Integrating Symmetry, Dominance, and Bound-and-Bound in a Multiple Knapsack Solver

Alex S. Fukunaga

Global Edge Institute, Tokyo Institute of Technology, Meguro, Tokyo, Japan
fukunaga@is.titech.ac.jp,

Abstract. The multiple knapsack problem (MKP) is a classical combinatorial optimization problem. A recent algorithm for some classes of the MKP is bin-completion, a bin-oriented, branch-and-bound algorithm. In this paper, we propose path-symmetry and path-dominance, which are instances of the symmetry detection by dominance detection approach for pruning symmetric nodes in the MKP branch-and-bound search space. In addition, we integrate the "bound-and-bound" upper bound validation technique used in MKP solvers from the OR literature. We show experimentally that our new MKP solver, which integrates symmetry techniques from constraint programming and bound-and-bound techniques from operations research, significantly outperforms previous solvers on hard instances.

1 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.

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. Then 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.

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

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

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

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

The MKP has numerous applications, including task allocation among autonomous agents, continuous double-call auctions [7], multiprocessor scheduling [9], vehicle/container loading [1], and the assignment of files to storage devices in order to maximize the number of files stored in the fastest storage devices [9]. 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] Thus, state-of-the-art algorithms for finding optimal solutions are based on branch-and-bound. Previous work has shown that for problems where the ratio of items to bins is relatively small (i.e., $n/m < 4$), the state-of-the-art algorithm is bin-completion, a bin-oriented branch-and-bound algorithm [6].

The search space explored by bin-completion has many symmetric states. Previous work introduced some techniques for exploiting the symmetry and demonstrated their utility. In this paper, we further investigate methods for exploiting symmetries in the MKP bin-completion algorithm. We propose new techniques that result in significant improvements over the previous state of the art. These techniques are instances of the general symmetry breaking via dominance detection (SBDD) approach [2; 3].

A technique which is responsible for much of the power of previous branch-and-bound MKP solvers in the OR literature is "bound-and-bound" [10; 12], which seeks to prune nodes by heuristically seeking to validate the (optimistic) upper bound on the total profit at each search node. We integrated this technique into our extended bin-completion based MKP solver.

The paper is organized as follows. We start by reviewing the bin completion algorithm (Section 2). Section 3 defines the basic framework we use for symmetry detection and breaking, and reviews previous algorithms for exploiting symmetry in the MKP. We then introduce new, generalized symmetry detection techniques which are more powerful than the previous techniques. We discuss methods for combining various symmetry mechanisms, and compare these methods with related work on symmetry detection and breaking and in the constraint programming literature. We describe the bound-and-bound technique and our

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

```
1 search_MKP(bins, items, sumProfit)
2   if bins==∅ or items == ∅
3     if sumProfit > bestProfit then bestProfit = sumProfit; return
4   ri = reduce(bins,items) /* Pisinger's R2 reduction */
5   if ri ≠ ∅
6     search_MKP(bins, items \ ri, sumProfit)
7     return
8   upperBound = compute_upper_bound(items,bins)
9   if (sumProfit + upperB ≤ bestProfit
10    return /* upper-bound based pruning using SMKP bound */
11  if (validate_upper_bound(upperBound))
12    return /* bound-and-bound */
13  bin = choose_bin(bins)
14  undominatedAssignments = generate_undominated(items,capacity(bin))
15  foreach A ∈ sort_assignments(undominatedAssignments)
16    if not(symmetric(A))
17      assign A to bin
18      search_MKP(bins \ bin, items \ A, sumProfit+∑_{j∈A} p_j)
```

Fig. 1. Bin-completion-based algorithm for the MKP. The top-level call is `search_MKP(bins,items,0)`.

integration of bound-and-bound into bin-completion in Section 4. In Section 5, we experimentally evaluate various combinations of symmetry mechanisms, and conclude with a discussion of results and directions for future work.

2 Bin-Completion Algorithm for the MKP

Bin-completion is a branch-and-bound algorithm for finding optimal solutions to multi-container assignment problems including the MKP and bin packing problems [6]. We briefly describe this algorithm. **For simplicity of exposition, in the examples below, we assume (unless stated otherwise) multiple-subsets sum problem (MSSP) instances, where $\forall j, p_j = w_j$. Thus, whenever possible in the description below, we simply refer to an item by its weight.**

A *bin assignment* $B_i = (item_1, ..., item_k)$ is a set of all of the items that are assigned to a given bin i, $1 \leq i \leq m$. Thus, a valid solution to a MKP instance consists of a set of bin assignments, where each item appears in exactly one bin assignment. A bin assignment is *feasible* with respect to a given bin j if the sum of its weights does not exceed the capacity of the bin, c_i. Otherwise, the bin assignment is *infeasible*. We say that a bin assignment S is *maximal* with respect to bin i if S is feasible, and adding any other remaining items would make it infeasible.

