Solving Integer Programming Problems Using Genetic Algorithms

Pham Nguyen Anh Huy* and Chu Tat Bich San* and E. Triantaphyllou**
* Faculty or Inoformation Technologies, Natural Sciences University, Hochiminh City, Vietnam
Tel : +084-08-8 354 266 Fax : +084-08-8 350096
E-mail: pnahuy@fit.hcmuns.edu.vn, ctbsan@fit.hcmuns.edu.vn
**Department of Industrial Engineering, Louisiana State University, Baton Rouge, LA 70803-6409, USA
Tel : 225-578-5372 Fax : 225 578-5990 E-mail: trianta@lsu.edu

Abstract: There are many methods to find solutions for Integer Programming problems (IPs) such as the Branch-Bound philosophy or the Cutting Plane algorithm. However, most of them have a problem that is the explosion of sets in the computing process. In addition, GA is known as a heuristic search algorithm for solutions of optimization problems. It is started from a random initial guess solution and attempting to find one that is the best under some criteria and conditions. The paper will study an artificial intelligent method to solve IPs by using Genetic Algorithms (GAs). The original solution of this was presented in the papers of Fabrício Olivetti de Françaand and Kimmo Nieminen [2003]. However, both have several limitations which causes could be operations in GAs. The paper proposes a method to upgrade these operations and computational results are also shown to support these upgrades.

Keywords: Integer programming, Genetic algorithms, optimization.

1. INTRODUCTION

This paper will study an artificial intelligent method to solve IPs by using Genetic GAs. The original of this was presented in [1] and [2]. Moreover, the paper will show several limitations of [1] and [2] in some cases which a global optimization can not be obtained. Causes could be operations in GAs. Then the paper also proposes a method to upgrade them. Computational results are also shown which support these upgrades. Basic concepts supporting the paper will be presented in section 2 such as concepts in GAs, presentation of IPs in GAs. Section 3 is some discussions and proposes to upgrade the method. Finally, experiments and comparison will be presented in section 4.

2. BASIC CONCEPTS

2.1. GAs

There are many methods to find solutions for IPs, but most of them are often too complicated in calculations such as Bound-Branch philosophy. In addition, GA is known as a heuristic search algorithm for solutions of optimization problems. It is started from a random initial guess solution and attempting to find one that is the best under some criteria and conditions, [2].

In GA, a gene is presented by a real number or bit, and a chromosome is a set of genes. A population is a set of chromosomes which are produced in different generations. Operations are defined for a population as follows.

- Selection operation selects the best chromosomes from the population for the next generation.
- Crossover operation crosses two chromosomes to create two new chromosomes for the population.
- Mutation operation mutates a chromosome into a new chromosome by inverting randomly selected genes of the chromosomes. [2] is presented a GA in pseudo code as follows.

N = number of generations

$P_k = \{x_i: x_i \in R^n, i=1... I_k\}$ is a set of chromosomes in the k-th generation, and I_k denotes the size for k = 1, ..., N

Algorithm "Genetic Optimizer"

Create an initial population $P_1 = \{x_i: x_i \in R^n, i=1... I_1\}$.

For (k = 1; k<=N; k++) {

If a good enough solution $x \in P_k$ *is found then* $x^* = x$; *exit*

Else {

- *Crossover operation for a number of chromosomes of population* P_k. *This leads to an enlargement of* P_k.
- *Mutation operation on the chromosomes that were created during crossover.*
- *Select a set of best chromosomes from population* P_k *to population* P_{k+1} *for the next generation.*

2.2. *Presentation of IPs in GAs*

Let consider an IPs:

maximize P = CX

Subject to: AX = b

X are integers and X>=0

Where X= (x_1, x_2... x_n), $A \in R^{mxn}$ and b= (b_1, b_2... b_m)

Example 1: Maximize P = $8*x_1 + 9*x_2 + 10*x_3 + 9*x_4 + 12*x_5 + 11*x_6$

Subject to:

$x_1 + x_2 + x_3 <= 550$

$x_4 + x_5 + x_6 <= 700$

$x_1 + x_4 <= 390$

$x_2 + x_5 <= 460$

$x_3 + x_6 <= 370$

$x_1, x_2, x_3, x_4, x_5, x_6$ integers

$x_1, x_2, x_3, x_4, x_5, x_6 >= 0$

2.2.1. Genes and Chromosomes, population

[1] proposes a method to present genes and chromosomes as follows. A gene corresponds to a variable x_i, a chromosome is a set of x_i = X and its fitness is assigned by the value of object function. In addition a population is a set of chromosomes which is often presented in an array of chromosomes. The gene is presented in an array of bits, thus the chromosome is also an array of bits with its size = n * sizeof (gene). Using bits to present genes make for crossover and mutation operations more "active", because this way can generate new numbers by swapping or inverting bits in operations.

