Exploiting Symmetry in Multiple Knapsack Problems

Alex S. Fukunaga
Global Edge Institute
Tokyo Institute of Technology
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. We investigate mechanisms for detecting and breaking symmetry in the bin-completion search space. We propose path-symmetry and path-dominance, two new symmetry relations which are more powerful generalization of previous MKP symmetry breaking techniques. Experiments show that path-symmetry and path-dominance significantly reduce the number of nodes searched by bin-completion. In particular, a variant of path-symmetry is shown to significantly improve upon the previous state of the art for the MKP with respect to both runtime and nodes searched.

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 there are two bins with capacity 10 and 7, and 4 items (9,3),(7,3),(6,7),(1,5), where the first element of each pair is the weight of the item and the second element is the profit of that item. The optimal solution to this MKP instance is to assign (9,3) and (1,5) to the bin with capacity 10, and the item (6,7) to the bin with capacity 7 (total profit = 15). The MKP is a natural generalization of the 0-1 Knapsack Problem where there are m containers of capacities $c_1, c_2, ...c_m$.

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.

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

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

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

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

The MKP has numerous applications, including task allocation among autonomous agents continuous double-call auctions [Kalagnanam *et al.*, 2001], multiprocessor scheduling [Labbé *et al.*, 2003], vehicle/container loading [Eilon and Christofides, 1971], and the assignment of files to storage devices in order to maximize the number of files stored in the fastest storage devices [Labbé *et al.*, 2003]. 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 [Fukunaga and Korf, 2007].

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 [Fahle *et al.*, 2001; Focacci and Milano, 2001].

The paper is organized as follows. We start by reviewing the bin completion algorithm (Section 2. Section 3 defines the

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

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 (Sections 4-5) which are more powerful than the previous techniques, and we discuss methods for combining various symmetry mechanisms (Section 6). In Section 7, we experimentally evaluate various combinations of symmetry mechanisms. We compare our work with related work on symmetry detection and breaking and in the constraint programming literature in Section 8, 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 [Fukunaga and Korf, 2007]. We briefly describe this algorithm. For simplicity, we describe the algorithm in terms of the Multiple Subset-Sum Problem (MSSP), where each item's profit equals its weight. Thus, whenever possible in the description below, we simply refer to an item by its weight. Generalization of the algorithm to MKP instances where the profits are not the same as the weights is straightforward.

A *bin assignment* $B_j = (item_1, ..., item_k)$ is a set of all of the items that are assigned to a given bin j, $1 \leq j \leq m$. 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_j. Otherwise, the bin assignment is *infeasible*. We say that a bin assignment S is *maximal* with respect to bin j 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 2, where each call to `search_MKP` corresponds to a node in the branch-and-bound search tree (e.g., Figure 1).

Nodes are pruned according to an upper bound which is based on a relaxation of the problem by Martello and Toth [1990] (Line 16). Pisinger's R2 reduction procedure [Pisinger, 1999] is applied at each node (Line 11) in order to try to reduce the problem by eliminating some items for consideration. The `choose_bin` function (Line 18) selects the bin with least remaining capacity, and 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.

Figure 1 shows part of an example bin-completion search tree. In addition to the pruning mechanisms mentioned above, a *dominance criterion* is applied (Fig 2, Line 19) to limit the nodes considered. 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

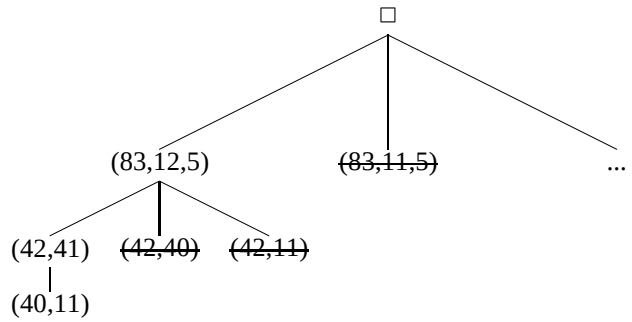

Figure 1: Part of the bin-completion search space for a MKP instance with capacity 100 and items {83,42,41,40,12,11,5}. Each node represents a maximal, feasible bin assignment for a given bin. Bin assignments shown with a ~~strikethrough~~, e.g., (83,11,5), are pruned because they are dominated according to the criterion in Proposition 1.

assignments which are dominated according to the following MKP dominance criterion [Fukunaga and Korf, 2007], which is based on the Martello-Toth dominance criterion for bin packing [Martello and Toth, 1990].

