Bin-Completion Algorithms for Multicontainer Packing and Covering Problems

Alex S. Fukunaga, Richard E. Korf
Computer Science Department
University of California, Los Angeles
fukunaga@cs.ucla.edu, korf@cs.ucla.edu

Abstract

Bin-completion, a bin-oriented branch-and-bound approach, was recently shown to be promising for the bin packing problem. We propose several improvements to bin-completion that significantly improves search efficiency. We also show the generality of bin-completion for packing and covering problems involving multiple containers, and present bin-completion algorithms for the multiple knapsack, bin covering, and min-cost covering (liquid loading) problems that significantly outperform the previous state of the art. However, we show that for the bin packing problem, bin-completion is not competitive with the state of the art solver.

1 Introduction

Many NP-hard problems involve packing some set of discrete objects into multiple containers ("bins"). In one class of problems, the objective is to *pack* some items into a set of containers without exceeding the containers' capacities. In a related class of problems, the goal is to *cover* a set of containers by filling them up to at least some minimal level using a set of items. When both the containers and the items are modeled as one-dimensional objects (possibly with an associated cost/value function), we refer collectively to these problems as *one-dimensional, multicontainer packing and covering problems*, or *multicontainer packing problems* in short. Multicontainer packing problems are ubiquitous, and model many AI applications including the allocation and rationing of resources or tasks among a group of agents, as well as operations research problems such as cargo loading and transport.

The most widely studied instance of a problem in this class is the *bin packing problem*: Given a set of items (numbers), and a fixed bin capacity, assign each item to a bin so that the sum of the items assigned to each bin does not exceed the bin capacity. For example, given the set of items 6, 12, 15, 40, 43, 82, and a bin capacity of 100, we can assign 6, 12, and 82 to one bin, and 15, 40, and 43, to another, for a total of two bins. This is an optimal solution to this instance, since the sum of all the items, 198, is greater than 100, and hence at least two bins are required.

In this paper, we consider *complete* algorithms for finding *optimal* solutions to multicontainer packing problems. Multicontainer packing problems are generally strongly NP-hard [Garey and Johnson, 1979] (including all of the problems covered in this paper). Therefore, state of the art, complete algorithms for finding optimal solutions are based on branch-and-bound. In contrast, note that single-container problems such as the classical 0-1 knapsack problem are weakly NP-hard, and pseudopolynomial time dynamic programming algorithms exist (c.f. [Kellerer *et al.*, 2004]).

Previous branch-and-bound algorithms for these problems have tended to be "item-oriented". For example, the classical algorithm for bin packing is the Martello and Toth algorithm [Martello and Toth, 1990]. Items are considered one at a time. Each node in the search corresponds to a decision regarding the assignment of an item to some non-full container. In contrast, *bin-completion* is a "bin-oriented" branch-and-bound strategy, where a node represents an assignment of a set of items to a single container. In addition to standard branch-and-bound pruning methods, a combinatorial *dominance criterion* is used to prune search.

This paper begins by reviewing the previous work on bin-completion. We then propose several improvements, including a generalization of the nogood pruning method proposed in [Korf, 2003] that significantly improves search efficiency. We demonstrate the generality of the bin-completion approach by presenting new bin-completion algorithms for the multiple knapsack, bin covering, and min-cost covering problems that significantly outperform the state of the art algorithms.

2 Bin-Completion

Bin-completion is a bin-oriented branch-and-bound strategy that exploits dominance properties to reduce the search space. For clarity, we describe the bin-completion algorithm and our extensions in the context of the bin packing problem due to its familiarity and simplicity, although the focus of our empirical work is on other multicontainer problems.

A *feasible set* is a set of items whose sum satisfies the capacity constraint (i.e., does not exceed the bin capacity). We say that a feasible set or bin assignment is *maximal* if no additional item can be added to the set without making the set infeasible. The items are sorted in decreasing order of size. We then generate maximal, feasible sets that include the largest

item. If there is more than one such set, the search may branch at that point. Each node of the search tree, except the root node, represents a complete, feasible assignment of items to a particular bin. The children of the root represent different ways of completing the bin containing the largest item. The nodes at the next level represent different complete, feasible sets that include the largest remaining item, etc. The depth of any branch of the tree is the number of bins in the corresponding solution. Bin-completion is a branch-and-bound algorithm. It starts with an upper bound, such as the best-fit decreasing solution, and applies a lower-bound heuristic function to prune the search. Rather than assigning items one at a time to bins, it branches on the different maximal, feasible sets that can be assigned to each bin.

The key to making bin-completion efficient is the use of a *dominance criterion* on the maximal, feasible sets that allows us to only consider a small subset of them.

Let A and B be two feasible sets. If the elements in B can be partitioned into subsets, and these subsets can be matched to the elements of A such that the sum of the elements in each subset doesn't exceed the corresponding element of A, then set A *dominates* set B. In other words, if the elements of B can be packed into bins whose capacities are the elements of A, then set A dominates set B. For example, let $A = \{20, 30, 40\}$ and let $B = \{5, 10, 10, 15, 15, 25\}$. Partition B into the subsets $\{5, 10\}$,$\{25\}$, and $\{10, 15, 15\}$. Since $5 + 10 \leq 20$, $25 \leq 30$, and $10+15+15 \leq 40$, set A dominates set B. Given all the feasible sets that contain a common element s, only the undominated sets need to be considered for assignment to the bin containing s. The reason is that if we complete the bin containing s with a dominated set, then we could swap each subset of items in the dominated set with the corresponding element of the dominating set, and get another solution without increasing the total number of bins.

