# Combining Multiple Representations in a Genetic Algorithm for the Multiple Knapsack Problem

Alex S. Fukunaga and Satoshi Tazoe

***Abstract*— We propose a new evolutionary algorithm for the multiple knapsack problem (MKP) which uses multiple representations. Previous, successful approaches for the MKP have included a weight-coded, order-based representation, as well as a grouping representation enhanced by a dominance condition to restrict search. We propose a representation-switching genetic algorithm which periodically transforms the representation of individuals between these two representations. We show that this new hybrid algorithm outperforms the previous approaches.**

## I. Introduction

Successful application of metaheuristics such as genetic algorithms to difficult combinatorial optimization problems usually depends on designing problem-specific representations and operators. Most work on applying metaheuristics to combinatorial optimization focuses on finding a particular, effective representation and operator set for a problem. However, for some problems, there may not be a single representation which dominates all others across the entire range of instances of interest. Furthermore, there may be instances on which no currently known representation is successful.

In this paper, we investigate the use of multiple representations in a genetic algorithm for the Multiple Knapsack Problem (MKP), which is the classical combinatorial optimization problem of assigning (packing) objects of various weights and values (profits) to a set of containers (bins) of various capacities, in order to maximize the total profit of the items assigned to the containers.

The recently proposed Undominated Grouping Genetic Algorithm (UGGA) for the MKP, uses a grouping representation, GA, where the genetic operators were restricted to only candidates which are undominated according to a dominance criterion [5]. The UGGA was compared experimentally with previous algorithms for the MKP, and was shown to perform very well on MKP instances with high correlations between item weights and values. However, for instances where item weights and values were not highly correlated, the UGGA was not competitive with the Weight-Coded GA (WCGA), an earlier, order-based GA [15].

The UGGA and WCGA use very different sets of representations and operators. In this paper, we propose a new, *Representation-Switching* GA (RSGA) for the MKP which effectively combines both of these representation strategies. By periodically transforming individuals from one representation to the other, we obtain a new hybrid algorithm which exploits the strengths of both algorithms. We experimentally evaluate the representation-switching strategy and show that this method significantly outperforms previous methods for some classes of large-scale multiple-knapsack problems.

The rest of this paper is organized as follows. Section II describes the MKP and review earlier approaches, including the Weight-Coded Genetic Algorithm and the Undominated Grouping Genetic Algorithm. In Section III, we propose a new, Representation-Switching GA which combines the Weight-Coded GA and Undominated Grouping GA. Section IV present experimental results comparing the various approaches for the MKP, showing that the new RSGA approach represents a new, state-of-the-art algorithm for the MKP. We conclude with a discussion of the results and directions for future work.

## II. Genetic Algorithms for the Multiple Knapsack Problem

Consider $m$ containers (bins) with capacities $c_1, ..., c_m$, and a set of $n$ items, where each item has a weight $w_1, ..., w_n$ and profit $p_1, ..., p_n$. Packing the items in the containers to maximize the total profit of the items, such that the sum of the item weights in each container does not exceed the container's capacity, and each item is assigned to at most one container is the *0-1 Multiple Knapsack Problem*, or MKP.

For example, suppose we have two bins with capacities $c_1 = 10, c_2 = 7$, and four items with weights 9,7,6,1 and profits 3,3,7,5. The optimal solution to this MKP instance is to assign items 1 and 4 to bin 1, and item 3 to bin 2, giving us a total profit of 15. Thus, the MKP is a natural generalization of the classical 0-1 Knapsack Problem to multiple containers.

The MKP has many applications, including continuous double-call auctions [8], multiprocessor scheduling [10], vehicle/container loading [2], and the assignment of files to storage devices in order to maximize the number of files stored in the fastest storage devices [10]. A special case of the MKP where the profits of the items are equal to their weights, i.e., $p_j = w_j$ for all $j$ is the *Multiple Subset-Sum Problem* (MSSP). The MKP (including the special case of the MSSP) is strongly NP-complete.[1]

### A. Incomplete algorithms for the MKP

This section gives an overview of incomplete approaches for the MKP. An important concept through the rest of this

Alex Fukunaga is with the Global Edge Institute, Tokyo Institute of Technology, Meguro, Tokyo, Japan. Satoshi Tazoe is with the Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Meguro, Tokyo, Japan

[1] The single-container 0-1 Knapsack problem is only *weakly* NP-complete, and can be solved in pseudopolynomial time using dynamic programming.

paper is the *efficiency* of an item $i$, defined as $p_i/w_i$, the ratio of its profit to its weight. Efficiency is a key concept in MKP heuristics because intuitively, it is usually desirable to pack the more efficient items and not use the least efficient items.

First-Fit Decreasing Efficiency Heuristic (FFDE) is a simple heuristic which is based on the well-known First-Fit Decreasing heuristic for bin packing [1]. The items are sorted in decreasing order of efficiency (i.e., $p_1/w_1 \geq p_2/w_2, .., \geq p_n/w_n$), and the bins are sorted in increasing order of capacity (i.e., $c_1 \leq c_2 ... \leq c_m$). A "first-fit" heuristic is applied, where we loop over each (sorted) item, and for each item, we loop over each (sorted) bin, and the item is placed in the first bin with sufficient remaining capacity to hold the item (if any such bin exists).

