To appear in Annals of Operations Research (Draft, October 2009)

A Branch-and-Bound Algorithm for Hard Multiple Knapsack Problems

Alex S. Fukunaga

the date of receipt and acceptance should be inserted later

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 criteria for pruning nodes in the MKP branch-and-bound search space. In addition, we integrate the "bound-and-bound" upper bound validation technique used in previous MKP solvers. We show experimentally that our new MKP solver, which successfully integrates dominance based pruning, symmetry breaking, and bound-and-bound, significantly outperforms previous solvers on some classes of hard problem 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.

Alex S. Fukunaga
Global Edge Institute, Tokyo Institute of Technology, Meguro, Tokyo, Japan E-mail: fukunaga@is.titech.ac.jp

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

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

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

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

The MKP has numerous applications, including task allocation among autonomous agents, continuous double-call auctions (Kalagnanam, Davenport, and Lee, 2001), multiprocessor scheduling (Labbé, Laporte, and Martello, 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. While this paper focuses on exact algorithms, there has also been work on approximation algorithms (Chekuri and Khanna, 2000; Caprara, Kellerer, and Pferschy, 2000; Caprara, Kellerer, and Pferschy, 2003), as well as heuristics for the MKP (Martello and Toth, 1981b; Raidl, 1999; Fukunaga, 2008)

In previous work (Fukunaga and Korf, 2007), we investigated a branch-and-bound algorithm called *bin-completion*, based on a bin-oriented branching structure and a powerful dominance criterion. We showed that for problems where the ratio of items to bins is relatively small (i.e., $n/m < 4$), bin-completion is a successful approach.

This paper extends the work on the bin-completion algorithm in several ways. First, the search space explored by bin-completion has many symmetric and dominated states. Previous work introduced some techniques for exploiting the symmetry and demonstrated their utility[2]. In this paper, we further investigate methods for exploiting symmetry and dominance in the MKP bin-completion algorithm, and propose *path-symmetry*, a dynamic symmetry-detection approach for the MKP. Path-symmetry is related to the general symmetry breaking via dominance detection (SBDD) approach in the constraint programming literature (Fahle, Schamberger, and Sellmann, 2001; Focacci and Milano, 2001). In addition, we further generalize upon path-symmetry and combine it with the dominance criterion used by bin-completion to derive *path-dominance*, a stronger, dominance-based pruning criterion.

Secondly, we integrate bin-completion with *bound-and-bound*, a technique which is responsible for much of the power of previous branch-and-bound MKP solvers (Martello and Toth, 1981a; Pisinger, 1999). Bound-and-bound seeks to prune nodes by heuristically seeking to validate the (optimistic) upper bound on the total profit at each search

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

[2] In this paper, these previous methods, formerly called nogood pruning and nogood dominance pruning are referred to as 2-swap-path-symmetry and 2-swap-path-dominance

node. We incorporated this technique into our extended bin-completion based MKP solver.

The resulting new MKP solver is shown to significantly improve upon the previous bin-completion MKP solver of (Fukunaga and Korf, 2007) on instances where the ratio of items to bins is small, resulting in a new, state-of-the art solver for these problem classes, and is also shown to be competitive with Pisinger's Mulknap solver (Pisinger, 1999) (the previous state-of-the-art solver) for problems with high ratios of items to bins.

The paper is organized as follows. We start by reviewing standard algorithms for the MKP (Section 2), and identify classes of problems which pose challenges for these algorithms. Next, Section 3 describes the bin-completion algorithm. Section 4 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 and dominance-based pruning techniques which are more powerful than the previous techniques. We discuss methods for combining various symmetry and dominance-based pruning mechanisms. In Section 5, we experimentally evaluate various combinations of symmetry and dominance-based pruning mechanisms, and we conclude with a discussion of results and directions for future work.

2 Previous Algorithms for the MKP: Item-Oriented, Bound-and-Bound

There is a long line of research on exact, branch-and-bound algorithms for the MKP. Early work by Ingargiola and Korsh (1975) proposed a reduction algorithm based on dominance relationships between pairs of items, and presented a branch-and-bound strategy that used this reduction procedure. In other early work, Hung and Fisk (1978) proposed a branch-and-bound algorithm using the Lagrangian relaxation for the upper bound.

The MTM algorithm of Martello and Toth (1981a) is an item-oriented branch-and-bound algorithm (see Figure 2(a)). 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, 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:

$$\text{maximize} \sum_{j=1}^{n} p_j x'_j \tag{5}$$

$$\text{subject to} \sum_{j=1}^{n} w_j x'_j \leq C, \tag{6}$$

$$x'_j \in \{0,1\}, j = 1, ..., n. \tag{7}$$

where the variable x'_j represents whether item j has been assigned to the aggregated bin. This *surrogate relaxed MKP* (SMKP) instance can be solved by applying any

algorithm for optimally solving the 0-1 Knapsack problem, and the optimal value of the SMKP is an upper bound for the original MKP. Thus, this upper bound computation is itself solving an embedded, weakly NP-complete (single-container) 0-1 Knapsack problem instance as a subproblem.

The MTM algorithm introduced a powerful extension to branch-and-bound for the MKP called *bound-and-bound*. 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 at each node: 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 have found the value of the optimal sub-solution under the current node and can backtrack. Furthermore, in this case, the value of L is updated to U.

The MTM algorithm applies a greedy heuristic algorithm for the MKP, which involves solving a series of m 0-1 Knapsack problems. First, container $i = 1$ is filled optimally using any of the remaining items, and the items used to fill container $i = 1$ are removed. Then, container $i = 2$ is filled using the remaining items. This process is iterated m times, at which point all containers are filled.

The Mulknap algorithm by Pisinger (1999) uses the same item-oriented branching structure as MTM, applying the SMKP upper bound at each node, as well as a bound-and-bound strategy. Mulknap differs from MTM in that it (1) uses a different validation strategy for the bound-and-bound based on splitting the SMKP solution, (2) applies item reduction at each node, and (3) applies capacity tightening at each node. Details on the reduction procedure and capacity tightening can be found in (Pisinger, 1999). Here, we focus on the improved bound-and-bound strategy.

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.

2.1 Where are the hard MKP problem instances?

The bound-and-bound technique used by MTM and Mulknap can be extremely powerful. In fact, for many random MKP benchmarks with a relatively large ratio of items to bins ($n/m > 5$), Mulknap's bound-and-bound strategy 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.*

To observe how the difficulty of MKP problem instances varies with problem characteristics, we consider the following, four standard classes of random problems from the MKP literature (Martello and Toth, 1990; Pisinger, 1999):

– *uncorrelated instances*, where the profits p_j and weights w_j are uniformly distributed in $[min, max]$.

