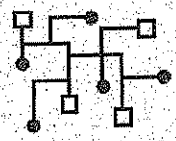

ISE

Industrial and Systems Engineering

Problem Reduction for One-Dimensional Cutting And Packing Problems

Peter A. Huegler
Bethlehem Steel Corporation

And

Joseph C. Hartman
Lehigh University

Report No. 02T-003

LEHIGH

University

200 West Packer Avenue
Bethlehem, PA 18015

Problem Reduction for One-Dimensional Cutting and Packing Problems

Peter A. Huegler

Homer Research Laboratories
Bethlehem Steel Corporation
Bethlehem, PA 18016
USA

And

Industrial and Systems Engineering Dept.
Lehigh University
Bethlehem, PA 18015
USA

Joseph C. Hartman

Industrial and Systems Engineering Dept.
Lehigh University
Bethlehem, PA 18015
USA

1 Abstract

This paper presents a polynomial time reduction procedure for one-dimensional cutting and packing problems. This reduction procedure is based upon separating the original problem into two sub problems whose optimal solutions can be combined into an optimal solution to the original problems. This reduction procedure is compared with the reduction method of Martello and Toth [13], [14]. A second reduction method is presented that combines the separation-based method and Martello and Toth's method. The effect of the combined method on upper and lower bounds is analyzed. The application of the combined reduction procedure and a column generation based upper bound on 39,970 test problems found optimal solutions 94.55% of the time, a gap of one large object 5.21% of the time and a gap of two large objects .24% of the time.

2 Introduction

The one-dimensional bin packing problem (BPP) can be stated as follows:

> Given a positive integer C and a set $I = \{1,2,...,n\}$ of n items, each having a positive integer size s_i $(i = 1,...,n)$ satisfying $s_i \leq C$, the *bin packing problem* (BPP) is to find the smallest integer m such that there is a partition $B = \{B_1, B_2, ..., B_m\}$ of set I where the sum of the sizes in each B_j does not exceed C. Informally we can think of the problem of packing n items into a minimum number of identical bins of capacity C; each subset B_j is in this case the contents of the bin. [14]

The one-dimensional cutting stock problem (CSP) differs slightly from the BPP in terminology but is mathematically equivalent. Instead of packing n items into a minimum number of identical bins, the CSP cuts n items from a minimum number of inventory items.

Dyckhoff [6] and Dyckhoff and Finke [7] developed a classification system for cutting and packing problems. The system is as follows:

1. *Dimensionality*
   - (N) Number of dimensions
2. *Kind of assignment*
   - (B) All large objects and a selection of small items
   - (V) A selection of large objects and all small items
3. *Assortment of large objects*
   - (O) One object
   - (I) Identical figure
   - (D) Different figure
4. *Assortment of small items*
   - (F) Few items (of different figures)
   - (M) Many items of many different figures
   - (R) Many items of relatively few (non-congruent) figures
   - (C) Congruent figures

In the bin packing and cutting stock problems, large objects refer to the bins and inventory items and small items refer to the items to be packed or cut. Using this typology, the BPP is 1/V/I/M and the CPP is 1/V/I/R. The difference is in the assortment of small items. The BPP has many items of many different sizes and the CPP has many items of relatively few sizes.

There are many solution procedures for both the BPP and CPP. The following sections discuss the procedures analyzed within. For more information refer to Haessler and Sweeny [12], Cheng, Feiring, and Cheng [2], Coffman, Galambos, Martello, and Vigo [4] and Coffman, Garey and Johnson [5].

Section 3 presents the bounds used in the remaining sections of this paper. Section 4 discusses the separation-based reduction method and the combined Martello and Toth and separation-based method. Section 5 reviews the test problems analyzed within the analysis section. Section 6 is the analysis of the reduction methods. Conclusions are presented in section 7.

3 Problem Bounds

3.1 Lower Bounds

Martello and Toth [13], [14] developed one of the better lower bounds for the cutting and packing problem by partitioning the small items into three groups and computing the bound based on the sizes in each partition. By evaluating all of the possible partitions, the algorithm generates a high-quality lower bound. The lower bound is calculated without constructing a solution. This lower bound method is referenced at MTL2 in the analysis section. The algorithm is polynomial time running in $O(n)$ time.

Martello and Toth also present a second, improved lower bound in [13] and [14]. This lower bound is a combination of the MTL2 lower bound and the Matello and Toth reduction procedure. This lower bound method is referenced in the analysis section as MTL3. This algorithm is also polynomial time running in $O(n^3)$ time.

A third lower bound is based on the Gilmore and Gomory column generation (CG) algorithm presented in [10] and [11]. Vanderbeck proves in [18] that the CG

solution of the CSP rounded up to the next highest integer is a lower bound and dominates the lower bounds due to Martello and Toth [13], [14].

3.2 Upper Bounds

The any fit decreasing (AFD) algorithms are heuristic methods to solve the bin packing and cutting stock problems. The algorithms differ in the rule used to assign items to a particular bin. There are three common fit rules for the AFD algorithm. The best fit decreasing (BFD) rule assigns the item to the bin that leaves the most remaining bin capacity. The worst fit decreasing (WFD) rule assigns the item to the bin that leaves the least remaining bin capacity. The first fit decreasing (FFD) algorithm assigns the item to the first bin in which the item fits. These algorithms are polynomial time algorithms.

