A New Grouping Genetic Algorithm for the Multiple Knapsack Problem

Alex S Fukunaga, *Member, IEEE*

***Abstract*— The Multiple Knapsack Problem (MKP) is the 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. We propose a new genetic algorithm for the MKP which searches a space of undominated candidate solutions. We compare the new algorithm to previous heuristics for the MKP, as well as alternative evolutionary algorithms, and show experimentally that our new algorithm yields the best performance on difficult instances where item weights and profits are highly correlated.**

I. Introduction

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.[1]

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. Let the binary decision variable x_{ij} be 1 if item j is placed in container i, and 0 otherwise. The 0-1 MKP can be formulated as the integer program below, where constraint 2 encodes the capacity constraint for each container, and constraint 3 ensures that each item is assigned to at most one container.

$$\text{maximize} \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \quad (1)$$

$$\text{subject to:} \sum_{j=1}^{n} w_j x_{ij} \leq c_i, \qquad i = 1, ..., m \quad (2)$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \qquad j = 1, ..., n \quad (3)$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j. \quad (4)$$

Alex Fukunaga is with the Global Edge Institute, Tokyo Institue of Technology, Meguro, Tokyo, Japan; email: fukunaga@is.titech.ac.jp.

[1]The MKP has also been called the "Multiple Container Packing Problem" [1]. The term "Multiple Knapsack Problem" has also been used by some authors in the evolutionary computation field to describe a very different problem which is known as the Multi-Dimensional Knapsack Problem in the standard terminology of the knapsack literature (See [2], [3] for a comprehensive overview of knapsack-related problems).

The MKP has many applications, including continuous double-call auctions [4], multiprocessor scheduling [5], vehicle/container loading [6], and the assignment of files to storage devices in order to maximize the number of files stored in the fastest storage devices [5]. 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.[2] The MKP is closely related to the classical *bin-packing problem* (BPP), which is the problem of minimizing the number of (uniform capacity) bins required to contain every item in a set of n items with weights $w_1, ..., w_n$. Key differences between the MKP and BPP are: (1) MKP items have both a weight and a profit, whereas BPP items only have a weight attribute. (2) Bins in the MKP have varying sizes, while BPP bin capacities are uniform. (3) In the MKP, we are trying to maximize the total profits of the items packed in a fixed set of bins, and we only pack a subset of the items – some items are unused. In the BPP, all items must be packed.

In this paper, we propose a new evolutionary algorithm for the MKP, the Undominated Grouping GA (UGGA), which differs significantly from other algorithms in that it only generates candidate solutions which are guaranteed to *not* be dominated according to a particular dominance criterion. Thus, the UGGA searches a space of locally optimal solutions according to one particular, weak notion of local optimality (dominance). We experimentally compare the UGGA with previous standard algorithms for the MKP, as well as previous genetic algorithms, and show that the UGGA significantly outperforms previous algorithms on hard, benchmark instances where item weights and profits are highly correlated.

The rest of this paper is organized as follows. Sections II-III describes previous heuristic and genetic approaches for the MKP, including adaptations of some algorithms from the bin-packing literature. Section IV defines a dominance criterion for the MKP Then in Section V, we propose the UGGA, a new algorithm which more exploits our dominance criterion by only generating undominated candidate solutions. Experimental results are shown in Section VI. Section VII concludes with a discussion of our results.

II. Heuristic algorithms for the MKP

This section gives an overview of previous non-evolutionary algorithms for solving the MKP. An important

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

978-1-4244-1823-7/08/$25.00

concept through the rest of this 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.

A. First-Fit Decreasing Efficiency Heuristic (FFDE)

This is a simple heuristic which is based on the well-known First-Fit Decreasing heuristic for bin packing [9]. 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).

B. MTHM

MTHM is an approximation algorithm for the MKP [10]. As with the FFDE heuristic described above, the items are sorted in increasing order of efficiency, and the bins are sorted in decreasing order of capacity. First, the items are greedily assigned to the knapsack. This greedy phase is followed by two improvement phases: (1) items are exchanged among knapsacks in order to allow the insertion of additional items, and (2) included items are swapped with non-included items in order to improve the score. Further details are in [10], [2]. In our experiments, we used Martello and Toth's Fortran implementation for the MTHM.

C. Mulknap

The current state-of-the-art algorithm for large MKP instances is Pisinger's Mulknap algorithm [11]. Mulknap is a complex branch-and-bound algorithm, which applies sophisticated upper-bounding and lower-bounding techniques at every node, in addition to problem reduction and bin capacity tightening. See [11] for details. 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. In our experiments, we used Pisinger's C implementation of Mulknap.

