of the usefulness of infeasible solutions in evolutionary search. They also tested a multi-objective approach [5] to constraint handling, and an additional test problem demonstrated the superiority of this multi-objective approach over previous single-objective approaches. Deepak Sharma [6] (2013) proposed an infeasibility-driven approach for bi-objective evolutionary optimization, in which some extreme solutions are allowed to recombine only with extreme infeasible solutions. Tapabrata Ray [7] (2009) maintained a small percentage of infeasible solutions close to constraint boundaries during its course of evolution for constrained optimization. Patryk Filipiak [8] (2011) and Maristela OliveiraSantos [9] (2010)

1874-110X/14 2014 Bentham Open

separately proposed an infeasibility-driven evolutionary algorithm and infeasibility-handling approach in genetic algorithm (GA). Both techniques use infeasible solutions in evolutionary computation.

Studies focusing on GA and knapsack problems are common [10-19]. In the present paper, we propose an immune genetic algorithm (IGA) to revive and use the LCs, which combine the evolutional information of the chromosome with the characteristic information of the problem. Applying the proposed algorithm to the typical multidimensional knapsack problem (MDKP) and comparing the results to those obtained by a GA without immune operation (SGA) reveals the validity of the proposed method.

2. DESIGN OF MDKP

The proposed algorithm, IGA, is mainly composed of a GA and an immune operation module and runs on the double islands model. Immune operation is used to replace the means of performing the genetic operations again to obtain non-lethal chromosomes. Immune operation involves extracting the useful features from the LCs, training a vaccine during evolution, and vaccination of LCs. We introduce these operations one by one in the following.

2.1. Multidimensional Knapsack Problem

The MDKP is first described so that we can explain the proposed algorithm using the MDKP as an example. The MDKP is an NP-hard problem that has several practical applications, such as processor allocation in a distributed system, cargo loading, stock cutting, project selection, or capital budgeting. The goal of the MDKP is to find a subset of objects that maximizes the total profit while satisfying some resource constraints, which can be formulated as:

$$Maximize \quad \sum_{j=1}^{n} v_j x_j \tag{1}$$

$$s.t. \quad \sum_{j=1}^{n} w_{ij} x_j \le c_i \quad (i = 1 \sim m) \tag{2}$$

$$x_j \in \{0,1\} \quad (j = 1 \sim n),$$

where n is the number of objects, m is the number of resources, v_j is the value associated with object j, w_{ij} is the consumption of resource i for object j, c_i is the available quantity of resource i (capacity of knapsacks for the i^{th} resource), and x_j is the decision variable with object j and is set to 1 if j is selected (and is otherwise set to 0). Constraints c_i (i = 1,…,m) described in Eq.(2) are referred to as knapsack constraints, so the MDKP is referred to as the m-dimensional knapsack problem. A number of authors also include the term zero-one when referring to the problem, e.g., the multidimensional zero-one knapsack problem.

2.2. Algorithm Model

The proposed GA, which uses LCs based on immune operation, has two types of chromosome pools, namely, the living island, which contains chromosomes, referred to as non-lethal chromosomes, that satisfy all constraints, and a lethal island, which contains LCs. In the living island, chromosomes are evolved by genetic operations, and in the lethal island, LCs are revived by immune operations.

In the IGA model, after initializing the population, the population is divided into two islands according to whether the chromosome is lethal or non-lethal. In the living island, LCs are created by genetic operations and are moved to the lethal island. In the lethal island, chromosomes are revived by the immune operation and are moved to the living island. A flowchart of the double islands model is shown in Fig. (**1**).

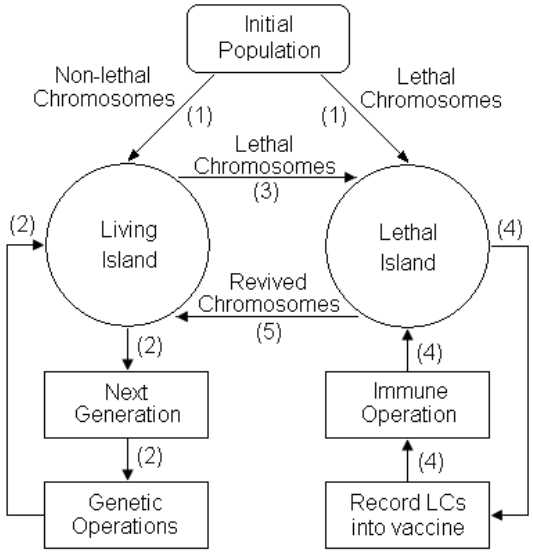

Fig. (1). Flowchart of the double islands model.

As shown in Fig. **1**, in the double islands model, the process of the algorithm consists of two main process lines, one is the process of GA and the other is the process of the immune operation. Section 2.3 describes the immune operations in detail.