The modified subset sum greedy (MSSG) heuristic is based on the subset sum problem. This heuristic solves a series of subset sum problems to generate packing patterns. The outline of the algorithm is as follows:

1. Apply the small items larger than one-half the large object size to separate large objects.
2. Starting with the large object with the least remaining capacity, find the largest subset of the remaining small items that fits within the remaining capacity and assign the small items to that large object.
3. Repeat step 2 until all partially filled large objects are filled or no small items remain. Quit if no small items remain.
4. Taking an unused large object, find the subset of the remaining small items that fills the large object and assign the small items to that large object.
5. Repeat step 4 until no more small items remain.

Two different exact methods are used to solve the subset sum sub problems. First, the problem is solved using an enumeration algorithm similar to the algorithm presented on page 201 in Chvátal [3]. Secondly, the problem is solved using Pisinger's minknap [15]

algorithm to solve the sub problems. These methods are referred to as MSSGEnum and MSSGMinKnap respectively. These algorithms execute fast in practice although the subset sum problem is NP-Hard.

The column generation residual (CGRes) algorithm is based on Gilmore and Gomory's column generation algorithm [10], [11]. The outline of the algorithm is as follows:

1. Solve the LP relaxation of the CSP using column generation.
2. For each pattern of small items identified in the column generation solution, apply the pattern to $\lfloor x_i \rfloor$ large objects.
3. Using only the remaining small items, solve the problem using
   a. FFD
   b. BFD
   c. WFD
   d. MSSGEnum
   e. MSSGMinKnap
4. Use the best solution from step 3 to complete the problem.

This algorithm incorporates upper bound methods that solve the subset sum problem. Therefore, this algorithm is not polynomial time, but in practice, executes sufficiently fast.

4 Problem Reductions

4.1 Martello and Toth Reduction

Martello and Toth describe a method for reducing BPP and CSP problems in [13] and [14]. This method identifies dominant cutting or packing patterns, which are removed and thereby reduce the original problem. The initial reduction method presented is exponentially inefficient because all the possible packing patterns must be generated and compared. Martello and Toth present a modified reduction method that

limits the cardinality of the packing patterns to three or less small items. The modified method is a polynomial time algorithm.

4.2 Separation-Based Reductions

Our separation-based reduction method attempts to reduce the problem size by identifying separable sub problems whose optimal solutions combine to be an optimal solution for the original problem. It is based on the following lemma.

Consider a BPP with items $I = \{1,2,...,n\}$, sizes $s_i \ (i = 1,...,n)$, and a bin capacity of C. Select a size s, such that $\frac{1}{2}C < s \leq C$. Construct a partition $I = \{I_1, I_2\}$. Where I_1 contains all items in I such that $s_i \geq s$ or $s_i \leq C - s$. Let k be the number of items in I_1 with $s_i > \frac{1}{2}C$. Let I_2 contain the remaining items, or the set $I \setminus I_1$, with $C - s < s_i < s$.

Lemma 1: If there exists an optimal solution to the partition I_1 with $Z^*_{I_1} = k$, then the optimal solution to the master problem is $Z^* = k + Z^*_{I_2}$.

Proof: Assume there exists an optimal solution to the partition I_1 with $Z^*_{I_1} = k$. By construction, no items from I_2 can be moved to I_1 without creating a new bin in the I_1 optimal solution. This is because each pattern in I_1 contains a small item with a size $s_i \geq s$ and all items in I_2 have a size $C - s > s_i > s$. Further, the smaller items ($s_i \leq C - s$) in I_1 can be moved to I_2 but their removal will not improve upon the optimal solution to I_1 (as the number of small items greater than $\frac{1}{2}C$ is equal to k) and at best, the solution to I_2 will remain the same and at worst increase the solution by one bin per small item moved. Similarly, moving one larger item from I_1 to I_2 will create a new bin in the

optimal solution to I_2 and will reduce the optimal solution to I_1 by at most one bin with possibly no reduction at all. Therefore, there is no swapping of items between I_1 and I_2 that can improve the optimal solutions to I_1 and I_2 and therefore, the optimal solution to the master problem can be expressed by $Z^* = k + Z^*_{I_2}$. □

The difficultly with applying the result of this lemma is that it requires an optimal solution to BPP, which is NP-hard. However, the above lemma can be used to examine an existing solution of any BPP to determine if the problem is reducible. Consider a feasible solution, P, to the BPP. Select an s, such that $\frac{1}{2}C < s \leq C$. Partition the packing patterns into two groups, P_1 and P_2, where P_1 contains all of the patterns that contain an item with size greater than or equal to s. Let I_1 be the items that are included in the patterns in P_1 and let I_2 be the items that are included in the patterns in P_2. Let S_I be the set of items in I such that $s_i < C - s$. Let S_{I_1} be the set of items in I_1 such that $s_i < C - s$. If $S_{I_1} = S_I$ then the partition $I = \{I_1, I_2\}$ has the properties required by Lemma 1. The patterns in P_1 are an optimal solution to I_1 because of the construction of P_1. Therefore, using Lemma 1, the problem is separable and the individual sub problems can be solved independently. And since P_1 is an optimal solution to I_1, only I_2 is considered further to optimize the entire problem with I.