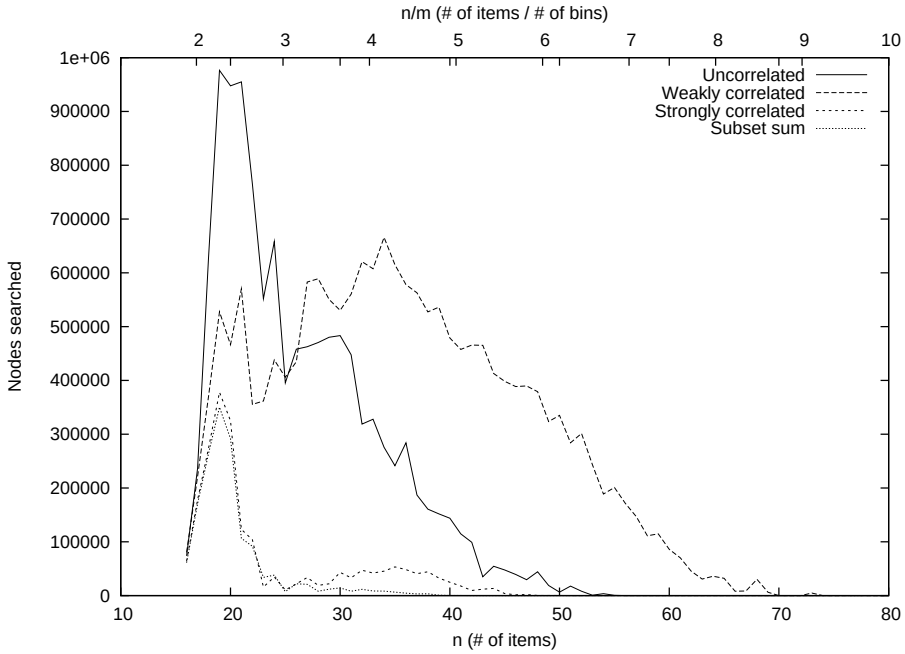


Fig. 1 Difficulty of problem instances as the ratio of items to knapsacks (n/m) varies. The number of bins was fixed at $m = 8$, and n was varied from 16 to 80. Each data point represents the mean over 100 instances solved with `Mulknap`, where each run had a time limit of 30 seconds. For runs that timed out, the number of nodes generated within the time limit is counted.

- *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, $min = 10$, $max = 1000$. 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 (Pisinger, 1999).

For $m = 8$ knapsacks, we considered n (number of items) ranging from 16 to 80, i.e., $2 \leq n/m \leq 10$. For each value of n, we generated 100 instances, and ran `Mulknap` on each instance with a 30-second time limit. Figure 1 shows the mean number of nodes searched by Mulknap on these instances. For runs that timed out, we counted the number of nodes generated within the time limit.

For all four classes of standard MKP benchmarks, problems with ratios of n/m slightly greater than 2 are the hardest instances, and as n/m increases, the problems become easier, requiring less search, and with sufficiently high values of n/m, these instances require no search, i.e., the upper bound is immediately validated at the root

node by bound-and-bound. Similar results were obtained for lower precision (items in the range [10,100]) and higher precision (items in the range [10,10000]) instances.

We also 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 any branching) in less than 0.01 seconds per instance. This further demonstrates the relative "easiness" of problems with high n/m ratios for bound-and-bound based MKP solvers.

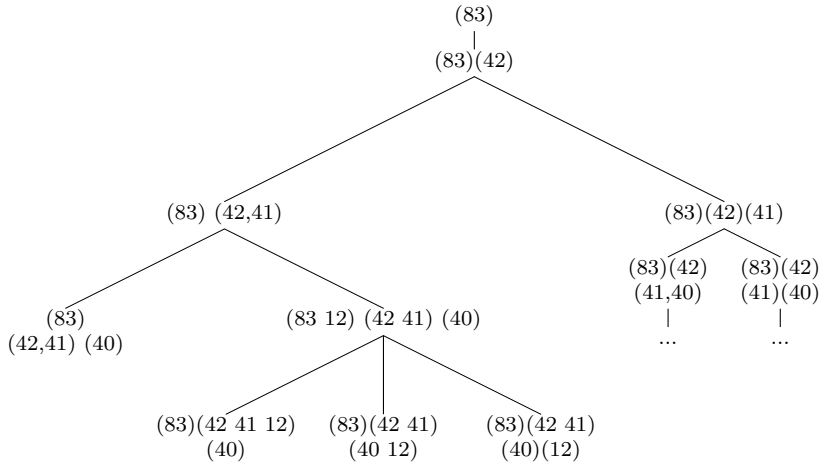
Our approach to generating difficult instances by identifying a critical parameter (n/m) for a simple uniform generator is analogous to the method of generating hard uniform satisfiability instances using the clause to variable ratio (Mitchell, Selman, and Levesque, 1992). However, there are other approaches to generating difficult instances. First, we can generate large instances (with large number of items and/or bins) which can not be solved in a reasonable amount of time simply because of their size, although this is a general observation and does not yield much insight into MKP problem difficulty. Second, as with knapsack problems in general, increasing the numerical precision of the instances (i.e., the number of significant digits in the representation of item weights) increases MKP problem difficulty (Pisinger, 1999). As noted earlier, changing the precision yields results similar to Figure 1, so the role of precision seems to be distinct from that of n/m in determining problem difficulty. A third approach is to develop artificial models specifically designed to generate hard instances. While the standard benchmark sets used in the MKP as well as single-bin 0-1 Knapsack literature uses a simple, natural uniform, random model, Pisinger has investigated methods designed specifically for generating hard, single bin, 0-1 Knapsack problem instances (Pisinger, 2005); investigation of extensions of these models for the MKP is an area for future work.

3 Bin-Completion Algorithm for the MKP

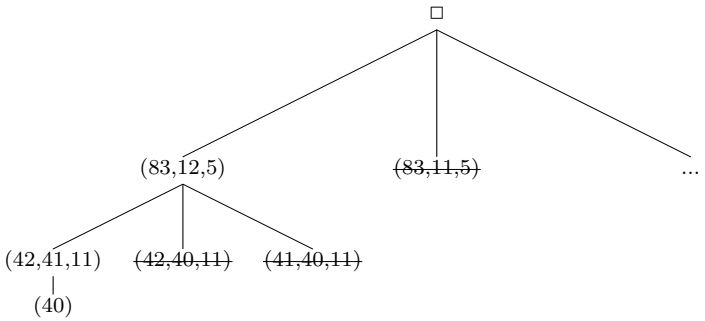
Bin-completion is a branch-and-bound algorithm for finding optimal solutions to multi-container assignment problems including the MKP, bin packing, and bin covering problems (Fukunaga and Korf, 2007). We briefly describe this algorithm. For simplicity of exposition, in the examples below, we assume (without loss of generality), 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 (unless explicitly noted otherwise).