More sophisticated approaches for solving large, difficult MKP instances included the Martello-Toth Heuristic Method (MTHM) [11], an iterative improvement algorithm, and Mulknap, a branch-and-bound algorithm [14]. Mulknap is an exact algorithm – given enough time, it is eventually guaranteed to return the optimal solution to a MKP instance. However, the time required for a Mulknap run to terminate may be too long for a difficult problem instance. Thus, in this paper, we run all algorithms (including Mulknap) with a time limit, and return the best solution found by the time limit.

Several genetic algorithms have been investigated for the MKP. The most straightforward class of evolutionary approaches to the MKP uses a *direct* encoding, where a candidate solution is represented is a vector $B = (b_1, ..., b_n)$, where the $i$-th element represents the bin to which the $i$-th item is assigned. This basic mapping may be enhanced with local improvement or repair phases [16]. Previous work in both the MKP [15] and the closely related bin-packing problem [17], [4] has shown that this approach is not competitive with other approaches described below, and we have confirmed this with our own experiments on the MKP with various direct encoding variants.

### B. Weight-Coded Genetic Algorithm (WCGA)

Raidl proposed a *Weight-Coded GA* (WCGA) for the MKP. This is a variant of an order-based encoding such as bin-packing encoding of Reeves [17]. Each individual is a vector of *modifiers* $M = (m_1, m_2, ..., m_n)$.[2] The WCGA is a type of *order-based* representation where the chromosome encodes the order in which a packing heuristic (which Raidl calls a *decoder*), such as the First-Fit or Best-Fit heuristic, packs items into bins. Items are sorted (ordered) according to a *modified efficiency* (e.g., the sort key for an item $i$ can be set by multiplying the efficiency of the item ($p_i/w_i$) by the item modifier $m_i$ specified in the chromosome. See [15] for details on weight generation. In particular, we implemented a WCGA using log-normal multiplicative biasing and decoding heuristic $B$[3]), using relative value ordering.

### C. The Undominated Grouping Genetic Algorithm

In contrast to the WCGA, which uses an ordering-based representation, the *Undominated Grouping Genetic Algorithm* (UGGA) uses a group-based representation. Falkenauer proposed the Grouping GA (GGA) for the bin packing problem [3].

A straightforward adaptation of the GGA for the MKP works as follows: A *bin assignment* $g = \{i_1, ..., i_k\}$ is a set of all the items that are assigned to a given bin. A valid solution to a MKP instance is a set of bin assignments, where each item appears in at most one bin assignment. A bin assignment is *feasible* with respect to a given bin capacity $c$ if the sum of its item weights does not exceed $c$. Otherwise, the bin assignment is *infeasible*. A *feasible solution* is a candidate solution for the MKP where all bin assignments are feasible.

In a GGA, each individual consists of a list of groups $\{g_1, g_2, ..., g_m\}$, where the $j$-th group is a bin assignment representing the set of items assigned to the $j$-th bin ($1 \leq j \leq m$). In other words, this is a direct representation of a set of bin assignments, where the genetic operators (crossover and mutation) are specialized in order to directly manipulate *groups* (bin assignments). The GGA searches the space of feasible solutions. The crossover and mutation operators are designed such that at each step, every bin assignment in every individual is a feasible bin assignment.

A dominance relation between search states $s_1$ and $s_2$ in a combinatorial optimization problem is used to establish that no extension of the dominated state has a solution of better quality than the best extension of the dominating state. Thus, if $s_2$ dominates $s_1$, it suffices to search extensions of $s_2$ and discard extensions of $s_1$, without discarding the optimal solution [7]. We now describe a dominance condition for the MKP, proposed in [6].

We say that a bin assignment $S$ is *maximal* with respect to capacity $c$ if $S$ is feasible, and adding any of the remaining items would make it infeasible. A feasible bin assignment $F_1$ *dominates* another feasible assignment $F_2$ if the value of the optimal solution which can be obtained by assigning $F_1$ to a bin is no worse than the value of the optimal solution that can be obtained by assigning $F_2$ to the same bin. Consider an MKP instance where we have a bin of capacity 100, and three items $i_1, i_2, i_3$ with weights $w_1 = 96, w_2 = 4, w_3 = 3$ and profits $p_1 = 90, p_2 = 10, p_3 = 8$. Assume that there are some other bins and items, and that no other item has weight less than or equal to 4. Two maximal, feasible, candidate bin assignment for the bin of capacity 100 is $F_1 = \{i_1, i_2\}$ and $F_2 = \{i_1, i_3\}$. A feasible but non-maximal candidate bin assignment is $F_3 = \{i_1\}$. Clearly, $F_3$ is dominated by both $F_1$ and $F_2$. Leaving wasted space where an additional item can be packed can never improve upon a solution that fills the wasted space.

[2]Raidl called these modifiers "weights", but we renamed them in order avoid confusion with item weights.

[3]Raidl's $B$ decoding heuristic is similar to the well-known Best-Fit packing heuristic for bin-packing problem [1]

The following, general dominance criterion is based on the Martello-Toth dominance criterion for the bin packing problem [12], and was originally used as the basis for an exact, branch-and-bound algorithm [6].