Item Number	Size
a	9
b	9
c	8
d	8
e	7
f	4
g	3
h	3
i	2
j	1

Table 1 – Example Problem

Pattern Number	Pattern
1	9-1
2	9
3	8-2
4	8
5	7-3
6	4-3

Table 2 – Solution to Example Problem

As an example consider the solution in Table 2 to the example problem in Table 1. First, let $s = 7$. P_I contains the patterns 1, 2, 3, 4, and 5. $I_1 = \{9,9,8,8,7,3,2,1\}$, $I_2 = \{4,3\}$, $S_{I_1} = \{3,2,1\}$, and $S_I = \{3,3,2,1\}$. Since $S_{I_1} \neq S_I$, the partition $I = \{I_1, I_2\}$ does not have the same properties required for Lemma 1. Therefore, it cannot be proven that this problem can be reduced using this partition.

Now consider $s = 8$. P_I contains the patterns 1, 2, 3, and 4. $I_1 = \{9,9,8,8,2,1\}$, $I_2 = \{7,4,3,3\}$, $S_{I_1} = \{2,1\}$, and $S_I = \{2,1\}$. Since $S_{I_1} = S_I$, the partition $I = \{I_1, I_2\}$ has the properties required for Lemma 1. The patterns in P_I are an optimal solution to I_I as $Z^*_{I_1} = k$. Using Lemma 1, the problem can be separated using this partition. Now, the problem can be reduced and only I_2 needs to be optimally solved.

The above example suggests the following algorithm for calculating the smallest s for which Lemma 1 identifies a separable problem.

1. Create two sets of items.
   a. Let I_1 be the set of items where $s_i \leq \frac{1}{2}C$.
   b. Let I_2 be the set of items where $s_i \leq \frac{1}{2}C$ and item i is in a pattern with another item i_1 where $s_{i_1} > \frac{1}{2}C$.
2. Sort the patterns in descending order based on the largest small item in each pattern.
3. Find the pattern in the list with the smallest small item with a size $s_i > \frac{1}{2}C$.
4. Set s to the size of the largest small item in the current pattern.
5. Remove the items from set I_1 with a size $s_i > C - s$.
6. If $I_1 = I_2$ then stop.
7. Remove the small items in the current pattern from I_2.
8. Move the previous pattern in the sorted pattern list.
9. Go to step 4.

Step 1 can be executed in $O(n)$ time. Each item in each pattern must be examined and there are n items. Step 2 can be executed in $O(n\log n)$ time. At the worst case, there will be n patterns in the list. Therefore, step 3 will execute in $O(n)$ time. At most, there will be n items to be removed and step 5 will also execute in $O(n)$ time. For the same reason, step 7 will execute in $O(n)$ time. The loop from step 4 to step 9 will execute at most n times. Therefore, the whole algorithm will execute in $O(n^2)$ time.

4.3 Hybrid Reductions

The Martello and Toth and Problem Separation methods can be combined into a hybrid reduction method. First, methods are selected for the hybrid algorithm (e.g. Martello and Toth, Problem Separation using the FFD solution, Problem Separation using the MSSG solution). The basic outline for the hybrid method is as follows:

1. Select the reduction method
2. Execute the current method.
3. Reduce the problem using the results.
4. If the problem is completely reduced, stop.
5. Execute subset sum to determine a new capacity.
6. Repeat steps 2 through 5 for each selected method.
7. Repeat steps 1 through 6 until no methods reduce the problem.

Four hybrid methods are examined. The hybrid methods are selected based on the analysis in section 6. Table 3 lists the hybrid methods.

Hybrid Method	Selected Reduction Methods
Hybrid FFD	Martello and Toth Problem Separation using FFD solution
Hybrid MSSGEnum	Martello and Toth Problem Separation using MSSGEnum solution.
Hybrid MSSGMinKnap	Martello and Toth Problem Separation using MSSGMinKnap solution.
Hybrid ALL	Martello and Toth Problem Separation using FFD solution Problem Separation using MSSGEnum solution Problem Separation using MSSGMinKnap solution

Table 3 – Hybrid Reduction Methods

The Hybrid methods are not polynomial time algorithms because step 5 solves the subset sum problem. Although this problem is NP-Hard, in practice, solution times are fast. In practice, the execution times of the Hybrid methods are also fast.

5 Test Problems

Seven problems sets are used as test problems. Problem generators generate two of the data sets. BPPGEN [17] is a generator used to create 35,000 bin packing problem instances. The source code for BPPGEN can be obtained from http://prodlog.wiwi.uni-halle.de/sicup/non_pub/research-support/bppgen.html. CUTGEN1 [9] is a problem generator for the one-dimensional cutting stock problem. It is very similar to BPPGEN.

CUTGEN1 is used to generate 3,600 problem instances. The source code for CUTGEN1 can be obtained at http://prodlog.wiwi.uni-halle.de/sicup/non_pub/research-support/cutgen1.html.

The remaining problem sets are not from problem generators. The BISON [16] problem instances consist of three sets of problems. These problems were used to test the BISON algorithm against Martello and Toth's MTP [13] algorithm. The first problem set, referred to as BisonDS1 contains 720 problem instances. BisonDS2, the second problem set contains 480 instances. The final data set is referred to as BisonHard. This set contains 10 instances. The total number of instances for all three BISON data sets is 1,210. The BISON data sets can be obtained at http://www.bwl.tu-darmstadt.de/bwl3/forsch/projekte/binpp/. The remaining problem sets are taken from J. E. Beasley's OR-Library [1]. These problems originate from E. Faulkenauer [8] and were used to test a grouping genetic algorithm. The OR-Library contains two sets of problems. One set of instances contains uniformly generated item sizes and referred to as BeasleyUniform. The other set of instances contains specially generated triplets and is referred to as BeaslyTriple. Both BeasleyUniform and BeasleyTriple contain 80 instances. These test problems can be obtained at http://www.ms.ic.ac.uk/info.html.

