Genetic Algorithm with Directional Mutation Based on Greedy Strategy for Large-scale 0-1 Knapsack Problems

1 Wei Liu, *2 Gang Liu
1, First Author
Faculty of Computer Science, China University of Geosciences (Wuhan), Wuhan 430074, China, E-mail: fallsnowsakura@qq.com
*2, Corresponding Author
Faculty of Computer Science, China University of Geosciences (Wuhan), E-mail: liugang67@163.com

Abstract

In view of the lack of efficiency or accuracy of solving large-scale 0-1 knapsack problems by the classic genetic algorithm, a directional mutation operator is designed to reduce the probability of resampling in the search process. Meanwhile an initializing operator and an individual correction operator are added to the algorithm to modify individual after every amendment, both of which are combined with the greedy theory. The proposed algorithm uses truncation selection and longest-distance fitness selection in the crossover, takes elite selection strategy combining with steady-state propagation as a correction operation, and uses the common 0-1 exchange mutation of the binary code, but the mutation probability of each bit of the binary string is adaptively modified. Comparison of experiment results of this algorithm and Active Evolution Genetic Algorithm is given based on the scales of 1000, 2000 and 5000. Experiments proved that the improved algorithm with directional mutation based on greedy theory for solving large-scale 0-1 knapsack problems has high accuracy and high efficiency.

Keywords: 0-1 Knapsack Problem, Genetic Algorithm, Greedy Theory, Directional Mutation

1. Introduction

0-1 Knapsack problem is also named subset problem. It is a classic NP-complete problem in the area of computer sciences, and it is widely used in areas of network design, network routing, task scheduling, cargo loading and funds allocation, etc [1-4]. It could be described as: given n objects and one knapsack with capacity c, each object has a weight \(w_i\), and a value \(v_i\) (i=1,2,...,n). In order to fill the knapsack with the selected objects that yield the maximal value and their total weight no larger than c, a solution of selecting the objects is wanted.

At present, most of the algorithms for solving 0-1 knapsack problem could be divided into two categories: the classic exact algorithms and novel artificial intelligence algorithms including genetic algorithms [5-6]. Even though the classic algorithms could be used to find the exact optimal solution, the time and space spending could become tremendous in solving large-scale problems. However, the advantage of genetic algorithm could be prominent, which can search the entire solution space by iterating selection, mutation and recombination of a population of potential solutions, and take the fitness function as the standard to direct the search towards regions of the search space where the average fitness of the population increases [7-8]. When the procedure reaches the termination condition, the approximate optimal solution is found. Excellent genetic algorithms could even find the exact optimal solution.

Because of the random research of genetic algorithms, some points in the solution space are usually resampled [9]. This deficiency is caused by the random mutation operation in the classic genetic operators. In recent years, many improvements are already made on classic Standard Genetic Algorithm (Referred to as SGA) [13-15], like the Genetic Quantum Algorithm from the reference [16] and the Active Evolution Based Genetic Algorithm (Referred to as AEBGA) from the reference [10]. But both of them have some deficiencies: the Genetic Quantum Algorithm doesn't deal with the random of classic genetic algorithm; even though AEBGA proposes the directional mutation, its implementation is too complicated, with too many parameters to control, and this leads to constraint on its performance and application [17-18]. A novel genetic algorithm with directional mutation based on greedy theory for solving large-scale 0-1 knapsack problems is proposed in this paper.
2. The genetic algorithm with directional mutation based on greedy strategy

2.1 The method of directional mutation

In the classic SGA, the mutation operator is random, and it could make the algorithm resample in the solution space [9], that is to say, a same object may be selected many times. This could result in several adverse consequences: (1) the sampled point is near the area of local optimal point, which leads to local optimization and a final solution that is not the global optimal solution; (2) The resampling of a same point could bring useless redundant calculation, and greatly influence the solution efficiency.

The research result of modern biology indicates that organism is not totally negative and passive during the mutation process. It cannot be excluded that organism could take part in the mutation actively to produce directional mutation to adapt the change of environment. So we can introduce this idea to improve traditional genetic algorithms based on the new mutation mechanism.

2.2 Greedy strategy

Greedy algorithm used for solving 0-1 knapsack problem falls into 3 categories: Sort the objects (1) by weight ascending, (2) by profit descending, or (3) by value-to-weight ratio descending, and put them into the knapsack by order till the capacity of knapsack is reached. The third category is more effective than the other two, but none of them could guarantee the optimal solution.