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 item would make it infeasible. The bin-completion algorithm searches a tree where each node at depth d, $1 \leq d \leq m$, represents a set of maximal, feasible bin assignments. Figure 2(b) shows part of an example bin-completion search tree.

Figure 3 shows the bin-completion algorithm for the MKP, where each call to `search_MKP` corresponds to a node in the branch-and-bound search tree. At each node, we first check whether we are at a leaf node (no more bins or items) and adjust the



(a) Partial, item-oriented branching structure. Each node corresponds to a decision about which bin an item is assigned to. Items are considered in decreasing order of weight



(b) Partial, bin-completion search tree. 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.

Fig. 2 Standard item-oriented search space vs. Bin-completion search space: Portions of the search trees for a MKP instance with capacity 100 and items with weights {83,42,41,40,12,11,5} ($\forall i, p_i = w_i$).

lower bound (`sumProfit`) if a new, best-so-far solution is found (lines 2-3). Pisinger's R2 reduction procedure (1999) is applied in order to try to reduce the problem by eliminating some items for consideration (line 4). If items are eliminated by reduction, then we call `search_MKP` on the reduced subproblem (lines 5-7). Then, an upper bound for the remaining subproblem is computed using Martello and Toth's SMKP bound, and the node is fathomed if the upper bound plus the current profit can not exceed the profit of the best solution so far (lines 8-10).

Next, lines 11-13 we attempt to prune the node and improve the lower bound using bound-and-bound as described in Section 3.1.

At this point, bin-completion is ready to branch and recursively solve the remaining subproblem. First `choose_bin` selects the bin b with minimal remaining capacity (line 14). Then, in line 15, the `generate_undominated` function generates the candidate children of the current node, which are the set of all maximal, feasible assignments for b which are not dominated by any other assignment according to a *dominance*

```
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 + upperBound ≤ bestProfit
10    return /* upper-bound based pruning using SMKP bound */
11  if (validate_upper_bound(upperBound))
12    bestProfit = upperbound
13    return /* bound-and-bound */
14  bin = choose_bin(bins)
15  undominatedAssignments = generate_undominated(items,capacity(bin))
16  foreach A ∈ sort_assignments(undominatedAssignments)
17    if not(symmetric(A))
18      assign A to bin
19      search_MKP(bins \ bin, items \ A, sumProfit+∑_{j∈A} p_j)
```

Fig. 3 Bin-completion-based algorithm for the MKP. The top-level call is `search_MKP(bins,items,0)`, with `bestProfit` initialized to $-\infty$.

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 eliminates feasible assignments which are dominated according to the following MKP dominance criterion proposed in (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 to the sum of the item weights 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). Generating the undominated bin assignments according to this criterion requires solving some small bin packing instances, but can be done using space linear in the number of remaining items. Details of this procedure are in (Fukunaga and Korf, 2007).

At this point, we have a set of undominated bin assignments which are the children of the current node in the branch-and-bound tree. These undominated bin assignments are sorted in order of increasing cardinality, and ties are broken in order of decreasing profit (line 16). For each of these children, we first attempt to prune the child with the symmetry/dominance based pruning methods described in Section 4 (line 17). Finally, if the child is not pruned, then we call `search_MKP` recursively on the child.

In some instances, `generate_undominated` can generate a very large number of undominated bin assignments. In fact, for some instances, there can be thousands (or more) undominated bin assignments. It is possible to incrementally generate and pro-

cess the undominated assignments in small batches, which allows us to avoid spending too much time on a single node, at the cost of losing some of the benefits of the value ordering (`sort_assignments`). Details for this method, called hybrid incremental branching, are in (Fukunaga and Korf, 2007).

3.1 Bound and Bound

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). We use Pisinger's Minknap 0-1 Knapsack solver code to solve the SMKP instances. As in Pisinger's Mulknap solver, this Minknap code is modified so that if there are multiple optimal solutions to the SMKP instance, the one with the smallest weight sum is returned, because this makes it more likely that the solution can be split by the bound-and-bound subset sum solver.

4 Exploiting Symmetry and Dominance in the Bin-Completion Search Tree

We now describe several techniques for detecting and pruning symmetries and additional dominated nodes in the bin-completion branch-and-bound search tree.

To describe these mechanisms, which are related to 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 the pruning methods described in this section.

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 which is expanded prior to E^i by the bin-completion search algorithm is a *nogood* with respect to X^d.

In Figure 4, $(8, 2)^1$ 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 (Focacci and Shaw, 2002).

4.1 Path-Symmetry

Consider the search tree shown in Figure 4. 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:

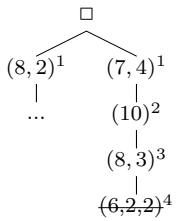


Fig. 4 The bin assignment $(6, 2, 2)^4$ can be pruned by Path-Symmetry. ($c_1 = 11, c_2 = 11, c_3 = 12, c_4 = 10$).

$(8, 2)^1, (7, 3)^2, (10, 2)^3, (6, 4)^4$. 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 4, 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 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,[3] 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

[3] 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.

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 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 Figure 4, 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 (Fukunaga and Korf, 2007): 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.[4]

Thus, we have five methods for detecting variants of path-symmetry: (1) 2-swap-path-symmetry, the restricted, 2-bin version of path symmetry, (2) path-symmetry using full packing and backtracking (BT), (3) full packing and FFD, (4) limited packing + BT, (5) limited packing + FFD. Full packing and backtracking captures the full pruning power of the path-symmetry criterion, while the other variants make various tradeoffs between pruning and per-node overhead. We present results using the 2 bin limited version (2-swap-path-symmetry), one approximation of full path-symmetry (limited packing + FFD), and the full path-symmetry test (full packing + BT). We do not present results using full packing+FFD and limited packing+BT because preliminary tests showed that these were not particularly promising or interesting.

4.2 Path-Dominance

Path-dominance is a generalization of path-symmetry. Consider the search tree shown in Figure 5 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: $(7,2)^1, (5,6)^2, (9,4)^3$. The optimal solution under the first sequence of bin assignments

[4] 2-swap-path-symmetry was previously called "nogood pruning" in (Fukunaga and Korf, 2007); 2-swap-path-dominance was previously called "nogood dominance pruning" in (Fukunaga and Korf, 2007). They have been renamed in this paper for clarity.