*Proposition 1 (MKP Dominance Criterion):* Let $A$ and $B$ be two assignments that are feasible with respect to capacity $c$. $A$ *dominates* $B$ if $B$ can be partitioned into $i$ subsets $B_1, ..., B_i$ such that each subset $B_k$ is mapped one-to-one to (but not necessarily onto) $a_k$, an element of $A$, and for all $k \leq i$, (1) the weight of $a_k$ is greater than or equal the sum of the item weights of the items in $B_k$, and (2) the profit of item $a_k$ is greater than or equal to the sum of the profits of the items in $B_k$.[6]

The *Undominated Grouping Genetic Algorithm* (UGGA) is a steady-state GA that uses the same group-based representation by the GGA, but exploits the above dominance criterion by using genetic operators which ensure that each step of the search, every group (bin assignment) in every individual is *undominated* (according to Criterion 1), and the genetic operators (mutation, crossover, and individual initialization) are guaranteed to generate individuals consisting only of undominated bin assignments.

The key genetic operations rely upon a method for generating an *undominated* bin assignment, given a bin and a set of available items as input. One obvious approach is to generate all feasible bin assignments, then apply pairwise dominance tests to eliminate the dominated assignments until we find an assignment which is not dominated by any remaining assignment. This is impractical, since the number of feasible assignments is exponential in the number of remaining items, and the memory and time required to generate and store all assignments would be prohibitive. Earlier work [5] described the `FillUndominated` algorithm, which, given a set of remaining items and some bin capacity, generates a random, undominated bin assignment using memory which is linear in the number of remaining items. This allows us to implement a grouping genetic algorithm which only explores undominated candidate solutions.

The *initialization* operator creates a new individual by iterating over each bin in increasing order of bin capacity, and filling it using `FillUndominated`. During reproduction, two parents are selected using rank-based selection [19] to select and copy two parents. created by copying the parents. *Mutation* iterates over each bin, and with probability $p_m$, marks the bin. Then, all of the marked bins are simultaneously emptied. Finally, we iterate over the marked bins in order of decreasing capacity, and fill the bin with an undominated bin assignment by calling `FillUndominated`. The *crossover* operation is a uniform crossover operation which exchanges the contents of corresponding bins between two individuals, where each bin is crossed with probability $p_c$. After the exchange of bins, each individual may have multiple instances of items (i.e., items that are simultaneously assigned to two bins). If there are any duplicates, the instance which was newly introduced to the genome via the bin exchange is kept. The bins which contain the redundant copies of the items are marked and emptied (all items assigned to those bins are unassigned). We then iterate over the marked bins in order of increasing capacity, and fill the bin with an undominated bin assignment by calling the `FillUndominated` routine.

## III. Using Multiple Representations

Earlier experiments [5] showed that while the UGGA performed very well on problem instances where the correlation between item weights and values was high, it performed poorly on instances where the correlation between item weights and values was weaker. For those instances, the WCGA had the best performance. Thus, the UGGA and WCGA, which are based on very different problem representations, were shown to be somewhat complementary.

A natural question is whether the UGGA and WCGA can be combined somehow into a new, better algorithm. We propose a *Representation Switching Genetic Algorithm* (RSGA), a new, hybrid algorithm for the MKP which uses both the UGGA and WCGA representations.

The RSGA is a simple, generic approach for combining multiple problem representations in a GA. Given two alternate problem representations, $R_1$ and $R_2$, the RSGA first chooses one of the representations, say $R_1$, and creates a population of individuals using representation $R_1$. We run the GA using $R_1$, until no progress has been made (i.e., the score of the best solution in the population has not improved) for $thresh$ seconds. Then, we apply a transformation to all individuals in the population which converts their representation from $R_1$ to $R_2$, and then we continue running the GA using representation $R_2$, until $thresh$ seconds have passed without improvement. We repeat this until some termination condition is met.

The intuition behind this method is that if a GA gets stuck using one problem representation, $R_1$, then it is in a state where the population is trapped in a local optimum with respect to the current genetic operators. By changing the representation of the individuals to a different representation, $R_2$, the population might now be in a state which is *not* trapped in a local optimum with respect to the genetic operators for $R_2$, which might enable us to make progress more rapidly.

We believed that this approach would be effective for the MKP because we have two problem representations which induce very different fitness landscapes. The WCGA searches an order-based space, where genetic operators change the order in which items are fed to a packing heuristic. The UGGA searches a group-based representation, where genetic operators reassign items among bins. Thus, it seemed that if the GA becomes stuck in one representation, then switching to the other representation would allow further progress to be made.

We use the following methods to convert individuals between the two representations. Transforming individuals from the WCGA representation to the UGGA representation is straightforward, since applying the decoding (packing) heuristic used by WCGA to a WCGA individual will give us

an assignment of items to bins, i.e., a grouping representation For every WCGA individual, there is a unique mapping to a grouping representation. This new grouping representation individual may include dominated bin assignments, and would not normally be generated by the UGGA. However, as genetic operators are applied, the bins that are affected by the crossover and mutation operators will consist solely of undominated bins, and after several generations, all members of the population are individuals which only contain undominated bin assignments (we have experimentally verified this). Thus, this transformation method starts takes a WCGA population and converts it to a grouping GA (GGA) population, which is then gradually converted to a UGGA population by the UGGA genetic operators. It is possible to view this process as being a type of "local search/optimization", where the UGGA operators are being used to eliminate dominated bin assignments from the individuals found by the WCGA.

