International Journal of Innovative
Computing, Information and Control
Volume **9**, Number **9**, September **2013**

ICIC International ©2013 ISSN 1349-4198
pp. 3537–3548

A HYBRID OF ROUGH SETS AND GENETIC ALGORITHMS FOR SOLVING THE 0-1 MULTIDIMENSIONAL KNAPSACK PROBLEM

Hsu-Hao Yang, Ming-Tsung Wang and Chung-Han Yang

Department of Industrial Engineering and Management, M608 Management Building
National Chin-Yi University of Technology
No. 57, Sec. 2, Zhongshan Rd., Taiping Dist., Taichung 41170, Taiwan
yanghh@ncut.edu.tw

Received July 2012; revised December 2012

Abstract. *The multidimensional 0-1 knapsack problem (MKP) is a well-known NP-hard combinatorial optimization problem. This paper uses a methodology that integrates a reduct of rough sets (RS) into the crossover operator of a genetic algorithm (GA) to solve the MKP. Two algorithms are presented in this paper; one selects the crossover points either randomly or via the reduct, whereas the other selects the crossover points solely by the reduct. The performance of these two algorithms was compared with a standard GA using test cases from the literature. According to the experimental results, this integration obtains both better quality and more clustered solutions, and could possibly improve the performance if some mechanisms are developed in the algorithm. The results justify the integration and demonstrate an alternative for improving the performance of the GA.*

Keywords: Multidimensional knapsack problem, Rough sets, Genetic algorithms, Attribute reduction

1. **Introduction.** The multidimensional 0-1 knapsack problem (MKP) can be stated as follows:

$$\text{Maximize} \quad z = \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} r_{ij} x_j \leq b_i, \quad i \in M = \{1, \ldots, m\}, \tag{2}$$

$$x_j \in \{0, 1\}, \quad j \in N = \{1, \ldots, n\}, \tag{3}$$

where n is the number of items, and m is the number of knapsack constraints. Each constraint in (2) is referred to as a knapsack constraint. Each item, $j \in N$, consumes r_{ij} resources in the ith knapsack and generates p_j units of profit. The objective is to choose the subset of items that generates the maximum profit without exceeding the set capacity constraints. According to [1], many practical problems can be formulated as the MKP. For example, given a capital budgeting problem where project j has a profit p_j and consumes r_{ij} units of resource i, the goal is to find a subset of n projects so that the total profit is maximized, and all resource constraints are satisfied.

The MKP is a well-known NP-hard combinatorial optimization problem commonly solved via dynamic programming, branch and bound, and so on. Readers are referred to [2] for a recent survey of the exact methods and heuristics for solving the MKP. Given the intractability of the MKP solutions, this paper extends the work based on [3], which

integrates the *attribute reduction* (interchangeably referred to as the *reduct*) in rough sets (RS) introduced by [4] into the *crossover* operator in the genetic algorithm (GA) introduced by [5]. GAs are evolutionary algorithms (EAs) that have been widely applied and found useful for solving computationally intractable problems, such as NP-complete problems. The RS has the capability to process data sets containing uncertainty, reduce the size of such data sets, and generate decision rules from these data sets. In these data sets, certain attributes may be redundant and can be eliminated without reducing the original classification quality. The process of finding a smaller set of attributes in an RS that ensures the same classification quality is called attribute reduction, and the underlying set is referred to as a reduct.

The technical content in terms of significance and innovation in the paper is stated as follows. First, since the successful implementation of a GA to solve the MKP by [1], GAs have been widely applied to the MKP. Tavares et al. [6] classified the GAs proposed in the literature into two main groups according to their representation and variation operators. Second, Raidl and Gottlieb [7] suggested that the success of EAs mainly depend on the representation of the solution candidates and variation operators used. Varnamkhasti and Lee [8] also suggested that GA performance is dependent on the genetic operators and the type of crossover operator in particular. These two articles highlight the importance of crossover operators. Third, according to [9], the integration of a reduct into the crossover operator appears to provide higher quality and more clustered solutions for 0-1 knapsack problems. Motivated by these three facts, we present two algorithms in this paper to investigate how the integration performs when solving the MKP. On the one hand, we further the area of MKP research by developing several problem solving innovations. On the other hand, our approach primarily differs from the proposed GAs in that we use the reduct notion for the crossover operator. As will be reviewed in more details later, to the best of our knowledge, we are the first to integrate a reduct into a crossover when solving the MKP. Our positive results reveal that this integration is a promising approach for tackling the MKP. In summary, the major contribution of this study is its variation of the crossover operator based on the reduct.

This paper presents two algorithms based on the reduct for the crossover operator. The first algorithm, referred to as CRGA, uses either the attributes obtained from the reduct, or random points if no reduct is found. In contrast, the second algorithm, referred to as SRGA, only uses those attributes obtained from the reduct to crossover, and no crossover occurs if no reduct is found. To investigate the results, we compare the performance of the two algorithms to a standard GA in terms of the solution quality and computational convergence, which will both be defined later. According to the computational results, the integration primarily provides an advantage by producing higher quality and more clustered solutions. This advantage justifies the integration and shows an alternative to standard GAs while improving their performance. However, the algorithms have trouble finding optimal solutions for certain test cases. Despite these not uncommon difficulties in finding optimal solutions for computationally intractable problems, the size of our test cases was limited relative to those reported in the literature. With regard to testing sizes, Khuri et al. [10] only tested a small number of problems, and Thiel and Voss [11] obtained results based on moderately sized test problems.