This dominance criterion was proposed by Martello and Toth [Martello and Toth, 1990], and was used in their branch-and-bound algorithm for bin-completion. However, their branch-and-bound algorithm was item-oriented, and they only exploit this dominance property in a limited way. In particular, they take each item j, starting with the largest element, and check if there is a *single* assignment of one or two more elements that dominates all feasible sets containing j. If so, they place j with those elements in the same bin, and apply the reduction to the remaining subproblem. They also use dominance relations to prune some element placements as well. Korf's bin-completion algorithm [Korf, 2002] makes much greater use of the Martello-Toth dominance criterion. In particular, when branching on the completion of any bin, it only considers undominated completions.

Historically, the first bin-completion algorithm we are aware of was proposed by Christofides, Mingozzi, and Toth in 1979 for the liquid loading problem, which we call the *min-cost covering problem* (see Section 6) [Christofides *et al.*, 1979]. Although their algorithm performs a bin-oriented branch-and-bound which considers only undominated completions, they use a much weaker dominance criterion (see Section 6). However, subsequent research in branch-and-bound algorithms for multicontainer problems focused on item-oriented approaches, and the bin-completion approach was apparently not investigated further until recently.

The BISON algorithm for bin packing by Scholl, Klein, and Jurgens [Scholl *et al.*, 1997] is a hybrid algorithm that integrates a suite of complex lower bounding procedures, upper bounding heuristics, and a branch-and-bound algorithm. The branch-and-bound component of BISON is a bin-completion algorithm where each node corresponds to a maximal, feasible bin assignment. A very limited form of the Martello-Toth dominance criterion is applied, as follows. If a maximal, feasible bin assignment has a pair of items that can be replaced by single unassigned item without decreasing the sum of the assignment, then it is dominated, and this node can be pruned.

Korf implemented a bin-completion algorithm for bin packing using the full Martello and Toth dominance criterion, and showed that it significantly outperformed the Martello-Toth item-oriented branch-and-bound algorithm [Korf, 2002]. Further improvements were achieved by using a more efficient algorithm for generating the undominated bin assignments, and nogood pruning (see below) [Korf, 2003].

3 Extensions to Bin-Completion

3.1 Nogood Dominance Pruning

Suppose we have a bin packing instance with the numbers 10,9,8,7,7,3,3,2,2, and bin capacity c=20. Let B_d represent a bin assignment at depth *d*. $\{10,8,2\}$ and $\{10,7,3\}$ are two undominated feasible bin assignments, and thus $B_1 = \{10, 8, 2\}$ and $B_1 = \{10, 7, 3\}$ are two possible assignments of the bin (node) at depth 1.

Korf [Korf, 2003] proposed the following *nogood pruning* technique. After exhausting the subproblem below the assignment $B_1 = \{10, 8, 2\}$, and while exploring the subproblem below the assignment $B_1 = \{10, 7, 3\}$, assume we find a solution that assigns $B_2 = \{9, 8, 2\}$. We can swap the $\{8, 2\}$ from B_2 with the $\{7, 3\}$ from B_1, resulting in a solution with $B_1 = \{10, 8, 2\}$ and and $B_2 = \{9, 7, 3\}$. However, we have already exhausted the subtree below $B_1 = \{10, 8, 2\}$, and therefore, we can prune this branch because it is redundant.

In general, given a node with more than one child, when searching the subtree of any child but the first, we don't need to consider bin assignments that assign to the same bin all items used to complete the current bin in a previously-explored child node, except for the largest element. More precisely, let $\{N_1, N_2, ..., N_m\}$ be a set of brother nodes in the search tree, and let $\{S_1, S_2, ..., S_m\}$ be the sets of items used to complete the bin in each node, excluding the first item assigned to the bin, which is common to all the brother nodes. When searching the subtree below node N_i for $i > 1$: for all $j < i$, we exclude any bin assignment B that (1) includes all the items in S_j in the same bin, and (2)swapping S_j from B with the items S_i in N_i results in two feasible bin assignments. By rejecting these bin assignments as redundant, the number of node generations is reduced.

We now propose an extension to this idea that allows pruning even more nodes, which we call *nogood dominance pruning*, or **NDP**. Suppose that after exhausting the subproblem below the assignment $B_1 = \{10, 8, 2\}$, and while exploring the subproblem below the assignment $B_1 = \{10, 7, 3\}$, we consider the assignment $B_2 = \{9, 7, 2\}$. We can swap the

$\{7,2\}$ from B_2 with the $\{7,3\}$ from B_1 and end up with a solution with the $B_1 = \{10,7,2\}$ and $B_2 = \{9,7,3\}$. However, according to the Martello-Toth dominance criterion, $B_1 = \{10,7,2\}$ is dominated by $B_1 = \{10,8,2\}$, and we have already exhausted the search below the node with the subtree under $B_1 = \{10,8,2\}$, so we can prune the search because it is not possible to improve upon the best solution under $B_1 = \{10,8,2\}$.