The bin-completion algorithm searches a tree where each node at depth d, $1 \leq d \leq m$, represents a maximal, feasible bin assignment. The bin-completion

algorithm for the MKP is shown in Figure 1, where each call to `search_MKP` corresponds to a node in the branch-and-bound search tree (e.g., Figure 2).

Nodes are pruned according to an upper bound which is based on a relaxation of the problem by Martello and Toth [11] (Line 8). Pisinger's R2 reduction procedure [12] is applied at each node (Line 4) in order to try to reduce the problem by eliminating some items for consideration. The `choose_bin` function (Line 13) selects the bin b with least remaining capacity. The `generate_undominated` function generates the set of all maximal, feasible assignments for b, with the additional constraint that these assignments are not dominated by any other assignment according to a *dominance criterion.* Given two feasible bin assignments F_1 and F_2, F_1 *dominates* 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. Bin-completion prunes feasible assignments which are dominated according to the following MKP dominance criterion [6], which is based on the Martello-Toth dominance criterion for bin packing [11].

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 to 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.*

The undominated bin assignments are sorted (Line 20) in order of non-decreasing cardinality, and ties are broken in order of non-increasing profit. The `symmetric` function (Line 21) applies one of the symmetry detection strategies described in this paper, and `validate_upper_bound` implements the bound-and-bound strategy described in Section 4. For example, given a bin with capacity 10 and items 9,8,7,3,2, the undominated, feasible bin assignments are (9),(8,2), and (7,3). It is possible for there to be a very large number of undominated bin assignments generated by `generate_undominated`, but this problem can be avoided by processing these in smaller batches, and the only thing we lose is part of the benefits of the value ordering (`sort_assignments`). This is called hybrid incremental branching, and details are in [6]. Figure 2 shows part of an example bin-completion search tree.

3 Exploiting Symmetry

To describe our symmetry breaking mechanisms, which are instances of the general SBDD approach [2; 3], we first introduce some notation and define the notion of a *nogood*, which is central to all of our symmetry exploitation methods.

Let B^d denote a bin assignment which assigns the elements of set B to a bin at depth d. Thus, $(10, 8, 2)^1$ and $(10, 7, 3)^1$ denote two possible bin assignments for a bin at depth 1.

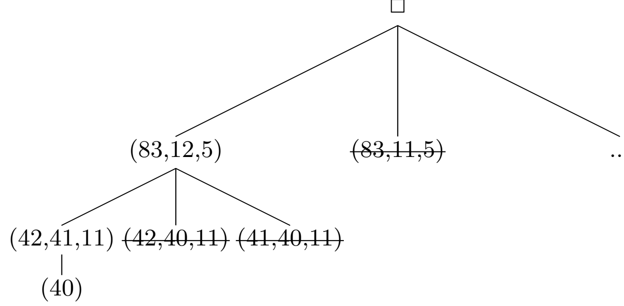

Fig. 2. Bin-completion search tree for a MKP instance with capacity 100 and items with weights {83,42,41,40,12,11,5} ($\forall i, p_i = w_i$). Each node represents a maximal, feasible bin assignment Bin assignments shown with a ~~strikethrough~~, e.g., (83,11,5), are pruned because they are dominated according to the criterion in Proposition 1.

Definition 1 (Nogood). *Let X^d be some node in the bin-completion search tree at depth d. Let $E^1, ..., E^{d-1}$ be ancestors of X^d at depths $1, ..., d-1$, respectively. For each such ancestor E_i, we say that every sibling of E^i to the left of E^i in the depth-first bin-completion search tree is a* nogood *with respect to X^d.*

In Figure 3, $(8, 2)^1$ is a nogood with respect to the descendants of $(7, 4)^1$. Since bin-completion is a depth-first branch-and-bound algorithm, a nogood denotes a bin assignment (node) whose descendants have been exhaustively searched in the current search tree. The union of all current nogoods is a concise description of the entire portion of the search tree which has been searched so far. This is similar to the use of the term "nogood" in [4].

3.1 Path-Symmetry

Consider the search tree shown in Figure 3. Assume that the capacities for bins 1-4 are 11,11,12, and 10, respectively. Assume that we have already exhaustively searched the subtree under $(8, 2)^1$, and we have generated the node $(7, 4)^1, (10)^2, (8, 3)^3, (6, 2, 2)^4$. By rearranging the items in bins 1-4, we can obtain a new set of bin assignments: $(8, 2)^1, (7, 3)^2, (10, 2)^3, (6, 4)^4$. This is a symmetric rearrangement, as the optimal solution under the first set of bin assignments is the same as the optimal solution under the latter set of assignments. Thus, we can prune the node at $(6, 2, 2)^4$.