The remainder of this paper is organized as follows. Section 2 briefly reviews the literature. Section 3 provides background on the standard GA and RS. Section 4 introduces the CRGA and SRGA in more detail. Section 5 compares and discusses the empirical results. Section 6 presents conclusions and future research directions.

2. **Literature Review.** The literature is reviewed, though far from exhaustively, from primarily two perspectives, either (1) EAs are included or (2) exact and/or heuristic methods are used to solve the MKP. We begin by including EAs.

Varnamkhasti and Lee [8] proposed several techniques to tackle premature convergence in GAs by controlling the population diversity. The authors compared the results to those of other genetic operators, heuristics, and local search algorithms commonly used for solving the MKP. Al-Shihabi and Ólafsson [12] presented a hybrid algorithm and compared to the state-of-the-art solution techniques. Ke et al. [13] proposed an ant colony optimization approach to deal with the MKP and performed computational experiments on benchmark problems. Tavares et al. [6] investigated five representations of the MKP to better understand of the role of both representation and heuristics in the MKP. Raidl and Gottlieb [7] provided guidelines and practical help for designing appropriate representations and operators for EAs by studying six different representations and associated variation operators for the MKP. Chu and Beasley [1] applied a GA to the MKP and showed that it is effective for finding solutions with good quality. Khuri et al. [10] developed a GA using standard operators, tested it on a small number of standard test problems, and reported only moderate results. Thiel and Voss [11] used simple heuristic operators based on local searches, tested them on a set of standard test problems, and obtained promising results for test problems of moderate size.

Angelelli et al. [14] applied a kernel search framework to solve the MKP and concluded that this method is effective and efficient with respect to known, problem-specific approaches. Boyer et al. [15] presented a method to combine dynamic programming and branch-and-bound to solve the MKP and tested several randomly generated test sets and problems from the literature. Boyer et al. [16] presented two heuristics for the MKP and showed their approaches gave better results than existing heuristics. Puchinger et al. [17] studied the MKP and showed the effectiveness of their proposed methods. Akcay et al. [18] proposed a new greedy-like heuristic method and demonstrated that significantly improves the computational efficiency of the existing methods and generates robust, near-optimal solutions. Bertsimas and Demir [19] presented an approximate dynamic programming approach for solving the MKP and concluded that the base-heuristic approach used to approximate the value function is a promising computational approach worthy of further investigation. For a comprehensive review of the exact methods and heuristics for solving the MKP, we refer readers to [2,20,21].

3. **GA and RS.** In this section, we briefly introduce the standard GA and RS.

3.1. **GA.** A GA is an evolutionary algorithm developed from the natural evolutionary process. In general, GAs start by selecting an initial population then iteratively appliesy operators to reproduce new populations, evaluate these populations, and decide whether the algorithm should continue. GAs differ from classical optimization algorithms primarily in that they operate on a population of individuals instead of parameters. This population-based operation enables GAs to process in parallel and search a solution space rather than a single solution, which reduces the likelihood of converging to a local solution in the early stages. Each individual in a population is encoded as a *chromosome* that represents a candidate solution. A chromosome is composed of *genes* that are usually binary. The evaluation of an individual is determined by the *fitness function value*. A standard GA includes the following steps.

- Generate an initial random population of chromosomes.
- Evaluate the chromosome population using an appropriate fitness function.
- Select a subset of chromosomes with better fitness values as parents.

• Cross-over the pairs of parents with the probability (P_c) to produce offspring.
• Mutate the offspring chromosomes with the probability (P_m) to avoid early convergence into a local solution.
• Reevaluate the fitness values of the offspring.
• Terminate the algorithm if the stopping criteria are satisfied.

In the selection step, an "elitism" strategy may be considered where a proportion (P_e) of the chromosomes is placed with the offspring, and one of the selection schemes, such as "roulette wheel", can be used. In the crossover step, one can use either a single-point or uniform-point to interchange the chromosome parts. For more details about GAs, refer to [22].

3.2. **RS.** In the RS, a database can be treated as an information table that is a quadruple $S =< U, Q, V, f >$, where U is the universe consisting of a set of objects, Q is a set of attributes, V is a set of values $= \cup_{q \in Q} V_q$ with V_q the value for attribute q, and f: $U \times Q \to V$ is a function such that $f(x, q) \in V_q$ for every $q \in Q$, $x \in U$.

Objects characterized by the same amount of knowledge (or information) are said to be *indiscernible*. That is, given the set of attributes $A \subseteq Q$ and objects x, $y \in U$, x and y are indiscernible by A if and only if $f(x, a) = f(y, a)$ for every $a \in A$. Every set of attributes A forms an equivalent relation in the universe U. This relation is referred to as the A-*indiscernibility* relation and can be denoted by $IND(A)$, which partitions the universe, U, into a family of the equivalence classes $\{X_1, X_2, \ldots, X_n\}$. This partition is commonly referred to as a *classification* and denoted by $U/IND(A)$. An equivalence class, X_i, for the relation $IND(A)$ is called an A-*elementary* set and denoted by $[x]_A$ if it contains an object x.