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.*

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).

3 Exploiting Symmetry

To describe our symmetry breaking mechanisms, which are instances of the general SBDD approach [Fahle *et al.*, 2001; Focacci and Milano, 2001], 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.

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.*

For example, in Figure 3, $(8, 2)^1$ is a nogood with respect to the descendants of $(7, 3)^1$ and $(9)^1$; $(7, 3)^1$ is a nogood with respect to the descendants of $(9)^1$

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

```
1 MKP_bin_completion(bins,items)
2  bestProfit = −∞
3  search_MKP(bins,items,0)

4 search_MKP(bins, items,sumProfit)
5  if bins==∅ or items == ∅
6   /*we have a candidate solution*/
7   if sumProfit > bestProfit
8    bestProfit = sumProfit
9   return
10 /* Attempt to reduce problem by eliminating some items from consideration*/
11 ri = reduce(bins,items)
12 if ri ≠ ∅
13  search_MKP(bins, items \ ri, sumProfit)
14  return
15 /* Attempt to prune based on Martello-Toth SMKP upper bound */
16 if (sumProfit + compute_upper_bound(items,bins)) ≤ bestProfit
17  return

18 bin = choose_bin(bins)
19 undominatedAssignments = generate_undominated(items,capacity(bin))
20 foreach A ∈ sort_assignments(undominatedAssignments)
21  if not(symmetric(A))
22   assign A to bin
23   search_MKP(bins \ bin, items \ A,sumProfit+∑_{i∈A} profit(i))
```

Figure 2: Outline of bin-completion for the multiple knapsack problem.

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 [Focacci and Shaw, 2002]. The number of possible nogoods at a particular search depth d in the worst case is the number of undominated bin assignments considered at depths $1, ..., d-1$.

3.1 2-Swap-Symmetry

The *2-swap-symmetry* method was proposed by Korf in a bin-completion algorithm for the bin packing problem [Korf, 2003].[2] Suppose we have a MSSP instance with the items {9,8,7,3,2...} where all bins have a capacity of 10. A portion of the bin-completion search tree is shown in Figure 3. After exhausting the subproblem below the assignment $(8,2)^1$, and while exploring the subproblem below the assignment $(7,3)^1$, assume we find a solution that assigns $(8,2)^2$. We can swap the pair of items (8,2) from the assignment $(8,2)^2$ with the pair of items (7,3) from the assignment $(7,3)^1$, resulting in a solution with $(8,2)^1$ and $(7,3)^2$ and the same total profit. However, we have already exhausted the subtree below $(8,2)^1$ and we would have found a solution with the same total profit as the best solution in the subtree below $(7,3)^2$. Therefore, we can prune the branch below $(8,2)^2$, because it is redundant (in other words, we have detected that the current partial solution is symmetric to a partial state that has already been exhaustively searched).

Similarly, in Figure 3, the bin assignment $(7,3)^2$ under the $(9)^1$, as well as the $(8,2)^2$ under the bin $(9)^1$ can both be pruned using 2-swap-symmetry.

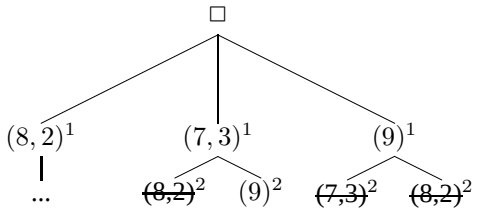

Figure 3: Both of the $(8,2)^2$ bin assignment under the $(7,3)^1$ and the $(9)^1$, as well as the assignment $(7,3)^2$ can all be pruned according to 2-swap-symmetry ($c_1 = 10, c_2 = 10$).

More generally, 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. The correctness follows straightforwardly from the definition of nogoods and the assumption that the nogood represents a node which has been exhaustively searched.

