RC 22013 (W0104-007) 4 April 2001 Computer Science

IBM Research Report

Column Generation Approach to the Multiple Knapsack Problem with Color Contraints

Laszlo Ladanyi, John J. Forrest, Jayant Kalagnanam
IBM Research Division
Thomas J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

Column Generation Approach to the Multiple Knapsack Problem with Color Constraints

Laszlo Ladanyi • John J. H. Forrest • Jayant Kalagnanam

IBM Research Division, Thomas J. Watson Research Center, P.O.Box 218, Yorktown Heights, NY 10598

jjforre@us.ibm.com • jayant@us.ibm.com • ladanyi@us.ibm.com

In this paper, we study a new problem which we refer to as the *multiple knapsack with color constraints (MKCP)*. Motivated by a real application from the steel industry, the MKCP can be formulated by generalizing the multiple knapsack problem along two directions: (i) adding assignment restrictions on items which can be assigned to a knapsack, (ii) adding a new attribute (called "color" in the paper) to an item and then adding the associated "color" constraints which restrict the number of distinct colors which can be assigned to a knapsack to two. These generalizations make the problem computationally difficult to solve to optimality in practice as well as in theory. A real life instance (called *mkc*) of this problem class is available through MIPLIB and a larger instance (*mkc7*) is downloadable from the IBM Deep Computing Institute [12].

The focus of this paper is to present computational results for the two mentioned instances of this problem. We present the original (natural) formulation for the problem then reformulate it and use column generation to solve it. The column generation problem is modeled and solved as a knapsack problem with color constraints. We solve *mkc* to optimality and use Dantzig-Wolfe decomposition for lower bounding the other instance. Solving *mkc* to optimality took less time than it takes to solve the LP relaxation of the original formulation. The larger instance is solved to near-optimality (within .7% of optimality) in a fraction of the time required to solve the original relaxed LP.

(*Integer Programming; Column Generation; Inventory-Production Applications*)

1. Introduction

In this paper we introduce a multiple knapsack problem with color constraints (MKCP). Conventional multiple knapsack problem considers a set of n items which need to be packed into m knapsacks to maximize profit. We generalize this conventional problem by associating a color attribute with each item and restricting the number of colors in any knapsack. Although the number of allowable colors in any knapsack can in general be any K, the instances studied arose from an application allowing no more than two colors. Additional

assignment restrictions are specified in terms of a subset of knapsacks which can hold a given item. While the knapsack problem and the multiple knapsack problem are theoretically hard to solve in practice they (especially the knapsack problem) tend to be relatively easy to optimize. The generalizations we have introduced, however, make the MKCP difficult to solve to optimality in practice as well as in theory. In this paper we first introduce the original, "natural" Integer Programming formulation to the MKCP then provide an alternate formulation that is very well suited for a pricing (column generation) based algorithm. We show that the new formulation is much more efficient in quickly solving large real instances to near-optimality. In fact, the *mkc* instance is solved to optimality.

The multiple knapsack problem with color constraints is motivated by the *surplus inventory matching* problem in the steel industry [6]. Production planning begins with an order book which contains a list of orders that need to be satisfied. Before planning production, an attempt is made to satisfy orders using leftover slabs from surplus inventory. The problem of applying orders against an existing surplus inventory is the surplus inventory matching problem and can be formulated as a multiple knapsack problem with color and assignment constraints. The goal of inventory matching is to maximize the total weight of the order book that is applied against existing inventory and to minimize the leftover weight of each used slab. In the following paragraphs we explain the background for the assignment and color constraints starting with the assignment constraints.

For each order in the order book we can identify a set of applicable slabs from the surplus inventory. These assignment restrictions are based on quality and physical dimension considerations. For any given order only slabs which are of the same quality or better can be applied. In addition, the thickness and width requirements for each order need to be compatible with those of the slab applicable. These considerations restrict the number of applicable slabs for each order.