Any objects that cannot be distinguished exactly given the set of the attributes could be approximated. Such an approximation allows us to define a set by a pair of sets, i.e., the *lower* and *upper* approximations. Let $A \subseteq Q$ and $X \subseteq U$; the A-*lower* approximation, denoted by $\underline{A}X$, and A-*upper* approximation, denoted by $\bar{A}X$, for set X are defined as follows:

$$\underline{A}X = \{x \in U : [x]_A \subseteq X\} \tag{4}$$

$$\bar{A}X = \{x \in U : [x]_A \cap X \neq \phi\} \tag{5}$$

According to these definitions, $x \in \underline{A}X$ means x *certainly* belongs to X, while $x \in \bar{A}X$ *possibly* belongs to X. The difference between $\bar{A}X$ and $\underline{A}X$ is called the A-*boundary* of X and is denoted as follows:

$$BN_A(X) = \bar{A}X - \underline{A}X \tag{6}$$

$BN_A(X)$ consists of objects that do not certainly belong to X given A. A set X is said to be *rough* (or *crisp*) if its $BN_A(X)$ is non-empty (or empty).

In an information system, some attributes may be redundant in the sense that eliminating them will not reduce the classification power of the original system. We described in the introduction that the RS can eliminate redundant attributes via attribute reduction to determine the set reduct. Formally, given A and $B \subseteq Q$, a reduct is a minimal set of attributes such that $IND(B) = IND(A)$. In other words, a reduct is the minimal non-redundant set of attributes that ensures the same quality for classifications for universe U.

4. **The Algorithms.** In this section, we describe the CRGA and SRGA, beginning with the basic idea behind the algorithms.

As stated earlier, a standard GA relies on operators such as selection, crossover, and mutation. Also recall that, in the RS, attribute reduction process can ensure the same

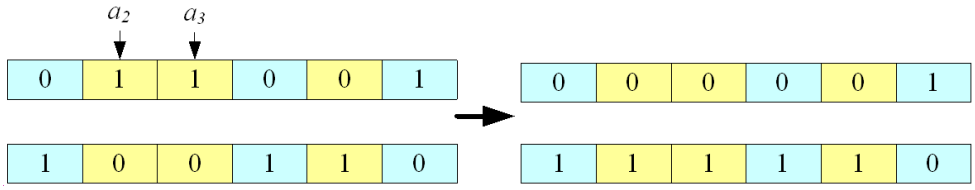
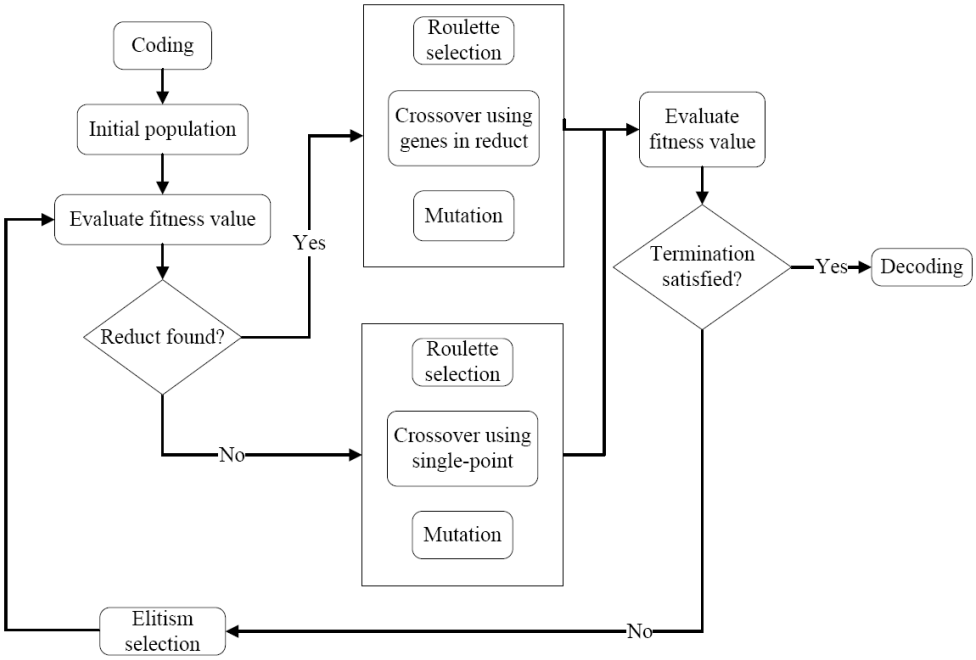
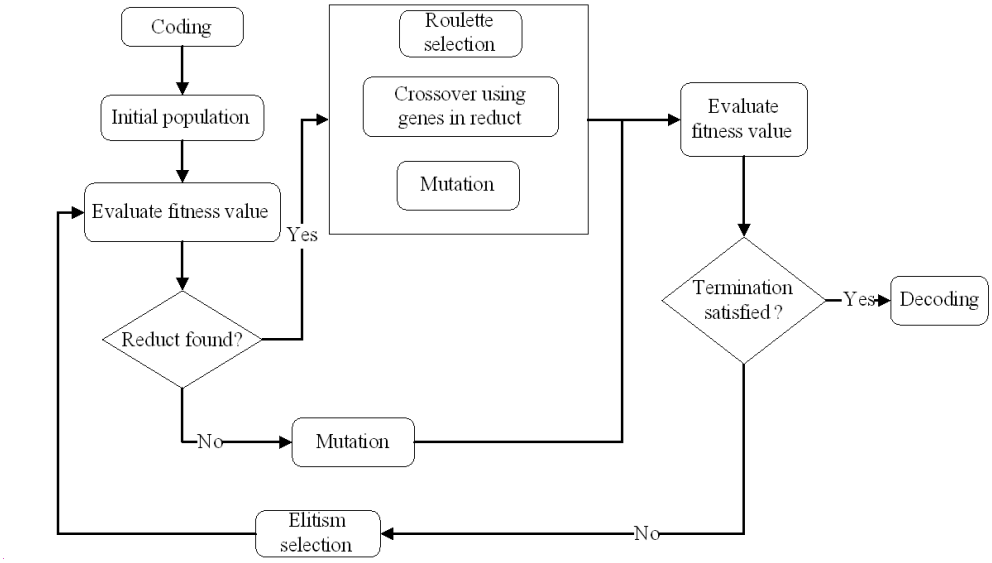


FIGURE 1. An illustration of applying the reduct to the crossover



(a)



(b)

FIGURE 2. (a) Flowchart for the CRGA, (b) flowchart for the SRGA

classification quality using a smaller set of attributes. We hope that more representative genes can be generated from the reduced attributes and used as the basis for crossover.