On the other hand, transforming individuals from the UGGA representation to the WCGA representation is not as straightforward, because there is a one-to-many mapping from the UGGA representation to the WCGA representation – there are many WCGA individuals which, when decoded using the packing heuristic, map to identical bin assignments. We have implemented one method where we generate and sort in descending order a vector $R$ of random numbers according to a probability distribution (we have tried log-normal, normal, and beta distributions). The bins are sorted in order of increasing capacity, and we loop over the items in the bins, assigning a weight $w_i = R_i/e_i$, where $e_i$ is the item efficiency. Unfortunately, although this method provides a mapping from the UGGA representation to the WCGA representation, we have not obtained successful results based on this mapping – the experimental results are not competitive with other methods. Therefore, in the rest of this paper, we focus on methods which only use a WCGA $\rightarrow$ UGGA mapping, and leave the development of a successful UGGA to WCGA mapping as future work.

We investigated several strategies for combining the WCGA and UGGA representations. The Representation-Switching GA (RSGA) strategy works as follows: We first initialize a WCGA population, and run the WCGA, until progress has not been made for $thresh$ seconds. Then, we convert the individuals to the UGGA representation, reset the progress timer, and execute the UGGA until progress stalls for $thresh$ seconds. At this point, we reinitialize the population to a random WCGA population, and repeat this procedure until a termination condition is reached. In other words, we run a WCGA until progress stalls, then switch to the UGGA and run until progress stalls again, at which point, we reinitialize the population from scratch and start again.

The representation alternation (ALT) strategy is an even simpler strategy which is similar to the RSGA, except that when the WCGA phase fails to make progress, it simply resets and creates a random UGGA. In other words, this is a trivial strategy which alternate between the WCGA and UGGA, where the WCGA and UGGA phases are completely independent. In effect, this strategy simply allocates time among the WCGA and UGGA strategies.

There have been several previous works which investigated the use of multiple representations in evolutionary computation. Schnier and Yao investigated an evolutionary algorithm which used a population consisting of individuals using 2 different representations: a cartesian and pseudo polar representation [18]. Okabe et al proposed a GA where each individual had two chromosomes, where one had a binary representation, and the other had a real-valued representation [13]. Both of these works used a mixed population which includes individuals using both representations, whereas we switch between homogeneous populations.

## IV. Experimental Results

We experimentally compare the WCGA, UGGA and the hybrid RSGA, and ALT strategies. In addition, we evaluate two simple *restarting* strategies, WCGA-R and UGGA-R. These algorithms run WCGA and UGGA, respectively, but after $T_L$ seconds without improvement, they reset and start from scratch with a new population. The best individual found throughout the entire run is saved. We use the WCGA-R and UGGA-R strategies as controls to test whether simply applying periodic random restarts is sufficient for obtaining improved performance compared to the WCGA and UGGA, respectively.

In the experiments below, The WCGA, as well as the WCGA components of the WCGA-R, ALT, and RSGA hybrid algorithms, used a population of 100, uniform crossover rate of 0.5, and mutation rate of 3/(# of items). The UGGA, as well as the UGGA components of the UGGA-R, ALT, and RSGA hybrid algorithms, used a population of 100, uniform crossover rate of 0.01, and mutation rate of 0.01. These are the same parameter settings as in [15], [5]. Note that because they are different algorithms, the crossover and mutation rates for the WCGA and UGGA are not directly comparable and need to be interpreted differently.

### A. Comparison of time to find solutions of a given quality

We first evaluate the new variants of the WCGA and UGGA using the set of MKP instances used in [15]. These are a set of 21 instances with different numbers of items $(n = 30, 50, 200)$, bins $(m = 3, ..., 80)$, where the capacities of all bins in a given instance are set to the same value $(c = 100, ..., 400)$.

In this experiment, we compare the amount of time required for each algorithm to find a solution with at least a certain target objective function score, i.e., $\sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{ij} \geq Target$. For each instance $i$, the target score $Target_i$ was determined in a set of preliminary experiments where the WCGA and UGGA were both executed 3 times on each instance for 10 minutes, $Target_i$ was set to the best score achieved by either algorithm over all of these preliminary runs.

Each of the WCGA, WCGA-R, UGGA, UGGA-R, ALT, and RSGA algorithms was executed 10 times on each of the

21 instances from the Raidl benchmark set with a 10 minute time limit per run. After a set of preliminary parameter tuning experiments, the restart interval $T_L$ was set to 10 seconds (we observed little difference for $T_L = 5, 10, 20, 30$.

Table I shows the results. The *Target* column indicates the target objective function score (see above). The *fail* columns indicate the number of times the algorithm *failed* to find a solution with at least the target objective function score. The *time* columns indicate the mean time for each algorithm to find a solution with the target objective function score or higher. When computing the mean time, unsuccessful runs were treated as if they took 600 seconds to find the solution. The boldfaced entries for each row highlight the best algorithm for that particular instance.

The results show that although there is no single algorithm which clearly outperforms all others on this benchmark set, there are some clear trends. The WCGA performed fairly well overall, while the UGGA performed very badly on this benchmark set, consistently underperforming all of the WCGA and hybrids, except on one benchmark (200/20/200).

Adding restarts to the WCGA and UGGA yield mixed results, as shown by the performance of WCGA-R and UGGA-R compared to WCGA and UGGA. In some cases, restarts significantly improved the performance of WCGA and UGGA, whereas in other cases, there are significant degradations of performance.

The hybrid RSGA strategy clearly outperformed the hybrid ALT strategy. The RSGA also significantly outperformed all other algorithms, including the WCGA and WCGA-R strategies, on some of the most difficult instances (#9-#12),

### B. Comparison of solution quality

In this set of experiments, we compared the quality of the solutions found by running various MKP algorithms for a fixed amount of time. We used three classes of benchmarks which are frequently used in the MKP literature [9], [12].