III. Evolutionary Approaches for the MKP

The most straightforward class evolutionary approaches to the MKP uses a *direct* encoding, where an individual 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 [12]. Previous work in both the MKP [1] and bin-packing [13], [8] 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. Due to space, we do not further detail this class of representations. Below, we describe two more promising approaches: an order-based representation, and a group-based representation.

A. Weighted-Coding Genetic Algorithm (WCGA)

Raidl proposed a *Weight-Coded GA* (WCGA) for the MKP. Each individual is a vector of *modifiers* $M = (m_1, m_2, ..., m_n)$[3]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 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 [1] for the details of weight generation. We implemented Raidl's WCGA as described in [1]. In particular, we implemented a WCGA using decoding heuristic B[4]), using relative value ordering.

B. Grouping Genetic Algorithm (GGA)

Falkenauer proposed the Grouping GA (GGA) for the bin packing problem [7]. We describe a straightforward adaptation of the GGA for the MKP in some detail, since this is the basis for the new UGGA described in Section V.

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.

The GGA crossover operator exchanges a set of corresponding bins (of the same size) between two individuals. 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, and the other instances are removed (because the purpose of the crossover is to move coherent bins from one individual to another). Then, FFDE (Section II-A) is executed in order to fill any empty spaces created by the bin exchange and subsequent duplicate removal. Our

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

[4]Raidl's B decoding heuristic is similar to the well-known Best-Fit packing heuristic for bin-packing problem [9]

GGA crossover is a uniform crossover: each bin is marked for exchange with probability p_c.

For example, consider a MKP instance with 3 bins and 13 items labeled with indices from $a, b, c, ..., m$. Let $P_1 = \{(a, b, c), (d, e, f), (g, h, i), (j, k)\}$ be one parent, where the first bin contains items with indices a, b, c the second bin contains the items with indices d, e, f, and so on. Let $P_2 = \{(a, d, i), (b, c, g), (e, f, k), (j, h)\}$ be the other parent (the items are assigned to different bins). The items l, m are not assigned to either parent. The uniform crossover determines whether to cross each bin with probability p_c. Assume that the second and fourth bins are marked for crossover.

If we cross bins 2 and 4, the resulting two children are: $C_1 = \{(a, b, c), (b, c, g), (g, h, i), (j, h)\}$ and $C_2 = \{(a, d, i), (d, e, f), (e, f, k), (j, k)\}$ Items $b, c, g, h,$ appear twice in C_1. We remove duplicate items, keeping the instances which arrived as a result of the bin exchange. This leaves us with $C_1 = \{(a), (b, c, g), (h, i), (j, h)\}$, We then apply the FFDE heuristic, in order to fill any remaining space in C_1 using some subset of items d, e, f, k, l, m, n.

The GGA mutation operator iterates through the bins of an individual GGA genome. Each bin is selected for mutation with probability p_m. Then, all of the selected bins are cleared (emptied), and then FFDE is applied in order to refill the bins.

There are several differences between our implementation of GGA for the MKP compared to Falkenauer's GGA [7] for bin-packing. First, Falkenauer has an inversion operator. This is reasonable for the bin packing problem, where all bins have the same capacity. However, capacities for the MKP are variable, so inversion does not translate well into the MKP. Falkenauer used a 2-point crossover; we experimented with 1-point, 2-point, and uniform crossover, and found that uniform crossover performed best on the MKP.

IV. Exploiting group dominance in a GGA

It is possible to improve upon the GGA by focusing the search to avoid bin assignments that can be identified as suboptimal. The notion of a dominance relationship between bin assignments enables us to implement this idea.

Given a set of k remaining items, we say that a bin assignment S is *maximal* with respect to capacity c if S is feasible, and adding any of the k remaining items would make it infeasible. We say that 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. One feasible, candidate bin assignment for the bin of capacity 100 is $F_1 = \{i_1, i_2\}$. Another feasible candidate bin assignment is $F_2 = \{i_1, i_3\}$. Finally, another feasible candidate bin assignment is $F_3 = \{i_1\}$. Note that F_1 and F_2 are maximal, feasible assignments; F_3 is feasible but non-maximal. 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. In other words, non-maximal bin assignments are always dominated. Similarly, regardless of what the other remaining items and bins are, the best solution that can be obtained using F_2 is no better than the best solution that can be obtained using F_1, so F_1 dominates F_2.

A formal criterion for dominance which generalizes these examples is the following:

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.