Binary value coding is adopted in the present paper as a general method. Binary value encoding has been proven to be well suited for different combinatorial optimization problems. We do not present the proof here. Since the proposed IGA faces primarily constrained combinatorial optimization problems, we explain the problem using an example of the MDKP. In the initial population, we perform the initialization for each chromosome using following algorithm:

(Algorithm: initialization for chromosomes)

1: let: I=(1,2,…,m);

2: set: chromosome $l_1 l_2 \ldots l_n \leftarrow (0,0,\ldots,0)$;

3: $W_i \leftarrow 0, \forall i \in I$;

4: **while**($W_i < c_i, \forall i \in I$) do:

5: select one l_k=0 randomly;

6: **if** ($W_i + w_{ik} \le c_i, \forall i \in I$) then

7: set ($l_k \leftarrow 1$; $W_i \leftarrow W_i + w_{ik}$, $\forall i \in I$);

8: **else** break while;

9: end if

10: end while;

The initial population and the initial lethal island contain no LCs at the beginning of process of the GA. The LCs primarily appear as the process of the GA evolves the population through genetic operations. The closer the solution is to the optimal solution, the more LCs will be produced by genetic operations when forming the next generation. The following is the definition of fitness for chromosome $l_1 l_2 ... l_n$,

$$f = \begin{cases} \sum_{j=1}^{n} v_j l_j, & \sum_{j=1}^{n} w_{ij} l_j \le c_i \quad (i = 1 \sim m) \\ 0, & \sum_{j=1}^{n} w_{ij} l_j > c_i \quad (i \in 1 \sim m) \end{cases} \tag{3}$$

Where $f \neq 0$ when the chromosome is non-lethal, and $f = 0$ when lethal.

2.3. Deal with the LCs through Immune Operations

In order to facilitate the description, we first present several definitions. A chromosome is denoted by a binary value string composed of n items, each of which is referred to as a gene. Some genes can make up an incomplete chromosome, which is a part of all genes in the chromosome. We refer to these genes as a chromosome block. The useful traits contained in the LCs are blocks of LCs. We refer to such a block as an excellent block of an LC. An excellent block that has been extracted from an LC is referred to as an extracted excellent block.

In solving the MDKP, the GA would produce a number of LCs, which produces an unsatisfactory solution. However, in the evolution process, the LCs contain a number of useful traits, which are similar to the parts of the optimal solution and should be used for resource conservation. In order to use these traits, we should first determine methods by which to these traits and vaccinate the LCs with a trained vaccine.

(1) Extract the Useful Traits of the LC

In order to extract the excellent block from the LCs, $l_1 l_2 ... l_n$, a three-value string $f_1 f_2 ... f_n$ is used to denote the block of $l_1 l_2 ... l_n$, where $\forall i \in (1,2, \ldots, n)$, $f_i = 0$ or 1 denotes the original value l_i of the LC, and $f_i = *$ denotes the i^{th} gene that does not belong to the block. For example, "01*0*001*0**110*00*1" is a block of chromosome "01101001001011010001". Each chromosome has several blocks. We provide an estimate for the block as basis for extracting the excellent block from the LC:

$$e(f_1 f_2 \cdots f_n) = \frac{v(1-\mu)}{k} \tag{4}$$

Where $v = \sum_{j \neq *,} f_j v_j$, $\mu = \max_i \left(\frac{\frac{n}{k} \sum_{j \neq *} f_j w_{ij} - c_i}{c_i} \right)$, the value v is different from the fitness value of the chromosome and is not bounded by c_i of the knapsack, and k is the number of 0 or 1(except *) in the block $f_1 f_2 ... f_n$ and is also referred to as the length of block.

There exists at least one optimal solution (chromosome) to a combinatorial optimization problem. The purpose of extracting the excellent block is to obtain a block from LCs that is similar to a corresponding block of an existent optimal chromosome. The shorter the length of the block, the higher probability of being identical to corresponding block of optimal chromosome. However, the longer block is helpful for improving the speed of the immune operation in dealing with the LC. Since a randomly generated gene has a 50% probability of being identical to the corresponding gene of the optimal chromosome for binary value coding, a randomly generated chromosome has a greater likelihood to obtain half of the genes of the optimal chromosome. In the present paper, the excellent block is searched the block with the length of block from $n/2$ to n.

The steps for extracting the excellent block from the LCs are outlined as follows:

(Algorithm: extract the excellent block from the LCs)

1: r←0;

2: **for** (j=n/2 to n) do:

3: $t_1 t_2 \ldots t_n \leftarrow l_1 l_2 \ldots l_n$;

4: select n-j genes randomly from $t_1 t_2 \ldots t_n$ and set them as *;

5: **if**($e(t_1 t_2 \ldots t_n) > r$) then

6: $f_1 f_2 \ldots f_n \leftarrow t_1 t_2 \ldots t_n$;

7: $r \leftarrow e(t_1 t_2 \ldots t_n)$;

8: end if

9: end for

10: get the $f_1 f_2 \ldots f_n$ as the extracted excellent block form $l_1 l_2 \ldots l_n$;

(2) Vaccine Training

By extracting the excellent block, we introduce the immune theory to the GA to revive the LCs created by the genetic operation. According to immune theory, it is necessary to train the vaccine during evolution. As a result, we can vaccinate the LCs when they appear during the evolution of the GA.