- *strongly correlated instances*. where the $w_j$ are uniformly distributed in [min,max] and $p_j = w_j + (max - min)/10$ (in other words, for each item, its weight and profit are strongly correlated).
- *multiple subset-sum instances* where the $w_j$ are uniformly distributed in $[min, max]$ and $p_j = w_j$, and
- *weakly correlated instances*, where the $w_j$ are uniformly distributed in [min,max] and the $p_j$ are randomly distributed in $[w_j - (max - min)/10, w_j + (max - min)/10]$ such that $p_j \geq 1$.

The first $m-1$ bin capacities $c_i$ were uniformly distributed in $[0.4 \sum_{j=1}^{n} w_j/m, 0.6 \sum_{j=1}^{n} w_j/m]$ for $i = 1, ..., m-1$. The last bin capacity $c_m$ is chosen as $c_m = 0.5 \sum_{j=1}^{n} w_j - \sum_{i=1}^{m-1} c_i$ to ensure that the sum of the capacities is half of the total weight sum. Degenerate instances were discarded as in [14]. We used items with weights in the range [1,10000].

Each algorithm was run on each instance 10 times with a different random seed. Each run was given a 180 second time limit. All runs were executed on a 2.4 GHz Intel Core2 Duo (all of the code used in the experiments was single-threaded and only used a single core). The FFDE heuristic terminated almost immediately (less than 0.02 seconds) on all instances.

We ran the algorithms on a set of 4 strongly correlated MKP instances, 4 instances of weakly correlated instances, and 4 instances of multiple subset-sum instances.

The parameters for the WCGA are the same as in [15] (the mutation rate is 3/(number of items) = 0.01). Note that the uniform crossover probability is interpreted differently in the WCGA than in the GGA variants.

The results are shown in Table II. We mainly compared the scores obtained by each of the algorithms on each instance. The *final score* of an algorithm is the total profit achieved by the best candidate solution generated during the entire run of the algorithm (i.e., the fitness value of the best-so-far individual). For each algorithm, Table II shows the mean and standard deviation ($\sigma$) of the final scores attained by the algorithm. We also show the final score of the *worst* and *best* runs of each algorithm on that problem instance. The "pop", "cross", and "mut" columns indicate the population size, crossover rate, and mutation rate, respectively for the genetic algorithms.

On the strongly correlated and multiple subset-sum instances, the UGGA consistently attained the highest scores. As mentioned in [5], this is because the dominance criterion used by the UGGA is most effective when item weights and values are highly correlated.

On the other hand, for the weakly correlated instances, the new RSGA strategy outperformed all of the other algorithms. Furthermore, the final score for the *worst* run of RSGA was better (higher) than the final score of the *best* run on any of the other algorithms, except in the case of Instance 4, where the WCGA-R best score was slightly higher than the worst score attained by the RSGA.

### C. Discussion

There are three components which distinguish the RSGA from the previously proposed, WCGA and UGGA algorithms. These are (1) usage of both the WCGA and UGGA representation, (2) conversion of individuals from the WCGA to UGGA representation, (3) making a large change (e.g., switching representations) when progress stopped. It is possible that any possible performance improvements observed in the RSGA may be due to (1), (2), or (3) above. The UGGA-R and WCGA-R strategies, which added restarts to the UGGA and WCGA, were designed to test whether making any large change to population which gets stuck (in other words, avoiding wasting more time when we're stuck) is sufficient to significantly improve performance. The results show that in the case of the UGGA, adding restarts does not seem to improve performance, and for the WCGA, adding restarts yields mixed results, but the performance obtained by WCGA-R is not competitive with the RSGA. On the Raidl benchmark set as well as weakly correlated instances, the ALT strategy performed significantly worse than the RSGA, indicating that merely allocating computational resources among the UGGA and WCGA representation by itself is not sufficient. Therefore, with our control experiments, we

