Journal of Theoretical and Applied Information Technology
30th September 2019. Vol.97. No 18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

APPLICATION OF FITNESS SWITCHING GENETIC ALGORITHM FOR SOLVING MULTIDIMENSIONAL KNAPSACK PROBLEM

KIM JUN WOO[1]

[1]Dong-A University, Department of Industrial and Management Systems Engineering, South Korea

E-mail: [1]kjunwoo@dau.ac.kr

ABSTRACT

Combinatorial optimization problems with constraints typically have many infeasible solutions, which cannot be used as a final solution. Therefore, metaheuristic algorithms for such problems must be carefully designed so that the infeasible solutions are dealt with appropriately. For example, repair and penalization are well-known feasibility handling approaches for genetic algorithm. However, those conventional approaches are problem-specific, which means that they must be appropriately tailored in order to be applied for solving a specific problem. On the contrary, fitness switching strategy is a general search strategy that can be used to develop genetic algorithms for solving a wide range of combinatorial optimization problems with constraints. Genetic algorithms based on fitness switching strategy need not to be equipped with repair or penalization procedures. Moreover, the strategy enables to utilize the infeasible solutions, typically ignored in conventional genetic algorithms. In order to investigate the usefulness of fitness switching strategy, this paper aims to extend the existing fitness switching strategy and develop a fitness switching genetic algorithm for multidimensional knapsack problem, which is a generalization of classical 0-1 knapsack problem. The experiment results demonstrate that fitness switching strategy can be used to develop effective metaheuristic algorithms for solving combinatorial optimization problems with multiple constraints.

Keywords: *Fitness Switching Strategy, Genetic Algorithm, Infeasible Solution, Multidimensional Knapsack Problem, Metaheuristic*

1. INTRODUCTION

Multidimensional knapsack problem (MKP) is a well-known extension of classical 0-1 knapsack problem (KP), where MKP has two or more resource constraints while KP has only one resource constraint [1][2]. Since MKP is a practical combinatorial optimization problem that has many application domains, it has been well-studied during past decades [3].

Typically, MKP is characterized by a number of items and a number of resources. Let N denote a set of n discrete items, 1, 2, …, n and v_i denote the value of item i (i = 1, 2, …, n). The objective of MKP is to find a subset of N which maximizes total value, where each item consumes m resources and t_j is the total amount of resource j (j = 1, 2, …, m). Let w_{ij} be the amount of resource j consumed by an item i. Then, MKP can be formulated as follows [3][4]:

$$\text{Max. Total Value} = \sum_{i=1}^{n} v_i x_i \quad (1)$$

$$\text{Subject to } \sum_{i=1}^{n} w_{ij} x_i \le t_j,\ j = 1, 2, \ldots, m \quad (2)$$

$$x_i \in \{0, 1\},\ i = 1, 2, \ldots, n \quad (3)$$

For convenience, let us assume that $w_{ij} > 0$ for all i and j. When n is relatively small, the optimal solution of MKP can be obtained by applying exact solution methods such as branch and bound method. However, MKP is a sort of NP-hard combinatorial optimization problem, which is difficult to solve by applying exact solution method when n is large. Therefore, approximate solution methods such as metaheuristic algorithms can be more effective for large MKP [4][5].

Conventional metaheuristic algorithms such as genetic algorithm (GA) [6], tabu search (TS) [7], simulated annealing (SA) [8] and particle swarm (PS) [9] provide stochastic search strategies that can be tailored to specific combinatorial optimization problems. Such strategies are designed to effectively explore the search space of given problems, however, they have to be carefully

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

customized to solve a specific problem. One of the main challenges in applying metaheuristic algorithms to a specific problem is handling the infeasible solutions.

A solution for s specific combinatorial optimization problem is infeasible if and only if it violates one or more constraints. For example, a solution for MKP is likely to be infeasible if it contains too many items, since it may consume too much resources. Moreover, it is straightforward that an infeasible solution cannot be a final solution for given problem. Therefore, they must be handled appropriately during the search procedure of metaheuristic algorithms. Especially, population-based metaheuristic algorithms such as GA and PS have to generate many alternative solutions, so they should be equipped with appropriate feasibility handling procedures [10]. However, traditional feasibility handling procedures are often problem-specific or require non-trivial configurations.

On the contrary, fitness switching (FS) strategy is a recently proposed feasibility handling approach for GA that can be applied to a wide range of combinatorial optimization problems [11]. Hence, this strategy can help to address the feasibility issue of metaheuristic algorithms, however, it had been not applied to MKP, yet. In this context, this paper aims to apply FS strategy to develop fitness switching genetic algorithm (FSWGA) for MKP and investigate the features of FSWGA for combinatorial optimization problems with multiple constraints.

The remainder of this paper is organized as follows: Section 2 provides a brief literature review on GA and MKP. Moreover, the concept of existing FSWGA is also explained in this section. Section 3 explains the search procedures of FSWGA for MKP, which is developed by extension of previous FSWGA for KP. Section 4 demonstrates the experiment results obtained by applying the extended FSWGA to MKP. Finally, Section 5 represents some discussions on FSWGA and concluding remarks.

2. RESEARCH BACKGROUNDS

GA is a sort of population-based metaheuristic algorithm, and it has been successfully applied to a wide range of combinatorial optimization problems during the last decades [5][11][12]. In order to apply GA to a specific combinatorial optimization problem, solutions of given problem must be represented as a simple string called chromosome. Moreover, a chromosome consists of a number of elements called gene, where single gene represents a feature of the associated solution [13]. For n-item KP and MKP, binary encoding scheme that utilizes the chromosomes with n genes indicating the count of a specific item is most widely used [14][15]. In other words, if an item i is included within a solution, the value of i th gene of the associated chromosome is 1, and otherwise, the gene's value is 0. This encoding scheme is also used in this paper.

Standard GA (SGA) is a most popular version of GA, and it is characterized by three genetic operators, selection, crossover, and mutation [13][16]. The objective of selection is to choose a number of solutions in population, which will be included within mating pool. Moreover, the desirability of a solution is evaluated by using fitness function, and the goal of GA is to discover the optimal solution which maximizes or minimizes the fitness function. For example, the total value in (1) or its modification can be used as the fitness function of GA for KP or MKP. Crossover operator is used to create a new solutions (offspring) by recombining the genes of solutions in mating pool (parents). 1-point, 2-point and uniform crossovers are the most well-known crossover operators [17]. On the contrary, mutation is used to slightly modify the offspring in hopes that better solutions would be obtained. Mutation operator must be applied to a very small number of genes, and bit flip mutation is the most widely used one for binary encoding scheme. Note that this mutation operator changes the value of a gene from 0 to 1, and vice versa [18].