This dominance criterion was proposed and proved in [14]; this was, in turn, based on the Martello-Toth dominance criterion for the bin packing problem [2]. While the MKP dominance criterion was originally proposed to prune search nodes in a branch-and-bound algorithm, we now show that it can be used very effectively in a genetic algorithm.

A. "Hybrid" Grouping Genetic Algorithm (HGGA)

Falkenauer [8] proposed a "Hybrid Grouping GA" (HGGA) for the bin packing problem, which extended his previous GGA for bin packing (adapted for the MKP above in Section III-B). The HGGA exploits a dominance criterion by Martello and Toth [2] for the bin packing problem similar to the MKP dominance criterion described above. We describe our adaptation of the HGGA to the MKP.

As described in Section III-B, the crossover and mutation operators for GGA involve reinserting objects using a FFDE decreasing heuristic. In both the crossover and mutation operators: Previous to reinserting objects using FFDE, the HGGA applies the following local optimization:Iterate over each bin, and attempt to replace the bin with a bin assignment of cardinality D or less which dominates the current bin assignment according to criterion 1 (after experimenting with $D = 2$ $D = 3$, we found that D=3 performed slightly better). Then, the FFDE heuristic is applied in order to fill any remaining spaces.[5]

This is a limited approach to exploiting dominance: while a single pass of the local optimization can replace a bin assignment A with another bin assignment A' which dominates A, A' in turn might be dominated by some other bin assignment A''. Furthermore, looking for a bin assignment which dominates a bin assignment A can be quite expensive – in the worst case, if A is an undominated assignment, we must show that there is no subset of the remaining elements

[5]We also implemented a scheme closer to Falkenauer's algorithm, which repeatedly applied the local optimization until no further improvements can be made. However, this led to worse results on the MKP.

of cardinality $\leq D$ which dominates A. This involves an expensive, backtracking-based enumeration of the possible candidate bin assignments which could dominate A.

V. The Undominated Grouping Genetic Algorithm (UGGA)

We now present the *Undominated Grouping Genetic Algorithm* (UGGA), a novel, genetic algorithm which fully exploits the dominance criteria described in Section IV. The main feature of the UGGA is that each step, 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 representation of individuals is the same group-based representation as the GGA.

First, we generate a population of individuals. Then, we execute a steady-state genetic algorithm, where at each generation, we select two parents and mate them (using the reproduction operators described below) to generate two children, which are then evaluated for fitness. The children are inserted into the population, replacing the two worst members of the population. The score of the best individual generated in the entire run is stored and returned at the end of the UGGA run. The key operations (initialization, crossover, mutation) are detailed below, but first, we describe a method for generating only undominated bin assignments, which is required by these genetic operators.

A. Generating Undominated Bin Assignments

A key component of the UGGA is 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.

We now describe *FillUndominated*, an algorithm that generates an undominated bin assignment using only linear space. Given n elements and a container capacity c, the *feasible bin assignments* are the subsets of the n elements, the sum of whose weights do not exceed c. We can enumerate all feasible bin assignments by recursively traversing a binary tree of depth n, where the left branch represents including an item, and the right branch represents excluding the item. A parameter representing the remaining capacity of the bin is incrementally computed and passed down the tree.

Now, we further extend the algorithm to generate only the *undominated*, feasible bin assignments. Suppose that we have a feasible bin assignment A whose sum of weights is t, and whose sum of profits is p. The *excluded items* are all items that are not in A. A set (bin assignment) A will be dominated according to Criterion 1 if and only if it contains any subset $s \subset A$ whose weight sum is less than or equal to the weight of an excluded item x, and the profit of x is greater than or equal to the sum of profits of the items in s, such that replacing the subset with x will not exceed the bin capacity c.

Therefore, to check if A is undominated, we enumerate each possible subset of its items, and for each subset $s \subset A$, we compare it against each excluded item x to verify that we *cannot* replace the subset with x (where the weight of x is greater than or equal to the sum of the weights in s) to obtain a feasible bin with equivalent or higher profit. If A passes this verification, it is undominated.

As soon as we find an undominated bin assignment, we can terminate the algorithm and return the assignment. This algorithm generates feasible bin assignments and immediately tests them for dominance, so it never stores multiple dominated bin assignments. Furthermore, the dominance test is done by comparing the included elements to excluded elements, and *does not involve any comparison between a candidate bin assignment and previously generated bin assignments*. Therefore, the memory required for dominance testing is linear in the number of items.

The algorithm described above is deterministic, so given a set of n items and a bin capacity c, this algorithm will always return the *same* undominated bin assignment. This is undesirable because we can get stuck in a local optimum. Therefore, when generating the feasible assignments, we randomize the order in which the items are considered during the backtracking generation of feasible assignments. The resulting randomized backtracking algorithm is equally likely to generate any of the undominated assignments (given a particular bin and set of items as input).