| | | | WCGA | | WCGA-R | | UGGA | | UGGA-R | | ALT | | RSGA | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ID | $n/m/c$ | Target | fail | time | fail | time | fail | time | fail | time | fail | time | fail | time |
| 1 | 30/3/100 | 33082 | 0 | 0.15 | 0 | 0.01 | 0 | 2.80 | 0 | 1.26 | **0** | **0.01** | **0** | **0.01** |
| 2 | 30/6/100 | 63464 | 0 | 0.19 | 0 | 0.34 | 2 | 321.88 | 0 | 127.86 | 0 | 0.14 | **0** | **0.11** |
| 3 | 30/9/100 | 96624 | 0 | 0.06 | 0 | 0.09 | 0 | 31.73 | 0 | 7.38 | 0 | 0.08 | **0** | **0.05** |
| 4 | 30/12/100 | 117328 | **0** | **0.29** | 0 | 0.43 | 1 | 168.18 | 0 | 19.30 | 0 | 0.44 | 0 | 0.39 |
| 5 | 50/5/100 | 56581 | 0 | 0.17 | **0** | **0.13** | 0 | 20.84 | 0 | 13.30 | 0 | 0.14 | 0 | 0.25 |
| 6 | 50/10/100 | 111597 | 9 | 595.72 | **5** | **448.41** | 10 | n/a | 9 | 547.88 | 6 | 498.55 | 6 | 509.71 |
| 7 | 50/15/100 | 159314 | 0 | 3.64 | 0 | 3.05 | 0 | 80.33 | 0 | 81.10 | 0 | 4.96 | **0** | **1.84** |
| 8 | 50/20/100 | 199201 | 0 | 1.81 | 0 | 0.60 | 4 | 433.05 | 2 | 268.53 | 0 | 3.89 | **0** | **1.23** |
| 9 | 200/20/100 | 231395 | 4 | 325.02 | 1 | 280.27 | 7 | 453.76 | 1 | 233.41 | 0 | 105.08 | **0** | **62.02** |
| 10 | 200/40/100 | 442019 | 8 | 558.11 | 6 | 477.76 | 8 | 547.66 | 10 | n/a | 7 | 517.27 | **0** | **106.65** |
| 11 | 200/60/100 | 644902 | 6 | 467.98 | 8 | 530.79 | 8 | 552.30 | 10 | n/a | 10 | n/a | **0** | **248.24** |
| 12 | 200/80/100 | 830819 | 6 | 377.87 | 0 | 202.26 | 5 | 473.71 | 10 | n/a | 8 | 570.17 | **0** | **86.38** |
| 13 | 30/3/200 | 66327 | 0 | 5.77 | 0 | 1.73 | 3 | 213.58 | 0 | 59.84 | **0** | **0.71** | 0 | 10.54 |
| 14 | 30/3/300 | 95708 | 0 | 39.96 | **0** | **37.06** | 10 | n/a | 7 | 544.03 | 0 | 81.90 | 0 | 45.97 |
| 15 | 30/3/400 | 130520 | 4 | 377.55 | **0** | **57.97** | 10 | n/a | 10 | n/a | 0 | 64.75 | 2 | 244.18 |
| 16 | 50/5/200 | 111792 | 2 | 201.66 | 0 | 24.28 | 5 | 421.07 | 3 | 394.82 | 0 | 30.86 | **0** | **19.79** |
| 17 | 50/5/300 | 164948 | 0 | 29.88 | 0 | 18.87 | 10 | n/a | 10 | n/a | 0 | 10.89 | **0** | **17.59** |
| 18 | 50/5/400 | 211102 | 0 | 23.43 | **0** | **15.34** | 9 | 585.92 | 9 | 557.23 | 0 | 33.09 | 0 | 21.35 |
| 19 | 200/20/200 | 451162 | 7 | 488.56 | 8 | 528.85 | **2** | **282.70** | 9 | 545.10 | 8 | 521.07 | 5 | 469.70 |
| 20 | 200/20/300 | 646849 | **6** | **510.50** | 10 | n/a | 10 | n/a | 10 | n/a | 8 | 519.29 | 10 | n/a |
| 21 | 200/20/400 | 834475 | **1** | **299.89** | 7 | 493.09 | 10 | n/a | 10 | n/a | 9 | 555.05 | 8 | 493.02 |

TABLE I

COMPARISON OF TIME TO FIND A SOLUTION WITH SCORE OF $Target$ OR BETTER (10 ATTEMPTS PER INSTANCE, 600 SECOND TIME LIMIT PER INSTANCE). $n$ = NUMBER OF ITEMS, $m$ = NUMBER OF BINS, $c$ = BIN CAPACITY (ALL BINS HAVE SAME CAPACITY).

showed that (1) and (3) by themselves are insufficient, and it appears that the conversion of individuals from the WCGA to UGGA has a significant impact on performance. This supports the hypothesis that the UGGA, with its mechanism for eliminating dominated bin assignments, serves as an effective, optimization phase when applied to solutions which have been found by the WCGA representation.

The ALT strategy slightly outperformed the RSGA on the strongly correlated and multiple subset-sum instances, where the UGGA consistently had the best performance of all algorithms, and significantly better than the WCGA. An explanation for this is that on these instances, the WCGA searches the "wrong" parts of the search space. Thus, when the RSGA converting the individual from the WCGA representation to the UGGA representation, this in effect starts the population off at local minima which are hard to escape, which is worse than restarting the population from scratch, as the ALT strategy does. We conjecture that in the ALT runs on these instances, the best results are always found in the UGGA phase, and that time spent in the WCGA phase is essentially wasted.

## V. CONCLUSIONS

We investigated a hybrid algorithm for the multiple knapsack problem which combines the WCGA, an order-based GA and the UGGA, a grouping GA As previously shown in [5], the UGGA is very effective when applied to MKP instances where the item weights and values are tightly correlated, but the UGGA was not competitive with the WCGA on instances where the correlation was not as strong. Previous work [5] had also compared the UGGA and WCGA to other algorithms, including simple heuristic algorithms, other GA approaches, and branch-and-bound, and found that for the problems we use here, the state of the art algorithm was either the WCGA or the UGGA.

In this paper, we sought to combine the strengths of both algorithms, and proposed the hybrid RSGA, which converts individuals from the WCGA representation to the UGGA representation when progress stalls. The intuition was that even on instances where the WCGA performed relatively well compared to the UGGA, adding the UGGA's ability to replace dominated bin assignments with undominated bin assignments would be useful in further optimizing a population which had stopped making progress under the WCGA representation. Our experiments indicate that this approach is successful. While the UGGA continued to be the best performer on instances where weights and values are tightly correlated, the new, hybrid RSGA strategy significantly outperformed both the UGGA and WCGA on instances where previously, the WCGA clearly had the best performance and the UGGA was previously not competitive. Thus, this work presents a new, state of the art algorithm for this class of MKP instances.