6 Analysis

6.1 Implementation Notes

All of the problem generators, reductions and algorithms are programmed in FORTRAN, C, and C++. Original source code from the authors was used where available. The programs were compiled with Microsoft Visual Studio and DEC Visual

Fortran compilers. The programs were executed on a 900 MHz Pentium III Compaq Presario 1800T with 384 MB of RAM running Microsoft Windows 2000 Professional.

For each problem, a subset sum problem is solved to determine the minimum required large object capacity. If the solution returns a sum less than the current large object capacity, the large object capacity is replaced by the sum.

The column generation algorithm requires an initial set of patterns. Chvátal [3] (page 199) proposes using a pattern for each small item made up of $\left\lfloor \frac{s_i}{C} \right\rfloor$ of the small items. This implementation of the column generation algorithm adds to the Chvátal suggested patterns, the patterns generated by solving the BPP with FFD, BFD, WFD, MSSGEnum, and MSSGMinKnap. All duplicates are removed. The additional patterns significantly decrease the column generation execution time. The knapsack sub problem is solved using a $O(nC)$ dynamic programming based algorithm presented in [19].

6.2 Problem Reductions

Initially, six reduction methods are analyzed. The first method is the limited cardinality Martello and Toth (MT) reduction method. The remaining five are variations of the Problem Separation method using the following starting solutions; First Fit Decreasing (SepFFD), Best Fit Decreasing (SepBFD), Worst Fit Decreasing (SepWFD), Modified Subset Sum Greedy – Enumeration (SepMSSGEnum) and Modified Subset Sum Greedy – Minknap (SepMSSGMinKnap). The reduction procedures are performed on problems from six different test problem sets as described in section 5.

Generator	Problem Count	MT	SepFFD	SepBFD	SepWFD	Sep MSSG Enum	Sep MSSG MinKnap
BeasleyTriple	80						
BeasleyUniform	80	80					
BisonDS1	720	720	396	338	128	397	395
BisonDS2	480	36					
BisonHard	10						
BPPGEN	35,000	22,630	10,900	8,724	4,301	10,751	10,747
CUTGEN1	3,600	817	1,160	377	351	1,116	1,115

Table 4 – Reduction Count by Method – All Problems

Table 4 displays the number of problems reduced by each reduction method. The first column, "Generator", identifies the test problem set. The column, "Problem Count", is the number of problems in each set. The remaining columns are the number of problems reduced by the respective methods. In general, the MT method reduced more problems than the problem reduction methods, the only exception being the CUTGEN1 problem set. This set is the only cutting stock problem set with the remaining being bin packing problem sets.

Generator	Problem Count	MT	SepFFD	SepBFD	SepWFD	Sep MSSG Enum	Sep MSSG MinKnap
BeasleyTriple	80						
BeasleyUniform	80	49.64%					
BisonDS1	720	73.50%	65.66%	62.64%	72.98%	65.64%	65.37%
BisonDS2	480	14.99%					
BisonHard	10						
BPPGEN	35,000	60.15%	71.73%	72.49%	78.55%	71.44%	71.45%
CUTGEN1	3,600	20.81%	70.66%	90.57%	92.32%	69.70%	69.73%

Table 5 – Percent Reduction by Method – All Problems

Table 5 displays the percent reduction by each reduction method. The "Problem Count" column is again the number of problems in each problem set. The remaining columns display the percent reduction for each reduction method. The percent reduction is only calculated from reduced problems. As an example, consider the BisonDS1 results for the SepWFD reduction procedure. From Table 4, this method reduces 128 problems.

From these problems, SepWFD, on average, removes 72.98% of the small items. The problem separation based reductions tend towards a larger reduction percentages than the Martello and Toth reduction. SepWFD generally outperforms the other reductions but reduces few problems. SepFFD is the second best separation-based reduction method and it reduces more problems than the other separation-based methods.

Generator	Problem Count	Reduced Count	MT	SepFFD	SepBFD	SepWFD	Sep MSSG Enum	Sep MSSG MinKnap
BeasleyTriple	80							
BeasleyUniform	80	80	100.00%					
BisonDS1	720	720	92.50%	8.89%	4.86%	2.50%	8.89%	8.61%
BisonDS2	480	36	100.00%					
BisonHard	10							
BPPGEN	35,000	22,871	87.95%	16.27%	9.87%	7.23%	15.57%	15.54%
CUTGEN1	3,600	1,271	24.31%	77.58%	25.89%	26.36%	73.72%	73.64%

Table 6 – Percent Best Reduction by Method

Table 6 displays the percentage each reduction method finds the best reduction as compared to the other methods. The Martello and Toth method performs well on all problem sets but the CUTGEN1 problem set. The SepFFD, SepMSSGEnum and SepMSSGMinKnap perform the best on the CUTGEN1 problem set. SepBFD and SepWFD perform the worst.