B. Individual creation and reproduction

Given the randomized, *FillUndominated* algorithm described above, it is now possible to detail the key genetic operators of the UGGA.

The *initialization* operator creates a new individual during population creation. A new individual with empty bin assignments is created. We iterate over each bin in increasing order of bin capacity, and each bin is filled by calling *FillUndominated*.

Reproduction is implemented in the UGGA as follows. We select two parents using rank-based selection [15]. Two children are created by copying the parents. Then, the crossover operator described below is applied. After the crossover, the mutation operator described below is applied.

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 (similar to the policy in the crossover operator for the GGA in Section III-B). The bins which contain the redundant copies of the items are then 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*.

The *mutation* operator 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*.

One interesting question is whether the UGGA, which limits its search space to undominated candidate solutions, can actually find an *optimal* solution to a MKP instance. This turns out to be the case, at least in principle. Let S be a set of m bin assignments $A_1, ..., A_m$. We say that S is an *undominated candidate solution* if there is no way to replace some or all of the any bin assignment A_i with items currently not assigned to a bin in S to obtain a bin which dominates A_i. Otherwise, S is a *dominated candidate solution*.

Proposition 2: For every MKP instance, there exists at least one undominated solution.

Proof: Let S be an optimal solution to an MKP instance M. If S is undominated, then we are done. However, suppose the optimal solution S is a dominated candidate solution. Then by definition, there exists some bin $A_k \in S$ such that A_k can be replaced with a bin assignment A'_k such that $SumProfit(A') \geq SumProfit(A)$, where $SumProfit(b)$ denotes the sum of profits in a bin assignment b. Since S was assumed to be optimal, $SumProfit(A'_k) = SumProfit(A_k)$. Repeat this procedure on bin k until bin k is undominated (guaranteed to terminate because the number of items n is finite). We can repeat this process for all dominated bins, until all bin assignments are undominated. Since the solution remains optimal at each step, this process must eventually result in an optimal solution which is undominated. Thus, there exists at least one undominated, optimal solution. □

Based on the above, it is easy to see that the UGGA can, in theory, eventually find an optimal solution. Both the initialization operator and the mutation generator are clearly able to randomly generate any undominated candidate solution (given either sufficient population size or sufficiently high mutation probability and sufficient time). According to the above Proposition, there is at least one optimal solution which is an undominated solution, the UGGA will, with probability approaching 1, eventually discover it given enough resources.

Of course, this only guarantees that in theory, UGGA has sufficient representational power to *eventually* find the optimal solution. There is no guarantee that the UGGA can actually find the optimal solution in a limited amount of time for any particular run on a given MKP instance, or even that the UGGA will perform well compared to other algorithms.

C. Random-Undominated: A Simple Randomized Algorithm

In order to isolate the power of the dominance criteria in focusing search, we also implemented *Random-Undominated*, a simple algorithm based on the individual initialization for the UGGA. Given an initial set of items and bins, *Random-Undominated* clears all of the bins, and repeatedly calls *FillUndominated* on all bins (in order of decreasing capacity) to generate an undominated, candidate solution. Since *FillUndominated* is randomized, we repeat this procedure until time runs out. This algorithms therefore samples the space of undominated candidate solutions – the only heuristic guidance used is that small bins are filled first. It turns out that this simple procedure is surprisingly powerful for some problems.

VI. Experimental Results

We experimentally evaluated the heuristics and evolutionary algorithms described in the previous sections using three classes of benchmarks which are frequently used in the MKP literature [3], [2].

- *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 Pisinger's experiments [11]. That is, we only used instances where: (a) each item fits into at least one of the bins, (b) the smallest bin is large enough to hold at least the smallest item, and (c) the sum of the item weights is at least as great as the size of the largest bin. We used items with weights in the range [1,10000].

In our experiments, we focus on problems that are known to be hard for the state of the art algorithm, Mulknap. Pisinger has shown that Mulknap can optimally solve problems with tens of thousands of items and few (up to $O(10)$) bins within seconds [11]. However, when the ratio of items to bins is approximately 2-5, problem instances become extremely difficult for all known solvers. For example, Mulknap can not solve instances with 15 bins and 45 items within given a 10 minute time limit [14]. This is similar to the existence of "phase transitions" in other problems such as satisfiability testing [16]. We therefore focused on instances with 300 items and 100 bins. We have similar results on a range of problem sizes where the ratio of items to bins is 3, ranging from 30 items and 10 bins to 300 items and 100 bins; due to space, we only show the largest instances.