3.2 2-Swap-Dominance

The following *2-swap-dominance symmetry* technique[3] [Fukunaga and Korf, 2007] allows even more pruning: Consider the partial search tree in Figure 4. Assume that all bins

[2] 2-swap-symmetry was previously called "nogood pruning", but we change the terminology in order to avoid confusion.

[3] 2-swap-dominance symmetry was previously called "nogood dominance pruning".

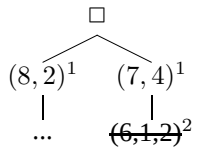

Figure 4: The bin assignment $(6,1,2)^2$ can be pruned by 2-swap-dominance ($c_1 = 11, c_2 = 11$).

have capacity 11. Suppose that after exhausting the subproblem below the assignment $(8,2)^1$, and while exploring the subproblem below the assignment $(7,4)^1$, we consider the assignment $(6,2,1)^2$. We can swap the items (6,2,1) from bin 2 with the pair of items (7,4) from bin 1 and end up with a solution with $(6,2,1)^1$ and $(7,4)^2$. However, according to the MKP dominance criterion, the set of items (6,2,1) is dominated by (8,2), and we have already exhausted the search below the node $(8,2)^1$, so we can prune the search under $(6,1,2)^2$ because it is not possible to improve upon the best solution under $(8,2)^1$.

More generally, 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. The correctness follows from the Proposition 1 and the assumption that the nogood represents a node which has been exhaustively searched.

To check whether a candidate assignment B is dominated by some nogood N, our current implementation uses a brute-force algorithm, which in the worst case takes time exponential in the cardinality of the bin assignment for each nogood.

4 Path-Symmetry

The techniques in the previous section only consider symmetries involving three bin assignments: a nogood at depth g, the current node at depth $d > g$, and the ancestor of the current node at depth g. We now generalize the symmetry techniques to detect and break symmetries involving more than these assignments.

Consider the search tree shown in Figure 5. 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 $(6,2,2)^4$. Note that 2-swap-dominance would not allow us to prune this node, since we can not swap the $(7,4)$ from bin 1 into bin 4 without exceeding bin 4's capacity. We define this symmetry more generally as follows.

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 5, is 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 $7, 4, 8, 3, 12, 6, 2, 2$.

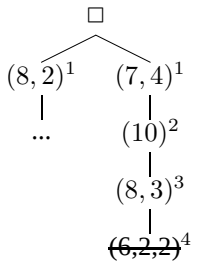

Figure 5: The bin assignment $(6,2,2)^4$ can be pruned by Path-Symmetry. ($c_1 = 11 c_2 = 11, c_3 = 12, c_4 = 10$).

Definition 2 (Path-Symmetry) *Let B^d be a candidate bin assignment. Let N^g be a nogood with respect to B^d, and let P be the current path items from depth g to 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. As with 2-swap symmetry, this 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,[4] 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 packing all of the

[4] 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.

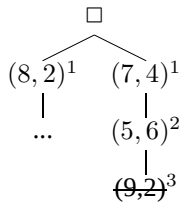

Figure 6: The bin assignment $(9,2)^2$ can be pruned by Path-Dominance ($c_1 = 11, c_2 = 12, c_3 = 13$)

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 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 $g+4, ..., d$, 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 Figure 5, the unlocked items for this approximation would be the 7 and 4 from bin 1. Thus, this limited packing problem involves packing the 7 and 4 into three bins: bin #2 with remaining capacity 8 (the 8 is moved to bin #1, the original capacity is 11, and there is a 3 which is locked, so the remaining capacity is 11-3=8), bin #3 with capacity 0 (no free space because there is a locked 12 occupying the bin), and bin #4 with remaining capacity 4. We can apply either BT or FFD to this limited packing problem. It is easy to see that the original 2-swap-symmetry check is a more restricted case of this approximate approach to path-symmetry.

Thus, we have four implementations path-symmetry, depending on the choice of methods for checking condition 2 in the path symmetry definition: (a) full packing with BT, (b) full packing with FFD, (c) limited packing with BT, and (d) limited packing with FFD.

5 Path-Dominance

Path-Dominance is the generalization of both 2-swap-dominance and path-symmetry. Consider the search tree shown in Figure 6 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 $(9,2)^3$. More generally:

Definition 3 (Path-Dominance) *Let B^d be a candidate bin assignment. Let N^g be a nogood with respect to 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. As with 2-swap dominance, this follows from the definition nogoods and Proposition 1.