Besides the above three genetic operators, additional procedures can be applied to GA, which enables more effective search procedure. For example, feasibility handling procedures are widely used in designing GA for a specific combinatorial optimization problem, and they are classified into two categories, repair and penalization. The former is used to convert the infeasible solutions into feasible ones [19][20], while the objective of the latter is to apply penalty term to the fitness value of infeasible solutions [21][22][23]. The repair and penalization procedures are widely used in previous GAs for MKP [24][25][26][27], however, they are often problem-specific and complex to implement. Moreover, infeasible solutions can contain some genes useful for exploring the search space in effective manner, even though they cannot be final

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

solution of given problem. Such useful information can be lost when repair procedure is applied [11].

On the contrary, FS strategy is a generalized search strategy that can be applied to a wide range of combinatorial optimization problems. This strategy suggests that both feasible and infeasible solutions should be utilized during search procedure, and their fitness values should be evaluated in different manners.

In more detail, FS strategy provides three distinguishing procedures, fitness switching, fitness leveling, and simple local search. Among them, fitness switching is the most important procedure, and it suggests that the fitness values of feasible and infeasible solutions should be evaluated in different manners. That is, a fitness value of a solution s, fitness(s), is computed as follows:

$$\text{fitness}(s) = \text{fitness}^{+}(s), \text{ if } s \text{ is feasible,} \quad (4.1)$$

and

$$\text{fitness}(s) = \text{fitness}^{-}(s), \text{ if } s \text{ is infeasible.} \quad (4.2)$$

Note that $\text{fitness}^{+}(s)$ and $\text{fitness}^{-}(s)$ are the fitness functions for feasible and infeasible solutions, respectively, and fitness switching procedure also suggests that these two functions should be inversely proportional to each other. Let $X(s)$ denote the desirability of solution s. Then, $\text{fitness}^{+}(s)$ and $\text{fitness}^{-}(s)$ should satisfy

$$\text{fitness}^{+}(s) \propto X(s) \propto 1 \,/\, \text{fitness}^{-}(s). \quad (5)$$

Note that the fitness value of infeasible solution should be inversely proportional to the desirability of solution, which is directly proportional to the fitness value of feasible solution. Moreover, $\text{fitness}^{+}(s)$ and $\text{fitness}^{-}(s)$ for maximization problem such as MKP should satisfy

$$\text{fitness}^{+}(s) \geq \text{fitness}^{-}(s). \quad (6)$$

However, if $|\ \text{fitness}^{+}(s) - \text{fitness}^{-}(s)\ |$ is too large, high selection pressure is imposed on the solutions within population, and this can occur premature convergence to local optima. In order to address this problem, FS strategy recommends modifying the initial $\text{fitness}^{+}(s)$ and $\text{fitness}^{-}(s)$ by applying fitness leveling procedure as follows:

$$\text{fitness}^{+}\text{'}(s) = 1 + L \times (\text{fitness}^{+}(s) - \min_{s \in F} \text{fitness}^{+}(s)) \,/\, (\max_{s \in F} \text{fitness}^{+}(s) - \min_{s \in F} \text{fitness}^{+}(s)) \quad (7)$$

and

$$\text{fitness}^{-}\text{'}(s) = (1 - \alpha) \times \text{fitness}^{-}(s) \,/\, \max_{s \in I} \text{fitness}^{-}(s), \quad (8)$$

where

$$F = \{\ s \mid s \in N \text{ and } s \text{ is feasible}\ \} \quad (9)$$

and

$$I = \{\ s \mid s \in N \text{ and } s \text{ is infeasible}\ \}. \quad (10)$$

L in (7) is the scale factor of fitness leveling, which is used to prioritize feasible solutions over infeasible ones, and its value should be larger than or equal to 1. Moreover, α in (8) is the location factor of fitness leveling, which is used to slightly decrease the modified fitness value of the best infeasible solution., and $0 < \alpha < 1$.

Similarly with the initial $\text{fitness}^{+}(s)$ and $\text{fitness}^{-}(s)$, the modified fitness values, $\text{fitness}^{+}\text{'}(s)$ and $\text{fitness}^{-}\text{'}(s)$ also satisfy

$$\text{fitness}^{+}\text{'}(s) \geq \text{fitness}^{-}\text{'}(s), \quad (11)$$

since

$$1 \leq \text{fitness}^{+}\text{'}(s) \leq L \quad (12)$$

and

$$0 \leq \text{fitness}^{-}\text{'}(s) \leq (1 - \alpha) \quad (13)$$

Consequently, FSWGA uses the modified fitness values, $\text{fitness}^{+}\text{'}(s)$ and $\text{fitness}^{-}\text{'}(s)$, in evaluating the solutions within population.

In addition, simple local search procedure is used to slightly modify the infeasible solutions in hopes that it may become feasible. Note that the role of this procedure is similar with traditional mutation operator, however, simple local search procedure is applied to only infeasible solutions.

Initially, FS strategy was developed to solve combinatorial optimization problems with rare feasible solutions, where it is hard to find any feasible solutions from scratch [11][28]. More

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

recent research paper has demonstrated that FSWGA can be applied to combinatorial optimization problems with many feasible solutions such as classical 0-1 KP, which has a single resource constraint [29]. However, FSWGA has been not applied to combinatorial optimization problems with many feasible solutions and multiple constraints. In this context, this paper aims to extend the existing FS strategy to solve MKP.

3. FITNESS SWITCHING GENETIC ALGORITHM FOR MULTIDIMENSIONAL KNAPSACK PROBLEM

3.1 Fitness Switching Strategy

In this paper, previous FSWGA for classical KP is extended to solve MKP with n items and m resource constraints. However, the fitness leveling and simple local search procedures of FSWGA for KP can be also applied to MKP. Hence, this section focuses on developing a refined fitness switching procedure for MKP.