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). All of the algorithms we implemented (FFDE, WCGA, GGA, HGGA, UGGA) were implemented using the CMUCL Common Lisp compiler. We used Martello and Toth's Fortran implementation of MTHM and Pisinger's C implementation of Mulknap. The FFDE and MTHM heuristics both terminated almost immediately (less than 0.1 seconds) on all instances. Mulknap will terminate as soon as it verifies that it has found an optimal solution (by exhausting the branch-and-bound search tree). However, on all of the instances tested in this paper, Mulknap did not terminate before the full time limit expired.

We ran the algorithms on a set of 10 strongly correlated MKP instances, 4 instances of weakly correlated instances, and 4 instances of multiple subset-sum instances.

After running some preliminary experiments (not shown here) to tune the control parameters (population, mutation rate, crossover rate), we compared the MKP heuristics and evolutionary algorithms on our benchmark instances. The parameters for the WCGA are the same as in [1] (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 (Section III-A).

The results are shown in Tables I-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 I shows the mean and standard deviation (σ) 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. The "individuals" columns show the average number of individuals generated by mating or randomized generation. Local optimization steps (such those taken by GGA, HGGA, and UGGA) are *not* counted in this statistic. This statistic is indicative of both: (a) the cost per "step" in various algorithms, and (b) the amount of effort expended for local optimization relative to the GA. For example, the "individuals" columns for the HGGA indicate that the HGGA spends most of its time executing local optimization,

The new UGGA significantly outperformed all other algorithms for strongly correlated and multiple subset-sum benchmarks. Furthermore, its performance is very stable – for *every* strongly correlated and multiple subset-sum problem instance, the final score for the *worst* run of UGGA was better (higher) than the final score of the *best* run on any of the other algorithms.

On the other hand, the UGGA performed poorly on the weakly correlated instances (significantly worse than the WCGA and GGA). The Weight-Coded GA performed significantly better than all other algorithms on weakly correlated instances, and second only to the UGGA on the strongly correlated and multiple subset-sum instances.

As an additional experiment, we ran UGGA and our implementation of WCGA on the instances used in the paper by Raidl [1], which originally proposed the WCGA for the MKP. This is a set of 21 problems with n between 30-200, and m between 3-80. For each item, j, the weight w_j is uniformly distributed, and the profit p_j was in the range $[0.8w_j, 1.2w_j]$. Thus, all of Raidl's instances are similar to the *weakly correlated* class of instances used in the previous experiment described above. The WCGA performed identically to the results in Raidl's paper [1], achieving the same gap relative to an LP upper bound using roughly the same resources (evaluations). As with the weakly correlated instances, the UGGA results were poor on these instances, and significantly underperformed the WCGA on all instances.

A possible explanation for the poor performance of the UGGA and HGGA on weakly correlated instances is that the dominance test does not adequately reduce the search space when the item weights and profits are less correlated. Establishing that some bin assignment A dominates another bin assignment A' according to Criterion 1 requires both the weights and profits of A' be packable in the weights and profits of A. This occurs more frequently when item weights and profits are correlated (in other words, the probability that a given feasible bin assignment A dominates another assignment A' is higher for correlated instances than for uncorrelated instances). Thus, the set of all undominated bin assignments is relatively larger for weakly correlated instances compared with strongly correlated instances. In this case, there is insufficient advantage to searching the space of undominated assignments, particularly given the large overhead required in order to guarantee that we are only generating undominated solutions. As shown in Table I UGGA generates significantly fewer solutions as GGA, given the same time limit, so unless the dominance criterion sufficiently limits the search space, it may be better not to use dominance at all.

The behavior of *Random-Undominated* somewhat confirms this explanation. *Random-Undominated* depends entirely on the effectiveness of the dominance criterion to sample promising regions of the space of all bin assignments (Section V-C). This simple procedure performed surprisingly well on strongly correlated and multiple subset-sum instances – it performs significantly better than the GGA and HGGA. On the other hand, this simple algorithm performed very poorly on the weakly correlated instances.