Generator	Problem Count	Reduced Count	MT	SepFFD	SepBFD	SepWFD	Sep MSSG Enum	Sep MSSG MinKnap
BeasleyTriple	80							
BeasleyUniform	80	80	80					
BisonDS1	720	720	656					
BisonDS2	480	36	36					
BisonHard	10							
BPPGEN	35,000	22,871	19,134	78	1			1
CUTGEN1	3,600	1,271	281	32			1	

Table 7 – Strictly Dominant Reduction Count by Method

Table 7 displays the number of strictly dominant reductions for each method. A reduction is strictly dominant when the reduction removes more small items from a

problem than any other reduction method. The "Reduced Count" column displays the total number of problems reduced by any of the methods. The remaining columns display the number of problems where the reduction method generated strictly dominant reductions. Consider the BisonDS1 problem set. The six reduction methods reduce a total of 720 problems. The Martello and Toth method reductions of 656 problems strictly dominate. For the remaining 64 reduced problems, multiple reduction methods find the largest reduction.

A second method of improving the reduction results is to combine the different reduction methods into a meta-method. The Hybrid method (see section 4) is a structure that combines several different reduction methods. The following is an analysis on four Hybrid methods. Hybrid FFD combines the Martello and Toth and SepFFD methods. HybridMSSGEnum combines the Martello and Toth and SepMSSGEnum methods. HybridMSSGMinKnap combines the Martello and Toth and SepMSSGMinKnap methods. And finally, HybridALL combines Martello and Toth, SepFFD, SepMSSGEnum, and SepMSSGMinKnap methods. The four methods combined into the Hybrid methods are selected for different reasons. The Martello and Toth method is selected because of the large number of strictly dominant reductions. The separation methods are selected because of the high percent reduction.

Generator	Problem Count	Hybrid FFD	Hybrid MSSG Enum	Hybrid MSSG MinKnap	Hybrid ALL
BeasleyTriple	80				
BeasleyUniform	80	80	80	80	80
BisonDS1	720	720	720	720	720
BisonDS2	480	36	36	36	36
BisonHard	10				
BPPGEN	35,000	22,870	22,862	22,862	22,871
CUTGEN1	3,600	1,270	1,229	1,229	1,271

Table 8 – Reduction Count by Hybrid Method – All Problems

Table 8 displays the reduction counts for all of the Hybrid methods. There is no improvement in the results for both the BeasleyTriple and BisonHard problem sets. The BisonDS1 and BisonDS reduction counts are the same as the reduction counts for the Martello and Toth method by itself. The Hybrid methods improve number of reductions in the BPPGEN and CUTGEN1 problem sets.

Generator	Problem Count	Hybrid FFD	Hybrid MSSG Enum	Hybrid MSSG MinKnap	Hybrid ALL
BeasleyTriple	80				
BeasleyUniform	80	49.64%	49.64%	49.64%	49.64%
BisonDS1	720	76.45%	76.47%	76.37%	76.43%
BisonDS2	480	14.99%	14.99%	14.99%	14.99%
BisonHard	10				
BPPGEN	35,000	69.53%	68.85%	68.82%	69.45%
CUTGEN1	3,600	79.89%	78.63%	78.48%	80.11%

Table 9 – Percent Reduction by Hybrid Method – All Problems

Table 9 displays the Hybrid methods percent reductions for all problems. The hybrid methods do not improve the percent reductions for the BeasleyUniform and BisonDS2 problem sets. The methods do improve the reductions for the BisonDS1, BPPGEN and CUTGEN1 problem sets.

Generator	Problem Count	Reduction Count	Hybrid FFD	Hybrid MSSG Enum	Hybrid MSSG MinKnap	Hybrid ALL
BeasleyTriple	80					
BeasleyUniform	80	80	100.00%	100.00%	100.00%	100.00%
BisonDS1	720	720	99.86%	100.00%	99.72%	98.06%
BisonDS2	480	36	100.00%	100.00%	100.00%	100.00%
BisonHard	10					
BPPGEN	35,000	22,871	99.93%	99.23%	99.21%	97.99%
CUTGEN1	3,600	1,271	99.69%	95.83%	95.67%	99.53%

Table 10 – Percent Best Reduction by Hybrid Method

Table 10 displays the percentage each hybrid method finds the best reduction as compared to the other hybrid methods.

Generator	Problem Count	Reduced Count	MT	SepFFD	Sep MSSG Enum	Sep MSSG MinKnap	Hybrid ALL
BeasleyTriple	80						
BeasleyUniform	80	80	100.00%				100.00%
BisonDS1	720	720	90.00%	1.67%	1.67%	1.67%	100.00%
BisonDS2	480	36	100.00%				100.00%
BisonHard	10						
BPPGEN	35,000	22,871	87.50%	6.11%	6.03%	6.02%	99.92%
CUTGEN1	3,600	1,271	23.76%	14.32%	13.69%	13.69%	100.00%

Table 11 – Percent Best Reduction – Summary Comparison

Table 11 displays a comparison of the percent best reductions between the HybridALL reduction method and the component methods. The HybridALL method significantly improves the individual methods.