The objective of MKP is to maximize the total value. Hence, total value is used as fitness value of feasible solutions in this paper. That is,

$$\text{fitness}^{+}(s) = \sum_{i=1}^{n} v_i x_i. \quad (14)$$

On the contrary, total value of infeasible solutions should be decreased by deleting some items, in order to convert it into a feasible solution. In this context, this paper proposes a basic alternative for fitness$^{-}(s)$ as follows:

$$\text{fitness}^{-}_1(s) = 1 / \sum_{i=1}^{n} v_i x_i \quad (15)$$

Also, it can be said that an infeasible solution can be converted into a feasible solution by decreasing its resource consumptions. Since MKP has multiple resource constraints, this paper proposes a second alternative for fitness$^{-}(s)$ as follows:

$$\text{fitness}^{-}_2(s) = 1 / \sum_{j=1}^{m} c_{rm}, \quad (16)$$

where consumption of jth resource,

$C_j = (w_{1j}x_1 + w_{2j}x_2 + \ldots + w_{nj}x_n) /$
$(w_{1j} + w_{2j} + \ldots + w_{nj})$, if s contains one or more items, (17.1)

and

$C_j = 1$, if $x_1 = x_2 = \ldots = x_n = 0$. (17.2)

Note that fitness$^{-}_2(s) = 1 / m$ when $x_1 = x_2 = \ldots = x_n = 1$ or $x_1 = x_2 = \ldots = x_n = 0$. Otherwise, fitness$^{-}_2(s)$ is larger than $1 / m$, and especially, a solution s that contains a small number of items tend to have larger fitness$^{-}_2(s)$. Moreover, it is not guaranteed that fitness$^{-}_2(s) \leq 1$. Therefore, fitness leveling procedure must be applied to this type of fitness function before it is used to evaluate the fitness values of the solutions within population.

In addition, infeasibility of a solution occurs due to its exceeding one or more t_j s. Taking this into account, this paper proposes third and fourth alternatives for fitness$^{-}(s)$ as follows:

$$\text{fitness}^{-}_3(s) = 1 / V(s) \quad (18)$$

and

$$\text{fitness}^{-}_4(s) = \prod_{j=1}^{m} E_j \quad (19)$$

where $V(s)$ is the number of resource constraints violated by a solution s and excess consumption of resource j, E_j is defined as follows:

$$E_j = (\sum_{i=1}^{n} w_{ij} - \sum_{i=1}^{n} w_{ij} x_i) /$$

$$(\sum_{i=1}^{n} w_{ij} - t_j), \text{ if } \sum_{i=1}^{n} w_{ij} x_i > t_j \quad (20.1)$$

and

$E_j = 1$, otherwise. (20.2)

The fitness$^{-}_3(s)$ simply indicates that an infeasible solution that violates a larger number of resource constraints should have smaller fitness value.

On the other hand, fitness$^{-}_4(s)$ utilizes more sophisticated information on the resource consumption of s. In the right hand side of (20.1), the denominator is the minimum unconsumed amount of resource j, while the numerator is the actual unconsumed amount of resource j. In other words, E_j in (20.1) is the proportion of the unconsumed amount of resource j, and fitness$^{-}_4(s)$ suggests that an infeasible solution with larger E_j s should have larger fitness values. Note that both fitness$^{-}_3(s)$ and fitness$^{-}_4(s)$ are smaller than or equal to 1.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In this paper, the extended FSWGAs for MKP are developed by applying fitness$^{-}_{1}(s)$~ fitness$^{-}_{4}(s)$, and their performances are compared by conducting numerical experiments.

3.2 Design of Genetic Algorithm

The proposed fitness switching procedure can be easily applied to the conventional SGA algorithm. The FSWGA in this paper uses binary encoding scheme and the solutions within the initial population are randomly generated, similarly with the existing FSWGA for KP.

```
01: single fitness_switching(int[] s, int n, int m, int type)
02: {
03:   SET weight[][] = array of weights
04:   SET value[] = array of values
05:
06:   SET cst[] = array of resource consumptions
07:   SET initial_fitness = 0
08:   SET t[] = array per bounds of resource
                consumptions
09:
10:   FOR j = 1 TO m
11:     SET cst[j] = getConsumption(s, n, weight, j)
12:   NEXT j
13:
14:   IF isFeasible(cst, t, m) THEN
15:     SET initial_fitness = getValue(s, n, value)
16:   ELSE IF type = 1 THEN
17:     SET initial_fitness = 1/getValues(s, n, value)
18:   ELSE IF type = 2 THEN
19:     SET r_consumption = 0
20:
21:     FOR j = 1 TO m
22:       r_consumption += conRatio(s, n, weight, j)
23:     NEXT j
24:
25:     SET initial_fitness = 1 / r_consumption
26:   ELSE IF type = 3 THEN
27:     SET initial_fitness = 1 / vCount(cst, t, m)
28:   ELSE IF type = 4 THEN
29:     SET r_excess = 1
30:
31:     FOR j = 1 TO m
32:       r_excess = r_excess × excessRatio(s, n,
                  weight, t, j, cst[j])
33:     NEXT j
34:
35:     SET initial_fitness = r_excess
36:   END IF
37:
38:   RETURN initial_fitness
39: }
```

Figure 1: Fitness Switching Procedure

Next, the fitness values of the solutions are computed by applying the proposed fitness switching procedure, which is summarized in figure 1. Note that the procedure in figure 1 is used to compute the fitness value of an individual solution *s*. Moreover, the integer variable *type* indicates that fitness$^{-}_{type}(s)$ is used to obtain the initial fitness values.