Example 2: from example 1 we have six genes x_1, x_2... x_6. Each x_i is an array of bits which size is calculated by assign "zero" the others. Let's take x_1 variable as an example

$x_1 + 0 + 0 <= 550$

$x_1 + 0 <= 390$

From this we can see that x_1 is no more than 390 and it means that x_1 can be presented in 9 bits (because 390 is 110000110 in binary). Calculating number of bits for others we get:

$x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 9$ bits

And a chromosome is an array of bits with it size = 9 *6 = 54 bits. To visualize, we present a gene and a chromosome in figure 1.

Gene

Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8	Bit 9

And Choromosome

Bit 1	..	Bit 9	Bit 1	..	Bit 9	Bit 1	..	Bit 9	Bit 1	..	Bit 9	Bit 1	..	Bit 9

Fig 1: An example of a gene and chromosome

2.2 2 Fitness value

The fitness value is an important factor in GAs. It defines a score which gives each chromosome the probability to be chosen for breeding or to live. [1] proposes the fitness value based on equation 1 as follows.

$$fitness\_value = \begin{cases} P(X) - \sum_{i=1}^{m} MCi * VCi(X) & if\ P(X) > \sum_{i=1}^{m} MCi * VCi(X) \\ 0 & otherwise \end{cases} \quad (1)$$

Where P(X) is value of chromosome X from object function P.

MC_i is measure of importance of i-th constraint.

$VC_i(X)$ is value of X at i-th constraint and it is calculated in equation 2

$$VCi(X) = \begin{cases} Ai(X) - bi & ifAi(x) > bi \\ 0 & otherwise \end{cases} \quad (2)$$

Example 3 from example 1, suppose X= (1,1,1,1,1 1) and MC_i = 20 for each constraint, the value of $VC_i(X)$:

$VC_1(X) = (1 + 1 + 1 - 550 > 0)? (1 + 1 + 1 - 550): 0 = 0$

$VC_2(X) = (1 + 1 + 1 - 700 > 0)? (1 + 1 + 1 - 700): 0 = 0$

$VC_3(X) = (1 + 1 - 390 > 0)? (1 + 1 - 390): 0 = 0$

$VC_4(X) = (1 + 1 - 460 > 0)? (1 + 1 - 460): 0 = 0$

$VC_5(X) = (1 + 1 - 370 > 0)? (1 + 1 - 370): 0 = 0$

And P(X) = 8 + 9 + 10+ 9 + 12 + 11 = 59, thus fitness_vlaue = 59

2.2.3. Selection, crossover, and mutation operations

2.2.3.1 Crossover operation

Inputs of this operation are two chromosomes in the population and outputs are two new chromosomes made by crossing between two input chromosomes. [1] is proposed in a simplest form which chooses a random point and let's slice each chromosome into two parts and exchange those parts to generate two new chromosomes. The operation is visualized by figure 2.

Inputs: Chromosome 1 & Chromosome 2

Chromosome 1

Bit 1	..	Bit 9	Bit 1	..	Bit 9	Bit 1	..	Bit 9	Bit 1	..	Bit 9

Chromosome 2

Bit 1	..	Bit 9	Bit 1	..	Bit 9	Bit 1	..	Bit 9	Bit 1	..	Bit 9

Outputs: Chromosome 3 & Chromosome 4

Chromosome 3

Bit 1	..	Bit 9	Bit 1	..	Bit 9	Bit 1	..	Bit 9	Bit 1	..	Bit 9

Chromosome 4

Bit 1	..	Bit 9	Bit 1	..	Bit 9	Bit 1	..	Bit 9	Bit 1	..	Bit 9

Figure 2: An example of crossover operation

2.2.3.2 Mutation operation

A mutation operation is also a method to create a new chromosome from another chromosome in the population. The new chromosome bases on changing value at a bit from 0 to 1 or otherwise. In [1], the position doing the mutation operation is a random point. This operation is visualized by figure 3.

Chromosome

Bit 1	..	Bit 9	*Bit Inv*	..	Bit 9	Bit 1	..	Bit 9	Bit 1	..	Bit 9

Figure 3: An example of mutation operation

Where Bit Inv is a bit which is chosen to mutate.

2.2.3.3 Selection operation

After doing crossover and mutation operations, a new population is created and next operation will select chromosomes continued alive or eliminable in next generation. [1] proposes a select operation based on the Roulette works by calculating probability of a chromosome to be selected based on its fitness and sum of all fitness of the population and then this probability is compared with a random value for the selection. This operation is visualized by figure 4.