More generally: Given a bin-completion search tree where we are considering a bin assignment for depth d, we define the *current path from depth g to depth d* as the union of bins g,$g+1$,...,d. The *current path items* are the union of all items in the current path. For example, in Figure 3, if we are at node $(6, 2, 2)^4$, the current path from depth 1 to 4 is the set of bins 1, 2, 3, and 4, and the current path items are $7, 4, 10, 8, 3, 6, 2, 2$.

Definition 2 (Path-Symmetry). *Let N^g be a nogood with respect to a candidate bin assignment B^d, and let P be the current path items from depth g to*

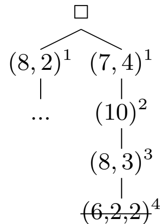

Fig. 3. The bin assignment $(6, 2, 2)^4$ can be pruned by Path-Symmetry. ($c_1 = 11, c_2 = 11, c_3 = 12, c_4 = 10$).

d. we say that there is a path-symmetry *with respect to nogood N^g if two conditions hold: (1) every item in N^g is a member of P, and (2) it is possible to (a) assign the items from the current path items corresponding to the items of N^g ($Items(N^g) \subset P$) to bin g , and (b) assign the remaining items ($P \setminus Items(N^g)$) to bins $g + 1, ..., d$ such that all bins $g, ..., d$ are feasible.*

If there is a path-symmetry between B^d and some nogood N^g as defined above, B^d can be pruned. The correctness follows directly from the definition of nogoods.

Checking the first condition of Definition 2 is straightforward. However, checking the second condition efficiently is not as straightforward, because it is essentially the decision version of a bin packing problem,[2] where we attempt to pack the items in $P \setminus Items(N^g)$ into bins with capacities $c_{g+1}, ..., c_d$. We describe several approaches:

In the first approach, we try to directly solve this bin packing problem using a simple backtracking algorithm (BT). The bin packing problem, like the MKP, is strongly NP-complete, and in the worst case, BT will take time which is $O(n^m)$, where n is the number of items and m is the number of bins. It is possible to avoid backtracking and use a standard bin packing heuristic such as first-fit decreasing (FFD), which has a polynomial complexity. Thus our second approach uses FFD to pack the items $P \setminus Items(N^g)$ into bins $g+1, ..., d$. The drawback of heuristics such as FFD is that it is not guaranteed to find a packing of the items into the bins even if one exists. However the symmetry check is still admissible – path-symmetry using a FFD check to test condition (2) may sometimes fail to prune a node that a BT check would have pruned, but will never prune a node that a BT check will not prune.

Another way to approximate the full check for condition (2) for path-symmetry is to limit the set of items that can be swapped among the bins. That is, instead of repacking all of the items $P \setminus Items(N^g)$ into bins $g + 1, ..., d$, we can "lock" some of the items into their current bins and only consider packing the unlocked items. We consider a *limited* packing problem (as opposed to the *full* packing

[2] In the decision version of bin packing, we are given m bins and n items, and the problem is to determine whether all n items can be packed into m bins such that the capacity constraints on all of the bins are not violated.

problem without locked items) where we (a) assign the items from the current path items corresponding to the items of $N^g (Items(N^g) \subset P)$ to bin g, and (b) pack the items $P \setminus Items(N^g)$ into bins $g+1, ..., d$, but in contrast to the full packing problem, we lock all of the items in $P \setminus Items(N^g)$ except for the items in bin g. In Fig. 3, the unlocked items would be the 7 and 4 from bin 1. The limited packing problem is to pack the 7 and 4 into three bins: bin #2 with remaining capacity 1 (the 10 is locked), bin #3 with remaining capacity 9 (the 8 is moved to bin #1, the original capacity is $c_3 = 12$, and there is a 3 which is locked, so the remaining capacity is 12-3=9), and bin #4 with remaining capacity 2 (one of items with weight 2 has moved to bin 1, but the remaining 6 and 2 are locked). In this case, the packing fails, so limited packing is insufficient, but a full packing (where all current path items were unlocked) would have enabled path symmetry detection. The choice of BT vs. FFD, and the choice of full vs. limited packing are orthogonal choices. Thus, full packing using BT will give us the full pruning power of path-symmetry (albeit at highest cost per node), while limited packing using FFD gives us a weaker (but cheaper) pruning test.

A more restricted version of this test was previously considered in [6]: Given a bin assignment B^d for the bin at depth d, we can prune B^d if there is a nogood N^g with respect to B^d such that (1) B^d includes all the items in N^g, and (2) if we swap the items in N^g from B^d with the items that are currently assigned to the bin at depth g, both resulting bin assignments are feasible. We call this strategy *2-swap-path-symmetry*, because it only considers symmetries that can be detected by swapping items between two particular bins.