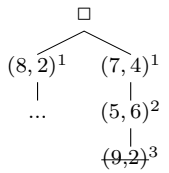


Fig. 5 The bin assignment $(9,2)^3$ can be pruned by Path-Dominance ($c_1 = 11, c_2 = 12, c_3 = 13$)

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:

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 relationship between B^d and some nogood N^g as defined above, B^d can be pruned (follows from the definition of 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 (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 in earlier work (Fukunaga and Korf, 2007): 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*.

Thus, as with path-symmetry, we have five methods for detecting variants of path-symmetry: (1) 2-swap-path-dominance, (2) path-dominance using full packing and backtracking (BT), (3) full packing and FFD, (4) limited packing + BT, (5) limited packing + FFD. As with path-symmetry, we present results using the 2 bin limited version (2-swap-path-dominance), one approximation of full path-dominance (limited packing + FFD), and the full path-dominance test (full packing + BT).

4.3 Combining Pruning Methods

We have defined a spectrum of techniques for exploiting symmetry and dominance above, ranging from the weakest, 2-swap-path-symmetry, to the strongest, full path-dominance with BT. The check for 2-swap-path-symmetry is very fast compared to all of the other pruning criteria. The check for 2-swap-path-dominance is very fast compared to general ($>$ 2-bin) path-dominance, but is much more expensive than 2-swap-path-symmetry, and comparable to general path-symmetry. Path-symmetry checks are significantly faster than path-dominance checks.

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 full 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 pruning method and the amount of overhead incurred at each node.

To alleviate this trade-off, we combine the strategies by *chaining* a set of tests so that the cheapest, least powerful method is applied first. If this prunes the node, then the costs of applying the more powerful (but costly) pruning methods are not incurred. However, if the node is not pruned, then we apply another, more powerful method, and so on. There are 4 components of a chained pruning strategy for bin-completion, executed in the following order: (1) 2-swap-path-symmetry pruning, (2) 2-swap-path-dominance pruning, (3) path-symmetry pruning, (4) path-dominance pruning.

In addition, for path-symmetry and path-dominance, we can use either limited or full packing, and either the backtracking (BT) or FFD test (see Section 4.1).

5 Experimental Results

We experimentally evaluated our bin-completion based MKP solvers, as well as Mulknap. As described in Section 4.3, we can combine various symmetry and dominance-based pruning methods by applying them in sequence. Table 1 shows the 12 configurations we evaluated. For each configuration, Table 1 indicates the search space searched, whether bound-and-bound was used, and the variant(s) of path-symmetry and path-dominance used. For example, the configuration labeled `2D/PSFull+B` is a bin-completion variant with bound-and-bound. It first applies 2-swap-path-symmetry at each node, and if the node is not pruned, then it applies 2-swap-path-dominance, then it applies path-symmetry pruning with full packing and BT. Preliminary experiments showed that 2-swap-path-symmetry is almost always helpful and incurs little overhead, so all bin-completion configurations except `BC` and `BC+BB` apply 2-swap-path-symmetry first.

All of our bin-completion algorithms were implemented in Common Lisp and compiled using the SBCL compiler version 1.0.22. These were compared against the Mulknap C code downloaded from Pisinger's web site compiled with `gcc` version 4.1.2 using the `-O3` optimization flag.

We evaluated the various solver configurations using the standard benchmark classes of uncorrelated, weakly correlated, strongly correlated, and multiple subset-sum instances, which are defined in Section 2.1. We used instances where the ratio of items to bins (n/m) ranged from 2 to 10. As shown in Section 2.1, this is a class of problems which is challenging for current MKP solvers.

	search space	bound & bound	2-swap-path-sym	2-swap path-dom	Path-Symmetry	Path-Dominance
Mulknap	item	y	-	-	-	-
BC	bin	n	n	n	n	n
BC+B	bin	y	n	n	n	n
2S	bin	n	y	n	n	n
2S+B	bin	y	y	n	n	n
2D	bin	n	y	y	n	n
2D+B	bin	y	y	y	n	n
PS	bin	n	y	n	Limited,FFD	n
PS+B	bin	y	y	n	Limited,FFD	n
2D/PS+B	bin	y	y	y	Limited,FFD	n
2D/PSFull+B	bin	y	y	y	Full,BT	n
2D/PD+B	bin	y	y	y	n	Limited,FFD
2D/PDFull+B	bin	y	y	y	n	Full,BT

Table 1 Algorithm configurations evaluated experimentally. All of these are exact, branch-and-bound algorithms.

The results are shown in Tables 2-4. Table 2 shows the impact of bound-and-bound on bin-completion solvers, and Tables 3-4 compare various combinations of the pruning techniques described in Section 4. All experiments were run on a 2.83 GHz Intel Xeon E5440 (all of our programs use a single core). Each entry in the tables represent data from 20 instances, and 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 (1 hour per 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 timeouts occurred, the *fail* column is the most significant result.

Overall, the `PS+B` configuration (bin completion with path-symmetry and bound-and-bound) resulted in the best performance, achieving good performance compared to both previous bin-completion based solvers from (Fukunaga and Korf, 2007) (i.e., `BC`, `2S`, `2D`), as well as Mulknap. In general, bin-completion based branching strategy is significantly more effective than the previous, item-oriented branching strategy when n/m is small, while the effect of successful bound-and-bound dominates for high values of n/m. Pruning based on path-symmetry and path-dominance is most effective for low n/m, and becomes less effective for high n/m.

5.1 Effectiveness of Bound-and-Bound and Comparison with Mulknap

Table 2 compares bin-completion solvers that use bound-and-bound at each node (`BC+B`, `2S+B`, `2D+B`) against configurations that are identical, except that bound-and-bound is not used at all (`BC`, `2S`, `2D`). Bound-and-bound becomes more effective as n/m increases, and per-node overhead of 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 some slowdown (less than 20%) in `BC+B`, `2S+B` and `2D+B` compared to `BC`, `2S`, and `2D`, 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.