The color constraints place restrictions on the sets of orders that can be matched to the same slab in the surplus inventory. Because of processing considerations in the finishing line of a steel mill not all orders assignable to a slab can be packed together on that slab. There is a route associated with each order that specifies the set of process operations that need to be applied in the finishing mill. Orders with different routes require different process operations and are referred to as being of different types. Slabs packed with different order types need to be cut before they are processed in the finishing mill. Since cutting slabs is expensive and often the cutting machine is a bottleneck, strong constraints are posed in terms of the number of allowed cuts per slab. The simplest and most commonly used constraint used is to limit the number of required cuts to one; i.e., no more that two order types are allowed on a slab. In order to describe this constraint formally we associate a unique *color* with each route code and restrict the number of colors on a slab to be no more than two. This restricts the number of different order types on a slab to two and the

Table 1: List of notations I.

N	: The set of orders.
M	: The set of slabs.
n	: $\|N\|$, the number of orders.
m	: $\|M\|$, the number of slabs.
i	: running index on the orders.
j	: running index on the slabs.
M^i	: Set of slabs incident to order i.
N_j	: Set of orders incident to slab j.
w^i	: Weight of order i.
W_j	: Weight of slab j.
C_j	: Set of colors incident on slab j.
c^i	: The color of order i.
x_j^i	: 1 if order i is assigned to slab j; 0 otherwise.
y_j^c	: 1 if orders of color c obtain material from slab j; 0 otherwise.
z_j	: 1 if any order is incident to slab j; 0 otherwise.

number of required cuts to be no more than one.

The rest of the paper is organized as follows. In the following section a natural formulation is given for MKCP. Section 3 describes a reformulation of MKCP. Due to the number of columns in the new formulation a pricing based algorithm is proposed. Results based on this algorithm are presented. Section 4 presents a discussion of the approach and computational results and provides directions for future research.

2. Natural formulation for MKCP

In this section, we provide a formulation for MKCP. We say that a slab and an order are *incident* to each other if the order can be manufactured from the slab. We will use the following notation.

Using this notation there are three sets of variables and four sets of constraints in the model:

$$\max \sum_{i=1}^{n} \sum_{j \in M^i} w^i x_j^i - \sum_{j=1}^{m} (W_j - \sum_{i \in M^i} w^i x_j^i) z_j$$

$$\sum_{i \in N_j} w^i x_j^i \leq W_j z_j \qquad 1 \leq j \leq m \tag{1}$$

$$\sum_{j \in M^i} x_j^i \leq 1 \qquad 1 \leq i \leq n \tag{2}$$

$$\sum_{c \in C_j} y_j^c \leq 2 \qquad 1 \leq j \leq m \tag{3}$$

$$x_j^i \leq y_j^{c^i} \qquad 1 \leq i \leq n,\ j \in M^i \tag{4}$$

$$x_j^i \in \{0,1\} \qquad 1 \leq i \leq n,\ j \in M^i$$

$$y_j^c \in \{0,1\} \qquad \forall c \in C_j,\ 1 \leq j \leq m$$

$$z_j \in \{0,1\} \qquad 1 \leq j \leq m$$

The total number of variables in this formulation is

$$\sum_{i=1}^{n} |M^i| + \sum_{j=1}^{m} |C_j| + m \quad = \quad \sum_{j=1}^{m} |N_j| + \sum_{j=1}^{m} |C_j| + m$$

while the total number of constraints is $\sum_{i=1}^{n} |M^i| + 2m + n$.

Constraints (1) specify that if a slab is used then the total weight of the orders assigned to the slab cannot exceed the weight of the slab; Constraint (2) describes that each order will be made at most once; while constraints (3) and (4) enforce the coloring restriction.

Notice that the objective function is non-linear. However, since $z_j = 0$ forces x_j^i to be zero for all $i \in N_j$ and $z_j = 1$ implies $x_j^i z_j = x_j^i$, for all feasible solutions the objective function is equivalent to

$$\sum_{i=1}^{n} \sum_{j \in M^i} w^i x_j^i - \sum_{i=1}^{n} (W_j z_j - \sum_{j \in M^i} w^i x_j^i) = \sum_{i=1}^{n} \sum_{j \in M^i} 2 w^i x_j^i - \sum_{i=1}^{n} W_j z_j$$

.

The final observation is that the objective function just combines the two stated goals (maximizing satisfied orders and minimizing wasted parts of slabs) with equal weights. This may or may not be the best composite objective, but this is how the creator of the application specified the problem. Also, all that a different composite weight would change is the multiplier 2 for w^i (the coefficient of x_j^i) and the multiplier 1 for W_j (the coefficient of z_j); nothing in the proposed algorithms would need to be changed.