In the beginning of the evolutional process, we construct vaccine schema, $s_1 s_2 ... s_n$, and set $s_i = 0$ ($i = 1,...,n$). During evolution, when each LC, $l_1 l_2 ... l_n$, appears as a genetic operation, the following operations are performed to train the vaccine:

(Algorithm: train vaccine with lethal chromosomes)

1: **for** i=1 to n do:

2: **if** $l_i = 1$ then

3: $s_i \leftarrow s_i + 1$;

4: end if

5: end for

Therefore, the vaccine is actually composed of the statistics of the genes from the LC, where $s_i(i = 1,...,n)$ will increase as the evolution progresses, and s_i and $s_j(i \neq j)$ are usually different from each other. We refer to $s_1 s_2 ... s_n$ as the vaccine, which plays an important role in vaccination.

(3) Vaccination

Both of excellent block and vaccine can be used as available resources to revive a valuable LC. The immune operation extracts the excellent block from the LCs and then restructure the remaining genes marked as * in the extracted excellent block. To restructure the remaining genes, immune operation sets the remaining genes, with the exception of the genes of the extracted excellent block, to 1 firstly, and then, according to the tendency of the vaccine to set the genes one-by-one to 0 until the LCs become to non-lethal chromosome. The pseudo-code is explained as follows:

(Algorithm: vaccination with vaccine)

1: set the LCs for vaccination as $l_1 l_2 \ldots l_n$;

2: extract the excellent block and record it with $f_1 f_2 \ldots f_n$;

3: **for** i=1 to n do:

4: if(f_i=*) then $l_i \leftarrow 1$;

5: end for

6: **while**($l_1 l_2 \ldots l_n$ is still lethal) do:

7: in $l_1 l_2 \ldots l_n$ select one gene from all genes whose value=1 and corresponding f_i=* by roulette according to the value of s_i in the vaccine $s_1 s_2 \ldots s_n$;

8: Set the selected gene to 0;

9: end while;

The vaccination of the LCs keeps the genes of the extracted excellent block unchanged, and resets residual genes (f_i = *) by immune operation based on the vaccine.

2.4. Steps of the IGA

The IGA is a GA with immune operation, which uses the immune operation of reviving the LCs to replace the means of retrying the genetic operations repeatedly to obtain non-lethal chromosomes. In order to use the LCs, the immune operation first extracts the excellent block of LCs, and then vaccinates LCs with vaccine. The IGA works under the framework of the GA. Under the double islands model, the steps of IGA are summarized as follows:

(Algorithm: steps of the IGA under the double islands model)

1) Initialize population

Move the non-lethal chromosomes into the living island and the lethal chromosomes into the lethal island.

2) Evolve population

(In the living island):

All of the chromosomes in the living island evolve into the next generation by genetic operations, and the new lethal chromosomes move to lethal island. In this process, the vaccine must be trained.

(In the lethal island):

Each chromosome in the lethal island is revived by immune operations and is then moved to the living island.

3) Repeat step 2 until the termination conditions of the GA are satisfied.

3. COMPUTATIONAL EXPERIMENTS

In order to check the practical performance of the IGA, we tested the IGA on the classical constrained combinatorial optimization problem, the MDKP. A set of standard test data of the MDKP was proposed by Chu and Beasley [10] and is publicly available from the OR-Library. For the experiment, we adopted instances of the MDKP from the OR-Library and changed its constraint conditions, i.e., the knapsack capacities, by following way:

$c_i \leftarrow c_i * l$, $i=(1,2,\ldots,m)$

Where l is a variable used to control c_i. To describe the tightness ratio of the constraints condition, we introduce the parameter α for the MDKP instance, which is defined as follows:

$$\alpha = \sum_{i=1}^{m} c_i \Big/ \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} \tag{5}$$

Where c_i and $w_{i,j}$ are as defined in Section 2.1. The purpose of the present paper is to obtain a method for the GA using the LCs, so we would rather observe the effects of the LCs by changing the tightness ratio α of the constraint condition to the same test instance. We present the results of the experiments for small- and large-scale problems with different α.

The present research proposes a method by which to deal with the LCs, replacing the present means of retrying genetic operations to obtain non-lethal chromosomes for the GA. This does not contradict other methods which are reported to improve the performance of the GA in other literatures [11-19]. Therefore, we compare the proposed IGA with the GA without using the LC and the SGA, in order to investigate the ability of the IGA.

3.1. Experiments on the Small-scale MDKP

We performed computer experiments for the IGA, taking the instances of the MDKP from the OR-Library and changing the tightness ratio α, and compared with the experimental results of the SGA. Ordinarily, the evolutional curve of the GA is plotted with respect to the number of generations of the GA. In the case of the present study, since the IGA has greater time complexity for one generation than the SGA, we set the terminating CPU time and plotted the evolutional curves with respect to the CPU time instead of the generation in order to compare the results of the IGA and the SGA and examine their performances. We refer to this type of evolutional curve as the time evolutional curve.

Table **1** summarizes the results of the experiments on the IGA and the SGA for a small-scale MDKP for the cases of α

Table 1. Experimental results for the small-scale MDKP.