All of the bin-completion solvers perform significantly better than Mulknap on instances with $n/m \leq 4$ on multiple subset-sum and strongly correlated instances, and

Strongly Correlated Instances (20 instances per set)									
	30 bins, 60 items			15 bins, 45 items			12 bins, 48 items		
	fail	time	nodes	**fail**	time	nodes	**fail**	time	nodes
Mulknap	**20**	-	-	**8**	1447.6	15065191	**1**	387.7	4618913
BC	**19**	2469.3	12919995	**7**	1245.7	11001154	**1**	214.5	2107106
BC+B	**19**	2634.0	12919994	**7**	1282.9	11000327	**1**	217.5	2093867
2S	**0**	401.4	2892773	**0**	955.6	6654960	**0**	211.3	1851019
2S+B	**0**	434.7	2892773	**0**	851.8	5902141	**0**	215.5	1839183
2D	**3**	497.2	1631516	**1**	850.1	5199276	**1**	175.5	1024321
2D+B	**3**	522.3	1631514	**1**	878.0	5198555	**1**	175.5	1012139
PS	**0**	346.6	2223863	**0**	360.4	2267173	**0**	103.6	689012
PS+B	**0**	397.6	2223862	**0**	375.2	2266460	**0**	105.2	678203
	15 bins, 75 items			10 bins, 60 items			10 bins, 100 items		
	fail	time	nodes	**fail**	time	nodes	**fail**	time	nodes
Mulknap	**0**	2.8	12831	**0**	0.5	5643	**0**	0.0	1
BC	**2**	673.2	2674564	**1**	644.1	3195946	**20**	-	-
BC+B	**0**	89.5	187546	**0**	13.2	22702	**0**	0.0	1
2S	**2**	857.2	2664605	**3**	496.1	2005870	**20**	-	-
2S+B	**0**	101.9	186907	**0**	14.5	22483	**0**	0.0	1
2D	**4**	608.4	1687894	**3**	548.4	1974650	**20**	-	-
2D+B	**0**	113.8	184984	**0**	14.8	22482	**0**	0.0	1
PS	**2**	876.6	2642081	**3**	508.7	1979216	**20**	-	-
PS+B	**0**	103.0	182546	**0**	14.2	21913	**0**	0.0	1
Uncorrelated Instances (20 instances per set)									
	30 bins, 60 items			15 bins, 45 items			12 bins, 48 items		
	fail	time	nodes	**fail**	time	nodes	**fail**	time	nodes
Mulknap	**20**	-	-	**17**	18516	80830867	**5**	502.7	21399885
BC	**20**	-	-	**7**	765.7	33450082	**1**	99.8	3834747
BC+B	**20**	-	-	**7**	808.7	33443730	**1**	90.6	3560519
2S	**6**	1296.3	42891188	**4**	170.1	5878005	**1**	84.3	2178953
2S+B	**8**	1303.9	31030003	**4**	190.6	5876272	**1**	81.7	2012293
2D	**5**	698.1	11298785	**4**	283.9	4010069	**1**	227.9	1865977
2D+B	**5**	858.1	11298783	**7**	808.7	33443730	**1**	219.1	1718940
PS	**0**	270.5	4607686	**3**	60.1	1176904	**0**	216.7	3853390
PS+B	**0**	355.2	4607685	**3**	66.2	1174424	**0**	80.0	684550
	15 bins, 75 items			10 bins, 60 items			10 bins, 100 items		
	fail	time	nodes	**fail**	time	nodes	**fail**	time	nodes
Mulknap	**6**	401.0	12462601	**0**	19.3	451517	**0**	0.0	1
BC	**1**	250.0	2927203	**0**	70.7	765242	**20**	-	-
BC+B	**1**	10.3	178772	**0**	0.8	13294	**0**	0.0	1
2S	**1**	404.1	2804649	**0**	120.5	764348	**20**	-	-
2S+B	**1**	16.7	158925	**0**	1.5	13209	**0**	0.0	1
2D	**1**	536.7	2761707	**0**	170.1	763175	**20**	-	-
2D+B	**1**	27.7	153950	**0**	2.5	13183	**0**	0.0	1
PS	**1**	414.8	2258201	**0**	130.5	748291	**20**	-	-
PS+B	**1**	17.2	93736	**0**	1.6	11732	**0**	0.0	1

Table 2 Impact of bound-and-bound on Strongly Correlated instances and Uncorrelated instances, $n/m \leq 10$. Item weights were in [10,1000]. The *fail* column indicates the # of instances (out of 20) that were not solved within the time limit (1 hour/instance). The *time* and *nodes* show average runtimes and nodes searched on the successful runs, excluding the failed runs.

on instances with $n/m \leq 5$ for weakly correlated and uncorrelated instances. Mulknap performs significantly better than bin-completion without bound-and-bound (`BC, 2S, 2D`) for problems with $n/m > 5$. However, the addition of bound-and-bound, greatly improves the performance of bin-completion for problems with high n/m. For 10 bins and 100 items, all of the solvers that use bound-and-bound solved all 20 instances at the root node of the branch-and-bound search tree (nodes=1). In principle, when Mulknap can solve a problem at the root node without branching, the bin-completion variants should also behave identically because they use the same bound-and-bound procedure. Although not shown here, results are similar for $n/m > 10$.

For weakly correlated instances and uncorrelated instances, bin-completion with bound-and-bound significantly outperforms Mulknap for 15 bins and 75 items, as well as 10 bins and 60 items. On the other hand, for multiple subset-sum instances and strongly correlated instances with 15 bins and 75 items, as well as 10 bins and 60 items, Mulknap outperforms the bin-completion solvers.

Although not shown here to avoid repetition, we also ran the same experiment with multiple subset-sum instances and weakly correlated instances, with results very similar to the strongly correlated and uncorrelated instances in Table 2, respectively.

Thus, it is clear that bound-and-bound significantly benefits bin-completion solvers. For weakly correlated and uncorrelated instances, bin-completion solvers with bound-and-bound consistently outperform Mulknap. For multiple subset-sum and strongly correlated instances, bin-completion solvers with bound-and-bound are competitive with Mulknap, except for instances where $5 \leq n/m \leq 6$.

The value of *integrating* both bound-and-bound and the bin-completion based branching structure can be seen in Table 2. For uncorrelated problems with 15 bins and 75 items, as well as 10 bins and 60 items, the solvers which used both the bin-completion search structure and bound-and-bound performed significantly better than both Mulknap and the bin-completion configurations which did not incorporate bound-and-bound. Thus, integrating both bin-completion and bound-and-bound resulted in performance that could not have been achieved by either technique alone, and could only be achieved with an integrated approach. Note that an alternative approach of running both Mulknap and a bin-completion solver (without bound-and-bound) in parallel with the same processor resource bounds would not have achieved the results obtained by the integrated solver.

5.2 On the utility of path-dominance

Our conclusions about the utility of the path-dominance pruning criterion are mixed. On one hand, 2-swap-path-dominance, the 2-bin, limited version of path-dominance originally proposed in (Fukunaga and Korf, 2007), is clearly a useful pruning criterion, as shown by the success of the `2D+B` configurations compared to pure bin completion (`BC+B`) and bin-completion with 2-swap-path-symmetrh (`2S+B`). Furthermore, `2D/PS+B`, which also incorporate 2-swap-path-dominance as one of the chained pruning rules (Section 4.3), performs well overall, although not as well as the approximate path-symmetry configuration `PS+B`. While `2D/PS+B` consistently searches fewer nodes than `PS+B` and is sometimes the best performer (e.g., multiple subset-sum instances with 30 bins and 60 items), it is usually slower due to the overhead incurred by the 2-swap-path-dominance test. It might be possible to reduce this overhead in an improved implementation.