3.2 Path-Dominance

Path-dominance is a generalization of path-symmetry. Consider the search tree shown in Figure 4 for an instance where the bin capacities for bins 1-3 are 11, 12, and 13, respectively. Assume that we have already exhaustively searched the subtree under $(8,2)^1$, and we have generated the current path in the search tree, $(7,4)^1, (5,6)^2, (9,2)^3$. By rearranging the items in bins 1-3, we can obtain a new set of bin assignments: $(7,2)^1, (5,6)^2, (9,4)^3$. This is a symmetric rearrangement, since the optimal solution under the first sequence of bin assignments must be the same as the optimal solution the latter sequence of assignments. Thus, we can prune the node $(9,2)^3$, since $(8,2)^1$ dominates $(7,2)^1$. More generally:

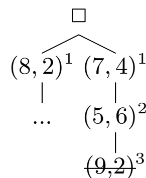

Fig. 4. The bin assignment $(9,2)^3$ can be pruned by Path-Dominance ($c_1 = 11, c_2 = 12, c_3 = 13$)

Definition 3 (Path-Dominance). *Let N^g be a nogood with respect to candidate bin assignment B^d, and let P be the current path items from depth g to d. We say that there is a* path-dominance symmetry *with respect to nogood N^g established at depth g if there exists some $s \subset P$ such two conditions hold: (1) s is dominated by N^g according to the MKP dominance criterion and (2) it is possible to (a) assign s to bin g, and (b) assign the remaining items ($P \setminus s$) to bins $g+1, ..., d$ such that all bins $g, ..., d$ are feasible.*

If there is a path-dominance symmetry between B^d and some nogood N^g as defined above, B^d can be pruned. This follows from the definition nogoods and Proposition 1.

Our current implementation of path-dominance works as follows. We enumerate subsets of the current path items such that each such subset s is dominated by N^g and is maximal, i.e., there is no other item which can be packed into the N^g. For each such s, we test whether condition (2) of the path-dominance symmetry definition (Definition 3 is satisfied. If so, then a path-dominance has been detected, so the current node can be pruned. The test for condition (2) is the same as the corresponding test for path-symmetry in the previous section. Thus, the same four implementations of the check are possible: (a) full packing with BT, (b) full packing with FFD, (c) limited packing with BT, and (d) limited packing with FFD. In the worst case, this check is executed for each subset s that satisfies condition (1) of Definition 3, so checking for path-dominance can be quite expensive.

The following, highly restricted form of Path-Dominance was proposed by Fukunaga and Korf [6]. Given a bin assignment B^d for depth d, we can prune B^d if there is a nogood N^g with respect to B^d such that (1) N^g dominates B according to the MKP dominance criterion (Proposition 1), and (2) The items in B^d can be swapped with the current items in bin g, such that the resulting bin assignments are both feasible. In other words, this is a restricted Path-Dominance test where all bins are frozen except for the bin at depth d. We call this strategy *2-swap-path-dominance*.

3.3 Combining Symmetry Breaking Strategies

We have defined a spectrum of symmetry-breaking techniques above, ranging from the weakest, 2-swap-path-symmetry, to the strongest, full path-dominance with BT. Path-dominance, using the full packing with backtracking implementation, clearly subsumes all of the other criteria. For example, every node which can be pruned by path symmetry will also be pruned by path-dominance (but not vice versa). However, there is a trade-off between the amount of pruning enabled by a symmetry relation and the amount of overhead incurred at each node in order to detect the symmetry. To alleviate this trade-off, we combine the strategies by *chaining* a set of tests so that the cheapest, least powerful symmetry is applied first. If this prunes the node, then the cost of applying the more powerful (but costly) symmetries is not incurred. However, if the node is not pruned, then we apply another, more powerful symmetry, and so on.

A preliminary study presented at a workshop reported results on 11 different configurations of symmetry-checking tests [5]. While we have found that path-symmetry (including 2-swap-path-symmetry) and 2-swap-path-dominance are relatively efficient and often offer a favorable trade-off between search reduction and increased cost per node, we have not yet found a way to reduce the cost of the more powerful variants (full/limited path dominance using either backtracking or FFD) sufficiently to justify their use. The specific configurations used this paper are described in Section 5.

3.4 Relationship to Previous Work on Symmetry Detection

Our MKP symmetry breaking mechanisms are domain-specific instances of the symmetry breaking via dominance detection (SBDD) approach [2; 3]. A significant difference is that in addition to detecting equivalences to previously explored subtrees (2-swap-path-symmetry and path-symmetry), our 2-swap-path-dominance and path-dominance algorithms also detect partial solutions which are dominated by previously explored subtrees (according to Proposition 1).

Our work is also similar to the pruning technique proposed by Focacci and Shaw [4] for constraint programming, which was applied to the TSP with time windows. Both methods attempt to prune the search by proving that the current node at depth j, which represents a partial j-variable (bin[3]) solution x, is dominated by some previously explored i-variable (bin) partial solution (nogood bin assignment) q, where $i < j$.