Our current implementation of path-dominance works as follows. We systematically enumerate every subset s of the current path items such that 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 considered: (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.

6 Combining Symmetry Breaking Strategies

Of the four symmetries defined above, 2-swap-symmetry is the weakest symmetry, and is subsumed by 2-swap-dominance, i.e., every node which would be pruned by 2-swap-symmetry would also be pruned by 2-swap-dominance Similarly, 2-swap-symmetry is also subsumed by path-symmetry. In turn, 2-swap-dominance and path-symmetry are both subsumed by path-dominance.

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* some subset of them in a sequence so that the cheapest, least powerful symmetry is applied first. If this prunes the node, then the cost of applying the more powerful symmetries is not incurred. However, if the node is not pruned, then we apply another, more powerful symmetry, and so on.

We compared 11 configurations of the MKP solver, each using a different combination of symmetry mechanisms. These configurations are shown in Table 1. The four symmetry techniques are shown left to right. A "Y" indicates that the symmetry is applied. For the new path symmetries, we also specify whether the limited or full configuration (Section 4) is used, and also whether BT or FFD is used.

The PureBC configuration does not apply any symmetry detection in line 21 of Figure 2. The 2-Sym configuration applies only 2-swap-symmetry. The 2-Dom configuration applies 2-swap-symmetry first; if the node is not pruned, then it applies 2-swap-dominance (this is the configuration labeled "BC+NDP" in [Fukunaga and Korf, 2007]. As a final example, PathDom-Full-BT first tests for Nogood symmetry. If the test fails, then Nogood Dominance symmetry is applied. If that fails, then a full Path Dominance test using backtracking is applied.

Name	2-Swap-Symmetry	2-Swap-Dominance	Path-Symmetry	Limited/Full	BT/FFD	Path-Dominance	Limited/Full	BT/FFD
PureBC	N	N	N	n/a	n/a	N	n/a	n/a
2-Sym	Y	N	N	n/a	n/a	N	n/a	n/a
2-Dom	Y	Y	N	n/a	n/a	N	n/a	n/a
PathSym-Full-BT	Y	Y	Y	Full	BT	N	n/a	n/a
PathSym-Full-FFD	Y	Y	Y	Full	FFD	N	n/a	n/a
PathSym-Lim-BT	Y	Y	Y	Limited	BT	N	n/a	n/a
PathSym-Lim-FFD	Y	Y	Y	Limited	FFD	N	n/a	n/a
PathDom-Full-BT	Y	Y	N	n/a	n/a	Y	Full	BT
PathDom-Full-FFD	Y	Y	N	n/a	n/a	Y	Full	FFD
PathDom-Lim-BT	Y	Y	N	n/a	n/a	Y	Limited	BT
PathDom-Lim-FFD	Y	Y	N	n/a	n/a	Y	Limited	FFD

Table 1: MKP solver configurations - each row corresponds to a combination of symmetry detection mechanisms

7 Experimental Results

We experimentally evaluated the 11 solver configurations described in the previous section, using the following four classes of problems (originally from [Pisinger, 1999]):

- *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 $p_j = w_j$.

The bin capacities were set as follows: The first $m - 1$ 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 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 [1999]. That is, we only used instances where: (a) each of the items fits into at least one of the containers, (b) the smallest container 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 container. In our experiments, we used items with weights in the range [1,1000].

We used instances where the ratio of items to bins is between 2 and 3, because it has been shown that for these instances, bin-completion is clearly the state of the art approach [Fukunaga and Korf, 2007]. We only report comparisons among bin-completion variants due to space; we also ran Pisinger's Mulknap solver [1999] on the same instances, but Mulknap's was not competitive with any of the bin-completion variants shown here. See [Fukunaga and Korf, 2007] for a full comparison of bin-completion with Mulknap and MTM, an older solver by Martello and Toth [1990].

The results are shown in Table 2. All experiments were run on a 2.4 GHz Intel Core2 Duo. Each experiment was run on 20 instances (all configurations were run on the same instances). 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).