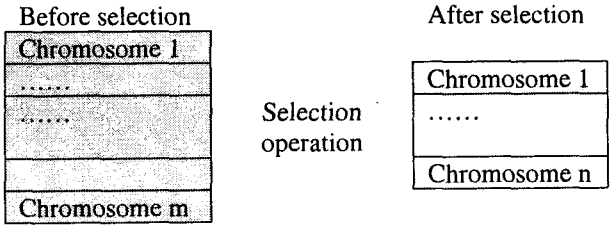

Figure 4: An example of selection operation

2.3. Flowchart of GAs using in IPs

The figure 5 is a presentation of the algorithm genetic optimizer in section 2.1.

3. DISCUSSIONS AND PROPOSES

GAs is a heuristic method to search an optimization. Sometimes, the method can be obtained the best value in several generations but also it has to pass many generations to obtain the best value. In this section, we will compare the method using GAs for IPs with others such as the most popular Branch Bound Algorithm for a visual theory and experiments proving these statements will be presented in section 4. Moreover, several limitations of [1] will be presented in company with proposing solutions to upgrade them.

3.1. Comparison between solving IPs by GAs and other algorithms

3.1.1 *Advantages*

Using GAs is a natural thinking which everybody can understand easy. In addition, GAs can avoid an explosion in branching on the tree which is a big limitation of Branch & Bound Algorithm. Moreover, its calculations are not too complicated in the approach finding a solution. Finally, GAs can be applied to solve others such as linear programming and non linear programming problems.

3.1.2 Disadvantages

Because GAs is a heuristic method to search an optimization, sometimes the method can obtain the best value after many generations and this value is only an approximation of the global solution. In addition, the answer of the question "How many generations can do to obtain the global solution?" does not have accuracy respond.

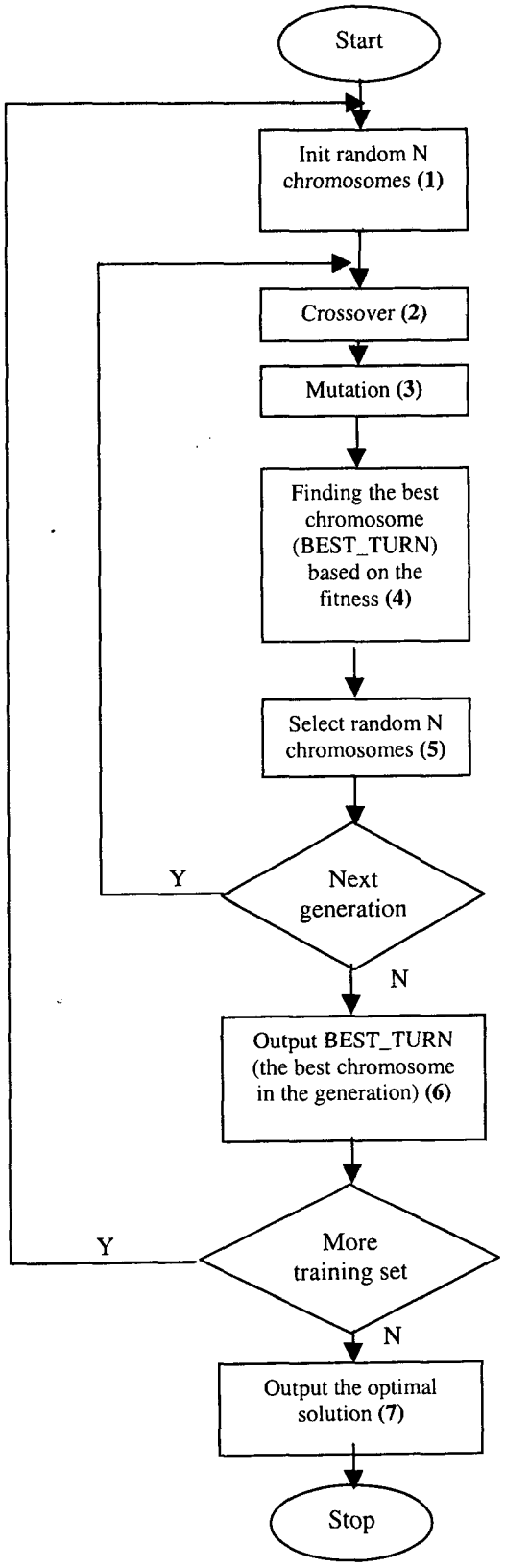

Figure 5: Flowchart of GAs using in IPs

3.2. Limitations of [1] and proposes

[1] has some limitations which affect converging speed to the optimal solution as follows. In mutation operation, chromosomes are not mutated at different positions, instead of this is only at a position corresponding to a variable. It means that only value of a variable in the chromosome is adjusted and others are not changed. It affects the converging speed to the optimal solution. To upgrade, we propose mutation at all variables and an algorithm for the mutation operation as follows.