To apply the RS, we treat a population as an information system consisting of chromosomes with each chromosome represented by a set of condition attributes and a decision attribute. For the knapsack, i, associated with the capacity constraint, b_i, each condition attribute represents the selection of an item, x_j, and the decision attribute represents the feasibility of the underlying chromosome in the population. In this way, encoding a gene simply becomes binary with a value of "1" meaning item x_j is selected and a decision with a value of "1" meaning the chromosome satisfies the knapsack's capacity b_i. After formulating the set of items $(x_1, x_2, \ldots, x_n)$ as a set of condition attributes, we can use the attribute reduction process to eliminate redundant items while preserving the feasibility.

To illustrate how the reduct is applied to the crossover, consider the two attribute reduct (a_2, a_3) in Figure 1. If these two attributes are identified as elements in the reduct, they are used as gene points for crossover. The CRGA and SRGA fundamentally differ in selecting gene points for crossover when no reduct is found. Figure 2(a) and Figure 2(b) show the flowcharts for both CRGA and SRGA. As shown in Figure 2(a), CRGA operates on the principle that, if a reduct can be found, then the genes in that reduct are selected for crossover; otherwise, a single-point selection is used. In contrast to the CRGA, the SRGA (Figure 2(b)) does not crossover and mutates directly if no reduct can be found, which means it only uses genes generated from the reduct. Both the CRGA and SRGA include a roulette selection and elitism selection.

5. **Experimental Results.** To investigate how CRGA and SRGA perform, we compared their performance to that of a standard GA (GA for short). Each algorithm executes 50 experiments for various numbers of items and knapsacks. We refer readers to [1] for more details about these experiments. Before presenting the results, we describe the parameters for the GA.

5.1. **The parameters and penalty function for the GA.** Some of the GA parameters are as follows: the crossover rate, P_c, is 0.6; the mutation rate, P_m, is $1/n$; the population size is 50; and the elitism rate, P_e, is 0.4. The stopping criteria of the algorithm are (1) the parents and offspring are entirely identical for 100 consecutive generations, or (2) a maximum of 2000 iterations was executed. We refer to the solution as the *final solution*, which may not be the optimal solution.

To deal with infeasibility over the course of the iterations, we implemented and slightly modified the penalty function proposed by [23]. The penalty function is briefly described below. Let $eval(x)$ denote the fitness value after the penalty and *item* be the number of the knapsack; $eval(x)$ is evaluated as the following:

$$eval(x) = (f(x) - p(x))/item, \text{ where} \tag{7}$$

$$f(x) = \sum_{j=1}^{n} p_j x_j, \tag{8}$$

$$p(x) = f(x) \times (dist/diff),$$

$$dist = \sum_{i=1}^{m} |r_{ij} - b_i|, \tag{9}$$

$$diff = \min\left(\sum_{i=1}^{m} b_i, \sum_{i=1}^{m} |TR_i - b_i|\right), \text{ and} \tag{10}$$

$$TR_i = \sum_{j=1}^{n} r_{ij}. \tag{11}$$

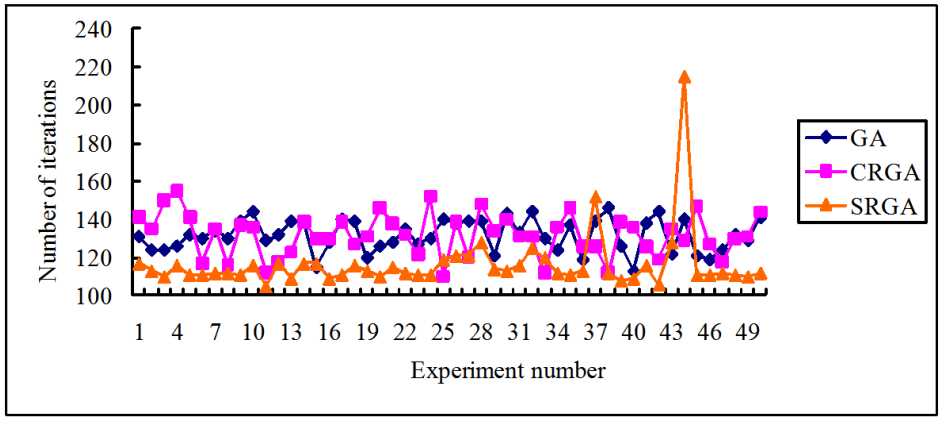
5.2. **The experiments.** We investigated the experimental results in terms of the computational convergence and solution quality. Our computational convergence was measured using the mean number of iterations required to both terminate the algorithm and reach the maximal solution for 50 experiments. The mean number of iterations was calculated by adding the number of iterations for each experiment and dividing by 50. For the maximal solution, consider an experiment that terminates with a final solution of 4300 after 126 iterations. However, before termination, the experiment obtained a solution of 4382 by the 17th iteration. In this case, the maximal solution is 4382. The reason we measure the number of iterations needed to reach the maximal solution is to determine when the algorithm generated better solutions. Note that the algorithm reports a solution of 4300 instead of 4381, primarily due to not satisfying the stopping criterion, which is an inherent problem for GAs as the following solution is not guaranteed to be better than the present one to prevent being trapped into a local solution during the early stages. Knowing the maximal solution can help provide information supporting the relaxation of the stopping criterion to find a better final solution.