As shown in line 10~12 of figure 1, the resource consumptions of given solution *s* are computed by using getConsumption() procedure, which is described in figure 2. Then, isFeasible() procedure in line 14 of figure 1, which is described in figure 3, is used to evaluate the feasibility of the solution *s*. If s satisfies all resource constraints, it is a feasible solution and its total value is used as the initial fitness value, as shown in line 15 of figure 1. The getValue() procedure described in figure 4 is used to compute the total value of a given solution *s*.

```
01: single getConsumption(int[] s, int n, int[][] weight, int
j)
02: {
03:   SET consumption = 0
04:
05:   FOR i = 1 TO n
06:     SET consumption += s[i] × weight[i][j]
07:   Next i
08:
09:   RETURN consumption
10: }
```

Figure 2: getConsumption Procedure

```
01: boolean isFeasible(single[] cst, int[] t, int m)
02: {
03:   SET feasible = TRUE
04:
05:   FOR j = 1 TO m
06:     IF cst[j] > t[j] THEN
07:       SET feasible = FALSE
08:     END IF
09:   Next i
10:
11:   RETURN feasible
12: }
```

Figure 3: isFeasible Procedure

```
01: single getValue(int[] s, int n, int[] value)
02: {
03:   SET value = 0
04:
05:   FOR i = 1 TO n
06:     SET value += s[i] × value[i][j]
07:   Next i
08:
09:   RETURN value
10: }
```

Figure 4: getValue Procedure

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

On the other hand, the initial fitness value of s is computed by using $\text{fitness}^{-}_1(s)$~ $\text{fitness}^{-}_4(s)$, as shown in line 16~35 of figure 1.

Figure 5~7 represent the additional procedures used to compute the initial fitness value of an infeasible solution. Among them, conRatio() procedure in figure 5 is used to compute the consumption of resource j, C_j in (17.1)~(17.2), which is in turn used to calculate $\text{fitness}^{-}_2(s)$ in (16). vCount() procedure in figure 6 counts the number of resource constraint violated by given solution s, which is used to compute $\text{fitness}^{-}_3(s)$ in (18).

The role of excessRatio() procedure summarized in figure 7 is to compute the excess consumption of resource j, E_j in (20.1)~(20.2). Moreover, E_j computed by excessRatio() procedure is in turn used to calculate $\text{fitness}^{-}_4(s)$ in (19). Consequently, the initial value of an arbitrary solution s can always be determined appropriately, by using the fitness switching procedure in figure 1.

After the initial fitness values are computed, simple local search procedure is applied to the solutions within the population, in order to slightly modify the infeasible solutions. Subsequently, fitness leveling procedure is applied to the solutions in order to adjust the initial fitness values.

In addition, conventional selection, crossover, and mutation operators are used to develop FSWGA for MKP. Above all, FSWGA proposed in this paper uses well-known roulette wheel selection operator to generate mating pool from the existing solutions. Note that the probability with which a solution s is selected by roulette wheel selection operator is proportional to the fitness value of the solution.

There are several elementary crossover operators that can be used to develop GAs based on the binary encoding scheme. Among them, the well-known uniform crossover is adopted by the proposed FSWGA for MKP. In order to create two offspring, offspring1 and offspring2, from two existing solutions, parent1 and parent2, uniform crossover operator generates random binary numbers for each gene. Moreover, the values of genes in offspring are determined according to the generated random binary number. For example, if a random binary number for ith gene (i = 1, 2, …, n) is 0, offspring1 inherits the ith gene of parent1, while offspring2 inherits the ith gene of parent2. On the other hand, if a random binary number for ith gene is 1, the ith gene of parent2 is inherited to offspring1, and ith gene of parent1 is inherited to offspring2.

```
01: single conRatio(int[] s, int n, int[][] weight, int j)
02: {
03:   SET actual_consumption = 0
04:   SET max_consumption = 0
05:   SET n_item = 0
06:
07:   FOR i = 1 TO n
08:     SET actual_consumption += weight[i][j]×s[i]
09:     SET max_consumption += weight[i][j]
10:
11:     IF s[i] = 1 THEN
12:       n_item++
13:     END IF
14:   NEXT i
15:
16:   IF n_item = 0 OR n_item = n THEN
17:     RETURN 1
18:   ELSE
19:     RETURN actual_consumption /
                  max_consumption
20: }
```

Figure 5: conRatio Procedure

```
01: int vCount(single[] cst, int[] t, int m)
02: {
03:   SET n_violation = 0
04:
05:   FOR j = 1 TO m
06:     IF cst[j] > t[j] THEN
07:       n_violation++
08:     END IF
09:   NEXT j
10:
11:   RETURN _violation
12: }
```

Figure 6: vCount Procedure

```
01: int excessRatio(int[] s, int n, int[][] weight, int[] t, int
j, single the_cst)
02: {
03:   SET total_weight = 0
04:   SET eRatio = 0
05:
06:   FOR i = 1 TO n
07:     total_weight += weight[i][j]
08:   NEXT i
09:
10:   IF the_cst <= t[j] THEN
11:     SET eRatio = total_weight – the_cst /
                    total_weight – t[j]
12:   END IF
13:
14:   RETURN eRatio
15: }
```

Figure 7: excessRatio Procedure

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The last genetic operator of SGA, mutation is used to randomly modified the offspring obtained by applying crossover operator. This operator is important in that it helps to maintain the diversity of the solutions within population. In other words, GA without mutation operator is more likely to converge to local optima. However, if mutation is too often applied, the useful information accumulated during previous search procedure cannot be exploited appropriately. Therefore, mutation operator is applied with very small probability, and this paper uses the well-known bit flip mutation to develop FSWGA for MKP.

From the practical perspectives, the proposed FSWGA has several benefits as follows: (i) Fitness switching, fitness leveling, and simple local search procedures are not associated with conventional genetic operators, selection, crossover, and mutation, since those procedures of FSWGA can be incorporated into the evaluation procedure of GA. This means that the conventional genetic operators are directly used in FSWGA without any modification. (ii) No problem-specific feasibility handling procedures such as repair and penalization are required in FSWGA. (iii) In order to develop FSWGA, the three procedures, fitness switching, fitness leveling, and simple local search, must be implemented. Among them, however, fitness leveling is not problem-specific, and we need not to customize this procedure to solve a specific combinatorial optimization problem. On the other hand, fitness switching and simple local search procedures are problem-specific, but the latter is inherently easy to design and implement. For example, FSWGA for MKP proposed in this paper uses a simple local procedure that is very similar with conventional bit flip mutation operator.

Consequently, FSWGA can be developed in very efficient manner, if appropriate fitness switching procedure is provided. Note that fitness switching procedure is the key element of FSWGA, and it is problem-specific. Especially, $\text{fitness}^{-}(s)$ should be carefully designed to solve given combinatorial optimization problem in an effective manner. In other words, the most important step in developing FSWGA is to define $\text{fitness}^{-}(s)$. Of course, $\text{fitness}^{-}(s)$ can be simply defined as a reciprocal of $\text{fitness}^{+}(s)$, as shown in (15), and this is the original concept of fitness switching. However, if such simple $\text{fitness}^{-}(s)$ does not work well, we have to develop alternative approaches for calculating $\text{fitness}^{-}(s)$, such as $\text{fitness}^{-}_{2}(s)$~ $\text{fitness}^{-}_{4}(s)$ in this paper.