The main difference between our method and Focacci and Shaw's method is the approach used to test for dominance. Focacci and Shaw's method extends q to a j-variable partial solution q' which dominates x. They apply a local search procedure to find the extension q'. In contrast, our methods start with a partial, j-bin solution x and try to transform it to a partial solution x' such that $\bar{x}'_i$, the subset of x' including the first i bins, is dominated by the i-bin partial solution q. We do this by transforming (via item swaps) the contents of bins $i, i+1, ..., j$ in x to derive a feasible partial solution x' such that $\bar{x}'_i$ is dominated by q.

4 Bound and Bound

A powerful technique for solving the MKP is *bound-and-bound*, which was originally implemented in Martello and Toth's MTM solver for the MKP [10]. In standard branch-and-bound, an upper bound U is computed at each node in the search tree. If $U \leq L$, L, where L is a lower bound, e.g., the best (highest) objective function score found so far by branch-and-bound, then exploring the node further is futile, so the node can be pruned. On the other hand, if $U > L$, then standard branch-and-bound does not prune the node. Bound-and-bound extends this by applying some heuristic technique to attempt to *validate* the upper bound: When $U > L$, bound-and-bound attempts to prove that the upper bound U can be achieved somehow in the current subtree – if so, then we

[3] Our analogues of CP variables and values are bins and bin assignments, respectively.

have found the value of the optimal subsolution under the current node and can backtrack.

The most powerful implementation of this idea is in Pisinger's Mulknap solver [12]. Mulknap is an item-oriented branch-and-bound algorithm. The items are ordered according to non-increasing *efficiency* (ratio of profit to weight), so that the next item selected by the variable-ordering heuristic for the item-oriented branch-and-bound is the item with highest efficiency that was assigned to at least one container by a greedy bound-and-bound procedure (see below). The branches assign the selected item to each of the containers, in order of non-decreasing remaining capacity.

At each node, an upper bound is computed using a relaxation of the MKP called the *surrogate relaxed MKP* (SMKP), which is obtained by combining all of the remaining m containers in the MKP into a single container with aggregate capacity $C = \sum_{i=1}^{m} c_i$, resulting in the single-container, 0-1 knapsack problem: where the items are the remaining items and the knapsack has the capacity of the aggregate container. The SMKP, which is currently the most effective upper bound for the MKP [8], can be solved by applying any algorithm for optimally solving the 0-1 Knapsack problem.

At each node, Mulknap attempts to validate the SMKP upper bound by showing that there exists a partition of the SMKP 0-1 Knapsack solution into the remaining empty spaces in the m bins of the original MKP instance. This is done by solving a series of m subset-sum problems which allocate the items from the SMKP solution to each bin, minimizing the unused capacity in each bin (without exceeding capacity). If this partition is successful then the SMKP upper bound can be achieved by partitioning the SMKP solution into the remaining spaces in the bins, so we have validated the upper bound possible under the current branch-and-bound node (and thus, we can backtrack).

Bound-and-bound can be extremely powerful for solving the MKP. In fact, for many random benchmarks with a relatively large ratio of items to bins ($n/m > 5$), bound-and-bound can often validate the SMKP upper bound at the root node of the search tree, which means that the instance is solved at the root node *without requiring any branch-and-bound search.*

We implemented Pisinger's bound-and-bound mechanism into our bin-completion solver: at each node, we attempt to validate the SMKP upper bound by partitioning the SMKP solution into the remaining bins (recall that in bin-completion, at depth b, $m - b$ bins are empty). Our implementation of the SMKP bound is a straightforward, primal branch-and-bound. Our implementation of the splitting procedure uses a standard branch-and-bound procedure using the max-cardinality bound [8].

5 Experimental Results

We compared the following bin-completion based MKP solver configurations:

- **PureBC**: bin completion with no symmetry checking and no bound-and-bound.

– **2-Dom**: Apply 2-swap-path-symmetry first, and if the node is not pruned, then try applying 2-swap-path-dominance. This corresponds to the "Bin-completion with nogood dominance pruning" algorithm reported in [6].
– **PathSym**: First, try 2-swap-path-symmetry, then try 2-swap-path-dominance, and finally, apply path-symmetry, using the limited packing with FFD implementation described above.
– **2-Dom-BB**: Same as 2-Dom, with bound-and-bound.
– **PathSym-BB**: Same as PathSym, with bound-and-bound.

All of our algorithms were implemented in Common Lisp and compiled using the CMUCL compiler version 19d. In addition, we also compared our algorithms with Pisinger's Mulknap algorithm (using Pisinger's C implementation, compiled using `gcc` version 4.12 with the `-O3` option.