Kalagnanam et. al [6] have studied the polyhedral structure of this problem and have designed a heuristic procedure to find feasible solutions. However, they were not able to solve even the smaller problem to optimality and there was a considerable gap between their lower bound and the best solution they found. In the forthcoming sections we describe a different formulation that helped us to solve *mkc* to optimality and gave us a much better upper bound and gap for *mkc7*. In Section 4 a comparison is given between their and our results.

3. A formulation suitable for column generation

In this section we propose a formulation for the MKCP that is well suited to be solved via column generation. This new formulation has significantly more columns than the original formulation, on the other hand it results in a well studied problem, the set packing problem [8].

There are two types of constraints in this formulation. The first type corresponds to the slabs in the problem, the second type to the orders. The variables represent feasible production patterns, that is, variable u_k has a 1 in the row corresponding to the slab the production pattern is to be made of and 1's in the rows corresponding to the orders in the production pattern. Each variable is a binary variable indicating whether that production pattern is chosen in the solution or not. Let us introduce the following notation:

Table 2: List of notations II.

P	: the set of feasible production patterns
P_j	: the set of set of feasible patterns manufacturable from slab j
P^i	: the set of set of feasible patterns containing order i
R_k	: the constraints corresponding to the slab the production pattern corresponding to u_k is made of
R^k	: the constraints corresponding to the orders in the production pattern corresponding to u_k

Let the cost of variable u_k be $\bar{c}_k = \sum_{i \in R^k} 2w^i - W_{R_k}$ and create the following set packing problem:

$$\max \sum_{k \in P} \bar{c}_k u_k$$

$$\sum_{k \in P^i} u_k \leq 1 \qquad \forall 1 \leq i \leq n \tag{5}$$

$$\sum_{k \in P_j} u_k \leq 1 \qquad \forall 1 \leq j \leq m \tag{6}$$

$$u_k \in \{0,1\} \qquad \forall k \in P$$

It is very easy to see that there is a one to one correspondence between the feasible solutions of this set packing problem and the feasible solutions of the original formulation. Moreover, the construction of the $\bar{c}$ cost vector ensures that the corresponding solutions have identical objective values. Therefore optimizing this problem is the same as optimizing the original formulation.

The obvious problem with this formulation is that the number of feasible production patterns is enormous. However, as we will see, this disadvantage is more than offset by the advantages of the formulation.

The first obvious advantage is that the capacity constraints and the color constraints are gone. They were needed in the original formulation to keep the production patterns feasible, but here we have only feasible production patterns in the formulation to start with. The second advantage is that, although the number of feasible patterns is huge, the number of patterns that are at nonzero level in any IP feasible solution, or in the solution to the LP relaxation of the formulation is very small. This property is exploited to solve the LP relaxation. We start off with a small number of patterns (actually, we can start off with none), solve the LP relaxation restricted to this set of variables, then generate new variables with positive reduced cost and add them to the formulation. If no column with positive reduced cost can be generated then we have reached an optimal solution to the LP relaxation of the full problem.

3.1 Generating columns with positive reduced costs

To improve the solution evenly, for each slab we generate a production pattern whose corresponding column has the highest reduced cost, i.e., the most positive if there is one with positive reduced cost. Finding each of these columns is again an optimization problem, since for a dual vector π the reduced cost of variable u_k

whose production pattern is made of slab j is simply

$$\bar{c}_k - \pi_j - \sum_{i \in R^k} \pi^i = \sum_{i \in R^k} 2w^i - W_j - \pi_j - \sum_{i \in R^k} \pi^i = -(W_j + \pi_j) + \sum_{i \in R^k} (2w^i - \pi^i) \tag{7}$$

and we want to maximize this over the set of production patterns that can be manufactured from slab j. For a fixed j the first term is constant. The feasible production patterns from slab j are those that satisfy the capacity and color constraints, thus this problem is equivalent to (using the notation from the original formulation):

$$\max \sum_j (2w^i - \pi_i) x_j^i \tag{8}$$

$$\sum_i w^i x_j^i \leq W_j \tag{9}$$

$$\sum_i y_j^{c(i)} \leq 2 \tag{10}$$

$$x_j^i \in \{0, 1\} \tag{11}$$

which is a knapsack problem with the side constraint that selected objects must have no more than two different colors. Moreover, the constant term in the reduced cost implies that we are only looking for production patterns whose reduced cost exceeds $W_j + \pi_j$. Since solving the LP relaxation of the knapsack problem (even with the side constraint) is rather simple, this required lower bound on the reduced cost can be very helpful in quickly concluding that there is no improving pattern for a particular slab.