4. EXPERIMENT RESULTS

Table 1: MKP With 50 Items

ID	Resource1	Resource2	Resource3	Value
1	19	22	39	66
2	10	22	19	60
3	16	24	34	21
4	15	11	13	139
5	13	22	15	65
6	16	15	17	95
7	19	12	31	6
8	11	10	12	91
9	11	18	21	12
10	19	11	24	85
11	10	29	23	12
12	10	11	11	53
13	20	26	40	99
14	14	16	32	48
15	18	27	36	96
16	15	13	38	81
17	15	29	20	70
18	11	24	13	122
19	17	16	28	83
20	19	19	17	129
21	11	14	26	54
22	17	28	36	31
23	20	29	33	52
24	19	15	29	96
25	17	21	17	57
26	19	18	33	45
27	16	12	35	16
28	10	24	38	130
29	17	30	35	58
30	15	22	13	37
31	15	21	27	116
32	14	22	20	64
33	20	29	14	43
34	17	19	14	108
35	19	18	13	75
36	14	14	23	121
37	13	11	35	67
38	17	17	25	60
39	18	13	33	33
40	15	30	34	48
41	17	30	23	98
42	18	27	15	91
43	18	12	24	102
44	20	13	39	120
45	10	11	23	61
46	11	17	28	58
47	16	15	23	119
48	14	15	20	122
49	17	26	32	71
50	10	17	28	64

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The performances of the alternatives fitness$^{-}(s)$ are compared by applying the proposed FSWGA to a MKP with 50 items and 3 resource constraints, and the resource consumptions and values of the 50 items are summarized in table 1. In addition, the descriptive statistics of resource consumptions are summarized in table 2, where we can see that Resouce3 is the most highly utilized resource on average, while the average consumption of Resource1 is smaller than the other 2 resources.

Furthermore, figure 8~10 provide the scatter plots that display the relationships between resource consumptions and item value, and it seems that no resource consumption is highly correlated with the item value.

Table 2: Summary Of Resource Consumptions

	Resource1	Resource2	Resource3
Average	15.47	19.27	25.42
Total	772	967	1271
Min.	10	10	10
Max.	20	30	40

In this paper, the MKP with given items are solved under 3 experiment conditions summarized in table 3, where different experiment conditions have different upper bounds of resource constraints, t_1, t_2 and t_3. Note that $t_1 < t_2 < t_3$ for all experiment conditions.

The numerical experiments have been performed under crossover rate = 0.8, mutation rate = 0.01, $L = 2$ and $\alpha = 0.01$, with varying population size N_p. In addition, the termination condition is defined by using maximum iteration number = 200, and elitism policy is applied so that 5 best solutions within every population are always included within the mating pool. Furthermore, 10 repetitions of experiments have been performed for each case, and the experiment results are summarized in table 4~6. Note that table 4~6 shows the average (Avg.) and standard deviation (St. Dev) of Best-Total-Value, the total value of final solution discovered by FSWGA with a specific fitness switching procedure.

Consequently, we can make following observations: (i) Above all, FSWGA has shown better performances under larger N_p. This indicates that the performance of FSWGA can be further enhanced by modifying the experiment parameters. (ii) The relative performance of FSWGA can vary with experiment conditions. For example, FSWGA with fitness$^{-}_4(s)$ shows relatively good performances under low upper bounds as shown in table 4. However, its performances are degraded under moderate or high upper bounds as shown in table 5~6. On the other hand, FSWGA with fitness$^{-}_1(s)$ shows good performances especially under moderate or high upper bounds and large N_p. (iii) Simple alternatives for fitness$^{-}(s)$ also have a significant competitiveness in terms of the total value of final solution. Among fitness$^{-}_1(s)$~fitness$^{-}_4(s)$, fitness$^{-}_1(s)$ and fitness$^{-}_3(s)$ are easy to compute and implement due to their simple structure. Especially, fitness$^{-}_1(s)$ is just a reciprocal of fitness$^{+}(s)$ for given problem, total value. On the contrary, the other alternatives, fitness$^{-}_2(s)$ and fitness$^{-}_4(s)$ has relatively complex structure.