For each x_i do
Position = Random (0, BIT-1)
// BIT is number of bits of x_i
Mutation (x_i, Position)

End for

In addition, the selection operation in [1] is not based on the best fitness value of the last generation to eliminate. Instead of this randomizes chromosomes selected for the next generation. Consequently, chromosomes selected may not be the best fitness value. This leads to decrease of the converging speed to the optimal solution. To upgrade, we propose an algorithm for the selection operation as follows

Step 1: Sort chromosomes bases on the fitness value

Step 2: Select N chromosomes from step 1 in order

The new algorithm selects chromosomes having the best fitness values. This works will be done by arranging chromosomes based on fitness values and select from first n chromosomes.

4. EXPERIMENTS

In this section, the paper will present computational results from running three different programs: Lingo Ver8.0 with Branch & Bound Algorithm, [1] and the upgrades using GAs. First, the paper will show a set of problems using in experiments in section 4.1 and then evaluations based on these problems present in section 4.2.

4.1. Problems

Problems in experiments come from sample files of Lingo in Table 1.

Name	Number of variables	Number of constraints
CAPLOC	15	8
IFCOS	25	35
JOBLST	7	8
MPSCHD	67	44
QASGN	48	44
SONGS	7	2
STAFFPTR	14	7
INTPTD	13	14
SAMPLE2	4	3

Table 1: Problems

4.2. Evaluations

Experiments will be deployed on a computer with configuration as follows.

Processor : P4 266MHZ
RAM : 1GB

Problems have results in execution time (in seconds), optimal solution and number of iterations in Table 2, 3 and 4 corresponding to the Branch & Bound algorithm in Lingo Ver8.0, [1] and the upgrades respectively.

Table 2: Solving Problems use Lingo Ver8.0

Name	Execution time (in second)	Optimal value	#Iterations
CAPLOC	2	327	22
IFCOST	3	150	9
JOBLST	1	19	0
MPSCHD	8	5385	42
QASGN	5	760	20
SONGS	1	5	0
STAFFPTR	2	0	0
INTPTD	>42	Infeasible	
SAMPLE2	5	Local optimal: 14750	1965

Table 3: Solving Problems use GAs in [1]

Name	Execution time (in second)	Optimal value	#Iterations
CAPLOC	8	320	150
IFCOS	11	139	150
JOBLST	5	19	150
MPSCHD	30	5340	150
QASGN	23	738	150
SONGS	5	5	150
STAFFPTR	8	0	150
INTPTD	42	1570	250
SAMPLE2	536	Global optimal: 14776	1965

Table 4: Solving Problems use the upgrades

Name	Execution time (in second)	Optimal value	#Iterations
CAPLOC	6	326	150
IFCOS	9	146	150
JOBLST	5	16	150
MPSCHD	27	5383	150
QASGN	20	758	150
SONGS	5	5	150
STAFFPTR	8	0	150
INTPTD	41	1800	250
SAMPLE2	514	Global optimal: 14792	1965

From above Table at column "Execution time", we see that Lingo gets results faster than [1] and the upgrades. However in INTPTD problem, Lingo gives an infeasible result while [1] and the upgrades find an integer solution. Moreover in SAMPLE2 problem, Lingo obtains a local optimal solution after 1965 iterations while [1] and the upgrades obtain a global optimal solution after 1965 iterations. Thus GAs is able to find a solution that satisfies integrality constraints if we do not limit to the execution time.

Moreover in column "optimal value" from Table 2, 3 and 4, we can see that the upgrades obtain an approximating optimal solution closer the global optimal solution than [1] in same number of iterations and execution time of the method is faster than [1].

5. CONCLUSION

Although the paper's results did not prove better than Lingo, using GAs in IPs opens a new method to solve not only IPs but also linear or nonlinear problems. Results encourage us to find a new method by justify, modify and refine this approach.

6. REFERENCES

[1]. Fabrício Olivetti de França – *"Solving Integer Programming Problems Using Genetic Algorithms"*.

Website
http://ai-depot.com/Articles/48/Programming.html.

[2]. Kimmo Nieminen, Sampo Ruuth – *"Genetic algorithm for finding a good first integer solution for MILP"*.

Website
http://www.doc.ic.ac.uk/research/technicalreports/2003/DTR03-4.pdf.

[3]. Maros, I., Nieminen, K, Ruuth, S – "Genetic algorithm for finding the first integer solution in Branch and Bound algorithms" - 20th IFIP Conference on System Modelling and Optimization, Trier, Germany, July 2001.

[4]. Simon Mardle and Sean Pascoe, *"An overview of genetic algorithms for the solution of optimisation problems"*.

Website
http://www.economics.ltsn.ac.uk/cheer/ch13_1/ch13_1p16.htm.