The greedy algorithm is referred to improve the individual correction operator, making it correct the individual operated with focus. This could make the individual get closer to a feasible solution.

Considering the character of 0-1 knapsack problem solution space (each object is either selected or not) and the advantage of greedy strategy, learning from the idea of directional mutation from the reference [10], the Genetic Algorithm with Directional Mutation Based on Greedy Strategy (Referred to as DGGA) is designed, which aims at working out an algorithm with superior solution efficiency and accuracy, and it should be controlled conveniently and implemented easily.

3. Algorithm design and implementation

3.1 Encoding and initialization

The traditional binary code is used to represent the state of every given object, 1 represents selected, and 0 represents unselected. Initialization is based on the method of random initialization. To obtain better initial population quality, the 3 initial individuals are first selected: the one with the highest profit, the one with the least weight, and the one with the highest value-to-weight ratio. Then, other individuals will be selected randomly.

3.2 Individual correction operator based on greedy selection strategy

The feasible solutions of knapsack problems are restricted by the knapsack capacity to a feasible solution region, however the total weight of all the objects is always larger than the capacity, which leads to the individual obtained after initialization or transformation may surpass the feasible solution region and become an unfeasible solution. Generally, this kind of individual has to be abandoned. However, on one hand, to simply abandon this unfeasible individual and generalize another new solution by initialization or re-operation will greatly decrease the algorithm efficiency. On the other hand, if the individual obtained after initialization or transformation possesses a total weight much smaller than the capacity, this individual is not considered to be an excellent solution. If this kind of individual is involved into the search process, the efficiency will also be decreased.

Based on the two situations above, we consider using individual correction operator to correct the individuals after being initialized or transformed in order to reduce the time and space cost. In the SGA the individual correction operator is already used, but it is only to simply delete or add objects randomly, therefore the effect of making the individual approach excellent solution is not obvious.

In view of the deficiency of the individual correction of the SGA, based on the idea that mating selection is performed on nondominated set [11] proposed by Zitzler (2001), the greedy selection
strategy is applied to correct the individual, where the individual with total weight larger than knapsack capacity is called as the superior individual, and the individual with total weight less than the knapsack capacity the inferior individual. For the superior individual, the greedy strategy is used to choose the object which should be removed, and for the inferior individual, the greedy strategy is used to choose the object which should be added [19-20]. Since the number of the objects added or removed is comparatively small, the greedy strategy is using the value-to-weight ratio as its optimal measurement standard. Specific operations are as follows:

1. If the individual is a superior individual, the selected objects are ordered as the individual indicates by their value-to-weight ratio ascending, the objects with comparatively low value-to-weight ratio successively are selected and removed, that is, changing their corresponding bits in the binary string from 1 to 0, till the total weight of all the selected objects smaller than the capacity. This operation is referred to as superior correction.

2. If the individual is an inferior individual, the unselected objects are ordered as the individual indicates by their value-to-weight ratio descending, the objects with highest value-to-weight ratio are selected by sequence while make sure the individual’s total weight not to surpass the capacity, that is, changing their corresponding bits in the binary string from 0 to 1, till the search is over or the knapsack cannot hold more objects. This operation is referred to as inferior correction.

### 3.3 Elite selection strategy and steady-state propagation

DGGA uses truncation selection and longest-distance fitness selection in the crossover. Given the population size $POPSIZE$, we order the individuals of the population by fitness descending, and number them from 0 to $POPSIZE-1$. The individual numbered $i$ ($i<(POPSIZE/2)$) is matching the individual numbered $POPSIZE-1-i$. As for the mutation, the parent individuals are selected by random selection method.

DGGA uses elite selection strategy combining with steady-state propagation (redundant individual removal) as a correction operation.

Elite selection strategy is a common selection strategy, which could guarantee the genetic ability of the excellent individuals in the population. It speeds up the convergence, while it will lead to the premature loss of the population diversity and result in a premature convergence. Therefore, we consider using a correction operation which removes the redundant individuals. The specific operations are: sort the individuals by fitness, and then compare the two consecutive individuals in sequence. If the two are equivalent, then random initialization is used to hash the latter one to the initial solution space of the population. This operation costs certain time, but reduces the probability of local convergence of the population. Certainly, the individuals of the population will not converge to one point, so we must modify the terminal condition: the degree of convergence is not used as terminal condition, but the largest fitness or the largest procession generation is set as the terminal condition.