Experimental parameters	Population size: 30, termination time: 20 s, number of simulations: 30			
problem	$m = 5, n = 39$			
	α	0. 5	0.25	0.125
	value of exact solution	8,244	4,296	2,562
IGA	average number of completed generations	7,308.83	5,763.77	4,907.6
	average number of LCs	201,840	199,646	223,696
	average CPU time for LCs	17.5098 (s)	17.9638 (s)	18.3162 (s)
	average of best solutions	8,238.97	4293.97	2559.97
	range of best solutions	8,181~8,244	4,285~4,296	2,501~2,562
	SD of best solutions	12.4592	3.72812	10.9498
	number of exact solutions obtained	14	20	29
SGA	average number of completed generations	23,789.8	14,634.1	4,577.63
	average of number of retries for LCs	754,389	714,875	480,826
	average CPU time of retry	12.368 (s)	15.266 (s)	18.509 (s)
	average of best solutions	8,209.93	4,263.5	2,519.27
	range of best solutions	8,144~8,244	4,244~4,296	2,505~2,562
	SD of best solutions	33.2174	23.6203	16.8225
	number of exact solutions obtained	8	8	4

= 0.5, 0.25, and 0.125. The time evolutional curves are shown in Figs. (**2**), (**3**), and (**4**). The problem parameters for each case and the experimental parameters of the IGA and the SGA are summarized in Table **1**.

In Table **1**, the experimental data were obtained by applying several algorithms to the same problems. The item "average number of completed generations" indicates the average number of generations completed by the IGA and the SGA in a given CPU time. The "average of number of LCs" indicates the average of the sum of the LCs for all completed generations for several simulation runs. The "average CPU time for LCs" indicates the CPU time spent to revive the LCs with the immune operation in the IGA. The item "average of best solutions" is the average fitness value of the best solutions obtained for all simulation runs of the algorithm. The items "range of best solutions" and "SD of best solutions" are the range and standard deviation of the fitness value of the best solution for all runs. The "number of exact solutions obtained" indicates how many simulation runs converge to the exact solution.

For the SGA, the "average of number of retries for LCs" indicates the average number of rerun operations of the GA to obtain a non-lethal chromosome when the LCs appeared in the chromosome pool after the genetic operations. The "average CPU time of retry" indicates the CPU time spent on the above rerun operations of the GA for the LCs. The other items have the same meaning as those described for the IGA.

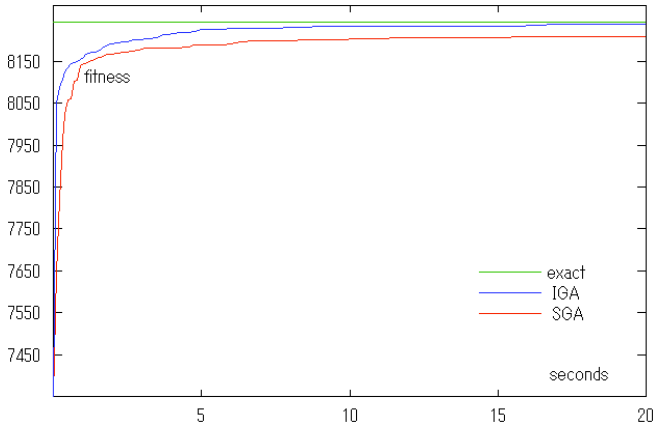

Fig. (2). Time evolutional curve of α = 0.5 on the small-scale MDKP.

From Table **1**, the IGA evolves generations no faster than the SGA. In the other words, the immune operations to revive the LCs in the IGA require more CPU time than rerunning the genetic operations to obtain the non-lethal chromosome in the SGA. However, during the given CPU time, the IGA can find a better solution than the SGA and has a smaller SD of the best solution compared to the SGA, especially for small values of α.

In order to investigate the ability to obtain an exact solution, we select test instance as possible as large scale for

which the exact solution can be obtained using the branch and bound method (BBM) within the available CPU time. For the three cases of α = 0.5, 0.25, and 0.125, the BBM requires CPU times of 200, 60, and 8 hours, respectively, to obtain the exact solution.

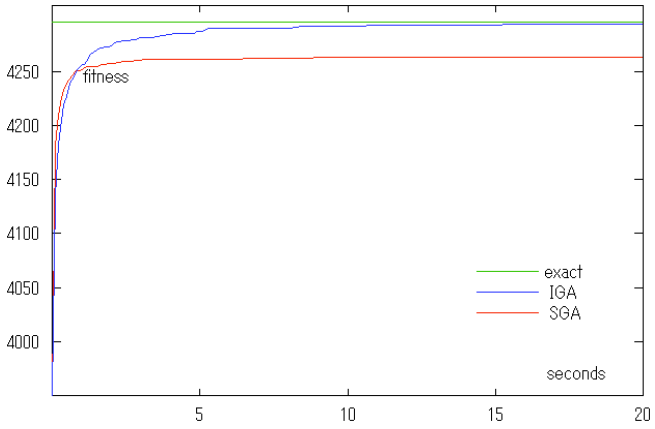

Fig. (3). Time evolutional curve of α = 0.25 on the small-scale MDKP.