In general, given a node with more than one child, when searching the subtree of any child but the first, we don't need to consider assignments that are dominated by a bin assignment in a previously-explored child node. More precisely, when searching the subtree below node N_i for $i > 1$, we exclude any bin assignments that are dominated by the items in S_j, for $j < i$. Note that an assignment dominates itself. Thus, no bin completion below node N_i can be dominated by the items in S_j, for $j < i$.

Nogood dominance pruning is strictly more powerful than Korf's nogood pruning. Any node pruned by nogood pruning will be pruned by NDP, but not vice versa. Of course, since NDP must detect dominance relationships as opposed to equivalence relationships, NDP will incur more overhead per node compared to nogood pruning. Our current implementation propagates a list of nogood sets along the tree. While generating the undominated completions for a given bin, we check each one to see if it is dominated by any current nogood. If so, we ignore that bin-completion.

Since the size of the nogood list increases with depth, and we compare each bin-completion against each nogood, the per-node overhead of NDP increases with depth. This means that pruning at the bottom of the tree (where pruning has the lowest utility) is more expensive than pruning at the top of the tree (where pruning has the highest utility). A simple strategy which address this issue is *depth-limited NDP*, where NDP pruning is applied only at nodes down to the NDP depth limit L. At nodes below the depth limit, Korf's original nogood pruning technique is applied.

In this section, we have described nogood pruning and NDP in the context of the bin packing problem. Adaptations of the technique for the multiple knapsack and bin covering problem are straightforward, given the dominance criteria used for these other problems.

3.2 Further modifications to the search algorithm

In Korf's algorithm for bin packing, at each node, the undominated feasible bin assignments are generated, and sorted according to decreasing sum. The algorithm branches on the completions according to the sorted order. In other words, a largest-sum-first *value ordering heuristic* is being applied at each node.

An issue with enumerating all undominated completions and applying value ordering is that computing the undominated sets is itself NP-complete. Korf [Korf, 2003] gives an algorithm that generates all and only undominated bin-completions, without the need to generate dominated completions as intermediate results. Let d be the average number of items that fit in a container. The time to generate all undominated feasible assignments of a bin increases with d. This is not an issue for bin packing, where problems with large d tend to be easily solved using heuristics such as best-fit decreasing. The solution found by the heuristic often equals the lower bound, and therefore no search is required, and hence there is no need to compute undominated completions. However, for the other three problems we considered, we have observed that it is much less likely that heuristics will match the optimistic bound and allow termination without search, and we have found that for instances with high d, the algorithm would either take a very long time to complete, or not terminate within a reasonable time limit because it was spending an inordinate amount of time computing the set of undominated completions at each node. Another significant drawback of completely enumerating the undominated bin assignments at each node is the memory required to store all of them.

An alternative approach is to start to go down the search tree and explore the children of a node without first enumerating and sorting the children. In cases where a good optimistic bound (i.e., lower bound for bin packing, and upper bound for bin covering and multiple knapsack) is available, and it takes relatively little search to find an optimal solution, this approach can result in significant speedups compared to the original scheme of generating all completions before going further down the search tree.

On the other hand, we have observed that the value-ordering strategy used to sort the undominated children of a node has a significant impact, and if we simply traverse the search tree depth-first without first enumerating and sorting the children, we lose the benefits of ordering the search tree according to the value-ordering heuristic. The negative impact of this can be alleviated by a hybrid strategy that generates a small number of children, applies the value-ordering heuristic to these, then recursively calls bin-completion on the remaining subproblem. We call this the *hybrid incremental branching strategy*.

Note that even in the absence of an explicit value ordering strategy, the algorithm used to generate the undominated children (for each problem, we use an adaptation of the algorithm given by Korf [Korf, 2003]) imposes an implicit ordering on the undominated assignments. We are currently evaluating various combinations of hybrid incremental branching and various value ordering strategies, and the choices reported in this paper for value ordering strategies and the number of undominated assignments generated at once in hybrid incremental branching are preliminary.

4 The Multiple Knapsack Problem

The *Multiple Knapsack Problem* (MKP) is a generalization of the classical *0-1 Knapsack Problem*. Consider m bins of capacities $c_1, c_2, ..., c_m$. and a set of n items that have a *weight* $w_1, ..., w_n$ and a *profit* $p_1, ..., p_n$. The goal of the MKP is to assign some subset of the items to each bin such that: (1) each item is assigned to no more than one bin, and (2) the sum of the weights of the items assigned to a bin does not exceed the bin's capacity, and (3) the total profit of the items that are assigned to a bin are maximized. 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. The MKP has numerous industrial applications. For example, the problem of loading m vehicles with cargo selected from n items in order to maximize the value of the transported items is a MKP instance.