On the other hand, the general path-dominance criterion seems to have limited utility, as shown by the relatively poor performance of the `2D/PD+B` and `2D/PDFull+B` configurations. Although full path-dominance (`2D/PDFull+B`) theoretically searches the fewest nodes, we have not found any configuration using path-dominance (i.e., configurations of full/limited swapping, backtracking/FFD packing) which achieves good performance compared to solvers relying on path-symmetry (e.g., `PS+B`). As shown in Tables 3-4, the `2D/PD+B` and `2D/PDFull+B` configurations consistently performed worse than `PS+B`. The `2D/PDFull+B` configuration performs very poorly due to the very large overhead of trying to detect all path-dominance relationships at every node.

Multiple Subset-Sum Instances (20 instances per set)									
	30 bins, 60 items			15 bins, 45 items			12 bins, 48 items		
	fail	time	nodes	**fail**	time	nodes	**fail**	time	nodes
Mulknap	**20**	-	-	**4**	755.5	6495827	**0**	170.0	1823987
BC+B	**16**	603.8	6267003	**0**	410.7	2989727	**0**	3.1	24489
2S+B	**0**	434.6	2892773	**0**	99.7	664153	**0**	2.3	17098
2D+B	**0**	178.0	955952	**0**	99.9	604300	**0**	2.7	16733
2D/PS+B	**0**	75.3	343477	**0**	66.7	370007	**0**	2.3	12927
2D/PSFull+B	**0**	70.0	95958	**0**	120.2	204171	**0**	5.7	10254
PS+B	**0**	98.1	489028	**0**	62.0	385275	**0**	1.9	13002
2D/PD+B	**1**	245.9	131693	**1**	720.6	212200	**0**	138.3	10398
2D/PDFull+B	**3**	68.3	122824	**3**	1373.8	120305	**1**	102.8	4626
	15 bins, 75 items			10 bins, 60 items			10 bins, 100 items		
	fail	time	nodes	**fail**	time	nodes	**fail**	time	nodes
Mulknap	**0**	0.0	2	**0**	0.0	1	**0**	0.0	1
BC+B	**0**	53.2	177017	**0**	21.5	80819	**0**	0.0	1
2S+B	**0**	65.4	175244	**0**	27.4	80690	**0**	0.0	1
2D+B	**0**	61.2	137884	**0**	26.5	67994	**0**	0.0	1
2D/PS+B	**0**	62.7	137871	**0**	26.9	67989	**0**	0.0	1
2D/PSFull+B	**0**	62.0	135805	**0**	26.1	64636	**0**	0.0	1
PS+B	**0**	66.5	175231	**0**	28.0	80413	**0**	0.0	1
2D/PD+B	**0**	68.5	123567	**0**	27.4	59130	**0**	0.0	1
2D/PDFull+B	**0**	68.3	122824	**0**	27.4	58394	**0**	0.0	1
Strongly Correlated Instances (20 instances per set)									
	30 bins, 60 items			15 bins, 45 items			12 bins, 48 items		
	fail	time	nodes	**fail**	time	nodes	**fail**	time	nodes
Mulknap	**20**	-	-	**8**	1447.6	15065191	**1**	387.7	4618913
BC+B	**19**	2634.0	12919994	**7**	1282.9	11000327	**1**	217.5	2093867
2S+B	**0**	434.7	2892773	**0**	851.8	5902141	**0**	215.5	1839183
2D+B	**3**	522.3	1631514	**1**	878.0	5198555	**1**	175.5	1012139
2D/PS+B	**0**	326.7	1576233	**0**	418.5	2146195	**0**	194.5	671526
2D/PSFull+B	**0**	248.0	345732	**0**	598.3	987785	**0**	424.5	374087
PS+B	**0**	397.6	2223862	**0**	375.2	2266460	**0**	105.2	678203
2D/PD+B	**3**	333.0	159603	**11**	856.4	601797	**13**	505.7	27701
2D/PDFull+B	**3**	472.4	54100	**11**	1959.5	332864	**13**	486.5	22500
	15 bins, 75 items			10 bins, 60 items			10 bins, 100 items		
	fail	time	nodes	**fail**	time	nodes	**fail**	time	nodes
Mulknap	**0**	2.8	12831	**0**	0.5	5643	**0**	0.0	1
BC+B	**0**	89.5	187546	**0**	13.2	22702	**0**	0.0	1
2S+B	**0**	101.9	186907	**0**	14.5	22483	**0**	0.0	1
2D+B	**0**	113.8	184984	**0**	14.8	22482	**0**	0.0	1
2D/PS+B	**0**	114.7	180802	**0**	14.5	21912	**0**	0.0	1
2D/PSFull+B	**0**	114.5	179972	**0**	14.4	21476	**0**	0.0	1
PS+B	**0**	103.0	182546	**0**	14.2	21913	**0**	0.0	1
2D/PD+B	**3**	360.7	86329	**0**	16.4	21725	**0**	0.0	1
2D/PDFull+B	**3**	355.1	85533	**0**	15.9	21277	**0**	0.0	1

Table 3 Evaluation on Multiple Subset-Sum instances and Strongly Correlated instances, with $n/m \leq 10$. Item weights were in [10,1000]. The *fail* column indicates the # of instances (out of 20) that were not solved within the time limit (1 hour/instance). The *time* and *nodes* show average runtimes and nodes searched on the successful runs, excluding the failed runs.

The `2D/PD+B` configuration uses limited packing and FFD to approximate full path-dominance, but also performs poorly compared to the solvers using path-symmetry. Consider the results for multiple subset-sum instances with 12 bins and 48 items. The path-symmetry configuration `PS+B` solves all 20 instances with 13002 nodes and 1.9 seconds per instance. The path-dominance configuration `2D/PD+B` requires less search (10398 nodes), but ran much slower (138.3 seconds per instance), and the full path-dominance configuration `2D/PDFull+B` only solved 19 out of 20 instances.

We performed additional experiments with smaller problem instances, where all bin-completion based configurations were able to find a solution within the time limit.