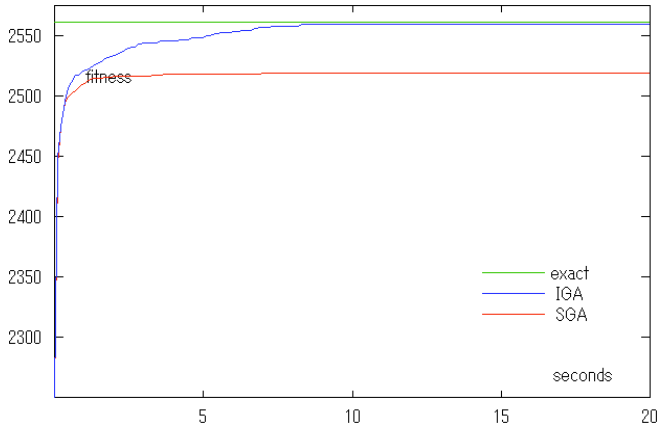

Fig. (4). Time evolutional curve of α = 0.125 on the small-scale MDKP.

3.2. Experiments on the Large-scale MDKP

For the large-scale MDKP experiments, we selected an instance with m = 30 and n = 500 from the OR-library and applied the IGA and the SGA. We also change the value of α to 0.5, 0.25, and 0.125. In the GA simulations, we set the population size of the chromosomes to 50 and the termination CPU time to 3,600 seconds, and we performed the simulations 10 times to obtain the results for each case of the MDKP for the IGA and the SGA.

Table **2** summarizes the results for the IGA and the SGA at different constraint ratios α. Figs. (**5**), (**6**), and (**7**) show the average time evolutional curves for different values of α.

When α = 0.5, within 3,600 seconds, IGA finished, on average, no more than 1,200 generations, in contrast to the SGA, which finished, on average, over 32,000 generations. Approximately 67,000 LCs were generated by genetic operations in the IGA, most of which were revived by immune operations, whereas over 1.6 million LCs were generated by genetic operations in the SGA. Furthermore, the SGA retried the genetic operations 4.5 million times, i.e., approximately three times (≈ 4.5/1.6) per LC, to obtain a non-lethal chromosome.

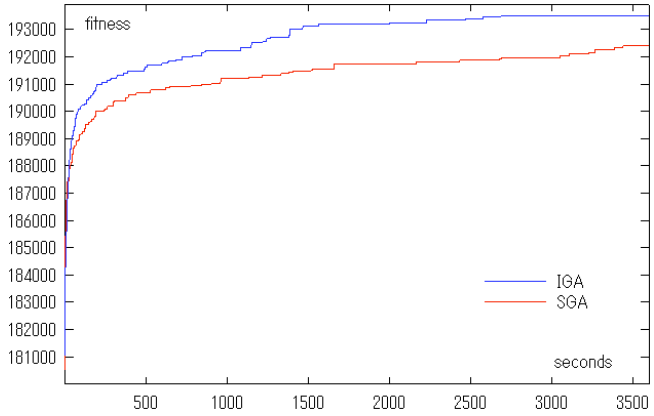

Fig. (5). Time evolutional curve of α = 0.5 on the large-scale MDKP.

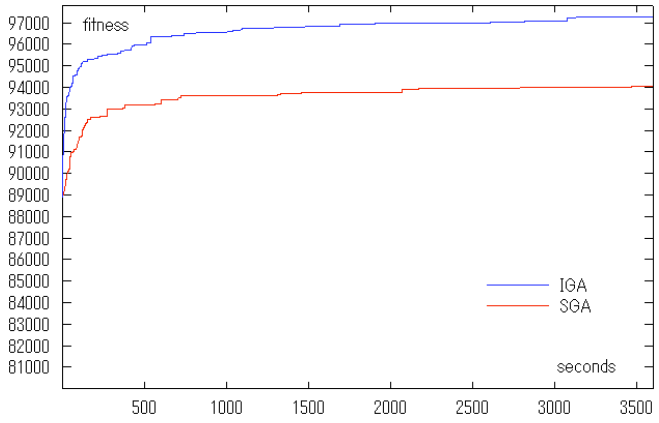

Fig. (6). Time evolutional curve of α = 0.25 on the large-scale MDKP.

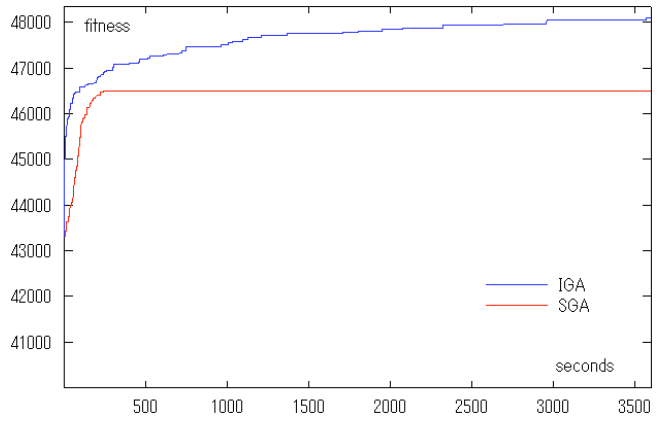

Fig. (7). Time evolutional curve of α = 0.125 on the large-scale MDKP.

In the SGA, the average probability of an LC being generated from one chromosome in the child generation pool

Table 2. Experimental results for the large-scale MDKP.