4.1 The Mulknap Algorithm

The state of the art algorithm for the MKP is Pisinger's Mulknap algorithm [Pisinger, 1999]. Mulknap is a branch-and-bound algorithm, where each node represents a decision as to which bin to place an item into, or to leave it out entirely. There are several well-known upper bounds for the MKP. Recent algorithms, including Mulknap, rely on the solution of the *Surrogate relaxed Multiple Knapsack Problem* (SMKP) instance [Martello and Toth, 1990], which is a single-container 0-1 Knapsack problem where the items are the same as for the original MKP instance, but there is a single container whose capacity is the sum of the capacities of the containers in the original MKP instance. Although the 0-1 Knapsack problem is also NP-complete (weakly NP-complete, since there is a pseudopolynomial algorithm), this upper bound computation is very fast in practice. At each node, Mulknap attempts to *validate* the upper bound for the remaining subproblem by distributing the set of items chosen as the solution for the SMKP into the bins. If such a distribution is possible without violating the bin capacities, then the upper bound has been achieved, and we can backtrack without further search down the current branch. The distribution is done by solving a series of subset-sum problems. First, the smallest knapsack is filled as much as possible with the items from the SMKP solution. Then the second smallest is filled with items from the remainder of the SMKP solution, and so on. Again, while this entails the solution of m subset sum instances, it is fast in practice (takes less time than the upper bound computation using the SMKP). In addition, Mulknap also incorporates techniques for reducing the problem instance at every node, as well as tightening the capacity constraints. See [Pisinger, 1999] for details.

4.2 Bin-Completion for the MKP

We define a dominance criterion for the MKP as follows: In a feasible bin assignment for a bin with capacity c_b, the sum of the item weights is no more than c_b.

Let A and B be two feasible assignments. 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 (but not necessarily onto) to 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.

Our bin-completion algorithm for MKP uses the same upper bound and reduction techniques as Mulknap. Our current implementation uses a hybrid incremental branching strategy (see 3.2) that generates two undominated children at a time, then uses a value ordering strategy that sorted them according to descending order of profit sums.

We evaluated our MKP algorithm using the same 4 classes of instances used by Pisinger [Pisinger, 1999]. We considered: (1) uncorrelated instances: p_j and w_j are uniformly distributed in R, (2) weakly correlated instances: w_j uniformly distributed in [10,1000] and p_j randomly distributed in $[w_j - 99, w_j + 99]$ such that $p_j \geq 1$, (3) strongly correlated instances: w_j uniformly distributed in [10,1000] and $p_j = w_j + 10$, and (4) multiple subset sum instances: w_j uniformly distributed in [10,1000] 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 (if the first $m-1$ capacities exceeded half the total weight, the instance was discarded). For each problem class, 30 instances were generated for various values of m and n. Trivial instances were discarded as in [Pisinger, 1999]. The class of uniform, random instances that require the most search for branch-and-bound solvers appear to be generated when n/m is relatively low [Pisinger, 1999; Martello and Toth, 1990]. Thus, the n/m ratio for the MKP appears to be a critical parameter that determines search difficulty. Pisinger has shown that for problems with high n/m ratio, Mulknap is highly effective, solving very large instances almost instantaneously with little or no search (when n ranged from 200 to 100,000, and m ranged 5 or 10). On the other hand, small n/m ratios between 2-3 result in the hardest problems. We therefore focus on these hard problems.

On each instance, we ran Mulknap, bin-completion, and bin-completion with Nogood Dominance Pruning (BC+NDP). For comparison, we used Pisinger's Mulknap code, available at his website, [1] compiled using the `gcc` compiler with `-O3` optimization settings. Our bin-completion code was implemented in Common Lisp.[2] Table 1 shows our results. All experiments were run on a 1.3GHz AMD Athlon. Each algorithm was given a time limit of 300 seconds to solve each instance. The *fail* column indicates the number of instances (out of 30) which could not be solved by the algorithm within the time limit. The *time* and *nodes* column show the total time spent and nodes generated **on the successful runs, excluding the failed runs**. Thus, the most important indicator of performance is the number of failed runs.

Both bin-completion and BC+NDP significantly outperform Mulknap, with the difference in performance becoming more pronounced as problem size was increased. This indicates that bin-completion is asymptotically more efficient than Mulknap for this class of problems. BC + NDP consistently outperforms bin-completion by a significant margin with respect to the number of nodes searched, and significant improvements in success rate and runtimes are observed for the larger problems sets. Similar results were obtained when the items had weights and profits in the range [10,100], as

[1] http://www.diku.dk/ pisinger/

[2] All of our bin-completion algorithms described in this paper were compiled using CMUCL version 18e. To serve as a point of reference, we ported Korf's bin-completion algorithm for bin packing to Common Lisp, and found that CMUCL generated code that ran at approximately half of the speed of Korf's C code compiled with `gcc` and `-O3` optimization.

	Mulknap		Bin-Completion			Bin-Completion + NDP		
(m,n)	**fail**	time	**fail**	time	nodes	**fail**	time	nodes
Uncorrelated Instances								
10,30	**10**	851	0	330	7994849	0	116	2059561
15,45	**29**	82	**27**	122	2818240	**27**	83	1389073
10,20	**8**	618	0	<1	1866	0	<1	1038
20,40	**30**	-	0	141	4703755	0	20	416409
25,50	**30**	-	**3**	1459	47252632	0	490	9229360
Weakly Correlated Instances								
10,30	**11**	841	0	129	2168245	0	99	1544200
15,45	**30**	-	**23**	852	13632608	**22**	908	13181578
10,20	**4**	1334	0	<1	1194	0	<1	794
20,40	**30**	-	0	16	634072	0	4	84270
25,50	**30**	-	**1**	734	25605562	0	205	3339290
Strongly Correlated Instances								
10,30	0	571	0	73	684006	0	62	585643
15,45	**30**	-	**26**	661	3697531	**25**	711	3816290
10,20	**8**	808	0	<1	797	0	<1	557
20,40	**30**	-	0	9	359645	0	3	58665
25,50	**30**	-	**1**	424	13008484	**1**	124	1757203
Subset Sum Instances								
10,30	0	694	0	15	104240	0	14	93727
15,45	**26**	711	**4**	1788	9452862	**3**	2017	10200510
10,20	**8**	941	0	<1	624	0	<1	462
20,40	**30**	-	0	3	126931	0	1	28840
25,50	**30**	-	0	137	4613658	0	72	1261483