Weakly Correlated Instances (20 instances per set)									
	30 bins, 60 items			15 bins, 45 items			12 bins, 48 items		
	fail	time	nodes	**fail**	time	nodes	**fail**	time	nodes
Mulknap	**20**	-	-	**20**	-	-	**16**	2170.3	51006850
BC+B	**20**	-	-	**10**	1012.1	34180356	**1**	818.0	26318491
2S+B	**1**	459.2	9852571	**7**	377.0	9621344	**0**	515.1	12717274
2D+B	**1**	179.8	2363906	**7**	431.9	6840630	**1**	730.9	9692496
2D/PS+B	**0**	22.7	249981	**2**	659.1	8315905	**0**	397.8	3624160
2D/PSFull+B	**0**	39.2	81723	**4**	473.0	1267518	**1**	593.7	1628314
PS+B	**0**	24.8	355782	**2**	470.3	9406073	**0**	223.6	3851828
2D/PD+B	**0**	178.0	133774	**9**	803.8	1122117	**13**	919.1	861377
2D/PDFull+B	**2**	298.0	28508	**11**	1116.3	291104	**10**	1689.3	510621
	15 bins, 75 items			10 bins, 60 items			10 bins, 100 items		
	fail	time	nodes	**fail**	time	nodes	**fail**	time	nodes
Mulknap	**20**	-	-	**1**	361.6	7777204	**0**	0.0	1
BC+B	**15**	669.3	18355153	**0**	139.0	2249349	**0**	0.0	1
2S+B	**15**	771.2	14345268	**0**	200.8	2029246	**0**	0.0	1
2D+B	**16**	1158.4	9113924	**0**	376.9	1967813	**0**	0.0	1
2D/PS+B	**15**	1429.9	7157359	**0**	316.0	1086322	**0**	0.0	1
2D/PSFull+B	**18**	987.1	2262094	**0**	425.3	942505	**0**	0.0	1
PS+B	**15**	605.1	7269343	**0**	179.6	1102114	**0**	0.0	1
2D/PD+B	**20**	-	-	**13**	1153.0	340866	**0**	0.0	1
2D/PDFull+B	**20**	-	-	**9**	1355.4	336225	**0**	0.0	1
Uncorrelated Instances (20 instances per set)									
	30 bins, 60 items			15 bins, 45 items			12 bins, 48 items		
	fail	time	nodes	**fail**	time	nodes	**fail**	time	nodes
Mulknap	**20**	-	-	**17**	18516	80830867	**5**	502.7	21399885
BC+B	**20**	-	-	**7**	808.7	33443730	**1**	90.6	3560519
2S+B	**8**	1303.9	31030003	**4**	190.6	5876272	**1**	81.7	2012293
2D+B	**5**	858.1	11298783	**7**	808.7	33443730	**1**	219.1	1718940
2D/PS+B	**0**	402.7	2889109	**3**	113.3	1040818	**0**	223.0	654034
2D/PSFull+B	**1**	410.9	479972	**3**	113.1	331310	**1**	158.2	174423
PS+B	**0**	355.2	4607685	**3**	66.2	1174424	**0**	80.0	684550
2D/PD+B	**3**	321.2	356468	**4**	377.4	546827	**5**	767.7	253232
2D/PDFull+B	**5**	631.6	93894	**4**	222.5	223246	**1**	803.3	144663
	15 bins, 75 items			10 bins, 60 items			10 bins, 100 items		
	fail	time	nodes	**fail**	time	nodes	**fail**	time	nodes
Mulknap	**6**	401.0	12462601	**0**	19.3	451517	**0**	0.0	1
BC+B	**1**	10.3	178772	**0**	0.8	13294	**0**	0.0	1
2S+B	**1**	16.7	158925	**0**	1.5	13209	**0**	0.0	1
2D+B	**1**	27.7	153950	**0**	2.5	13183	**0**	0.0	1
2D/PS+B	**1**	31.1	93456	**0**	2.6	11729	**0**	0.0	1
2D/PSFull+B	**2**	25.1	63926	**0**	3.0	11279	**0**	0.0	1
PS+B	**1**	17.2	93736	**0**	1.6	11732	**0**	0.0	1
2D/PD+B	**2**	212.5	66331	**0**	12.2	11636	**0**	0.0	1
2D/PDFull+B	**2**	143.4	63195	**0**	14.5	11202	**0**	0.0	1

Table 4 Evaluation on Weakly Correlated instances and Uncorrelated instances, with $n/m \leq 10$. Item weights were in [10,1000]. The *fail* column indicates the number of instances (out of 20) that were not solved within the time limit (1 hour/instance). The *time* and *nodes* show average runtimes and nodes searched on the successful runs, excluding the failed runs.

For uncorrelated instances with 10 bins, 30 items, `BC` solves all instances in an average of 9.13 seconds and 1,662,504 nodes. In comparison, the 2-swap-path-dominance configuration `2D` requires 0.57 seconds and 47,193 nodes, and the path-dominance configuration `PS+B` requires 0.25 seconds and 9,728 nodes. Thus, `2D` and `PS+B` are searching 2-3 orders of magnitudes fewer nodes than `BC`, respectively. Finally, the full path-dominance configuration `2D/PDFull+B`, solves all of these instances in 0.78 seconds and 5031 nodes. Thus, for these instances, exploiting the most powerful dominance criterion yields a factor of 2 reduction in nodes searched for these instances compared to `PS+B`, but the additional cost per node results in a factor of 3 slowdown compared to `PS+B`.

5.3 Approximate vs. full pruning for path-symmetry and path-dominance

In Sections 4.1-4.2, we described 5 variants of both path-symmetry and path-dominance, ranging from limited versions which only consider two bins (2-swap-path-symmetry and 2-swap-path-dominance), to the full implementation of the pruning criteria, using full packing and backtracking (BT) at each node, as well as approximations which considered a limited subset of the items in each bin and a heuristic packing algorithm (FFD) at each node to detect path-symmetry and path-dominance. All of these methods still result in an exact algorithm for the MKP, although the approximate and 2-bin variants can miss opportunities to fathom nodes. The relatively poor performance of the bin-completion configurations which used full path-symmetry (`2D/PSFull+B`) and full path-dominance checks (`2D/PDFull+B`) compared to configurations which used approximate checks (`2D/PS+B` and `2D/PD+B`, respectively) show that using limited packing and FFD to approximate these checks yields a favorable tradeoff in pruning vs. per-node overhead.

5.4 Comparison with Previous Bin-Completion Solvers

Overall, the combination of path-symmetry and bound-and-bound techniques (`PS+B`) significantly reduced the size of the branch-and-bound tree compared to bin-completion with 2-swap-path-dominance `2D`, the previous state of the art algorithm for MKP problems with low n/m ratios (Fukunaga and Korf, 2007). Tables 3-4 show that exploiting symmetry and dominance is a very effective technique for hard MKP instances with low n/m ratio. Furthermore, Table 2 shows that integrating bound-and-bound significantly improves performance on instances with higher n/m ratios, while modestly penalizing performance on instances with lower n/m ratios. Thus, the `PS+B` configuration, which successfully integrates bin-completion, path-symmetry based pruning, and bound-and-bound technique, is a new, state-of-the-art algorithm for instances for low n/m ratios.