Experimental parameters	Population size: 50, termination time: 3,600 s, number of simulations: 10			
Problem	$m = 30, n = 500$			
	α	0. 5	0.25	0.125
IGA	average number of completed generations	1,158.7	572.9	566.4
	average number of LCs	67,689.9	54,805.6	55,510.2
	average CPU time for LCs	3,574.95 (s)	3,590.16 (s)	3,589.05 (s)
	average of best solutions	193,506	97,272.3	48,097.7
	range of best solutions	192,530~194,788	96,528~98,230	47,278~48,827
	SD of best solutions	700.4	460.6	436.8
SGA	average number of completed generations	32,857.7	630.5	672.3
	average of number of retries for LCs	4.50658 e+06	7.61905e+05	8.11378e+05
	average number of LCs	1.65185e+06	55,975.6	63,363.8
	average CPU time of retry	2,833.5(s)	3,586.49(s)	3,587.31(s)
	average of best solutions	192,413	94,064.4	46,493.1
	range of best solutions	190,855~193,369	92,991~95,351	45,112~47,559
	SD of best solutions	668.9	811.4	865.7

by the crossover and mutation operation is approximately 0.5 [≈ (1.6×10^6 LCs)/(3.2×10^4 generations)/(50 chromosomes)/(2 child chromosomes)], which is less than 0.58 (≈ 67,689.9/1,158.7/50/2)in the IGA. In addition, for the cases of α = 0.25 and 0.125, the probability of LCs appearing is larger for the IGA than for the SGA, which implies that the population of the IGA is closer to the boundary of the constraints of the problem, making the generation of LCs by genetic operations easier than that for the SGA. The computational complexity of the IGA is greater than that of the SGA, and the IGA requires more CPU time to process one generation than the SGA. However, the performance of the IGA for obtaining higher fitness chromosomes is high.

For small α, the constraints are more severe, so the genetic operations break down the non-lethal chromosomes to the LCs more easily than for large α. For small α, the SGA could finish less than 700 generations, which is a significant decrease from over 32,000 generations for α = 0.5, whereas the IGA could finish almost the same number of generations, over 500, which is over 50% of the generations processed for the case of α = 0.5. The ability of the IGA is clear for the case in which α is small, i.e., the case in which the constraints were severe. These facts are shown in Figs. **6** and **7**. As shown in Fig. **7**, when α = 0.125, the SGA could not evolve the chromosomes in the pool after 300 seconds, whereas the IGA could evolve the chromosomes gradually as the generation progresses. In addition, the SD of the best solution of the IGA for 10 iterations was smaller than that of the SGA, which also shows the higher performance of the IGA. On the whole, the IGA is significantly different from the SGA, which works especially well for the optimization problem with very severe constraints.

The experiments discussed in subsections 3.1 and 3.2 showed the role LCs in the evolutionary process; thus, we tested various data for LCs and then compared IGA with SGA on a single test case. As an algorithm to solve the MDKP, IGA should be tested on a large number of experiments rather than an individual test instance. IGA should also be compared not only with SGA, but also with other approaches to solve MDKP. We tested IGA on all 270 large-scale MDKP instances provided by the OR-library [10]. We also compared the results with those of other algorithms for solving the MDKP [10-19]. These algorithms include *GA with H1* and *GA with H2* proposed by Günther R. Raidl [11], *Swap* and *Insert* from Jens Gottlieb [12], and the *Improved GA* also proposed by Günther R. Raidl [17]. The quality of a solution is measured by the percentage gap of the objective value, *fitns*, with respect to the optimal value of the LP-relaxed problem, $f^{\mathrm{LP}}{}_{\max}$: $\%\text{-}gap = 100\times(f^{\mathrm{LP}}{}_{\max} - fitns) / f^{\mathrm{LP}}{}_{\max}$.

Consistent with the representation in literature [10-19], various approaches were tested in different running conditions and with different objectives data. Comparing IGA with each of the other techniques directly is difficult. Nonetheless, IGA obtains an average %-gap of 0.54 on 270 instances, whereas those of other algorithms are between a %-gap of 0.64 and 0.54. IGA shows preliminary superiority in searching for optimal solutions. In another paper, we will report exclusively detailed test results on a large number of large-scale problems, and then compare the results of IGA with those of other algorithms under different running conditions.

4. DISCUSSION

In recent decades, several studies reported improvements in the performance of the GA. Various improvements were applied to the GA to solve a wide range of optimization problems. The IGA was proposed as a method to replace the retrying operations of the GA to obtain a non-lethal chromosome when an LC appeared as a result of the immune operation. The IGA does not contradict other methods [11~19] reported to improve the GA. The immune operation can be applied with other methods simultaneously.

4.1. General Ability of the IGA

Based on the results of the experiments, we can conclude that the IGA is better suited to problems with severe constraints than the SGA.

In general, for MDKP combinatorial problems, the constraints of the problem bound the solution space into two parts, namely, the feasible space, where all of the constraints are satisfied, and the infeasible space, where at least one of the constraints is not satisfied. Moreover, the exact solution of the problem generally exists near the boundary on the feasible space side. When applying the GA to such problems, the chromosomes in the pool evolve toward the exact solution. If the chromosome coding method in the GA could take all of the constraints into account and the genetic operations would be designed so as not to generate any LCs for the child chromosome, all of the chromosomes in the GA pool were non-lethal and should evolve within the feasible space. However, in general, it is difficult or too sophisticated to design such a coding method and genetic operations. However, when the simple coding method, such as the {0,1} coding system, is adopted, the genetic operations generate the LCs at a high rate, especially for severe constraints. Moreover, in the later stage of the GA, when the chromosomes evolve and approach the exact solution, and near the border of the constraints, the genetic operations tend to generate LCs at a high rate, and, in the worst case, the generated chromosomes are all lethal. The GA without using the LCs works on only one side of the boundary, the frequent appearance of the LCs impairs the ability of the GA to reach the exact solution within the feasible space.