First, note that the PathDom-Full-BT and PathSym-full-BT configurations, which implement the full versions of the path dominance and path symmetry relations, respectively, significantly reduce the size of the branch-and-bound tree compared to 2-Dom, which was the previous state of the art [Fukunaga and Korf, 2007]. However, the runtimes are not significantly better – in fact, for many instances, they run much slower than 2-Dom due to the overhead of symmetry detection.

Fortunately, configurations that use a more limited version of path-symmetry and path-dominance fare better: at a cost of some increase in the number of nodes searched, the overhead per node is drastically reduced. In particular, the PathSym-Limited-FFD configuration significantly outperforms the previous state of the art 2-Dom with respect to both runtime and nodes searched. The number of nodes searched by PathSym-Limited-FFD is about an order of magnitude smaller than that of 2-Dom for uncorrelated instances, and about a factor of 2 smaller for subset sum instances, (based on node counts from problems where both configurations solved all of the instances). The runtimes for PathSym-Limited-FFD are a factor of 2-3 faster than 2-Dom, which shows that the increased complexity per search node is more than offset by the search efficiency gained. Thus, bin-completion using PathSym-Limited-FFD is the new state of the art algorithm for optimally solving MKP instances for the class of problems considered here.

Furthermore, the number of nodes searched by PathSym-Limited-FFD is within a factor of 2 of the number searched by PathDom-Full-BT (the most "powerful" configuration tested with respect to the number of nodes searched), despite the fact that PathSym-Limited-FFD uses both a less powerful symmetry relation (path-symmetry as opposed to path-dominance) and uses a limited, approximate mechanism for detecting the symmetry (Limited-FFD) as opposed to Full-BT.

While full path-dominance with backtracking (PathDom-Full-BT) was the most efficient configuration with respect to nodes searched, the overhead of applying full path dominance at each node significantly increases runtime and does not appear worthwhile. We are currently investigating methods which apply path-dominance more selectively (or in combination with path symmetry), but have not yet found a combination of techniques which outperformed the PathSym-Lim-

FFD configuration.

Overall, these results show that exploiting symmetry is a very effective technique for the MKP. Even the simplest configuration which exploits symmetry, 2-Sym, consistently outperforms PureBC (bin-completion without any symmetry detection), by at least a 1-2 orders of magnitude difference in the number of nodes searched, and a consistent runtime speedup of 1-2 orders of magnitude. The new PathSym-Lim-FFD configuration searches 1-4 orders of magnitude fewer nodes, and also run up to 3 orders of magnitude faster than PureBC.

8 Related Work

Our MKP symmetry breaking mechanisms are domain-specific instances of the symmetry breaking via dominance detection (SBDD) approach [Fahle *et al.*, 2001; Focacci and Milano, 2001]. A significant difference is that in addition to detecting equivalences to previously explored subtrees (2-swap-symmetry and path-symmetry), our 2-swap-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 [2002] 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[5]) 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.

9 Conclusions

This paper investigated techniques for exploiting symmetry in a branch-and-bound algorithm for the multiple knapsack problem. Specifically, we focused on techniques for breaking symmetries in the search space explored by the bin-completion branch-and-bound algorithm. We proposed two new, symmetry breaking mechanisms (path symmetry and path dominance) which are generalizations of previously studied strategies (2-swap symmetry and 2-swap dominance). We showed that the new symmetries are significantly more powerful than their predecessors, reducing the branch-and-bound trees by up to an order of magnitude compared to the previous state of the art, 2-swap dominance. However, we also found that these new symmetry breaking mechanisms added a significant amount of overhead per node, which resulted in slower runtimes compared to the simpler 2-swap dominance symmetry. We showed that approximate (but admissible) implementations of path symmetry and path dominance combined with previous mechanisms (e.g., the PathSym-Lim-FFD configuration) are able to achieve a favorable tradeoff between pruning and per-node complexity, significantly outperforming the previous state of the art on the classes of MKP instances we considered.

There are several directions for future work. Although we have investigated approximate symmetry detection, our focus so far has been on identifying powerful symmetry relations such as path-symmetry and path-dominance. There are opportunities to develop more efficient implementations of path symmetry and path dominance. Although PathSym-Lim-FFD searched an order of magnitude less nodes than 2-Dom on some instances, the runtime speedup is only a factor of 2-3. More efficient implementation strategies may enable path-symmetry to achieve close to an order of magnitude speedup relative to the 2-Dom configuration.