Generator	MT	SepFFD	SepBFD	SepWFD	SepMSSG Enum	SepMSSG MinKnap
BeasleyTriple						
BeasleyUniform	0.0053					
BisonDS1	0.0031	0.0034	0.0050	0.0051	0.0057	0.0651
BisonDS2	0.0028					
BisonHard						
BPPGEN	0.0046	0.0042	0.0047	0.0045	0.0065	0.0621
CUTGEN1	0.0078	0.0061	0.0070	0.0071	0.0140	0.2045

Table 12 – Normalized Reduction Times by Method

Generator	Hybrid FFD	Hybrid MSSG Enum	Hybrid MSSG MinKnap	Hybrid ALL
BeasleyTriple				
BeasleyUniform	0.0113	0.0221	0.0555	0.0383
BisonDS1	0.0081	0.0103	0.0340	0.0226
BisonDS2	0.0072	0.0331	0.1177	0.0778
BisonHard				
BPPGEN	0.0130	0.0154	0.0435	0.0275
CUTGEN1	0.0187	0.0294	0.2592	0.1037

Table 13 – Normalized Reduction Times by Hybrid Method

Table 12 and Table 13 display the normalized average execution times for each of the analyzed methods. All of the times are reported in seconds. The times are normalized because within each problem set, there are problems with differing number of

small items. The value displayed in each table is the average execution time per 1,000 small items. Also, the times are an average only of those problems a method reduced. For example, the SepFFD method reduced 396 problems for the BisonDS1 problem set (Table 4). The times for only those 396 problems are included in the calculation for the time shown in Table 12. The calculations are performed this way because of the way the Problem Separation reduction executes. This method first tests whether a problem is a candidate by looking at the number of big items. If this number is zero, the reduction is stopped. Problems with no big items take no time in this reduction method. Including these problems would skew the times. Therefore, the times are only reported on those problems that were reduced. As shown by the data, executing the reduction methods is trivial.

6.2.1 Lower Bounds

Six lower bound methods are analyzed in this section. Three are Martello and Toth L2 (MTL2), Martello and Toth L3 (MTL3) and Column Generation (ColGen). Each of the remaining three lower bound methods first reduce the problem using the HybridALL reduction method, then execute the lower bound method. These three are the Reduced Martello and Toth L2 (RedMTL2), Reduced Martello and Toth L3 (RedMTL3) and Reduced Column Generation (RedColGen). The following discusses the results.

Generator	Problem Count	MTL2 Best	MTL3 Best	ColGen Best	Red MTL2 Best	Red MTL3 Best	Red ColGen Best
BeasleyTriple	80	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
BeasleyUniform	80	98.75%	98.75%	100.00%	98.75%	98.75%	100.00%
BisonDS1	720	74.17%	90.00%	100.00%	91.25%	91.94%	100.00%
BisonDS2	480	95.21%	95.21%	100.00%	95.21%	95.21%	100.00%
BisonHard	10	30.00%	30.00%	100.00%	30.00%	30.00%	100.00%
BPPGEN	35,000	72.73%	89.71%	100.00%	90.98%	91.30%	100.00%
CUTGEN1	3,600	65.47%	70.08%	100.00%	77.56%	78.33%	100.00%

Table 14 – Percent Best by Lower Bound Methods

Table 14 displays the percent of the problems each of the six lower bound methods found the best lower bound. There is no improvement for the BeasleyTriple and BisonHard problem sets because the HybridALL reduction method does not reduce any of these problem instances. The column generation methods dominate the remaining methods. ColGen and RedColGen always find the best lower bounds.

Reducing the problem prior to the executing the ColGen method does not improve any of the lower bound calculations. This is explained by the fact that the HybridALL reduction removes small items by identifying packing patterns and the ColGen lower bound procedure is based upon packing patterns. In effect, the reduction procedure identifies packing patterns that are included within an optimal column generation solution along with a minimum usage for those patterns. This will not improve the lower bounds calculated using the ColGen method. However, the reduction should improve the execution time of the ColGen lower bound method.

Generator	Problem Count	Improved Count	Average Improvement	Minimum Improvement	Maximum Improvement
BeasleyTriple	80				
BeasleyUniform	80	66	5.61	1	21
BisonDS1	720	634	3.58	1	35
BisonDS2	480	27	21.59	6	55
BisonHard	10				
BPPGEN	35,000	20,345	6.09	1	110
CUTGEN1	3,600	872	1.15	1	9

Table 15 – ColGen Reduction Improved Iteration Comparison

Generator	Problem Count	Deteriorated Count	Average Deterioration	Minimum Deterioration	Maximum Deterioration
BeasleyTriple	80				
BeasleyUniform	80	8	(1.88)	(1)	(4)
BisonDS1	720	36	(2.61)	(1)	(9)
BisonDS2	480	8	(3.63)	(1)	(7)
BisonHard	10				
BPPGEN	35,000	1,127	(6.43)	(1)	(39)
CUTGEN1	3,600	74	(2.69)	(1)	(12)

Table 16 – ColGen Reduction Deteriorated Iteration Comparison

Table 15 and Table 16 display the effects of the reduction on the ColGen lower bound method. Table 15 contains statistics for the problems whose iteration count was improved by the reduction. Table 16 contains statistics for the problems whose iteration count was made worse by the reduction. Significantly more problems had improvements (54%) than did not (3%). Reducing the number of iterations reduces the execution time of the ColGen method.