VII. Conclusions

We proposed the UGGA, a new genetic algorithm for the MKP. The UGGA uses a group-based representation, where each gene represents the set of items that are assigned to a particular group (bin). Although group-based representations have been used previously (e.g., Falkenauer's Grouping GA for bin-packing), the UGGA differs from all previous approaches in that it maximally exploits a dominance criterion among candidate bin assignments by only generating candidate solutions that are undominated

Configuration				Instance 1					Instance 2				
Algorithm	pop	cross	mut	individuals	Mean	σ	worst	best	individuals	Mean	σ	worst	best
FFDE	n/a	n/a	n/a	n/a	504429	n/a	n/a	n/a	n/a	521987	0	521987	521987
MTHM	n/a	n/a	n/a	n/a	699757	n/a	n/a	n/a	n/a	681330	0	681330	681330
Mulknap	n/a	n/a	n/a	n/a	745837	n/a	n/a	n/a	n/a	767114	n/a	n/a	n/a
WCGA	100	0.5	0.03	462470	751648	184	751260	751932	467439	768262	109	768066	768387
GGA	100	0.01	0.01	3264256	749516	343	748930	749958	3289261	765521	339	764890	766150
HGGA	100	0.01	0.01	1564	749030	202	748833	749414	1664	764922	331	764184	765373
UGGA	100	0.01	0.01	2452429	**753370**	47	753278	753428	2439979	**769614**	40	769569	769683
Random-Undominated	n/a	n/a	n/a	47557	750421	145	750242	750649	45109	766719	157	766518	766941
Configuration				Instance 3					Instance 4				
Algorithm	pop	cross	mut	individuals	Mean	σ	worst	bests	individuals	Mean	σ	worst	best
FFDE	n/a	n/a	n/a	n/a	481095	n/a	n/a	n/a	n/a	478171	n/a	n/a	n/a
MTHM	n/a	n/a	n/a	n/a	629253	n/a	n/a	n/a	n/a	673521	n/a	n/a	n/a
Mulknap	n/a	n/a	n/a	n/a	708087	n/a	n/a	n/a	n/a	722244	n/a	n/a	n/a
WCGA	100	0.5	0.03	462880	711446	136	711228	711592	462631	726772	144	726552	727051
GGA	100	0.01	0.01	3251066	709328	231	708990	709669	3272602	724378	158	724108	724634
HGGA	100	0.01	0.01	1604	708916	189	708590	709131	1690	724438	272	724210	724997
UGGA	100	0.01	0.01	2516927	**712937**	66	712796	713019	2502070	**728296**	41	728222	728362
Random-Undominated	n/a	n/a	n/a	54665	710213	192	710062	710650	53094	725781	126	725618	725985
Configuration				Instance 5					Instance 6				
Algorithm	pop	cross	mut	individuals	Mean	σ	worst	best	individuals	Mean	σ	worst	best
FFDE	n/a	n/a	n/a	n/a	516694	n/a	n/a	n/a	n/a	510440	n/a	n/a	n/a
MTHM	n/a	n/a	n/a	n/a	711381	n/a	n/a	n/a	n/a	661103	n/a	n/a	n/a
Mulknap	n/a	n/a	n/a	n/a	765050	n/a	n/a	n/a	n/a	734766	n/a	n/a	n/a
WCGA	100	0.5	0.03	462366	772854	170	772697	773166	457569	738881	148	738594	739048
GGA	100	0.01	0.01	3315200	770471	386	769927	771207	3233501	736526	310	736144	737023
HGGA	100	0.01	0.01	1579	770160	301	769670	770610	1683	736140	347	735429	736582
UGGA	100	0.01	0.01	2430143	**774614**	53	774528	774700	2491512	**740188**	44	740116	740255
Random-Undominated	n/a	n/a	n/a	49517	771148	79	771029	771254	54432	737611	95	737500	737800
Configuration				Instance 7					Instance 8				
Algorithm	pop	cross	mut	individuals	Mean	σ	worst	best	individuals	Mean	σ	worst	best
FFDE	n/a	n/a	n/a	n/a	512466	n/a	n/a	n/a	n/a	509252	n/a	n/a	n/a
MTHM	n/a	n/a	n/a	n/a	669063	n/a	n/a	n/a	n/a	704983	n/a	n/a	n/a
Mulknap	n/a	n/a	n/a	n/a	742112	n/a	n/a	n/a	n/a	749965	n/a	n/a	n/a
WCGA	100	0.5	0.01	459684	743443	106	743309	743614	455230	756413	136	756144	756652
GGA	100	0.01	0.01	3250968	741422	205	741134	741755	3255280	754028	406	753150	754512
HGGA	100	0.01	0.01	1615	740869	193	740535	741165	1689	753894	271	753490	754330
UGGA	100	0.01	0.01	2446398	**744783**	54	744674	744856	2430945	**757975**	36	757890	758016
Random-Undominated	n/a	n/a	n/a	53133	742424	122	742225	742642	50352	755464	72	755363	755585
Configuration				Instance 9					Instance 10				
Algorithm	pop	cross	mut	individuals	Mean	σ	worst	best	individuals	Mean	σ	worst	best
FFDE	n/a	n/a	n/a	n/a	502112	n/a	n/a	n/a	n/a	531489	n/a	n/a	n/a
MTHM	n/a	n/a	n/a	n/a	688309	n/a	n/a	n/a	n/a	720932	n/a	n/a	n/a
Mulknap	n/a	n/a	n/a	n/a	756370	n/a	n/a	n/a	n/a	783801	n/a	n/a	n/a
WCGA	100	0.5	0.01	453835	757323	113	757188	757560	461137	785046	201	784635	785313
GGA	100	0.01	0.01	3271950	755440	246	755138	755751	3320704	782614	302	782091	782979
HGGA	100	0.01	0.01	1609	755171	167	754953	755483	1592	782228	299	781837	782777
UGGA	100	0.01	0.01	2508361	**758685**	51	758612	758752	2403495	**786644**	32	786570	786704
Random-Undominated	n/a	n/a	n/a	52951	756309	70	756229	756444	43814	783993	122	783833	784212