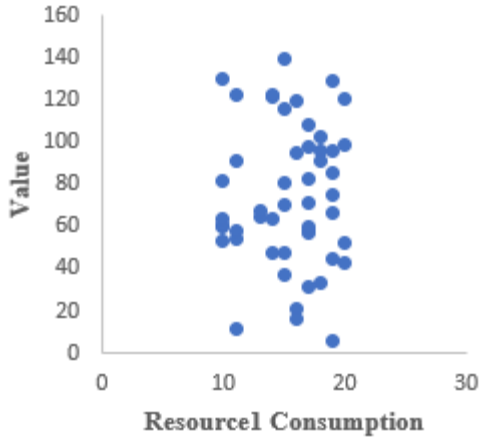

Figure 8: Item Values Versus Resource1 Consumption

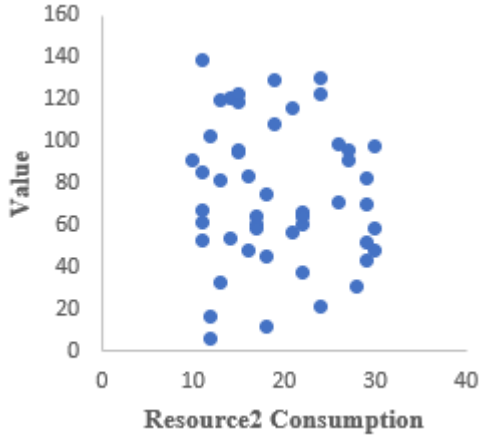

Figure 9: Item Values Versus Resource2 Consumption

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

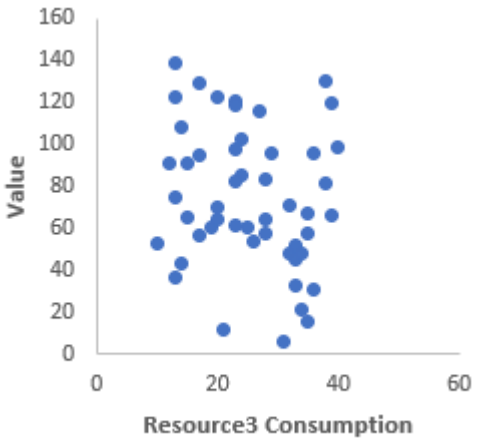

Figure 10: Item Values Versus Resource3 Consumption

Table 3: Upper Bounds of Resource Consumption

Condition	t_1	t_2	t_3
Low	150	175	200
Moderate	300	400	500
High	700	850	1000

Table 4: Best-Total-Values For Low Upper Bounds

		fitness^-_1	fitness^-_2	fitness^-_3	fitness^-_4
N_p=25	Avg.	1160.5	1169.5	**1185.2**	**1185.6**
	St. Dev.	35.74	26.51	13.06	14.17
N_p=50	Avg.	1179.3	**1192.0**	1179.5	**1189.8**
	St. Dev.	28.92	15.00	20.72	13.45

Table 5: Best-Total-Values For Moderate Upper Bounds

		fitness^-_1	fitness^-_2	fitness^-_3	fitness^-_4
N_p=25	Avg.	2091.8	2091.4	**2096.0**	2092.9
	St. Dev.	19.47	23.85	11.89	13.94
N_p=50	Avg.	**2102.6**	2100.7	**2102.7**	2101.6
	St. Dev.	11.97	11.82	9.12	9.37

Table 6: Best-Total-Values For High Upper Bounds

		fitness^-_1	fitness^-_2	fitness^-_3	fitness^-_4
N_p=25	Avg.	3433.3	3422.9	**3445.3**	**3441.6**
	St. Dev.	26.62	21.65	20.26	17.87
N_p=50	Avg.	**3452.2**	3449.9	**3452.5**	3446.6
	St. Dev.	13.31	16.11	10.94	14.66

However, the complex alternatives do not always outperform the simpler ones. This implies that we need not to define too complicated alternatives for $\text{fitness}^-(s)$, in order to develop FSWGA for specific combinatorial optimization problems. (iv) Overall, $\text{fitness}^-_3(s)$, a reciprocal of the number of resource constraints violated by given solution, is the best alternative for $\text{fitness}^-(s)$ in table 4~6, in terms of the best-total-values. Of course, $\text{fitness}^-_3(s)$ may be ineffective for KP, since the classical KP has a single resource constraint. However, the experiment results in this paper suggests that it is useful for solving combinatorial optimization problem with multiple constraints. Moreover, $\text{fitness}^-_3(s)$ is a simple type alternative for $\text{fitness}^-(s)$, which is appropriate for practical use.

Figure 11 shows an example of changes in total value of feasible solutions within population during the search procedure. In addition, this result is obtained from a single repetition of experiment under lower upper bounds. Due to the low upper bounds of resource constraints, FSWGA could not discover any feasible solutions during the early period of search procedure in figure 10. In other words, it is not easy to generate an arbitrary feasible solution from scratch in this case, and therefore, the populations at early iterations consist of only infeasible solutions. Note that both average and maximum values of the total value of feasible solutions are 0 in this figure. However, figure 10 also shows that FSWGA has succeeded in discovering feasible solutions at 11th iteration, and the total values of feasible solutions are continuously improved after this iteration. Therefore, we can conclude that the FS strategy is useful for solving combinatorial optimization problems with multiple constraints, such as MKP, in that FSWGA can effectively generate feasible solutions by utilizing the infeasible solutions within population and improve the fitness values of the feasible solutions in order to discover the optimal solution for given problem.

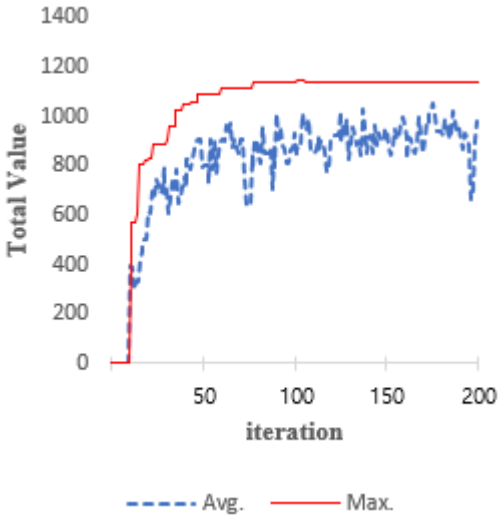

Figure 11: An Example Of Change In Total Value Of Feasible Solutions Under Low Upper Bounds

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

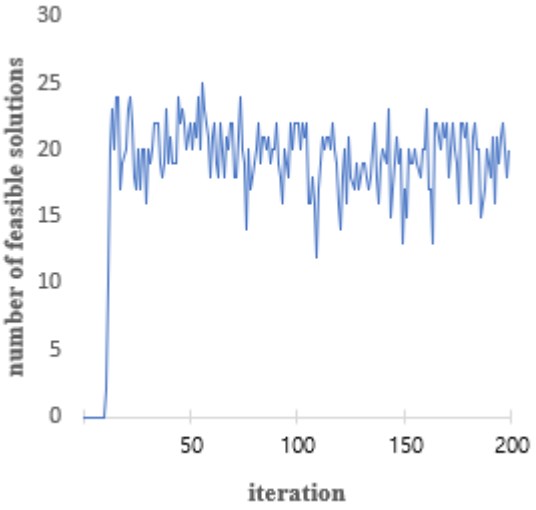

Figure 12: An Example Of Change In Number Of Feasible Solutions Under Low Upper Bounds

As shown in figure 12, the number of feasible solutions within population rapidly increases after some feasible solutions are discovered. However, the number of feasible solutions still fluctuate during the later period in search procedure, as shown in figure 12. It is straightforward that this fluctuation is occurred by the genetic operators, crossover and mutation. Especially, crossover operator is likely to generate infeasible solutions that contain too many items when it is applied to 2 parent solutions that contain considerable numbers of items, even if both of them are feasible solutions. It is possible to avoid such infeasible solutions by using more sophisticated crossover operators that is designed to generate only feasible offspring, however, such operators are also problem-specific and complicated inherently. In other words, designing and implementing crossover operators that do not generate infeasible solutions for a specific combinatorial optimization problem is often a non-trivial task. Moreover, infeasible solutions sometimes can have useful genes for discovering better solutions. Thus, the infeasible solutions can help to maintain the diversity of the solutions within population. In this context, FS strategy can be a very effective feasibility approach for combinatorial optimization problems with constraints.