We evaluated the various solver configurations using the following four standard classes of problems from the MKP literature.

– *uncorrelated instances*, where the profits p_j and weights w_j are uniformly distributed in $[min, max]$.
– *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 $p_j \geq 1$,
– *strongly correlated instances*, where the w_j are uniformly distributed in [min,max] and $p_j = w_j + (max - min)/10$, and
– *multiple subset-sum instances*, where the w_j are uniformly distributed in $[min, max]$ and $p_j = w_j$.

In our experiments, $min = 1$, $max = 1000$. 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 $1 \leq i < m$. The last 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 [12].

We used instances where the ratio of items to bins (n/m) ranged from 2 to 10. This is because for $n/m \geq 10$, Mulknap frequently finds a solution at the root node by succeeding in validating the SMKP upper bound with the subset-sum based bound-and-bound. For example, we generated 1000 instances each of the uncorrelated, weakly-correlated, strongly-correlated, and multiple subset-sum instances with 10 bins and 100 items, where $[min, max] = [1{,}1000]$. Mulknap solved all 4000 instances at the root node (i.e., without search) in less than 0.01 seconds per instance (see [12] for related results). On the other hand, for $n/m \leq 5$, the bound-and-bound at the root node usually fails, and Mulknap is forced to branch. It is therefore the instances with smaller n/m ratios that are in some sense the most difficult random MKP instances that can be generated using the model described above, so we focus on these problems.

The results are shown in Table 1. All experiments were run on a 2.4 GHz Intel Core2 Duo. Each experiment was run on 20 instances per (# bins, # items) pair (all configurations were run on the same instances), so a total of 480 instances

were used. The *fail* column indicates the number of instances (out of 20) that were not solved within the time limit (300 seconds/instance). The *time* and *nodes* show average time spent and nodes searched on the successful runs, excluding the failed runs. Thus, in the experiments where there were timeouts, the fail column is the most significant result.

There are several clear trends in the results. First, symmetry-based pruning is most effective for low n/m, and becomes less effective for high n/m. For $n/m < 5$, the variants that use some form of symmetry (2-Dom, 2-Dom-BB,PathSym, PathSym-BB) clearly search significantly less nodes than PureBC, and using less runtime. The only exception was for uncorrelated 12-bin, 48-item instances. For $n/m \geq 5$, the savings in nodes searched is insufficient to offset the cost of symmetry-based pruning. The % of nodes pruned due to symmetry techniques is highest for less correlated instances. This is because the dominance criterion is most powerful when item weights and profits are highly correlated, which means that most candidate bin assignments are pruned by the dominance criterion during `generate_dominated` (Fig 2, line 14), and are never considered.

Second, bound-and-bound becomes more effective as n/m increases, and the overhead associated with bound-and-bound *decreases* as n/m increases. For $n/m = 2$ (30-bin, 60-item instances), the overhead of bound-and-bound is sufficiently large enough that there is a significant performance degradation in 2-Dom-BB and PathSym-BB compared to 2-Dom and PathSym, respectively. However, for larger values of n/m, the relative overhead of bound-and-bound becomes less significant, and for $n/m \geq 5$, bound-and-bound is significantly enhancing the performance of the bin-completion variants.

The search behavior of Mulknap and bin-completion variants with bound and bound (2-Dom-BB and PathSym-BB) are similar when $n/m \geq 5$. In principle, when Mulknap can solve a problem at the root node without search, the bin-completion variants should also solve the same problem at the root node. Below the root node, the search behaviors of Mulknap and bin-completion with bound-and-bound can diverge, because Mulknap branches on individual items, using a variable ordering based on decreasing item efficiency (p/w ratio), while bin-completion is branching on undominated bin assignments, where the variable ordering is based on minimal cardinality, using profit as a tie-breaker.

The performance differences between Mulknap and our 2-Dom-BB/PathSym-BB variants on the strongly-correlated and multiple subset-sum instances for 10 bins/60 items, and 10 bins/100 items can be explained by a differences in the implementation of the 0-1 Knapsack solver used to compute the SMKP (lower bound) solution. There are cases where there exist multiple optimal solutions to the SMKP 0-1 Knapsack instance, all with the same total profit, but with differing total weight (such cases more common for multiple-subset sum instances and strongly correlated instances). Mulknap implements a specialized 0-1 Knapsack solver which is biased to find solutions with the smallest weight sum (which makes it more likely that the solution is splittable by the bound-and-bound subset sum solver) Our current 0-1 Knapsack solver did not implement this bias, and as a consequence, missed opportunities to successfully apply bound-and-bound.

Thus, Mulknap performed significantly better than 2-Dom-BB and PathSym-BB for the strongly-correlated and multiple subset-sum instances for $n/m \geq 5$, even though in principle (with a better implementation of the SMKP 0-1 Knapsack solver), the performances should have been identical, as both algorithms could have solved all of these instances at the root node.