Generator	MTL2	MTL3	ColGen	Red MTL2	Red MTL3	Red ColGen
BeasleyTriple	0.00104	0.63676	4.79562	0.01363	0.64269	4.97245
BeasleyUniform	0.00075	0.06480	0.87224	0.03842	0.14667	0.53833
BisonDS1	0.00157	0.01376	0.46487	0.02336	0.03936	0.23730
BisonDS2	0.00058	0.28682	5.20823	0.02321	0.31295	5.03325
BisonHard	-	0.09550	757.18900	0.47700	0.56700	758.15450
BPPGEN	0.00102	0.03713	4.44067	0.02220	0.05840	4.08619
CUTGEN1	0.00076	1.10868	1.52208	0.11091	1.11912	1.61980

Table 17 – Normalized Lower Bound Times by Method with Reduction Time

Table 17 displays the average normalized execution times for each lower bound method. All the times are reported in seconds. The times are normalized to a 1,000 small item problem because each problem set contains problems of different sizes. There is very little time penalty for reducing a problem before calculating the lower bound and in many instances, the reduction improves the execution time.

6.2.2 Upper Bounds

Six upper bound methods are analyzed on non-reduced problems and problems reduced with the HybridALL for a total of twelve upper bound methods. The six methods executed on non-reduced problems are First-Fit Decreasing (FFD), Best-Fit Decreasing (BFD), Worst-Fit Decreasing (WFD), Modified Subset Sum Greedy Enumeration (MSSGEnum), Modified Subset Sum Greedy Pisinger minknap (MSSGMinKnap) and Column Generation Residual (ColGenRes). The same six

methods are executed on reduced problems. The names of these six methods are RedFFD, RedBFD, RedWFD, RedMSSGEnum, RedMSSGMinKnap and RedColGenRes.

Generator	Problem Count	FFD	BFD	WFD	MSSG Enum	MSSG MinKnap	ColGen Res
BeasleyTriple	80				61.25%	83.75%	100.00%
BeasleyUniform	80	7.50%	7.50%	2.50%	56.25%	21.25%	91.25%
BisonDS1	720	76.67%	76.81%	61.81%	89.44%	76.67%	99.31%
BisonDS2	480	52.08%	52.08%	46.67%	88.13%	73.54%	99.58%
BisonHard	10					90.00%	100.00%
BPPGEN	35,000	87.16%	87.26%	76.15%	92.04%	82.26%	99.35%
CUTGEN1	3,600	52.86%	53.25%	44.67%	55.22%	36.64%	99.83%

Table 18 – Percent Best by Upper Bound Method

Generator	Problem Count	Red FFD	Red BFD	Red WFD	Red MSSG Enum	Red MSSG MinKnap	Red ColGen Res
BeasleyTriple	80				61.25%	83.75%	100.00%
BeasleyUniform	80	7.50%	7.50%	2.50%	56.25%	60.00%	97.50%
BisonDS1	720	76.81%	76.94%	69.58%	89.44%	84.86%	99.44%
BisonDS2	480	52.08%	52.08%	46.88%	88.13%	74.58%	99.38%
BisonHard	10					90.00%	100.00%
BPPGEN	35,000	87.40%	87.50%	78.53%	92.30%	82.68%	99.29%
CUTGEN1	3,600	52.86%	53.25%	46.97%	56.39%	37.81%	99.75%

Table 19 – Percent Best by Reduced Upper Bound Method

Table 18 and Table 19 display the percent of the time that each upper bound method calculates the best upper bound. Table 18 contains the best percentages for the non-reduced upper bound methods and Table 19 contains the percentages for the reduced upper bound methods. As with the lower bound methods, there is no improvement for the instances in the BeasleyTriple and BisonHard problem sets because these instances are not reduced by the HybridALL reduction method. The ColGenRes and RedColGenRes find the best upper bound more often than the other methods.

6.2.3 Gap Analysis

This section analyzes the gap (inventory items or bins) between the six reduced upper bound methods and the RedColGen lower bound, as it dominates the other lower bound methods.

Generator	Problem Count	Red FFD	Red BFD	Red WFD	Red MSSG Enum	Red MSSG MinKnap	Red ColGen Res
BeasleyTriple	80						13
BeasleyUniform	80						19
BisonDS1	720						36
BisonDS2	480					2	20
BisonHard	10						1
BPPGEN	35,000		1		9	75	416
CUTGEN1	3,600						1,294

Table 20 – Strictly Dominant Gaps

Table 20 displays the number of strictly dominant solutions from the reduced upper bound methods. An upper bound strictly dominates all other upper bounds when the upper bound is less than all of the other upper bounds. The RedColGenRes upper bound method finds the best upper bound in all but 87 instances. These results suggest a new upper bound method that takes the best upper bound from the RedBFD, RedMSSGEnum, RedMSSGMinKnap and theRedColGenRes methods. This upper bound method is referred to as RedBest. Table 21 displays the gap analysis of the RedBest upper bound method.

Generator	Problem Count	Percent Optimal	Average Gap	Minimum Gap	Maximum Gap
BeasleyTriple	80	13.75%	1.00	1	1
BeasleyUniform	80	93.75%	1.00	1	1
BisonDS1	720	97.64%	1.00	1	1
BisonDS2	480	90.00%	1.08	1	2
BisonHard	10	90.00%	1.00	1	1
BPPGEN	35,000	95.08%	1.04	1	2
CUTGEN1	3,600	93.58%	1.00	1	1

Table 21 – RedBest Gap Analysis

Generator	RedBest Gap	Problem Count
BeasleyTriple	Optimal	11
BeasleyTriple	1	69
BeasleyTriple	2	-
BeasleyUniform	Optimal	75
BeasleyUniform	1	5
BeasleyUniform	2	-
BisonDS1	Optimal	703
BisonDS1	1	17
BisonDS1	2	-
BisonDS2	Optimal	430
BisonDS2	1	46
BisonDS2	2	4
BisonHard	Optimal	9
BisonHard	1	1
BisonHard	2	-
BPPGEN	Optimal	33,196
BPPGEN	1	1,712
BPPGEN	2	92
CUTGEN1	Optimal	3,369
CUTGEN1	1	231
CUTGEN1	2	-

Table 22 – RedBest Gaps by Gap Amount

Table 22 displays a breakdown of the number of problems by the gap calculated using the RedBest upper bound method. The only problems with a gap of two are in the BisonDS2 and BPPGEN problem sets.