The RSGA converts individuals from the WCGA to UGGA representation, but so far, we have not found a successful way to apply a mapping from the UGGA to WCGA representation which results in improved performance. An algorithm which successfully cycles between the two representations remains an avenue for future research.

| STRONGLY CORRELATED INSTANCES | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Instance 1 | | | | | Instance 2 | | | | |
| Algorithm | individuals | Mean | $\sigma$ | worst | best | individuals | Mean | $\sigma$ | worst | best |
| FFDE | n/a | 504429 | n/a | n/a | n/a | n/a | 521987 | 0 | 521987 | 521987 |
| WCGA | 466752 | 751658 | 128 | 751426 | 751813 | 467782 | 768237 | 102 | 768066 | 768392 |
| WCGA-R | 467389 | 751545 | 91 | 751387 | 751690 | 469013 | 768067 | 132 | 767876 | 768278 |
| UGGA | 2452429 | **753370** | 47 | 753278 | 753428 | 2439979 | **769614** | 40 | 769569 | 769683 |
| UGGA-R | 2767450 | 752986 | 51 | 752901 | 753047 | 2742961 | 769356 | 30 | 769282 | 769387 |
| ALT | 1356612 | 752941 | 73 | 752863 | 753089 | 1323436 | 769304 | 58 | 769237 | 769434 |
| RSGA | 1491588 | 752862 | 82 | 752752 | 753052 | 1496421 | 769178 | 65 | 769069 | 769282 |
| | Instance 3 | | | | | Instance 4 | | | | |
| Algorithm | individuals | Mean | $\sigma$ | worst | bests | individuals | Mean | $\sigma$ | worst | best |
| FFDE | n/a | 481095 | n/a | n/a | n/a | n/a | 478171 | n/a | n/a | n/a |
| WCGA | 463753 | 711332 | 163 | 711038 | 711645 | 463296 | 726583 | 127 | 726454 | 726813 |
| WCGA-R | 464937 | 711253 | 101 | 711091 | 711378 | 463772 | 726544 | 150 | 726339 | 726732 |
| UGGA | 2516927 | **712937** | 66 | 712796 | 713019 | 2502070 | **728296** | 41 | 728222 | 728362 |
| UGGA-R | 2735163 | 712716 | 54 | 712621 | 712803 | 2732558 | 728024 | 60 | 727950 | 728144 |
| ALT | 1312243 | 712611 | 97 | 712524 | 712841 | 1356181 | 727961 | 90 | 727852 | 728091 |
| RSGA | 1520649 | 712622 | 86 | 712463 | 712750 | 1495412 | 727904 | 91 | 727769 | 728041 |
| **MULTIPLE SUBSET SUM INSTANCES** | | | | | | | | | | |
| | Instance 1 | | | | | Instance 2 | | | | |
| Algorithm | individuals | Mean | $\sigma$ | worst | best | individuals | Mean | $\sigma$ | worst | best |
| FFDE | n/a | 747026 | n/a | n/a | n/a | n/a | 762816 | n/a | n/a | n/a |
| WCGA | 470177 | 749855 | 192 | 749540 | 750176 | 466883 | 766468 | 107 | 766220 | 766579 |
| WCGA-R | 469720 | 749783 | 130 | 749549 | 749964 | 468152 | 766430 | 114 | 766240 | 766600 |
| UGGA | 2398800 | **751507** | 21 | 751475 | 751547 | 2392706 | **767773** | 22 | 767741 | 767803 |
| UGGA-R | 2767926 | 751326 | 19 | 751293 | 751355 | 2731756 | 767652 | 44 | 767595 | 767701 |
| ALT | 1344139 | 751269 | 49 | 751167 | 751333 | 1367280 | 767596 | 52 | 767512 | 767673 |
| RSGA | 1624572 | 751283 | 37 | 751244 | 751338 | 1681930 | 767602 | 39 | 767519 | 767659 |
| | Instance 3 | | | | | Instance 4 | | | | |
| Algorithm | individuals | Mean | $\sigma$ | worst | best | individuals | Mean | $\sigma$ | worst | best |
| FFDE | n/a | 707080 | n/a | n/a | n/a | n/a | 722512 | n/a | n/a | n/a |
| WCGA | 470075 | 709404 | 208 | 709021 | 709698 | 471433 | 724862 | 141 | 724622 | 725066 |
| WCGA-R | 469535 | 709436 | 109 | 709307 | 709633 | 471833 | 724734 | 177 | 724471 | 725038 |
| UGGA | 2474574 | **711062** | 23 | 711022 | 711087 | 2467649 | **726409** | 27 | 726368 | 726450 |
| UGGA-R | 2754892 | 710988 | 27 | 710942 | 711027 | 2771604 | 726315 | 37 | 726276 | 726371 |
| ALT | 1374349 | 710956 | 33 | 710906 | 711010 | 1415012 | 726280 | 67 | 726178 | 726347 |
| RSGA | 1534461 | 710937 | 50 | 710862 | 710991 | 1558087 | 726249 | 49 | 726183 | 726335 |
| **WEAKLY CORRELATED INSTANCES** | | | | | | | | | | |
| | Instance 1 | | | | | Instance 2 | | | | |
| Algorithm | individuals | Mean | $\sigma$ | worst | best | individuals | Mean | $\sigma$ | worst | best |
| FFDE | n/a | 806906 | n/a | n/a | n/a | n/a | 778781 | n/a | n/a | n/a |
| WCGA | 454258 | 851013 | 742 | 849529 | 852306 | 467653 | 807812 | 646 | 806913 | 809248 |
| WCGA-R | 453993 | 850175 | 414 | 849312 | 850703 | 460357 | 807804 | 450 | 806919 | 808354 |
| UGGA | 2518209 | 836076 | 2608 | 832881 | 841013 | 2544247 | 798572 | 1187 | 797279 | 801225 |
| UGGA-R | 2852438 | 825284 | 1074 | 823505 | 826935 | 2771233 | 793116 | 1416 | 791224 | 796048 |
| ALT | 1346604 | 852603 | 306 | 852174 | 853034 | 1326441 | 809874 | 240 | 809520 | 810363 |
| RSGA | 1241924 | **852721** | 249 | 852428 | 853152 | 1286191 | **810179** | 114 | 810034 | 810343 |
| | Instance 3 | | | | | Instance 4 | | | | |
| Algorithm | individuals | Mean | $\sigma$ | worst | best | individuals | Mean | $\sigma$ | worst | best |
| FFDE | n/a | 723833 | n/a | n/a | n/a | n/a | 755329 | n/a | n/a | n/a |
| WCGA | 465363 | 749197 | 367 | 748624 | 749742 | 455743 | 791930 | 352 | 791356 | 792470 |
| WCGA-R | 463653 | 748981 | 285 | 748551 | 749495 | 456358 | 792169 | 478 | 791524 | 793126 |
| UGGA | 2146420 | 741216 | 1764 | 738844 | 743754 | 2575501 | 776295 | 2782 | 771752 | 780933 |
| UGGA-R | 1929849 | 741421 | 1120 | 739565 | 743330 | 2825803 | 771906 | 1767 | 768904 | 774406 |
| ALT | 1118997 | 750787 | 272 | 750438 | 751189 | 1224836 | 793421 | 276 | 792917 | 793747 |
| RSGA | 1066917 | **750888** | 241 | 750537 | 751214 | 1286829 | **793454** | 214 | 793121 | 793728 |