The solution quality is measured from the mean value and standard deviation of the final feasible solutions. In other words, experiments terminating with infeasible solutions are excluded from the mean and standard deviation calculations. Again, each experiment may terminate with an infeasible solution or at the predefined maximum number of iterations

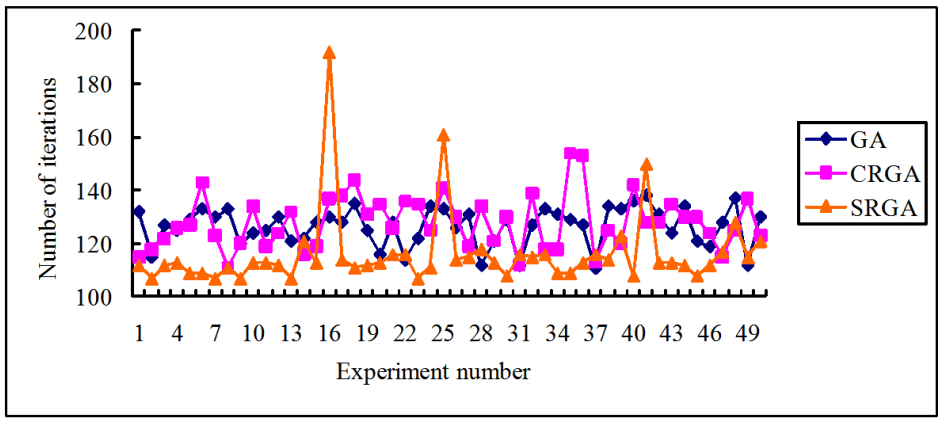
TABLE 1. The mean number of iterations required for termination, and the mean number of iterations required to reach the maximal solution

Instance	$n \times m$	GA			CRGA			SRGA		
		MIT	MIMS	NOS	MIT	MIMS	NOS	MIT	MIMS	NOS
Petersen1	6 × 10	110	1.3	48	111	6.7	45	129	4.7	50
Petersen2	10 × 10	116	9.5	21	116	7.89	22	128	31.2	44
Petersen3	15 × 10	119	8.4	8	119	7	1	127	52.3	13
Petersen4	20 × 10	121	12.7	1	127	8.2	0	117	65.5	0
Petersen5	28 × 10	129	20.9	0	129	7.2	0	128	73.3	0
Petersen6	39 × 5	276	33.8	0	138	53.1	0	129	77.9	0
Petersen7	50 × 5	141	58.1	0	139	59.7	0	120	63.2	0
HP1	28 × 4	132	35.3	0	130	36.4	0	138	79	0
HP2	35 × 4	133	41.1	0	134	42.4	0	111	71.7	0
Weing1	28 × 2	132	36.8	0	132	8.1	0	117	58.1	0
Weing2	28 × 2	129	28.4	0	128	37.3	0	118	67.4	0
Weing4	28 × 2	138	53.2	0	139	51.5	0	114	68.5	0
Weing5	28 × 2	128	28.8	0	131	37.9	0	120	58.2	0
Weing6	28 × 2	130	34.2	0	129	28.9	0	119	64	0
Weing7	105 × 2	163	88.7	0	157	83.6	0	147	44.5	0
PB1	27 × 4	130	35.8	0	130	7.8	0	141	78.1	0
PB2	34 × 4	130	34.4	0	133	10.2	0	130	73.2	0
PB4	29 × 2	127	20.3	0	129	29.2	0	113	70.4	0
PB5	20 × 10	122	6.8	0	120	8.9	0	127	58.8	0
Weish1	30 × 5	127	27.2	0	128	10.2	0	116	59.6	0

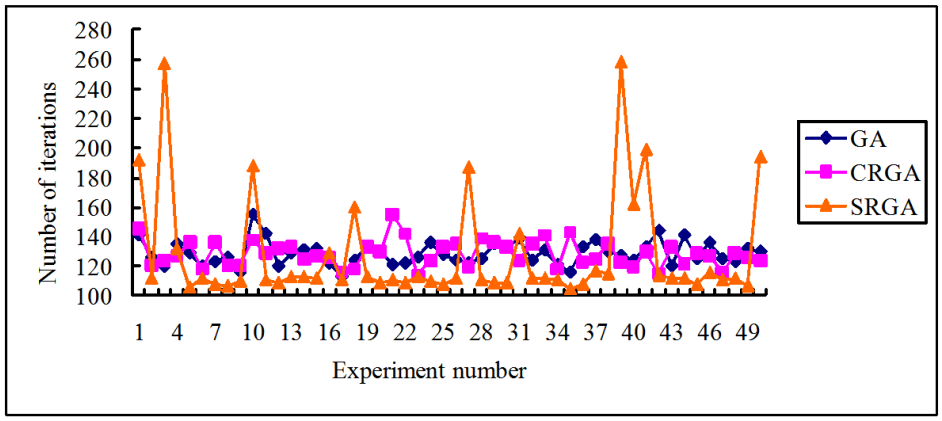
MIT = mean number of iterations to terminate.
MIMS = mean number of iterations to reach the maximal solution.
NOS = the number of optimal solutions obtained.