The LCs in the infeasible space, that are generated from the non-lethal chromosomes in the feasible space by genetic operations, must possess blocks that correspond to valuable parts composing the chromosome of the exact solution. In the double islands (non-lethal and lethal islands) model proposed in a previous study [3], the genetic operations applied to the LCs in the lethal island can revive an LC and migrate it to the non-lethal island. In the IGA, the immune operations can revive the lethal chromosomes at a fairly high rate compared with the double islands model. Immune operations provide more channels that lead to optimal solutions for the LCs.

For small α, the constraints are more severe, so the genetic operations more easily break down non-lethal chromosomes into LCs than for the case of large α. For the MDKP, when α is decreased from 0.5 to 0.25, and then to 0.125, the feasible solution space shrinks significantly as α^m. As α decreases, the "average number of completed generations" of the SGA decreases rapidly, which indicates that the performance of the SGA is reduced as α decreases, i.e., under the severe constraints. However, the IGA maintained the "average number of completed generations" despite the large decrease in α, which indicates that the IGA is weakly affected by α and shows a more excellent ability even under the severer constraints. The IGA has an advantage in searching the exact solution for problems with severe constraints.

For the large-scale MDKP with m = 30, as α decreases, the feasible space shrinks significantly compared to the small-scale MDKP with m = 5. The performance of the SGA to search for a better solution is reduced and evolution was not possible after 300 seconds when α = 0.125, whereas the IGA was able to evolve chromosomes when α changed from 0.5 to 0.25 or 0.125. This suggests that the constraints do not significantly affect the immune operation to revive the LCs, and the IGA may be insensitive to the strength of the constraints. The IGA can show more remarkable ability in the cases of severe constraints.

4.2. Vaccine and Vaccination in the IGA

With the vaccine, the vaccination operation transforms the LCs in the infeasible space and offers higher fitness values so that the chromosomes will move not only to the feasible space but will also approach the optimal solution. In this manner, the IGA expands the search space to the other side of the boundary and increases the ability of the GA to reach the optimal solution.

The method by which to train the vaccine and to vaccinate the LCs is important in the IGA. As for the method described in Section 2.3, the trained vaccine records the statistics information of the LCs. Other types of information can also be used to compose the vaccine, e.g., the information of the excellent chromosomes, such as the best chromosome in the chromosome pool of each generation. This type of vaccine records the excellent genes of the best chromosome of every generation.

(Algorithm: train the vaccine with excellent chromosomes)

Before starting the evolution process, set s1s2…sn ←(0,0,…,0). Whenever a new best chromosome l1l2…ln appears to replace the old best chromosome, perform the following training steps:

1: **for** i=1 to n do:

2: **if** l_i =1 then

3: $s_i \leftarrow s_i+1$;

4: end if

5: end for

Here, $s_i (i = 1,\ldots,n)$ will also increase as the GA evolution progresses, and s_i and $s_j (i \neq j)$ will usually be different. Note that $s_1 s_2 \ldots s_n$ can also be regarded as the vaccine. We refer to this as vaccine #2, and the vaccine introduced in the preceding sections is referred as vaccine #1. We performed computer simulation experiments using the IGA and vaccine

#2. Since vaccine #2 can provide more rapid immune operations than vaccine #1, the IGA with vaccine #2 can generate more generations within the simulation time and sometimes obtains better solutions than the IGA with vaccine #1. However, compared to the IGA with vaccine #1, the IGA with vaccine #2 failed to obtain better solutions for all cases. Moreover, compared the IGA with vaccine #2 had a larger SD of best solutions, despite it obtains a better solution sometimes for some problems.

5 CONCLUSIONS AND FUTURE RESEARCH

The present study proposed a method for reviving the LCs to replace the means of retrying the genetic operations repeatedly to obtain non-lethal chromosomes based on immune operations. Applying the IGA to the MDKP revealed that, in some cases, especially for optimization problems with severe constraints (small α), the IGA is more effective for dealing with the LCs than the SGA. In addition, this IGA improved the search performance of the GA in solving optimization problems with severe constraints. Since the extracted vaccine is related to the characteristic information of the problem, for other optimization problems the vaccine should be extracted according to the characteristics of the problem.

In the future, in addition to further improving the double islands model, the method of immune operations must be researched further. If the proposed immune operations are improved to decrease the time complexity and allow rapider operation, the IGA will be applicable to a wider range of constraint optimization problems.

CONFLICT OF INTEREST

The author(s) confirm that this article content has no conflict of interest.