To highlight the performance differences between the various symmetry pruning techniques, we describe some experiments with smaller problem instances, where all bin-completion based configurations were more likely to find a solution within the time limit. For uncorrelated instances with 10 bins, 30 items, PureBC solves all instances in an average of 9.13 seconds and 1,662,504 nodes. In comparison, 2-Dom solves all instances in 0.57 seconds and 47,193 nodes, and PathSym solves all instances in 0.28 seconds and 9,432 nodes. Thus, 2-Dom and PathSym are searching 2 and 3 orders of magnitudes fewer nodes than PureBC, respectively. Finally, we consider another configuration, *PathDom*, which first applies the same sequence of symmetry tests as PathSym, and finally applies the full path-dominance test using backtracking – thus, PathDom applies our most powerful pruning criterion and searches the fewest number of nodes. PathDom solves all of these instances in 0.78 seconds and 5031 nodes. Thus, exploiting the most powerful dominance criterion can yield almost another factor of 2 reduction in nodes searched for these instances, but the additional cost per node results in an overall 3x slowdown. We have not found any configuration using path dominance (other than the highly restricted 2-Dom case) where the search reduction was sufficient to offset the additional cost per node.

Overall, PathSym significantly reduced the size of the branch-and-bound tree compared to 2-Dom, the previous state of the art [6] algorithm for MKP problems with low n/m ratios. The results in Table 1 show that exploiting symmetry is a very effective technique for hard MKP instances with low n/m ratio. Furthermore, integrating bound-and-bound was shown to significantly improve performance on instances with higher n/m ratios, while modestly penalizing performance on instances with lower n/m ratios. Thus, the PathSym-BB configuration, which successfully integrates bin-completion, symmetry-based pruning (a combination of 2-swap-path-symmetry, 2-swap-path-dominance and path-symmetry), and Pisinger's bound-and-bound technique, can be considered a new, state-of-the-art algorithm for instances for low n/m ratios.

6 Conclusions

This paper presented an algorithm for the multiple knapsack problem which integrates techniques from constraint programming (symmetry-based pruning), operations research (bound-and-bound, as well as the SMKP upper bound and other techniques borrowed from Mulknap and earlier MKP solvers from the OR literature), and the AI literature (the bin-completion search space [6]). We proposed two new, symmetry breaking mechanisms (path symmetry and path dominance) which are generalizations of previously studied strategies (2-swap-path-symmetry and 2-swap-path-dominance). We showed that integrating path-

symmetry resulted in a new solver which significantly outperformed the previous state of the art, 2-swap dominance based bin-completion solver reported in [6]. We further showed that integrating bound-and-bound could significantly improve the performance on problems with higher n/m ratios.

There are several directions for future work. Although path-dominance is our most powerful symmetry relation, the current implementation is not competitive with path symmetry due to the large overhead incurred at each node. We are currently investigating improved implementations and approximate detection strategies to make path-dominance more viable. Likewise, our current implementation of bound-and-bound uses naive branch-and-bound algorithms for the SMKP upper bound and subset sum computation for bound validation. As discussed in Section 5, integration with more sophisticated algorithms is likely to result in significant performance improvements. Finally, the symmetry detection techniques described in this paper are not limited to the MKP. For example, it is straightforward to apply the symmetry techniques to improve the search efficiency of any of the bin-completion based solvers for bin packing, bin covering, and min-cost covering problems described in [6].

References

1. S. Eilon and N. Christofides. The loading problem. *Management Science*, 17(5):259–268, 1971.
2. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *Proceedings of the International Conference on Constraint Programming*, pages 93–107, 2001.
3. F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proceedings of the International Conference on Constraint Programming*, pages 77–92, 2001.
4. F. Focacci and P. Shaw. Pruning sub-optimal search branches using local search. In *Proc. CPAIOR*, pages 181–189, 2002.
5. A. Fukunaga. Exploiting symmetry in multiple knapsack problems. In *Proc. CP-07 Workshop on symmetry and constraint satisfaction problems*, 2007.
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. 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.
8. H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer-Verlag, 2004.
9. 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.
10. S. Martello and P. Toth. A bound and bound algorithm for the zero-one multiple knapsack problem. *Discrete Applied Mathematics*, 3:275–288, 1981.
11. S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, 1990.
12. D. Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114:528–541, 1999.