TABLE I

COMPARISONS ON 10 HARD, STRONGLY CORRELATED INSTANCES (10 RUNS PER INSTANCE, 180 SECONDS PER RUN).

according to the dominance criterion. This is enabled by a randomized, practical (linear-space) algorithm for generating only undominated bin assignments. Compared to Falkenauer's HGGA, which uses the dominance criterion in a local optimization phase, the UGGA more fully exploits the dominance criterion. Furthermore, the UGGA is in some sense a simpler search algorithm – there is no separate, local optimization procedure.

Our experimental results showed that on MKP instances where item weights and profits are highly correlated, the UGGA outperformed other heuristic and evolutionary approaches. On the other hand, for weakly correlated instances, the Weight-Coded GA performed best. We conjectured that this is because our dominance criterion does not sufficiently reduce the search space when item weights and profits are not correlated; this hypothesis is partially supported by the performance of *Random-Undominated*, which samples the space of undominated assignments. Future work will seek to determine more precisely the criteria under which UGGA is successful.

Thus, main the contributions of this paper are:

- We proposed the UGGA, a new grouping-based GA which searches the space of undominated candidate solutions.
- We showed that the UGGA is a new, state-of-the-art algorithm for difficult, large MKP instances where the item weights and profits are highly correlated.

Although this paper focused on the UGGA for the MKP, the general approach of the UGGA (i.e., searching the space of undominated solutions) is applicable to other problems involving the assignment/packing of items into containers/bins, where a similar dominance criterion can be derived. Similar dominance criteria have been found for the bin-packing problem [2], bin covering problem and for the min-cost covering