ACKNOWLEDGMENTS

This work is supported by the Scientific Research Foundation (Project No.Y201430654) funded by the Zhejiang Provincial Education Department (China), by the Quzhou Science and Technology Plan Projects (Project No.2013048) funded by the Quzhou Science and Technology Bureau (China), and by the Talents Cultivation Research Start-up Foundation (Project No. BSYJ201309) funded by the Quzhou University.

REFERENCES

[1] Iima Hitoshi, Sannomiya Nobuo, "The Influence of Lethal Gene on the Behavior of Genetic Algorithm", *The society of instrument and control engineers*, vol.31, no.5, pp.569-576, 1995.

[2] Z. Michalewicz, "Do not kill unfeasible individuals," in *4th Intelligent Information Systems Workshop*, 1995, pp. 110-123.

[3] Mengchun Xie, Tetsushi Yamaguchi, Tomohiro Odaka, Hisakazu Ogura, "An Analysis of Evolutionary States in the GA with lethal Genes", *The Information and Systems Society, Institute of Electronics, Information and Communication Engineers*, vol.J79-D-II, no.5, pp.870-878, 1996.

[4] Y. Yu and Z. Zhou, "On the usefulness of infeasible solutions in evolutionary search: A theoretical study," in *World Congress on Computational Intelligence*. IEEE, 2008, pp. 835-840.

[5] Lyndon While, Philip Hingston, "Usefulness of Infeasible Solutions in Evolutionary Search: an Empirical and Mathematical Study", *2013 IEEE Congress on Evolutionary Computation*, Cancún, México, pp.1363-1370, June 20-23, 2013.

[6] Deepak Sharma, Prem Soren, "Infeasibility Driven Approach for Bi-objective Evolutionary Optimization", *2013 IEEE Congress on Evolutionary Computation (CEC)*, Cancun, Mexico, pp.868-875, 20-23 June 2013.

[7] Tapabrata Ray, Hemant Kumar Singh, Amitay Isaacs, Warren Smith, "Infeasibility Driven Evolutionary Algorithm for Constrained Optimization", *Constraint-Handling in Evolutionary Optimization Studies in Computational Intelligence*, vol.198, pp.145-165,2009.

[8] Patryk Filipiak, Krzysztof Michalak, Piotr Lipinski, "Infeasibility Driven Evolutionary Algorithm with ARIMA-Based Prediction Mechanism", *Intelligent Data Engineering and Automated Learning - IDEAL 2011*, vol.6936, pp.345-352, 2011.

[9] Maristela Oliveira Santos, Sadao Massago, Bernardo Almada-Lobo, "Infeasibility handling in genetic algorithm using nested domains for production planning", *Computers &Operations Research*, 37(2010), pp.1113-1122.

[10] P.C. Chu and J.E. Beasley, "A Genetic Algorithm for the Multidimensional Knapsack Problem", *Journal of heuristics*, vol.4, no.1, pp.63-86, June 1998.

[11] Günther R. Raidl, "Weight-Codings in a Genetic Algorithm for the Multiconstraint Knapsack Problem", *Proceedings of the 1999 ACM symposium on applied computing*, Texas, United State, no.02, pp.291-296, February 1999.

[12] Jens Gottlieb, "Permutation-Based Evolutionary Algorithms for Multidimensional Knapsack Problems", *Symposium on Applied Computing Proceedings of the 2000 ACM symposium on Applied computing*, Como, Italy, vol.1, pp.408-414, March 2000.

[13] V. Gabrel, M. Minoux, "A scheme for exact separation of extended cover inequalities and application to multidimensional knapsack problems", *Operations research letters*, vol.30, no.4, pp. 252-264, August 2002.

[14] Guan-Chun Luh, Chung-Huei Chueh, "Multi-objective optimal design of truss structure with immune algorithm", *Computers and structures*, vol.82, Issues.11-12, pp.829-844, May 2004.

[15] Farhad Djannaty, Saber Doostdar, "A Hybrid Genetic Algorithm for the Multidimensional Knapsack Problem", *Int. J. Contemp. Math. Sciences*, vol. 3, no.9, pp.443-456, 2008.

[16] P. C. Chu, J. E. Beasley, "A genetic algorithm for the generalised assignment problem", *Computers and operations research*, vol.24, no.1, pp.17-23, January 1997.

[17] Günther R. Raidl, "An Improved Genetic Algorithm for the Multiconstrained 0-1 Knapsack Problem", *Proceedings of the 5th IEEE International Conference on Evolutionary Computation*, Anchorage, AK, pp.207-211, May 1998.

[18] Alex S. Fukunaga, "A New Grouping Genetic Algorithm for the Multiple Knapsack Problem", *2008 IEEE Congress on Evolutionary Computation (CEC 2008)*, Hong Kong, China, pp.2225-2232, June 2008.

[19] Alex S. Fukunaga, Satoshi Tazoe, "Combining Multiple Representations in a Genetic Algorithm for the Multiple Knapsack Problem", *2009 IEEE Congress on Evolutionary Computation (CEC 2009)*, Trondheim, Norway, pp.2423-2430, May 2009.

Received: September 16, 2014 Revised: December 23, 2014 Accepted: December 31, 2014

© Zhang *et al.*; Licensee *Bentham Open*.

This is an open access article licensed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted, non-commercial use, distribution and reproduction in any medium, provided the work is properly cited.