Although path-dominance is our most powerful symmetry relation, the current implementation is not competitive with path symmetry due to the large overhead incurred to check for path-dominance at each node. We are currently investigating improved implementations and approximate detection strategies which make path-dominance more viable as part of a combined (chained) strategy.

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

References

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

[Fahle *et al.*, 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *Proc. CP-01*, pages 93–107.

[Focacci and Milano, 2001] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proc. CP-01*,77–92.

[Focacci and Shaw, 2002] F. Focacci and P. Shaw. Pruning sub-optimal search branch brances using local search. In *Proc. CP-AI-OR*, pages 181–189.

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

[Kalagnanam *et al.*, 2001] 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.

[Korf, 2003] R. Korf. An improved algorithm for optimal bin packing. In *Proceedings of IJCAI*,pages 1252–1258.

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

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

[Pisinger, 1999] D. Pisinger. An exact algorithm for large multiple knapsack problems. *Eur. J. of Operational. Res.*, 114:528–541.

Uncorrelated Instances															
	15 bins, 30 items			20 bins, 40 items			25 bins, 50 items			10 bins, 30 items			15 bins, 45 items		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
PureBC	0	0.951	206419	7	49.436	7545714	1	90.290	8875301	0	9.130	1662504	12	123.233	16112720
2-Sym	0	0.011	1324	0	0.924	84266	2	39.127	2940614	0	0.597	63708	10	56.951	3941467
2-Dom	0	0.010	1059	0	0.600	43875	1	29.319	1735503	0	0.572	47193	10	62.507	3392116
PathDom-Full-BT	0	0.013	188	0	1.513	1924	3	36.779	13483	0	0.786	5031	20	-	-
PathDom-Full-FFD	0	0.006	199	0	0.236	2783	1	15.346	41349	0	0.975	8109	16	89.955	373482
PathDom-Lim-BT	0	0.005	199	0	0.262	2547	0	19.380	40216	0	0.778	7252	14	121.330	317797
PathDom-Lim-FFD	0	0.005	199	0	0.259	2547	0	19.398	40238	0	0.780	7258	14	121.665	318298
PathSym-Full-BT	0	0.005	236	0	0.155	2498	0	29.416	30607	0	0.431	6761	16	170.84	337027
PathSym-Full-FFD	0	0.003	245	0	0.108	3341	0	5.510	70684	0	0.322	10093	9	77.686	978742
PathSym-Lim-BT	0	0.003	239	0	0.099	3141	0	4.230	57896	0	0.284	9422	8	56.068	756632
PathSym-Lim-FFD	0	0.003	239	0	0.097	3142	0	4.208	57962	0	0.284	9432	8	55.568	758425
Weakly Correlated Instances															
	15 bins, 30 items			20 bins, 40 items			25 bins, 50 items			10 bins, 30 items			15 bins, 45 items		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
PureBC	0	0.717	69033	2	32.962	2328058	15	176.854	12712681	0	4.269	413635	14	125.483	5452363
2-Sym	0	0.026	1418	0	0.608	32680	0	14.699	551669	0	0.762	46564	10	69.61	2197822
2-Dom	0	0.016	787	0	0.302	12273	0	6.532	168474	0	0.703	36682	10	68.902	2024937
PathDom-Full-BT	0	0.015	214	0	0.370	881	1	39.83	4668	0	7.399	5056	18	117.355	109057
PathDom-Full-FFD	0	0.011	236	0	0.295	1251	0	4.969	10967	0	2.134	8894	14	97.217	305538