MULTIPLE SUBSET SUM INSTANCES													
Configuration				Instance 1					Instance 2				
Algorithm	pop	cross	mut	individuals	Mean	σ	worst	best	individuals	Mean	σ	worst	best
FFDE	n/a	n/a	n/a	n/a	747026	n/a	n/a	n/a	n/a	762816	n/a	n/a	n/a
MTHM	n/a	n/a	n/a	n/a	747026	n/a	n/a	n/a	n/a	762816	n/a	n/a	n/a
Mulknap	n/a	n/a	n/a	n/a	744773	n/a	n/a	n/a	n/a	764293	n/a	n/a	n/a
WCGA	100	0.5	0.01	459386	749922	82	749817	750051	462056	766492	159	766163	766665
GGA	100	0.01	0.01	3829413	748927	123	748638	749067	3971218	764187	296	763518	764470
HGGA	100	0.01	0.01	1464	747471	244	747081	747884	1467	763034	182	762808	763307
UGGA	100	0.01	0.01	2398800	**751507**	21	751475	751547	2392706	**767773**	22	767741	767803
Random-Undominated	n/a	n/a	n/a	47212	748674	103	748539	748872	45605	765032	173	764836	765325
Configuration				Instance 3					Instance 4				
Algorithm	pop	cross	mut	individuals	Mean	σ	worst	best	individuals	Mean	σ	worst	best
FFDE	n/a	n/a	n/a	n/a	707080	n/a	n/a	n/a	n/a	722512	n/a	n/a	n/a
MTHM	n/a	n/a	n/a	n/a	707080	n/a	n/a	n/a	n/a	722512	n/a	n/a	n/a
Mulknap	n/a	n/a	n/a	n/a	706549	n/a	n/a	n/a	n/a	721480	n/a	n/a	n/a
WCGA	100	0.5	0.01	459404	709486	91	709317	709612	460556	724818	175	724562	725161
GGA	100	0.01	0.01	3732500	708656	160	708408	708827	3853962	723963	194	723632	724229
HGGA	100	0.01	0.01	1514	707485	194	707175	707794	1484	722599	262	722357	723097
UGGA	100	0.01	0.01	2474574	**711062**	23	711022	711087	2467649	726409	27	**726368**	726450
Random-Undominated	n/a	n/a	n/a	55957	708362	103	708235	708545	54594	723947	105	723797	724114
WEAKLY CORRELATED INSTANCES													
Configuration				Instance 1					Instance 2				
Algorithm	pop	cross	mut	individuals	Mean	σ	worst	best	individuals	Mean	σ	worst	best
FFDE	n/a	n/a	n/a	n/a	806906	n/a	n/a	n/a	n/a	778781	n/a	n/a	n/a
MTHM	n/a	n/a	n/a	n/a	806906	n/a	n/a	n/a	n/a	778781	n/a	n/a	n/a
Mulknap	n/a	n/a	n/a	n/a	843374	n/a	n/a	n/a	n/a	801497	n/a	n/a	n/a
WCGA	100	0.5	0.01	468557	**851940**	1173	849272	853305	468550	**809522**	797	808099	810504
GGA	100	0.01	0.01	3356774	844882	1041	843185	846112	3366824	803587	1035	801304	805125
HGGA	100	0.01	0.01	2100	837125	1766	834605	839340	1999	797239	1956	794064	799693
UGGA	100	0.01	0.01	2518209	836076	2608	832881	841013	2544247	798572	1187	797279	801225
Random-Undominated	n/a	n/a	n/a	74678	805086	747	804024	806437	69673	770207	579	769204	770862
Configuration				Instance 3					Instance 4				
Algorithm	pop	cross	mut	individuals	Mean	σ	worst	best	individuals	Mean	σ	worst	best
FFDE	n/a	n/a	n/a	n/a	723833	n/a	n/a	n/a	n/a	755329	n/a	n/a	n/a
MTHM	n/a	n/a	n/a	n/a	723833	n/a	n/a	n/a	n/a	755329	n/a	n/a	n/a
Mulknap	n/a	n/a	n/a	n/a	740210	n/a	n/a	n/a	n/a	780777	n/a	n/a	n/a
WCGA	100	0.5	0.01	462478	**750382**	486	749667	751117	464390	**792639**	483	791975	793289
GGA	100	0.01	0.01	3245244	745631	997	743958	746782	3311285	787438	968	785953	788641
HGGA	100	0.01	0.01	2271	738710	1252	736057	739696	2119	780556	1129	778882	782737
UGGA	100	0.01	0.01	2146420	741216	1764	738844	743754	2575501	776295	2782	771752	780933
Random-Undominated	n/a	n/a	n/a	49918	714391	683	713235	715442	74995	744201	1070	743056	745879

TABLE II

COMPARISONS ON 4 MULTIPLE SUBSET-SUM INSTANCES AND 4 WEAKLY CORRELATED INSTANCES (10 RUNS PER INSTANCE, 180 SECONDS PER RUN).

problem (sometimes known as the liquid loading problem) [14]. Thus, we believe that the UGGA is a promising, general approach for a significant class of difficult, problems involving assignment of items to containers. Future work will investigate the application of the UGGA to other problems such as bin packing and bin covering.

REFERENCES

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

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

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

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

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

[6] S. Eilon and N. Christofides, "The loading problem," *Management Science*, vol. 17, no. 5, pp. 259–268, 1971.

[7] E. Falkenauer, "A new representation and operators for genetic algorithms applied to grouping problems," *Evolutionary Computation*, vol. 2, no. 2, pp. 123–144, 1994.

[8] ——, "A hybrid grouping genetic algorithm for bin packing," *Journal of Heuristics*, vol. 2, pp. 5–30, 1996.

[9] E. Coffman, M. Garey, and D. Johnson, "Approximation algorithms for bin packing: A survey," in *Approximation Algorithms for NP-Hard Problems*, D. Hochbaum, Ed. PWS Publishing, 1996.

[10] S. Martello and P. Toth, "Heuristic algorithms for the multiple knapsack problem," *Computing*, vol. 27, pp. 93–112, 1981.

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

[12] G. 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*, ser. Lecture Notes in Computer Science 1498, 1998, pp. 875–884.

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

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

[15] 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)*, 1989, pp. 116–121.

[16] D. Mitchell, B. Selman, and H. Levesque, "Hard and easy distributions of SAT problems," in *Proceedings of AAAI*, 1992, pp. 459–65.