7 Conclusions

In this paper, two new reduction procedures are presented. The first reduction, a polynomial time algorithm, is based on identifying separations of a BPP or CSP problem that can be optimally solved and combined back into an optimal solution of the original problem. Sufficient conditions are identified for when this reduction is valid. A polynomial time algorithm is presented that uses the conditions to identify a separation from any given solution. Five separation-based reduction procedures (using different initial solutions) are compared to the Martello and Toth reduction procedure which is

based on pattern dominance. The second reduction procedure is a combination of several separation-based reduction procedures and the Martello and Toth reduction procedure. This procedure, although not a polynomial time algorithm, executes fast in practice. The hybrid method outperforms the individual reduction procedures.

The effects of the hybrid reduction procedure on lower and upper bounds are examined. It is shown that the reduction improves the Martello and Toth L2 and L3 lower bounds [13], [14]. It does not improve the results of the lower bound based on Gilmore and Gomory's column generation procedure [10], [11]. The reduction, in general, does improve the execution time of this lower bound procedure.

The reduction also improves upon upper bound procedures. The upper bound procedures include three Any-Fit Decreasing procedures, First-Fit Decreasing, Best-Fit Decreasing and Worst-Fit Decreasing. Two other upper bound procedures are based on solving a series of subset sum problems, each using a different algorithm to solve the subset sum problem. The final upper bound is based on the column generation technique, removing the integer portion of the solution, and using other upper bound methods to solve the residual problem. It is shown that the hybrid reduction improves the solutions generated using these upper bound methods. The column generation based upper bound performed the best. An upper bound method that uses the best bound from all of the methods is also examined.

Finally, an analysis of the gap between the column generation based lower bound method and upper bound methods is examined. It was found that the maximum gap for all of the problems using the best upper bound is two large objects. The best upper bound, when executed on 39,970 test problems, found optimal solutions 94.55% of the

time, a gap of one large object 5.21% of the time and a gap of two large objects .24% of the time.

This research was sponsored, in part, by NSF Grant DMI-9984891.

8 References

[1] J. E. Beasley, OR-library: distributing test problems by electronic mail, Journal of the Operational Research Society 41 (1990) 1069-1072.

[2] C.H. Cheng, B.R. Feiring, T.C.E. Cheng, The cutting stock problem – A survey, International Journal of Production Economics 56 (1994) 291-305.

[3] V. Chvátal, Linear Programming (Freeman, New York, 1983).

[4] E.G. Coffman Jr., G. Galambos, S. Martello, D. Vigo, Bin packing approximation algorithms: combinatorial analysis, in: D.-Z. Du, P.M. Pardalos, eds., Handbook of Combinatorial Optimization (Kluwer Academic Publishers, 1998).

[5] E.G. Coffman Jr., M.R. Garey, D.S. Johnson, Approximation algorithms for bin packing: a survey, in: D. Hochbaum, ed., Approximation Algorithms for NP-Hard Problems (PWS publishing, Boston, 1996) 46-93.

[6] H. Dyckhoff, A typology of cutting and packing problems, European Journal of Operational Research 44 (1990) 145-159.

[7] H. Dyckhoff, U. Finke, Cutting and Packing in Production and Distribution (Phusica-Verlag, Heidelberg, 1992).

[8] E. Faulkenauer, A hybrid grouping genetic algorithm for bin packing, Working paper CP 106-P4, CRIF – Research Center for Belgian Metalworking Industry (1994).

[9] T. Gau, G. Wäscher, CUTGEN1: a problem generator for the standard one-dimensional cutting stock problem, European Journal of Operational Research 85 (1995) 572-579.

[10] P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting stock problem, Operations Research 9 (1961) 948-959.

[11] P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting stock problem – part II, Operations Research 11 (1963) 863-888.

[12] R.W. Haessler, P.E. Sweeny, Cutting stock problems and solution procedures, European Journal of Operational Research 54 (1991) 141-150.

[13] S. Martello, P. Toth, Knapsack Problems (Chichester, 1990).

[14] S. Martello, P. Toth, Lower bounds and reduction procedures for the bin packing problem, Discrete Applied Mathematics 28 (1990) 59-70.

[15] D. Pisinger, A minimal algorithm for the 0-1 knapsack problem, Operations Research 45 (1997) 758-767.

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

[17] P. Schwerin, G. Wäscher, The bin packing problem: a problem generator and some numerical experiments with FFD and MTP, International Transactions in Operational Research 4 (1997) 377-389.

[18] F. Vanderbeck, Computational study of a column generation algorithm for bin packing and cutting stock problems, Mathematical Programming, Series A, 86 (1999) 565-594.
[19] L. Wolsey, Integer Programming (John Wiley & Sons, Inc., New York, 1998).