(a)



(b)

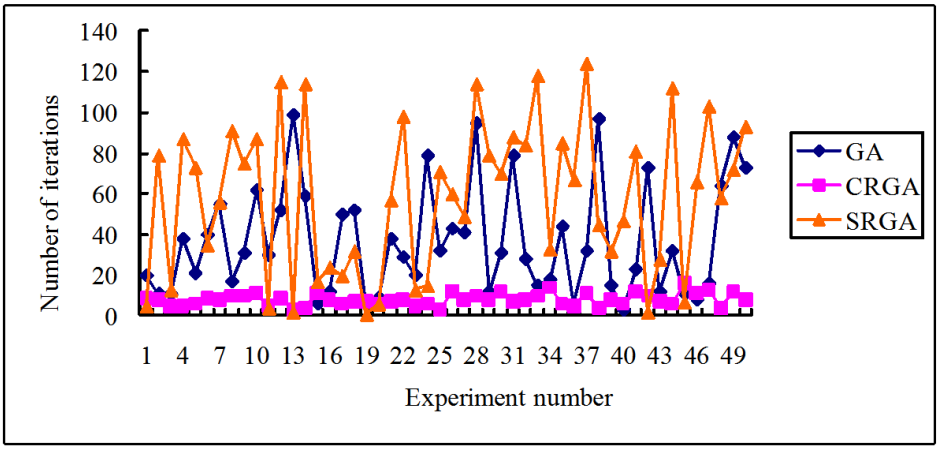


(c)

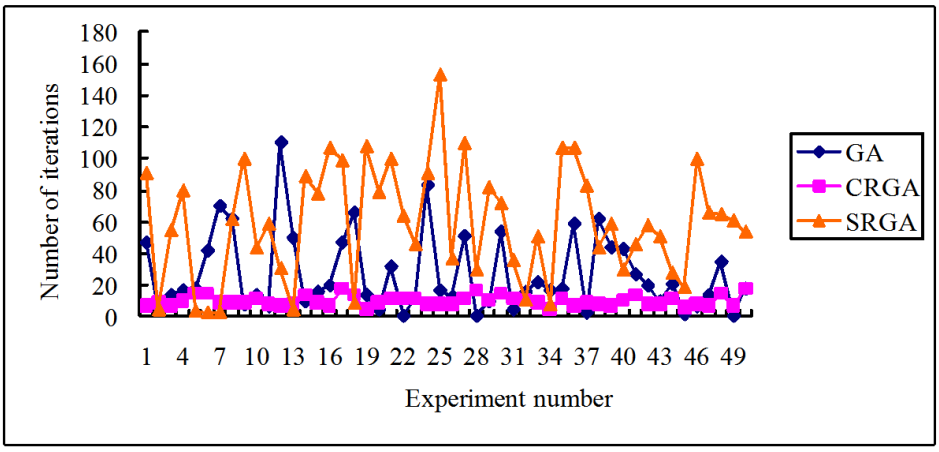
FIGURE 3. (a) The number of iterations required to terminate each experiment given Weingartner1. (b) The number of iterations required to terminate each experiment given Weish1. (c) The number of iterations required to terminate each experiment given Petersen5.

even if an optimal solution appeared. This outcome is due to the nature of GAs as described above.

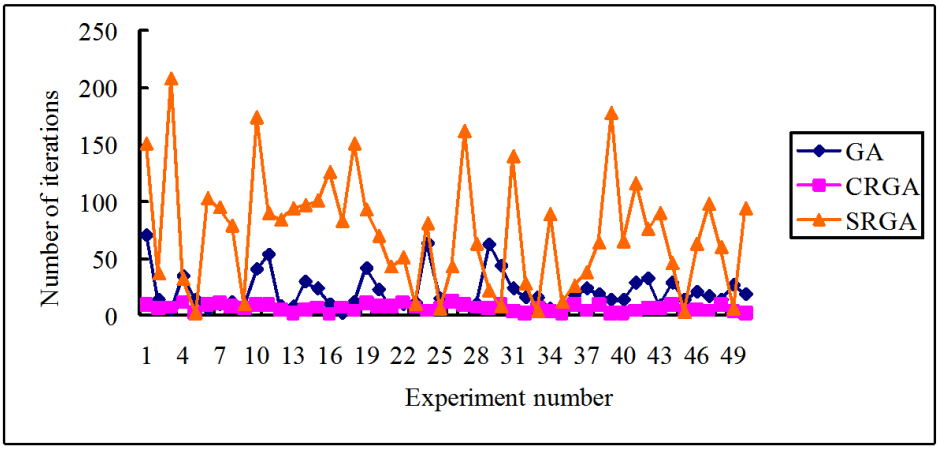
We compare the algorithm performances using 20 test instances listed in Table 1 of [1], who stated that "these problems are small real-world problems (p.77)". The best known solutions are also given in [1].



(a)



(b)



(c)

FIGURE 4. (a) The number of iterations required to reach the maximal solution for each experiment given Weingartner1. (b) The number of iterations required to reach the maximal solution for each experiment given Weish1. (c) The number of iterations required to reach the maximal solution for each experiment given Petersen5.

5.2.1. *Computational convergence.* Consider Table 1. We discuss with cases in order of which types of items, n, are present for a given knapsack number, m.