Uncorrelated Instances

	30 bins, 60 items			15 bins, 45 items			12 bins, 48 items			15 bins, 75 items			10 bins, 60 items			10 bins, 100 items		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
Mulknap	20	n/a	n/a	20	n/a	n/a	10	42.95	1202516	13	0.01	3	0	1.84	41271	**0**	< 0.01	**1**
PureBC	20	n/a	n/a	14	125.38	17593990	**3**	**38.69**	**4135959**	3	41.61	754025	0	21.30	415663	20	n/a	n/a
2-Dom	10	91.49	4180505	10	62.20	3725747	3	61.29	1930296	4	72.29	600277	0	53.34	414249	20	n/a	n/a
2-Dom-BB	12	79.987	2384741	10	74.36	3723605	3	62.84	1893539	2	6.65	38421	**0**	**1.04**	**7981**	**0**	< 0.01	**1**
PathSym	**3**	**40.30**	**752990**	**8**	**31.44**	**908677**	3	48.36	572037	4	78.07	556327	0	56.63	395254	20	n/a	n/a
PathSym-BB	4	37.16	549704	8	36.28	907531	3	48.89	555786	**2**	**6.50**	**35907**	0	1.11	7553	**0**	< 0.01	**1**

Weakly Correlated Instances

	30 bins, 60 items			15 bins, 45 items			12 bins, 48 items			15 bins, 75 items			10 bins, 60 items			10 bins, 100 items		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
Mulknap	20	n/a	n/a	20	n/a	n/a	20	n/a	n/a	20	n/a	n/a	11	116.11	2011539	**0**	< 0.01	**1**
PureBC	20	n/a	n/a	13	110.76	5008825	5	83.08	4596503	**19**	**278.44**	**10554776**	1	46.50	770159	20	n/a	n/a
2-Dom	6	70.26	1628331	9	65.27	2043285	4	79.95	2337020	20	n/a	n/a	2	59.96	650387	20	n/a	n/a
2-Dom-BB	7	72.14	1418034	9	71.29	2043123	4	83.14	2335930	20	n/a	n/a	1	37.08	546095	0	< 0.01	69
PathSym	**1**	**52.31**	**822188**	**6**	**57.67**	**1387753**	**4**	**43.36**	**799555**	19	297.59	2388087	2	56.86	367994	20	n/a	n/a
PathSym-BB	1	52.58	614972	6	62.53	1387665	4	45.12	798571	19	297.13	2349921	**1**	**33.38**	**276420**	0	< 0.01	69

Strongly Correlated Instances

	30 bins, 60 items			15 bins, 45 items			12 bins, 48 items			15 bins, 75 items			10 bins, 60 items			10 bins, 100 items		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
Mulknap	20	n/a	n/a	20	n/a	n/a	3	97.50	962223	0	14.28	137495	**0**	< 0.01	**1**	**0**	< 0.01	**1**
PureBC	17	129.07	2918108	5	62.61	1026434	2	31.39	609882	1	39.52	219995	1	45.65	251584	20	n/a	n/a
2-Dom	1	31.01	440718	3	36.93	676154	0	48.23	660830	1	59.52	219858	1	69.74	251495	20	n/a	n/a
2-Dom-BB	1	39.23	440718	3	41.29	676014	1	37.09	540183	0	4.61	17560	0	0.59	2850	3	< 0.01	1
PathSym	**0**	**11.65**	**143886**	**1**	**46.26**	**798791**	**0**	**29.23**	**321255**	1	62.41	218049	1	71.51	249179	20	n/a	n/a
PathSym-BB	0	15.20	143886	2	36.13	537265	0	30.37	318512	**0**	**4.68**	**17543**	0	0.61	2846	3	< 0.01	1

Multiple Subset-Sum Instances

	30 bins, 60 items			15 bins, 45 items			12 bins, 48 items			15 bins, 75 items			10 bins, 60 items			10 bins, 100 items		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
Mulknap	20	n/a	n/a	14	111.11	783653	2	25.52	168455	**0**	**0.01**	**2**	**0**	0.01	**1**	**0**	< 0.01	**1**
PureBC	16	122.46	2715571	4	34.54	989428	0	4.20	106243	0	7.51	18050	0	14.11	24830	15	0.54	10
2-Dom	1	9.40	225208	2	20.41	532674	0	3.13	64775	0	8.91	17993	0	16.60	24803	15	0.54	10
2-Dom-BB	1	11.48	225208	2	21.67	532671	0	3.13	64764	0	5.83	17993	0	6.19	24339	14	0.36	8
PathSym	**0**	**5.51**	**106551**	**1**	**27.92**	**565196**	**0**	**2.82**	**45671**	0	8.82	17902	0	16.81	24668	15	0.55	10
PathSym-BB	0	6.88	106551	1	29.91	565193	0	2.85	45660	0	5.76	17902	0	6.27	24213	14	0.36	8

Table 1. Comparison on random instances for $2 \leq n/m \leq 10$. Item weights were in [1,1000]. The *fail* column indicates the number of instances (out of 20) that were not solved within the time limit (300 seconds/instance). The *time* and *nodes* show average runtimes and nodes searched on the successful runs, excluding the failed runs.