Table 1: Multiple Knapsack results. (Times are in seconds).

well as the range [10,10000].

5 The Bin Covering Problem

Suppose we have n items with weights $w_1, ..w_n$, and an infinite supply of containers with quota q. The *bin covering* problem, also known as the *dual bin packing* problem, is to pack the items into containers such that the number of containers that contain sets of items whose sum is at least q is **maximized**. That is, the goal is to distribute, or ration the items among as many containers as possible, given that the containers have a specified quota that must be satisfied. Note that the total weight of the items placed in a container can be greater than q.

Bin covering is a model for resource or task allocation among multiple agents where the goal is to maximize the number of agents who fulfill some quota. It is also a model for industrial problems such as: (1) packing peach slices into cans so that each can contains at least its advertised net weight in peaches, and (2) breaking up monopolies into smaller companies, each of which is large enough to be viable [Assman *et al.*, 1984].

5.1 The Labbé, Laporte, and Martello Algorithm

The state of the art complete algorithm for the bin covering problem is by Labbé, Laporte, and Martello [Labbé *et al.*, 1995]. We shall refer to this as the *LLM* algorithm. LLM is a branch-and-bound algorithm. The items are sorted in decreasing order of size. Each node represents a decision as to which bin to put an item into, or to leave it out entirely. At each node, lower bounds based on combinatorial arguments are computed, and the remaining subproblem is reduced using two reduction criteria (see [Labbé *et al.*, 1995]). At the root node, a set of heuristics is applied in order to compute a lower bound. As with the bin packing problem (c.f. [Korf, 2002]), many instances of the bin covering problem require

n	Labb´e		Bin-Completion			Bin-Completion + NDP		
	fail	time	**fail**	time	nodes	**fail**	time	nodes
10,000 instances per row								
10	0	2	0	2	215	0	2	215
20	0	4	0	4	595	0	4	548
40	0	78	0	10	5860	0	10	3888
60	**2**	458	0	154	3610644	0	127	2467708
80	**17**	823	**2**	110	3191035	**1**	88	1135615
100	**28**	1210	**1**	198	6045642	**1**	79	613463
50,000 instances per row								
80	n/a	n/a	**6**	332	7597514	**4**	362	5763523
90	n/a	n/a	**6**	574	15079358	**4**	493	7320350
100	n/a	n/a	**8**	803	20739466	**6**	585	11600379
120	n/a	n/a	**20**	1941	65428014	**16**	1543	33040246

Table 2: Bin Covering results. (Times are in seconds).

no search, and can be immediately solved at the root node when the heuristics described in [Labbé *et al.*, 1995] compute a lower bound that equals the upper bound.

5.2 Bin-Completion for Bin Covering

We define a dominance criterion for bin covering as follows. Let A and B be two feasible assignments. A dominates B if B can be partitioned into i subsets $B_1, ..., B_i$ such that each item $A_k \in A$ is mapped one-to-one (but not necessarily onto) a subset B_k, such that for all $k \leq i$, the sum of the item weights of subset B_k is greater than or equal to the corresponding item A_k (i.e., B_k "covers" A_k).

To see that this dominance criterion gives a valid pruning criterion, suppose a solution to a bin-completion instance assigns B to a bin. If we exchange the items in A for the corresponding subsets of B, then this would result in a valid solution that satisfies at least as many bins.

As with LLM, we apply lower bounding heuristics at the root node, and apply the same upper bounds used in LLM [Labbé *et al.*, 1995] at each node. We use a hybrid incremental branching strategy that generates two undominated children at a time, then using smallest-sums-first value ordering to order them.

We compared our implementation of bin-completion for the bin covering problem with our implementation of the LLM algorithm (both implementations were in Common Lisp and run on a 600MHz Pentium 3). We generated 10,000 random instances for each set, where the items were chosen uniformly in the range [1,9999], and the quota was 10,000. We ran our implementations of LLM, bin-completion, and bin-completion + NDP on each instance, with a time limit of 180 seconds per instance. We also compared the two bin-completion variants on larger instances of 80-120 items, using sets of 50,000 random instances. Table 2 shows our results. Again, we show the number of failed runs that timed out before solving the instance, as well as the time spent and nodes generated, **excluding the failed runs**. Bin-completion significantly outperformed LLM. The increasing gap in performance as problem size increases suggests that bin-completion is asymptotically more efficient than LLM. For the harder problems, bin completion + NDP significantly outperforms bin-completion.