According to Table 1, the SRGA performed the best in 14 out of 20 cases in terms of the mean number of iterations to terminate (MIT for short); and both the CRGA and GA appear to have equivalent performance. In terms of the mean number of iterations to obtain the maximal solution (MIMS for short), the CRGA performed better in 10 cases. In contrast, the SRGA performed surprisingly poorly.

To gain insight into the two different mean numbers of iterations, we select Weingartner1, Weish1, and Petersen5 shown in Figures 3(a), 3(b) and 3(c), respectively, to showcase the number of iterations required to terminate the algorithm. Recall that the SRGA outperformed the GA and CRGA in terms of the MIT in 14 cases. The superiority of SRGA can be observed in Figures 3(a), 3(b) and 3(c), where several experiments have unusually high numbers of iterations. Based on this observation, we suggest that the SRGA has an advantage for terminating the algorithm. Similarly, number of iterations to reach the maximal solution for the three cases is shown in Figures 4(a), 4(b) and 4(c). Note that the CRGA consistently performs better when obtaining the maximal solution, which concurs with the results mentioned above. In summary, either CRGA or SRGA demonstrates advantages in terms of computational convergence.

5.2.2. *Solution quality.* After examining the computational convergence, we discuss the solution quality based on Table 2. According to Table 2 and by jointly considering the mean values and standard deviations, the SRGA outperforms both the GA and CRGA in seven cases. Considering the standard deviation alone, the SRGA produces the smallest value in nine cases. In these cases, a larger mean value represents better solution quality, and a smaller standard deviation represents more closely clustered solutions. Therefore, the results suggest that the SRGA can produce better quality and more clustered solutions. The GA performs the best when taking its mean value and standard deviation together, which demonstrates its continued competitiveness. To our surprise, the CRGA performed the worst.

5.3. **Summary.** In this section, we summarize the preceding findings focusing on the SRGA and CRGA. Overall, the SRGA had the advantage in terms of terminating the algorithm and producing both better quality and more clustered solutions. Recall that the SRGA either uses the genes generated from the reduct or mutates directly. These advantages justify our integration of the reduct into the crossover operator. In contrast, the performance of the CRGA was not on par with that of the SRGA, which might be partly due to the frequent crossover operations failing to satisfy the stopping criteria. However, the CRGA performs relatively well in terms of reaching the maximal solution, which suggests some form of mechanism, such as relaxing the stopping criteria, could significantly improve the CRGA performance.

Though this summary is based on moderately sized test instances, according to [1], these instances did present some challenges for other GA heuristics. Indeed, more convincing conclusions can be drawn by providing a clear comparison between our work and existing ones. Al-Shihabi and Ólafsson [12] suggested that a fair comparison needs to consider the effect of computation time. The solution time was given in [1,12]. However, it is widely known that the computation or CPU time largely depends on the type of machine or programming language used. Measurements such as the average value, average gap, and standard deviation of the solutions are given in [7,13]; however, their test instances are different from this study. In contrast, our use of the mean number of iterations rarely appears in the literature.

TABLE 2. The means and standard deviations for the GA, CRGA, and SRGA

Instance	$n \times m$	Best known	GA		CRGA		SRGA	
			Mean[a]	Std	Mean	Std	Mean	Std
Petersen1	6 × 10	3800	3796	19.6	3782	74	3800	0
Petersen2	10 × 10	8706	8562	159.3	8562	184.9	8662	120
Petersen3	15 × 10	4015	3873	128	3878	92.1	3941	80.5
Petersen4	20 × 10	6120	5455	300.9	5476	371.4	5630	292
Petersen5	28 × 10	12400	11595	404.1	11203	461	12240	413.6
Petersen6	39 × 5	10618	10031	269.1	10107	222.2	9953	222.8
Petersen7	50 × 5	16537	15498	355.6	15184	558.7	14915	352.2
HP1	28 × 4	3418	3273	61.6	3259	55.8	3214	76.3
HP2	35 × 4	3186	29101	213.5	2921	95.1	2864	86.3
Weing1	28 × 2	141278	131512	4946.6	130885	5856	131409	5820.3
Weing2	28 × 2	130883	115069	7990.2	113289	7122.4	116883	5692.7
Weing4	28 × 2	119337	110112	4964.2	107535	5686.6	106950	5365
Weing5	28 × 2	98796	80808	10643.2	79038	10691.4	75109	15618.7
Weing6	28 × 2	130623	117017	4985.1	116773	4815.2	115671	6007.2
Weing7	105 × 2	1095445	969541	37356.7	975269	41559.4	783196	44754.8
PB1	27 × 4	3090	2963	48.1	2953	48.7	2936	69.2
PB2	34 × 4	3186	2979	69.7	29655	71.5	2907	78.7
PB4	29 × 2	95168	84409	3276.7	83483	3530.6	81412	3543.6
PB5	20 × 10	2139	1984	64.9	1984	62.9	2016	51.1
Weish1	30 × 5	4554	3662	471.3	3774	416.7	3777	357.7

[a] All mean values are rounded off

6. **Conclusions.** The MKP is a well-known NP-hard combinatorial optimization problem. This paper uses a methodology that integrates an RS reduct into the crossover of a GA to solve the MKP. This paper presents two algorithms based on this methodology: one selecting the crossover points either randomly or using a reduct, the other selecting the crossover points solely by a reduct. The performance of these two algorithms was compared with that of a standard GA using test instances from the literature.