### 3.4 Directional mutation operator

DGGA uses the common 0-1 exchange mutation of the binary code, but the mutation probability of each bit of the binary string is adaptively modified. Every bit of the individual gene string corresponds an object, 1 represents selected, otherwise 0 represents removed. For object $i$ ($i=1,2,\ldots,n$), the correspondent bits of all the individuals are set a same mutation probability $P_i$ ($0<P_i<1$), that means the correspondent bit of every individual of this object is set to 1 with probability $P_i$, and set to 0 with probability $1-P_i$. Therefore, $P_i$ represents the probability of object $i$ being selected.

First, assumed that object $i$ has a profit $V_i$ and a weight $W_i$, total profit is $\sum V_i$ and total weight is $\sum W_i$, then,

$$P_i = 0.5 + (V_i / \sum V_i) \times (V_i / W_i - \sum V_i / \sum W_i)$$

(1)
In the mutation, all the selected mutation bits of every individual gene string mutate with the correspondent mutation probability $P_i$. If the bit is originally 0, then it is changed to 1 with $P_i$, otherwise it is changed to 0 with $1-P_i$. Every selected mutation individual must mutate once. After the mutation is completed, the new individual is corrected by the superior or inferior correction operator, and a feasible solution is yielded. Then, elite selection is applied to the feasible solutions:

If the new individual is superior to the parent individual, then the correspondent mutation $P_i$ of the mutation bit will be adaptively modified: if it is from 0 to 1, then $P_i = P_i + 0.01$, otherwise $P_i = P_i - 0.01$; If the new individual is inferior to the parent individual, $P_i$ is also adaptively modified: if it is from 0 to 1, then $P_i = P_i - 0.01$, otherwise $P_i = P_i + 0.01$. $P_i$ is always constrained to the range of 0.05-0.95; otherwise if $p_i$ is set to 0 or 1, the mutation operator will lose the random searching ability and deteriorate to a pure directional searching operator.

After many times of mutation, the objects which should be selected will be selected with high probability, and the ones not will be hard to be selected. This reduces the resampling times during the searching process and improves the searching efficiency. The specific operations are as follows:

a. Select the mutation individual $i$ according to the given mutation probability $P_m$.
   b. For the individual $i$’s binary string, determine whether mutation should be applied in sequence from bit 0 to bit $N-1$;
   c. Correct the new individual with superior or inferior correction operators, and obtain a feasible child individual.
   d. For the child individual, apply the elite selection, adaptively modify the mutation probability of the mutation bit, and finally terminate the mutation operation and return.

 Apparently, the adaptive modification of mutation probability is similar to double coding strategy, but the former is easier, more convenient and more efficient. This modification strategy could be applied to other kinds of genetic algorithms.

### 3.5 Crossover operators

DGGA uses a common crossover strategy in the binary code: multiple points’ transposition (Given the size of population $POPSIZE$): First, select a pair of parent individuals for mutation according to the set crossover probability and the selection method of parent individuals. Second, yield a random number $N$, which is the number of points to be exchanged ($1<N<POPSIZE/10$). Next, select $N$ positions from the binary string which numbers its bits from 0 to $POPSIZE-1$ according to the principle that the pair of crossover points is opposite (crossover is meaningless if equivalent). After the transposition, correct the individuals with superior and inferior correction operators. Apply elite selection to the four individuals including the new child individual pair and parent individual pair, and finally get the correspondent individual pair.

### 3.6 Control parameters

In this paper, the DGGA parameters are set as follows:

- The number of the factors of encoding depends on the number of unknown numbers of the problem; the population size $POPSIZE=90$; crossover probability $pCross$ ranges from 0.4 to 0.9; mutation probability ranges from 0.02 to 0.1; terminal condition is the maximal generation restriction, and the terminal generation is set to 300.

### 3.7 Implementation process

Main Flow Chart of DGGA is depicted in Figure 1. The corresponding processing steps of the algorithm are described as follows:

- **Step1**: Initiate the population with random number generation method combined with greedy strategy.
- **Step2**: Correct the new individuals with superior/inferior correction operator, yield the feasible initial population and select the optimal solution.
- **Step3**: Judge if the current generation reaches the restriction generation. If it reaches, terminate the
process. If not, continue processing.

Step4: Execute the crossover operation, and then correct the new individual with superior/inferior correction.

Step5: Execute the mutation operation, and then correct the new individual with superior/inferior correction.

Step6: Execute the adaptive modification on each mutated object’s probability of being selected.