PathDom-Lim-BT	0	0.010	238	0	0.214	1218	0	3.014	8664	0	1.472	7609	14	78.441	276743
PathDom-Lim-FFD	0	0.009	238	0	0.213	1218	0	3.004	8668	0	1.454	7609	14	77.115	276776
PathSym-Full-BT	0	0.015	254	0	0.096	1248	0	4.904	8417	0	4.643	7061	13	116.37	321874
PathSym-Full-FFD	0	0.007	273	0	0.075	1647	0	1.117	16347	0	0.422	11730	8	94.2817	1250029
PathSym-Lim-BT	0	0.008	273	0	0.070	1623	0	0.825	13094	0	0.348	10562	7	88.891	1172600
PathSym-Lim-FFD	0	0.008	273	0	0.071	1623	0	0.822	13133	0	0.346	10564	7	89.148	1174057
Strongly Correlated Instances															
	15 bins, 30 items			20 bins, 40 items			25 bins, 50 items			10 bins, 30 items			15 bins, 45 items		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
PureBC	0	0.141	3197	2	28.443	209643	16	202.955	792114	0	0.650	13489	17	192.077	2008851
2-Sym	0	0.021	285	0	1.707	6179	3	39.125	79392	0	0.349	6623	13	145.069	1422360
2-Dom	0	0.013	175	0	1.203	3174	2	33.937	50439	0	0.339	6373	13	136.639	1311413
PathDom-Full-BT	0	0.007	70	0	1.113	653	2	65.039	6341	0	1.177	3674	20	-	-
PathDom-Full-FFD	0	0.008	72	0	0.661	804	0	40.147	13374	0	0.835	4412	19	257.260	126024
PathDom-Lim-BT	0	0.008	72	0	0.675	833	1	24.341	11578	0	0.773	4328	19	180.463	107563
PathDom-Lim-FFD	0	0.008	72	0	0.673	833	1	24.309	11579	0	0.773	4329	19	180.150	107564
PathSym-Full-BT	0	0.006	74	0	0.619	781	1	40.185	8977	0	0.396	4108	20	-	-
PathSym-Full-FFD	0	0.008	75	0	0.646	931	1	25.359	16569	0	0.276	4814	12	130.194	901912
PathSym-Lim-BT	0	0.007	76	0	0.675	979	1	25.779	16267	0	0.273	4810	12	103.099	743737
PathSym-Lim-FFD	0	0.009	76	0	0.670	979	1	26.271	16275	0	0.271	4811	12	103.094	745103
Subset-Sum Instances															
	15 bins, 30 items			20 bins, 40 items			25 bins, 50 items			10 bins, 30 items			15 bins, 45 items		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
PureBC	0	0.049	1488	0	4.964	1246450	6	46.046	1386179	0	0.121	3630	4	49.785	1282436
2-Sym	0	0.005	132	0	0.073	1893	0	1.1612	23909	0	0.080	2137	2	30.118	722638
2-Dom	0	0.001	88	0	0.037	941	0	0.545	11534	0	0.078	2096	2	30.067	690547
PathDom-Full-BT	0	0.004	50	0	0.209	325	1	55.56	2178	0	0.709	1456	20	-	-
PathDom-Full-FFD	0	0.002	51	0	0.048	382	0	1.937	3721	0	0.239	1677	10	100.001	99212
PathDom-Lim-BT	0	0.002	51	0	0.042	393	0	1.176	3475	0	0.202	1629	10	73.015	86786
PathDom-Lim-FFD	0	0.003	51	0	0.043	393	0	1.167	3475	0	0.202	1629	10	73.240	86789
PathSym-Full-BT	0	0.002	51	0	0.047	371	0	9.975	3177	0	0.153	1637	18	200.315	45671
PathSym-Full-FFD	0	0.001	52	0	0.023	425	0	0.251	4679	0	0.078	1836	2	38.967	522329
PathSym-Lim-BT	0	0.003	53	0	0.019	451	0	0.254	4903	0	0.077	1838	2	29.725	450926
PathSym-Lim-FFD	0	0.001	53	0	0.020	451	0	0.250	4907	0	0.080	1839	2	29.632	452028

Table 2: Multiple knapsack problem results: Comparison of Bin-Completion with various combinations of 2-Swap-Symmetry, 2-Swap-Dominance Symmetry, Path-Symmetry, and Path-Dominance on random MKP instances. The *fail* column indicates the number of instances (out of 20) that were not solved within the time limit (300 seconds/instance). The *time* (seconds on 2.4GHz Intel Core2 Duo) and *nodes* show average time spent and nodes searched on the successful runs, excluding the failed runs.