6 Min-Cost Covering

We define the *Min-Cost Covering Problem* (MCCP) as follows. Given a set of m bins with quotas $\{q_1, ..., q_m\}$, and a set of n items with weights $w_1, ..., w_n$ and costs $p_1, ..., p_n$, assign some subset of the items to each bin such that (1) each item is assigned to no more than one bin, (2) the sum of the weights of the items assigned to a bin is at least the bin's capacity (i.e., the bin is covered, as in bin covering), and (3) the total cost of the items that are assigned to the bin are minimized. This problem has also been called the *liquid loading problem* [Christofides *et al.*, 1979], because it was originally motivated by the following industrial application: Consider the disposal or transportation of m different liquids (e.g., chemicals) that cannot be mixed. If we are given n tanks of various sizes, each with some associated cost, the problem is to load the m liquids into some subset of the tanks so as to minimize the total cost. Note that here, the "liquids" correspond to containers, and the "tanks" correspond to the items.

6.1 Christofides, Mingozzi, and Toth (CMT) Algorithm

The previous state of the art algorithm for the min-cost covering problem is an early version of bin-completion by Christofides, Mingozzi, and Toth [Christofides *et al.*, 1979]. The main difference between their algorithm and the bin-completion algorithm described in Section 2 is their use of a different dominance criterion. The Christofides et al. dominance criterion (CMT criterion) is as follows. A set of items A dominates another set B if is possible to map A one-to-one to (but not necessarily into) B by a function π such that $w_j \geq w_{\pi(j)} \forall j \in A$, and $p_j \geq p_{\pi(j)} \forall j \in A$.

Note that the CMT criterion only considers mappings between single elements of A and B, whereas the dominance criteria we have described above for bin packing, MKP, and bin covering considers mappings between elements of A and subsets of B.

Christofides et al. empirically evaluated a number of lower bounds for the MCCP. They found that the best bound was computed by solving m single-container MCCP problems, one for each of the bins, where all n items were available for the single-bin MCCP problem. Each single-bin instance was solved using a simple branch-and-bound algorithm. The lower bound is the sum of the optimal values for the m single-container MCCP problems. This is essentially a lower bound derived by relaxing the MCCP constraint that each item can be assigned to at most one bin.

6.2 Bin-Completion for the MCCP

Our new bin-completion algorithm for the MCCP is similar to the Christofides et al. algorithm. The major difference is that we use the following dominance criterion. A set of i items A dominates another set B if B can be partitioned into i subsets $B_1, ..., B_i$ such that each item $A_k \in A$ is mapped one-to-one (but not necessarily onto) to a subset B_k, and for all $k \leq i$, (1) the weight of A_k is less than or equal to the sum of the item weights of the items in B_k, and (2) the profit of item A_k is less than or equal to the sum of the profits of the items in B_k. The CMT dominance criterion is a special case of this criterion, where $|B_k| = 1$ for all k.

We evaluated our bin-completion algorithm and the CMT algorithm using the same classes of problem instances that we used for the MKP (Section 4). For each problem class, 30 nontrivial instances were generated for various values of m and n. The purpose of this experiment was to evaluate the relative impact of each component of bin-completion, that is: (1) the type of dominance criterion used, (2) whether nogood pruning was used, and (3) whether nogood dominance pruning was used.

On each instance, we ran (1) **CMT1**: the CMT algorithm as presented in [Christofides *et al.*, 1979], (2) **CMT2**: the CMT algorithm extended with nogood pruning, (3) **BC1**: bin-completion using our dominance criterion but *without* nogood pruning, (4) **BC2**: bin-completion using our dominance criterion and with nogood pruning, (5) **BC3**: bin-completion using our dominance criterion and NDP. For all algorithms, a hybrid incremental branching strategy that generated 100 children at a time was used, and the children were ordered in increasing order of profit sums.

Table 1 shows our results. All experiments were run on a 2.5GHz Pentium IV. Each algorithm was given a time limit of 60 seconds to solve each instance. The *fail* column indicates the number of instances (out of 30) which could not be solved by the algorithm within the time limit. The *time* and *nodes* column show the total time spent and nodes generated, excluding the failed runs. As shown by the data, each component of bin-completion has a significant impact. Although our dominance criterion requires much more computation per node to compute than the simpler CMT criterion, the performance is significantly improved. The nogood pruning strategy significantly improves performance for both the algorithm based on the CMT dominance criterion, as well as our dominance criterion. Finally, BC3, which uses NGD is results in the best performance. We also tested an item-oriented branch-and-bound algorithm that uses the same lower bounding procedure, but found that it performed even worse than the CMT algorithm on all of the test instances.

7 Bin Packing

We implemented our extended bin-completion algorithm for the bin packing problem, and summarize our results below. For further details, see [Fukunaga, 2005].