On the other hand, figure 13 and figure 14 depicts examples of changes in total value of feasible solutions and number of feasible solutions under high upper bounds, which are obtained from another single repetition of experiment. In these figures, we can see that even initial population contains feasible solutions. More exactly, all of the solutions within the initial population are feasible, as shown in figure 14. It is straightforward that an arbitrary feasible solution can be generated more easily under high upper bounds. Therefore, fitness$^{+}(s)$ is used to compute the fitness values of solutions more often in this case. In other words, FSWGA operates similarly with conventional GAs to a certain extent if feasible solutions can be generated easily.

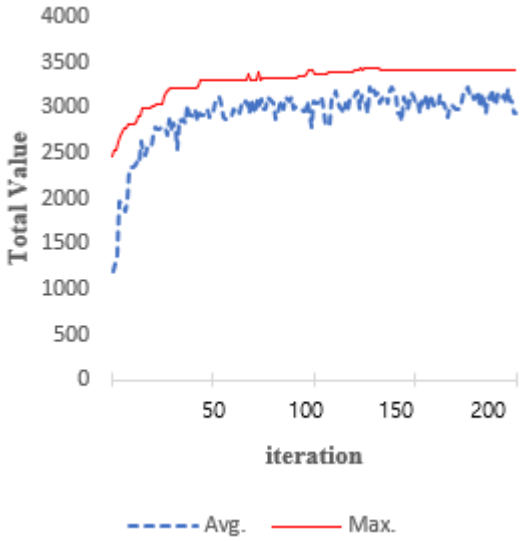

Figure 13: An Example Of Change In Total Value Of Feasible Solutions Under High Upper Bounds

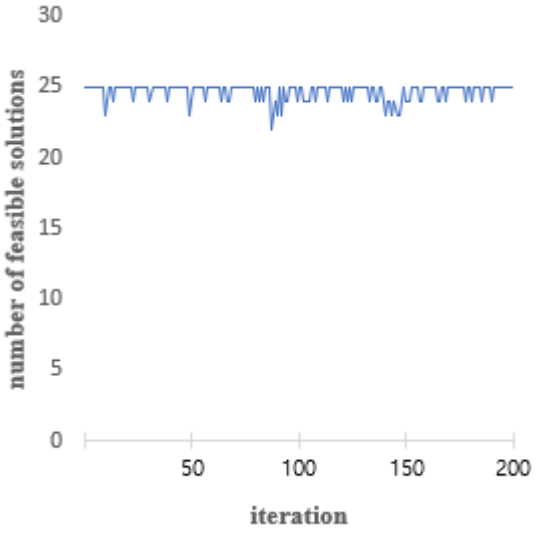

Figure 14: An Example Of Change In Number Of Feasible Solutions Under High Upper Bounds

However, figure 14 shows that the number of feasible solutions also fluctuate during the entire

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

search procedure even if upper bounds of resource constraints are high, and this means that the genetic operators still can generate infeasible solutions. Hence, infeasible solutions must be appropriately handled even if they rarely appear, and experiment results in this paper suggests that FS strategy also can be an effective feasibility handling approach for combinatorial optimization problems with rare infeasible solutions.

In conventional GAs, repair and penalization procedures are widely used to deal with infeasible solutions. However, they are not included within the framework of SGA. Thus, FSWGA is easier to implement than GAs with repair or penalization. Especially, repair procedure is used to convert infeasible solutions into feasible ones, which means that infeasible solutions are not allowed in GAs with repair procedure. On the contrary, FSWGA can maintain diversity in population more effectively in that it allows infeasible solutions to be included within population. Penalization procedure also can be used to handle the infeasible solutions within population, however, it must be carefully configured so that each infeasible solution has appropriate fitness value. In contrast, values of FSWGA's parameters, L and α, can be determined in an ad-hoc manner. Consequently, FS strategy can be an effective approach in solving combinatorial optimization problems with constraints.

5. CONCLUSIONS

Feasibility handling is one of the major challenges in developing metaheuristic algorithms for solving combinatorial optimization problems with constraints. For example, repair and penalization methods are widely used feasibility handling approaches for GA. However, such conventional approaches are often problem-specific or require non-trivial configurations.

On the contrary, FS strategy is a generalized fitness handling strategy, and it is easy to implement. That is, FS strategy can be applied to GAs for a wide range of combinatorial optimization problems without modifying the existing genetic operators. In summary, fitness value of a solution for a combinatorial optimization problem with constraints is typically improved by consumptions of given resources. Inversely, FS strategy suggests that infeasible solutions can be converted to feasible ones by worsening the fitness values. In order to investigate the applicability of FS strategy, this paper developed FSWGA for MKP.

The key element of FS strategy is fitness^{-}(s), which must be carefully defined in developing FSWGA. The original concept of FS strategy suggests that a reciprocal of fitness value of feasible solution can be used as is fitness^{-}(s), however, this paper demonstrated that more complicated alternatives can be defined if required.

Numerical experiments revealed that FSWGA is also useful for solving combinatorial optimization problems with multiple constraints, such as MKP. Moreover, experiment results showed that the complicated fitness^{-}(s) does not always outperform the simple fitness^{-}(s), such as the reciprocals of the total value or the number of violated constraints. Therefore, FSWGA can be designed in a more straightforward manner than other GAs that use conventional feasibility handling approaches.

MKP is a well-known combinatorial optimization problem with multiple constraints. However, the constraints are of same type in that all of them are resource constraints. In this context, the author plans to apply FS strategy to combinatorial optimization problems with multi-type constraints in future. Also, developing simple but more sophisticated alternatives for fitness^{-}(s) can be another valuable future research topic.

ACKNOWLEDGEMENTS

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(Ministry of Science, ICT & Future Planning) (NRF-2017R1C1B1008650).