TABLE II

SOLUTION QUALITY COMPARISON WITH A 180 SECOND TIME LIMIT PER RUN, 10 RUNS PER INSTANCE: COMPARISONS ON 4 STRONGLY CORRELATED PROBLEMS, 4 MULTIPLE SUBSET-SUM INSTANCES, AND 4 WEAKLY CORRELATED INSTANCES. FOR EACH ALGORITHM, WE SHOWS THE MEAN AND STANDARD DEVIATION ($\sigma$) OF THE SCORES ATTAINED BY THE ALGORITHM. THE *best* AND *worst* ARE THE SCORES ACHIEVED BY THE BEST AND WORST RUNS (OUT OF THE 10 INDEPENDENT RUNS) FOR EACH INSTANCE BY EACH ALGORITHM.

## ACKNOWLEDGEMENTS

This work was supported by the Japan MEXT program, "Promotion of Env. Improvement for Independence of Young Researchers", the JSPS Compview GCOE, and a JSPS Grant-in-Aid for Young Scientists 20700131.

## REFERENCES

[1] E.G. Coffman, M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing, 1996.

[2] S. Eilon and N. Christofides. The loading problem. *Management Science*, 17(5):259–268, 1971.

[3] E. Falkenauer. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2):123–144, 1994.

[4] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[5] A. Fukunaga. A new grouping genetic algorithm for the multiple knapsack problem. In *Proc. IEEE Congress on Evolutionary Computation*, pages 2225–2232, 2008.

[6] A. Fukunaga and R. Korf. Bin-completion algorithms for multicontainer packing, knapsack, and covering problems. *Journal of Artificial Intelligence Research*, 28:393–429, 2007.

[7] T. Ibaraki. The power of dominance relations in branch-and-bound algorithms. *The Journal of the Association of Computing Machinery*, 24:264–279, 1977.

[8] J.R. Kalagnanam, A.J. Davenport, and H.S. Lee. Computational aspects of clearing continuous call double auctions with assignment constraints and indivisible demand. *Electronic Commerce Research*, 1:221–238, 2001.

[9] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer-Verlag, 2004.

[10] M. Labbé, G. Laporte, and S. Martello. Upper bounds and algorithms for the maximum cardinality bin packing problem. *European Journal of Operational Research*, 149:490–498, 2003.

[11] S. Martello and P. Toth. Heuristic algorithms for the multiple knapsack problem. *Computing*, 27:93–112, 1981.

[12] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, 1990.

[13] T. Okabe, Y. Jin, and B. Sendhoff. Evolutionary multi-objective optimisation with a hybrid representation. In *Proc. IEEE Congress on Evolutionary Computation*, pages 2262–2269, 2003.

[14] D. Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114:528–541, 1999.

[15] G.R. Raidl. The multiple container packing problem: A genetic algorithm approach with weighted codings. *ACM SIGAPP Applied Computing Review*, pages 22–31, 1999.

[16] G.R. Raidl and G. Kodydek. Genetic algorithms for the multiple container packing problem. In *Proc. 5th International Conf. on Parallel Problem Solving from Nature V, Amsterdamm, The Netherlands*, Lecture Notes in Computer Science 1498, pages 875–884, 1998.

[17] C. Reeves. Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research*, 63:371–396, 1996.

[18] T. Schnier and X. Yao. Using multiple representations in evolutionary algorithms. In *Proc. IEEE Congerss on Evolutionary Computation*, pages 479–486, 2000.

[19] D. Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proc. International Conf. on Genetic Algorithms (ICGA)*, pages 116–121, 1989.