According to the experimental results, integration has the advantages of producing better quality and more clustered solutions and might improve the performance if some mechanisms such as manipulating the stopping criteria were developed. These advantages justify the integration and demonstrate an alternative for improving the performance of GAs.

Even though integrating the reduct into the crossover operator is motivated by solving the MKP, the major aim of the study was to investigate the integration when solving combinatorial optimization problems. The proposed approach can be applied to various other combinatorial optimization and subset problems. In essence, an appropriate representation is needed to apply the reduct to other problems.

This paper contains some possible extensions. First, the algorithm frequently produces infeasible solutions. To deal with this infeasibility, we modified the penalty function from [23], which primarily penalizes the function based on the number of knapsacks. Other penalty functions could improve the infeasibility handling and deserve investigation. Second, one may consider using different elitism strategies. The present elitism strategy places a proportion of the chromosomes into the offspring, which is likely to only find local solutions. A dynamic strategy selection for chromosome placement may help reduce the likelihood of being trapped in a local solution.

Acknowledgment. This research was supported in part by the National Science Foundation grant number 101-2410-H-167-002. The authors gratefully acknowledge the anonymous reviewers' helpful comments and suggestions for improving the presentation of the paper.

REFERENCES

[1] P. C. Chu and J. E. Beasley, A genetic algorithm for the multidimensional knapsack problem, *J. Heuristics*, vol.4, no.1, pp.63-86, 1998.
[2] A. Fréville, The multidimensional 0-1 knapsack problem: An overview, *Eur. J. Oper. Res.*, vol.155, no.1, pp.1-21, 2004.
[3] H.-H. Yang, M.-T. Wang, Y.-J. Chen, Y.-S. Huang and C.-J. Kao, Crossover based on rough sets – A case of multidimensional knapsack problem, *IEEE Int. Conf. Ind. Eng. Eng. Manage*, Macao, China, vol.489, no.89, 2010.
[4] Z. Pawlak, Rough set, *Int. J. Inform. Comput. Sci.*, vol.11, pp.341-356, 1982.
[5] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, USA, 1975.
[6] J. Tavares, F. B. Pereira and E. Costa, Multidimensional knapsack problem: A fitness landscape analysis, *IEEE T. Syst. Man Cy. B*, vol.38, no.3, pp.604-616, 2008.
[7] G. R. Raidl and J. Gottlieb, Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem, *Evol. Comput.*, vol.13, no.4, pp.441-475, 2005.
[8] M. J. Varnamkhasti and L. S. Lee, A fuzzy genetic algorithm based on binary encoding for solving multidimensional knapsack problems, *J. Appl. Math.*, 2012.
[9] H.-H. Yang, S.-W. Wang, H.-T. Ko and J.-C. Lin, A novel approach for crossover based on attribute reduction – A case of 0/1 knapsack problem, *IEEE Int. Conf. Ind. Eng. Eng. Manage*, Hong Kong, China, vol.352, no.273, 2009.
[10] S. Khuri, T. Bäck and J. Heitkötter, The zero/one multiple knapsack problem and genetic algorithms, *ACM SAC'94*, pp.188-193, 1994.
[11] J. Thiel and S. Voss, Some experiences on solving multiconstraint zero-one knapsack problems with genetic algorithms, *INFOR*, vol.32, pp.226-242, 1994.
[12] S. Al-Shihabi and S. Ólafsson, A hybrid of nested partition, binary ant system, and linear programming for the multidimensional knapsack problem, *Comput. Oper. Res.*, vol.37, no.2, pp.247-255, 2010.
[13] L. Ke, Z. Feng, Z. Ren and X. Wei, An ant colony optimization approach for the multidimensional knapsack problem, *J. Heuristics*, vol.16, no.1, pp.65-83, 2010.
[14] E. Angelelli, R. Mansini and M. G. Speranza, Kernel search: A general heuristic for the multidimensional knapsack problem, *Comput. Oper. Res.*, vol.37, no.11, pp.2017-2026, 2010.
[15] V. Boyer, D. El Baz and M. Elkihel, Solution of multidimensional knapsack problems via cooperation of dynamic programming and branch and bound, *Eur. J. Ind. Eng.*, vol.4, no.4, pp.434-449, 2010.
[16] V. Boyer, M. Elkihel and D. El Baz, Heuristics for the 0–1 multidimensional knapsack problem, *Eur. J. Oper. Res.*, vol.199, no.3, pp.658-664, 2009.
[17] J. Puchinger, G. R. Raidl and U. Pferschy, The multidimensional knapsack problem: Structure and algorithms, *Informs J. Comput.*, vol.22, no.2, pp.250-265, 2009.
[18] Y. Akcay, H. Li and S. H. Xu, Greedy algorithm for the general multidimensional knapsack problem, *Ann. Oper. Res.*, vol.150, no.1, pp.17-29, 2007.
[19] D. Bertsimas and R. Demir, An approximate dynamic programming approach to multidimensional knapsack problems, *Manage Sci.*, vol.48, no.4, pp.550-565, 2002.
[20] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, Wiley-Interscience, New York, NY, 1990.
[21] H. Kellerer, U. Pferschy and D. Pisinger, *Knapsack Problems*, Springer-Verlag, New York, NY, 2004.
[22] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*, Wiley-Interscience, New York, NY, 1997.
[23] A. L. Olsen, Penalty functions and the knapsack problem, *Proc. of the 1st IEEE Conf. Evol. Comput.*, Orlando, FL, USA, pp.554-558, 1994.