6 Related Work

Path-symmetry and path-dominance are closely related to generic, symmetry pruning techniques in constraint programming, such as the symmetry breaking via dominance detection (SBDD) approach (Fahle et al., 2001; Focacci and Milano, 2001), which seeks to prune partial solutions by using a symmetry function to map the current search node to one which has previously been fathomed (exhaustively searched). The symmetries detected by path-symmetry are based on swapping of items among bins in a candidate solution, where the swaps are constrained so that bin capacities are not violated. This does not directly correspond to the standard symmetries in constraint programming (e.g., variable / value symmetries). Thus, although path-symmetry is related to the generic SBDD approach, it is not clear how other approaches to symmetry detection from the constraint programming literature can be applied to the MKP. This remains an area for future work.

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) solution[5]) 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.

Another, related technique for dominance-based pruning called *local dominance* (LD) was originally proposed in (Fischetti and Toth, 1988) for the MKP, and generalized for mixed integer linear programming in (Fischetti and Salvagnin, 2008). In the LD approach, partial assignment α is pruned if it can be shown if there exists some other partial solution involving the variables in α which dominates α. Unlike our method and Focacci and Shaw's method, the LD approach does not necessarily require the identification of a *previously explored* partial solution which dominates α – it is sufficient to identify some partial solution β (previously explored or not) which dominates α. In (Fischetti and Toth, 1988), heuristics are used to find β, while in (Fischetti and Salvagnin, 2008), a small MILP instance is solved to seek β. In addition the LD check in (Fischetti and Salvagnin, 2008) also generates nogoods to accelerate the dominance check and speed up the search. Fischetti and Toth's MKP solver (1988) is an item-based branch-and-bound solver similar to the MTM algorithm (Martello and Toth, 1981a). Their pruning mechanisms "H3" and "H4" seeks to identify a single item assigned to a bin in the current partial assignment which can be replaced by another, "better" item – indicating that the current partial assignment is dominated and can be pruned. Although the specific pruning criteria used in (Fischetti and Toth, 1988) (i.e., H3, H4) are subsumed by the bin-completion MKP Dominance Criterion (Proposition 1), the general approach of LD pruning can be applied to a bin-completion search space, and is an area for future work.

7 Conclusions

This paper presented an algorithm for the multiple knapsack problem which successfully extends the bin-completion algorithm of (Fukunaga and Korf, 2007). We proposed two new, structural symmetry and dominance-based pruning methods (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 2-swap-path-dominance based bin-completion solver reported in (Fukunaga and Korf, 2007). We further showed that integrating bound-and-bound (Pisinger, 1999) could significantly improve the performance on problems with higher n/m ratios. The resulting, integrated solver (`2D/PS+B`) is competitive with Mulknap on problems with high n/m ratio, and outperforms the previous state of the art for problems with low n/m ratios. We showed that there are problems for which the integration of these techniques

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

clearly resulted in better performance than either bound-and-bound or bin-completion alone.

There are several directions for future work. Although path-dominance is our most powerful symmetry relation and prunes the most nodes, 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. Finally, the symmetry and path-dominance 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 (Fukunaga and Korf, 2007).

Acknowledgements This work was supported by the Japan MEXT program, "Promotion of Env. Improvement for Independence of Young Researchers", the JSPS Compview GCOE, and a JSPS Grant-in-Aid for Young Scientists 20700131. Thanks to David Pisinger for making his Mulknap and Minknap code publicly available. Thanks to Rich Korf for helpful discussions about this work. Thanks to the anonymous reviewers for helpful comments.

References

Caprara, A., Kellerer, H., and Pferchy, U. (2000). A PTAS for the multiple-subset sum problem with different knapsack capacities. *Information Processing Letters*, *73*, 111–118.

Caprara, A., Kellerer, H., and Pferschy, U. (2003). A 3/4-approximation algorithm for multiple subset sum. *Journal of Heuristics*, *9*, 99–111.

Chekuri, C. and Khanna, S. (2000). A ptas for the multiple knapsack problem. In *Proceedings of the 11th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 213–222.

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

Fahle, T., Schamberger, S., and Sellmann, M. (2001). Symmetry breaking. In *Proceedings of the International Conference on Constraint Programming*, pp. 93–107.

Fischetti, M. and Salvagnin, D. (2008). Pruning moves. *Technical Report*.

Fischetti, M. and Toth, P. (1988). A new dominance procedure for combinatorial optimization problems. *Operations Research Letters*, *7*(4), 181–186.

Focacci, F. and Milano, M. (2001). Global cut framework for removing symmetries. In *Proceedings of the International Conference on Constraint Programming*, pp. 77–92.

Focacci, F. and Shaw, P. (2002). Pruning sub-optimal search branches using local search. In *Proc. Fourth International Workshop on Integration of AI and OR Techniques in Constraing Programming for Combinatorial Optimisation Problems (CP-AI-OR)*, pp. 181–189.

Fukunaga, A. (2008). A new grouping genetic algorithm for the multiple knapsack problem. In *Proc. IEEE Congress on Evolutionary Computation*, pp. 2225–2232.

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

Hung, M. and Fisk, J. (1978). An algorithm for the 0-1 multiple knapsack problem. *Naval Research Logistics Quarterly*, *24*, 571–579.

Ingargiola, G. and Korsh, J. (1975). An algorithm for the solution of 0-1 loading problems. *Operations Research*, *23*(6), 1110–1119.

Kalagnanam, J., Davenport, A., and Lee, H. (2001). Computational aspects of clearing continuous call double auctions with assignment constraints and indivisible demand. *Electronic Commerce Research*, *1*, 221–238.

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

Martello, S. and Toth, P. (1981a). A bound and bound algorithm for the zero-one multiple knapsack problem. *Discrete Applied Mathematics*, *3*, 275–288.

Martello, S. and Toth, P. (1981b). Heuristic algorithms for the multiple knapsack problem. *Computing*, *27*, 93–112.

Martello, S. and Toth, P. (1990). *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons.

Mitchell, D., Selman, B., and Levesque, H. (1992). Hard and easy distributions of SAT problems. In *Proceedings of AAAI*, pp. 459–65.

Pisinger, D. (1999). An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, *114*, 528–541.

Pisinger, D. (2005). Where are the hard knapsack problems?. *Computers and Operations Research*, *32*, 2271–2284.

Raidl, G. (1999). The multiple container packing problem: A genetic algorithm approach with weighted codings. *ACM SIGAPP Applied Computing Review*, 22–31.