Extending Korf's algorithm [Korf, 2003] with NDP and a value ordering based on minimum cardinality (with ties broken according to decreasing sum of item weights) resulted in significant speedups over Korf's algorithm on uniform random problem instances, as well as the class of "uniform" OR-LIB[3] instances of 120-1000 items, for the same experiments described in Korf's paper [Korf, 2003]. In addition, extending the algorithm further by applying randomized restarts [Gomes *et al.*, 1998] resulted in significant speedups compared to Korf's algorithm on the "triplet" instances from OR-LIB. For example, our extended algorithm (implemented in Common Lisp and executed on a 1.3GHz Athlon) solves all 80 of the triplet instances from OR-LIB within 1500 seconds combined, whereas Korf's algorithm (implemented in C and executed on a 440MHz Sun Ultrasparc) failed to solve any

[3] see http://www.brunel.ac.uk/depts/ma/research/jeb/info.html

	CMT1		CMT2		BC1		BC2		BC3	
(m,n)	**fail**	time	**fail**	time	**fail**	time	**fail**	time	**fail**	time
Uncorrelated Instances										
5,15	0	3	0	2	0	<1	0	<1	0	<1
10,30	**30**	-	**30**	-	**30**	-	**8**	605	**6**	567
5,10	0	<1	0	<1	0	<1	0	<1	0	<1
10,20	**7**	324	0	5	0	168	0	1	0	1
15,30	**30**	-	**27**	49	30	-	**6**	360	**1**	511
Weakly Correlated Instances										
5,15	0	6	0	2	0	<1	0	<1	0	<1
10,30	**30**	-	**30**	-	**30**	-	**28**	78	**27**	77
5,10	0	<1	0	<1	0	<1	0	<1	0	<1
10,20	0	88	0	1	0	4	0	<1	0	<1
15,30	**30**	-	**6**	325	**25**	167	0	115	0	20
20,40	**30**	-	**30**	-	**30**	-	**28**	72	**13**	410
Strongly Correlated Instances										
5,15	0	2	0	1	0	<1	0	<1	0	<1
10,30	**30**	-	**30**	-	**4**	361	**1**	207	0	93
5,10	0	5	0	<1	0	<1	0	<1	0	<1
15,30	**17**	286	0	95	0	64	0	3	0	1
20,40	**30**	-	**19**	330	**23**	330	**6**	180	0	58
Subset Sum Instances										
5,15	**0**	2	0	1	0	<1	0	<1	0	<1
10,30	**30**	-	**30**	-	**4**	362	**1**	224	0	100
5,10	0	<1	0	<1	0	<1	0	<1	0	<1
10,20	0	5	0	<1	0	<1	0	<1	0	<1
15,30	**17**	289	0	100	0	65	0	3	0	<1
20,40	**30**	-	**19**	258	**23**	172	**6**	181	0	63

Table 3: Min-cost covering problem. (Times are in seconds).

of the twenty 501-item instances, and failed to solve 8 of the 249-item instances given a 10 minute time limit per instance. See [Fukunaga, 2005] for further results.

However, our bin-completion solver was not competitive with the state of the art, which is currently a branch-and-price integer linear programming solver by Belov and Scheithauer [Belov and Scheithauer, 2003]. Belov and Scheithauer [Belov and Scheithauer, 2003] provide a new benchmark set of 28 very hard bin packing instances; they solved most of these within seconds, although some took hours. Our current bin-completion code could not solve any of the 28 instances, given 15 minutes per instance. They also report that the largest triplet instances from OR-LIB (Triplet-501) were solved in an average of 33 seconds per instance (660 seconds total) on a 1GHz AMD Athlon XP.

Furthermore, Belov was kind enough to run a set of one hundred, 80-item, uniform, random instances that we generated with items in the range [1,1,000,000] and a bin capacity of 1,000,000 for us. His solver solved all of these instances in less than 1 second each at the root node (i.e., without any search), using rounding heuristics based on the linear programming LP solution, whereas our best bin-completion solver required 534 seconds and searched 75,791,226 nodes.

Recent branch-and-price approaches for bin packing such as the solver by Belov and Scheithauer (see [de Carvalho, 2002] for a survey of others) use a bin-oriented branching strategy, where decisions correspond to the instantiation of one or more complete bins. At each node, a column generation procedure is used to compute the LP lower bound. They derive much of their power from a very accurate LP lower bound based on a cutting stock formulation of bin packing [Gilmore and Gomory, 1961], which has been observed to almost always give the optimal value as the lower bound, and has never been observed to give a value that is more than one higher than the optimal value (c.f. [Wäscher and Gau, 1996]). In addition, rounding heuristics applied to fractional LP-solutions often yields the optimal, integral solution. The combination of a very tight LP lower bound and good upper bounding procedure result in very little search being performed for almost all problem instances. This domain-specific branch-and-price LP-based approach does not seem to generalize straightforwardly to the MKP and MCCP, in part due to the differences in the granularity of the objective function. While it is possible that a similar branch-and-price approach could be applied to the bin covering problem, we are aware of no such approach in the literature.

For completeness, we have applied a straightforward, integer-linear programming approach to the MKP, MCCP, and bin covering problems. We applied the GNU GLPK integer linear programming solver to the mathematical programming formulations of these problems, but not surprisingly, these performed very poorly in comparison with the domain-specific algorithms described in this paper.

8 Related Work

The multicontainer packing problems we studied in this paper can be formulated as constraint programming problems. In one possible formulation, the variables correspond to bins, and the values of the bins correspond to unique, feasible bin assignments (this is similar to the integer linear programming formulation of the cutting-stock problem [Gilmore and Gomory, 1961]). For example, given items with weights $\{3, 2\}$ let x_1 be the variable representing the first bin with capacity 4. The domain of x_1 is the two singleton sets $\{3\}$ and $\{2\}$.