Step7: Order the population’s individuals by fitness, and select the current optimal individual, calculate its solution and compare with the former optimal solution, choose the better one as the current optimal solution.

Step8: Remove the redundant individuals and then turn to Step 3.

4. Data experiments

4.1 Comparison algorithm description

4.1.1 Dynamic programming

The dynamic programming algorithm (Referred to as DP) selected in this paper is the 0-1 Knapsack Algorithm from the reference [12]. Since the time and space complexity of this algorithm is not relevant to the scale of the problem, and the solution is exactly optimal, it is appropriate to choose this algorithm as comparison standard algorithm. The specific idea of this algorithm is described as follows:

**Input:** Given \( n \) objects, object \( i \) has a profit \( b_i \) and a weight \( w_i \) \((i=1,2,...n)\). The knapsack has a capacity \( w \), the total weight of the objects is \( W \).

**Output:** From the array \( B[0\ldots W+1] \) select the biggest value from \( B[0] \) to \( B[w] \) \((0\leq w\leq W)\), that is, the optimal solution regarding that knapsack capacity is \( w \).

The core pseudo-code is as follows:

```plaintext
for w = 0 to W do B[w] = 0
for k = 1 to n do
    for w = W downto \( w_k \) do
        B[w] = \max(B[w], B[w-w_k] + \text{profit}_{\text{of object } k})
```

![Figure 1. Main flow chart of DGGAs](image-url)
if \( B[w-w_j]+b_j > B[w] \) then \( B[w] = B[w-w_j]+b_j \)

The time cost of this dynamic programming 0-1 Knapsack algorithm is \( O(n*W) \), and \( W \) is the total weight of the objects.

### 4.1.2 Active evolution based genetic algorithm

The Active Evolution Based Genetic Algorithm (denoted as AEBGA) is selected as one of the comparison algorithms. This algorithm refers the selective evolution of the creatures in nature, which is the active mutation of creature to adapt to the living environment. Here the idea of active evolution based on directional mutation is applied to genetic algorithm to make the algorithm search process more efficient and more directional. It has many control parameters, complex modification rules, so it is difficult to be implemented or controlled.

### 4.2 Generation of the experimental data

The data generation method is: set weight of the object \( i \) as \( w_i \in [0,4000] \), where \( w_i \) is a random integer. Its value \( v_i \) is a random integer as \( v_i = w_i + [0, 2000] \). Knapsack capacity \( C = \frac{1}{2} \sum w_i \), where \( n \) is the total number of the objects. All the integer data are generated according to these settings. In addition, in order to verify that algorithm is effective to real number, random floating-point numbers, regarding as \( n=1000, \ w_i \in [0,10], \ v_i=w_i+[0, 5] \) (a random integer), knapsack capacity \( C = \frac{1}{2} \sum w_i \), are generated.

### 4.3 The experiment parameter settings and data record

The parameters are set uniformly as followings:

For the small scale problems, the number of the objects should be 100, 200, 500; for the large scale problems, the number of the objects should be 1000, 2000, 5000. The population size \( PPOSSIZE=90 \), initial crossover probability \( pCross=0.5 \), the terminal generation is 300, initial mutation probability \( pMutation=0.05 \), for each problem 30 experiments are done.

Description of the convergence standard: If the error of the solution limits to 0.0001 of the accurate solution, the algorithm is considered to be convergent. For AEBGA, the experimental data comes from the reference [10].

The experiment results of DGGA on small scale problems are shown as Table 1.

<table>
<thead>
<tr>
<th>Scale</th>
<th>DP</th>
<th>Best</th>
<th>Worst</th>
<th>Average</th>
<th>Best</th>
<th>Average Convergence Generation</th>
<th>Average Accuracy Ratio</th>
<th>Average Time Cost</th>
<th>Convergence Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>19005</td>
<td>19005</td>
<td>19005</td>
<td>22.1667</td>
<td>1</td>
<td>1</td>
<td>1.5125</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>36837</td>
<td>36837</td>
<td>36837</td>
<td>37.3</td>
<td>1</td>
<td>1</td>
<td>1.76417</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>90832</td>
<td>90832</td>
<td>90826</td>
<td>61.4667</td>
<td>0.999965871</td>
<td>3.05097</td>
<td>100%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The experiment results of DGGA on large-scale problems are shown as Table 2. For the floating-point knapsack problem with number of objects as 1000 (1000f), the floating-point data should be converted to integer numbers. Then the optimal solution could be yielded by 0-1 knapsack algorithm from the reference [12]. The solution of exact dynamic programming algorithm could be referred directly to prove the high solution accuracy of DGGA, and no other improved algorithms need to be referred.