REFRENCES:

[1] A. Fréville, "The Multidimensional 0-1 Knpsack Problem: An Overview", *European Journal of Operational Research*, Vol. 155, No. 1, 2004, pp. 1-21.

[2] T. Lust, and J. Teghem, "The Multiobjective Multidimensional Knapsack Problem: A Survey and a New Approach", *International Transactions in Operational Research*, Vol. 19, No. 4, 2012, pp. 495-520.

[3] J. Puchinger, G. R. Raidl, and U. Pferschy, "The Core Concept for the Multidimensional Knapsack Problem", *Proceeding of European Conference on Evolutionary Computation in Combinatorial Optimization*, 2006, pp. 195-208.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[4] J. Puchinger, G. R. Raidl, and U. Pferschy, "The Multidimensional Knapsack Problem: Structure and Algorithms", *INFORMS Journal on Computing*, Vol. 22, No. 2, 2010, pp. 250-265.

[5] J. W. Kim, "Candidate Order Based Genetic Algorithm (COGA) for Constrained Sequencing Problems", *International Journal of Industrial Engineering: Theory, Applications and Practice*, Vol. 23, No. 1, 2016, pp. 1-12.

[6] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[7] F. Glover, "Tabu Search for Large Scale TimeTabling Problems", *European Journal of Operational Research*, Vol. 54, No. 1, 1991, pp. 39-47.

[8] S. Kirkpatrick, "Optimization by Simulated Annealing: Quantitative Studies", *Journal of Statistical Physics*, Vol. 34, No. 5-6, 1984, pp. 975-986.

[9] J. Kennedy, and R. Eberhaard, "Particle Swarm Optimization", *Proceedings of IEEE International Conference on Neural Networks*, Vol. 4, 1995, pp. 1942-1948.

[10] Z. Beheshti, and S. M. H. Shamsuddin, "A Review of Population-based Meta-heuristic Algorithms", *International Journal of Advances in Soft Computing and Its Applications*, Vol. 5, No. 1, 2013, pp. 1-35.

[11] J. W. Kim, and S. K. Kim, "Fitness Switching Genetic Algorithm for Solving Combinatorial Optimization Problems with Rare Feasible Solutions", *Journal of Supercomputing*, Vol. 72, No. 9, 2016, pp. 3549-3571.

[12] M. Kumar, M. Husian, N. Upreti, and D. Gupta, "Genetic Algorithm: Review and Application", *International Journal of Information Technology and Knowledge Management*, Vol. 2, No. 2, 2010, pp. 451-454.

[13] D. Whitley, "A Genetic Algorithm Tutorial", *Statistics and Computing*, Vol. 4, No. 2, 1994, pp. 65-85.

[14] Z. Michalewicz, and J. Arabas, "Genetic Algorithms for the 0/1 Knapsack Problem", *International Symposium on Methodologies for Intelligent Systems*, 1994, pp. 134-143.

[15] R. P. Singh, "Solving 0-1 Knapsack Problem Using Genetic Algorithms", *Proceedings of 2011 IEEE 3rd International Conference on Communication Software and Networks (ICCSN)*, 2011, pp. 591-595.

[16] J. Andre, P. Siarry, and T. Dongon, "An Improvement of the Standard Genetic Algorithm Fighting Premature Convergence in Continuous Optimization", *Advances in Engineering Software*, Vol. 32, No. 1, 2001, pp. 49-60.

[17] E. Semenkin, and M. Semenkina, "Self-configuring Genetic Algorithm with Modified Uniform Crossover Operator", *Proceedings of International Conference in Swarm Intelligence*, pp. 414-421.

[18] F. Chicano, A. M. Sutton, L. D. Whitley, and E. Alba, "Fitness Probability Distribution of Bit-Flip Mutation", *Evolutionary Computation*, Vol. 23, No. 2, 2015, pp. 217-248.

[19] P. Chootinan, and A. Chen, "Constraint Handling in Genetic Algorithms Using Gradient-based Repair Method", *Computer and Operations Research*, Vol. 33, No. 8, 2006, pp. 2263-2281.

[20] S. Salcedo-Sanz, "A Survey of Repair Methods Used as Constraint Handling Techniques in Evolutionary Algorithm", *Computer Science Review*, Vol. 3, No. 3, 2009, pp. 175-192.

[21] D. W. Coit, A. E. Smith, and D. M. Tate, "Adaptive Penalty Methods for Genetic Optimization of Constrained Combinatorial Problem", *INFORMS Journal on Computing*, Vol. 8, No. 2, 1996, pp. 173-182.

[22] Ö, Yeniay, "Penalty Function Methods for Constrained Optimization with Genetic Algorithms", *Mathematical and Combinatorial Applications*, Vol. 10, No. 1, 2005, pp. 45-56.

[23] M. Schlüter, and M. Gerdts, "The Oracle Penalty Method", *Journal of Global Optimization*, Vol. 47, No. 2, 2010, pp. 293-325.

[24] A. Hoff, A. Løkketangen, and I. Mittet, "Genetic Algorithms for 0/1 Multidimensional Knapsack Problems", *Proceedings of Norsk Informatikk Konferanse*, 1996, pp. 291-301.

[25] P. C. Chu, and J. E. Beasley, "A Genetic Algorithm for the Multidimensional Knapsack Problem", *Journal of Heuristics*, Vol. 4, No. 1, 1998, pp. 63-86.

[26] M. J. Alves, and M. Almeida, "MOTGA: A Multiobjective Tchebycheff based Genetic Algorithm for the Multidimensional Knapsack Problem", *Computers and Operations Research*, Vol. 34, No. 11, 2007, pp. 3458-3470.

[27] F. Djannaty, and S. Doostdar, "A Hybrid Genetic Algorithm for the Multidimensional Knapsack Problem", *International Journal of Contemporary Mathematical Sciences*, Vol. 3, No. 9, 2008, pp. 443-456.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[28] J. W. Kim, and S. K. Kim, "Genetic Algorithms for Solving Shortest Path Problem in Maze-type Network with Precedence Constraints", *Wireless Personal Communications*, forthcoming.

[29] J. W. Kim, "Application of Fitness Switching Genetic Algorithm for Solving 0-1 Knapsack Problem", *Journal of Theoretical and Applied Information Technology*, Vol. 96, No. 2, 2018, pp. 7339-7348.