Given this constraint programming formulation, it can be seen that the pruning techniques based on nogoods described in Section 3 are closely related to techniques proposed for constraint programming. Nogood pruning identifies and prunes nodes by detecting whether the bin assignment for the current node contains a nogood. This is very similar to nogood-based techniques for pruning symmetric solutions proposed by Fahle, Schamberger and Sellmann [Fahle *et al.*, 2001] and Focacci and Milano [Focacci and Milano, 2001],

The NGD technique proposed in this paper is similar to the nogood dominance pruning technique proposed by Focacci and Shaw [Focacci and Shaw, 2002] for constraint programming, who applied their technique to the symmetric and asymmetric traveling salesperson problem with time windows. See [Focacci and Shaw, 2002] for a full description of their technique.

Both methods attempt to prune the search by proving that the current node at depth j, which represents a partial j-variable solution x, is dominated by some previously explored i-variable partial solution (nogood), q.

The main difference between the two methods 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'.

On the other hand, our NGD method starts with a partial, j-variable solution x and tries to transform it to a partial solution x' such that $\bar{x}'_i$, the subset of x' including the first i

variables, is dominated by q. We do this by swapping the values of the ith and jth variables in x to derive x', and testing whether $\bar{x}'_i$ is dominated by q.

For efficiency, the current implementations of both nogood dominance pruning methods are *weak*, in the sense that if x is dominated by q, the procedures will not necessarily detect the dominance. Focacci and Shaw rely on an incomplete, local search to find the extension q'. We only consider transformations involving two variables, but to fully exploit the dominance criterion, we would need to consider transformations involving all variables $i, i+1, ..j$.

9 Conclusions

This paper makes two main contributions: First, we proposed extensions to bin-completion to improve its performance, including nogood dominance pruning, which generalizes Korf's nogood pruning method. Second, we demonstrated the utility and generality of bin-completion by applying it to three new problems: the bin covering problem, the multiple knapsack problem, and the min-cost covering problem. We proposed new dominance criteria for these problems, and showed that our bin-completion solvers significantly outperform , the previous state of the art on hard instances (in some cases by orders of magnitude with respect to runtime). For these three problems, our nogood dominance pruning technique significantly reduced search effort and runtimes. The comparison of our bin-completion algorithm to the early bin-completion algorithm by Christofides et al. for the MCCP showed that our new dominance criteria resulted in significant performance improvements.

Finally, we revisited the bin packing problem and showed that our extensions significantly improved the performance of bin-completion for some standard benchmarks. However, we found that bin-completion is not competitive with state of the art solvers for bin packing based on the cutting stock problem formulation.

While we have focused on four particular multicontainer problems in this paper, there are many similar problems involving the assignment of objects to multiple containers where similar dominance structures can be exploited. Examples include the generalized assignment problem, multiprocessor scheduling, and the segregated storage problem. Given our results, maximally exploiting powerful dominance criteria in a bin-completion framework appears to be a promising future direction for such multicontainer problems.

10 Acknowledgments

Thanks to Gleb Belov for running his solver on some of our bin packing test instances, and to the anonymous reviewers for helpful comments. This research was supported by NSF under grant No. EIA-0113313.

References

[Assman *et al.*, 1984] S.F. Assman, D.S. Johnson, D.J. Kleitman, and J.Y.T. Leung. On a dual version of the one-dimensional bin-packing problem. *J. Algorithms*, 5:502–525, 1984.

[Belov and Scheithauer, 2003] G. Belov and G. Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. Technical report, Technical University Dresden, 2003. Preprint MATH-NM-03-2003.

[Christofi des *et al.*, 1979] N. Christofi des, A. Mingozzi, and P. Toth. Loading problems. In N. Christofi des, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*. John Wiley & Sons, 1979.

[de Carvalho, 2002] J.M. Val´erio de Carvalho. LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141:253–273, 2002.

[Fahle *et al.*, 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *Proc. Int. Conference on Constraint Programming*, pages 93–107, 2001.

[Focacci and Milano, 2001] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proc. Int. Conference on Constraint Programming*, pages 77–92, 2001.

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

[Fukunaga, 2005] A. Fukunaga. *Bin-Completion Algorithms for One Dimensional, Multicontainer Packing Problems*. PhD thesis, UCLA, 2005. forthcoming.

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[Gilmore and Gomory, 1961] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.

[Gomes *et al.*, 1998] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proc. AAAI*, pages 431–437, Madison, Wisconsin, 1998.

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

[Korf, 2002] R. Korf. A new algorithm for optimal bin packing. In *Proc. AAAI*, pages 731–736, 2002.

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

[Labb´e *et al.*, 1995] M. Labb´e, G. Laporte, and S. Martello. An exact algorithm for the dual bin packing problem. *Operations Research Letters*, 17:9–18, 1995.

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

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

[Scholl *et al.*, 1997] A. Scholl, R. Klein, and C. Jürgens. BISON: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers in Operations Research*, 24(7):627–645, 1997.

[Wäscher and Gau, 1996] G. Wäscher and T. Gau. Heuristics for the integer one-dimensional cutting stock problem: A computational study. *OR Spektrum*, 18(3):131–144, 1996.