The experiment results of DGGA are compared with those of AEBGA from the reference [10], shown as the Table 3. Note that the best solution in the reference [10] is viewed as exact optimal solution; however, the best result of approximate algorithms always has some errors from exact optimal solution.
4.4 The analysis of experiment results

Shown as Table 1 and Table 2, DGGA could reach convergence at 100% under all scales of problems, according to the convergence standard. It is proved that DGGA shows high solution accuracy on all kinds of problems.

Though the test data for AEBGA from reference [10] is not the same with that for DGGA, their methods of generating data both come from the reference [13]. Therefore, for large-scale knapsack problems, the average result of many experiments could show the solution accuracy of the two algorithms.

Table 2. Experiment results of large-scale problems

<table>
<thead>
<tr>
<th>Scale</th>
<th>DP</th>
<th>Best</th>
<th>Worst</th>
<th>Average Best</th>
<th>Average Convergence Generation</th>
<th>Average Accuracy Ratio</th>
<th>Average Time Cost</th>
<th>Convergence Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>182437</td>
<td>182430</td>
<td>182430</td>
<td>182434</td>
<td>79.2</td>
<td>0.999983556</td>
<td>3.22063</td>
<td>100%</td>
</tr>
<tr>
<td>2000</td>
<td>358451</td>
<td>358450</td>
<td>358449</td>
<td>358450</td>
<td>113.533</td>
<td>0.99999721</td>
<td>4.30537</td>
<td>100%</td>
</tr>
<tr>
<td>5000</td>
<td>893128</td>
<td>893128</td>
<td>893125</td>
<td>893127</td>
<td>168.833</td>
<td>0.99999888</td>
<td>11.344</td>
<td>100%</td>
</tr>
<tr>
<td>1000f</td>
<td>2230.26</td>
<td>2230.26</td>
<td>2230.22</td>
<td>2230.25</td>
<td>98.733</td>
<td>0.999995516</td>
<td>2.7486</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 3. Comparison of experiment results of DGGA and AEBGA

<table>
<thead>
<tr>
<th>Scale</th>
<th>Algorithm</th>
<th>Average Accuracy Rate</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>DGGA</td>
<td>1</td>
<td>&lt;2s</td>
</tr>
<tr>
<td></td>
<td>AEBGA</td>
<td>0.995036</td>
<td>&lt;3s</td>
</tr>
<tr>
<td>500</td>
<td>DGGA</td>
<td>0.999965</td>
<td>&lt;4s</td>
</tr>
<tr>
<td></td>
<td>AEBGA</td>
<td>0.994277</td>
<td>&lt;9s</td>
</tr>
<tr>
<td>1000</td>
<td>DGGA</td>
<td>0.999983</td>
<td>&lt;4s</td>
</tr>
<tr>
<td></td>
<td>AEBGA</td>
<td>0.997474</td>
<td>&lt;16s</td>
</tr>
</tbody>
</table>

As shown in Table 3, DGGA is superior to AEBGA on average solution accuracy under the problems of which the number of objects is set to 1000 or 2000 or 5000.

For the solution efficiency, due to the uncertainties like the PC used for experiment and the executive environment, the results of experiments could only show the solution efficiency of the two algorithms to some extent. As shown in Table 3, DGGA is superior on average solution efficiency under the three scales of problems.

5. Conclusion

Aiming at the issue of low searching efficiency of classic genetic algorithm, a refined idea of directional mutation operator based on greedy strategy is proposed. Further, in view of the deficiency of each operator, the individual correction operator based on greedy strategy and the strategy of removing duplicate individuals is designed. Experimental results proved that Genetic Algorithm with Directional Mutation Based on Greedy Strategy (DGGA) has superiority on solution accuracy and efficiency. Since there might be minor differences between the optimal solutions and convergence generations of the same large-scale problem, it would be a direction of subsequent research to find how to improve the stability of the algorithm and how to further improve the solution efficiency and accuracy.
6. Acknowledgments

The work is supported by the Fundamental Research Funds for the Central Universities, China University of Geosciences (Wuhan) (No. CUG090109), China Postdoctoral Science Foundation (No. 20080440972), and by National Natural Science Foundation of China (No. 40772196).

7. References

Genetic Algorithm with Directional Mutation Based on Greedy Strategy for Large-scale 0-1 Knapsack Problems
Wei Liu, Gang Liu