3.2 Upper bounding

The previous subsection addresses the issue of how to solve the full LP relaxation by iteratively solving smaller LP relaxations and generating columns, but we need something more. We need to be able to derive an upper bound on the optimal objective value of the full LP relaxation in every iteration. There are two reasons for this. The first is that in a Branch-and-Price algorithm we can fathom a search tree node if the upper bound on the optimal objective value of the LP relaxation at the node is already lower than the value of a currently known feasible solution. Since we may not be able to solve the subproblems that generate the columns (after all, even though the knapsack problem is considered relatively easy, it *is* NP-complete) we still want to have an upper bound for fathoming purposes. The second reason is that without an upper bound on the optimal value of the LP relaxation of the full problem we couldn't tell how close we are to optimality, we wouldn't have a proven gap.

Fortunately, upper bounding is very easy using Dantzig-Wolfe decomposition [2]. Since the sum of all variables in P_i is not more than 1, the objective value of the LP relaxation cannot change more than the highest reduced cost (or an upper bound on that value) as a result of changing the values of the variables in

P_i. To get an upper bound on the reduced cost we can use the LP relaxation of the subproblem (the side constrained knapsack problem) which is very easy to solve. Now adding all "per slab" upper bounds to the optimal objective value of the current LP relaxation yields an upper bound on the optimum of the full LP relaxation.

3.3 Finding integral feasible solutions

Another advantage of the column generation based formulation is that it is very easy to generate feasible solutions. In each iteration we considered the fractional solution and started by including every variable above 0.5 in the solution. From the set of remaining fractional variables we excluded all that intersected the already selected variables. Whatever remained afterwards was always such a small set that we could solve the set packing problem on that set by enumeration.

4. Implementation details and computational results

We have implemented the column generation method using BCP, which is part of COIN-OR, the Common Optimization INterface for Operations Research ([11]). BCP is a framework for Branch and Cut and Price algorithms, it handles all bookkeeping related to search tree management, to column and cut generation, etc. It is available as an open source program downloadable from the COIN-OR home page ([11]), and one of the sample implementations for its use is our code to solve the multiple knapsack problem with color constraints. We have used OSL as the LP engine to solve the LP relaxations. We ran our experiments on an IBM RS6000 44P-375MHz workstation.

To solve the subproblems, the knapsack problem with the side constraint, we have used various algorithms. For those subproblems where the number of objects in the knapsack (i.e., the orders) is small all possible solutions were enumerated and stored in a pool. Since the subproblems have to be solved over and over again with only the costs of the items changing, this pool greatly improved the solution time when the number of possibilities is not to big (we have enumerated every knapsack containing no more than 50 orders).

For the larger subproblems we have considered every possible color pair for that knapsack, attempted to solve the knapsack problem with those items only that had the selected two colors and finally chose the best of these solutions. To solve the individual knapsack problems we have used a Dynamic Programming approach (see, e.g., [7]).

This approach worked fine for the problem *mkc*. We were always able to solve the subproblems to optimality. In four seconds we solved the full LP relaxation to optimality (i.e., proved that there are no columns not in the current formulation that has positive reduced cost). This optimal solution happened to be integral thus we have solved a previously unsolved problem. It is worthwhile to note that the optimal value

of the LP relaxation of the original formulation is 611.85 and the best previously known feasible solution is 553.75. With the new formulation the optimal value of the LP relaxation is 563.846 which is also the value of the integer solution. This indicates that the new formulation is significantly tighter then the original one.

However, in *mkc7*, which is a big brother to *mkc*, we have ran into difficulties. Some of the subproblems led to unsolvable knapsack problems. For example, in one subproblem there were 66 identically colored items, thus there was only one knapsack problem to solve, but the items had almost identical cost per weight ratios and almost any half of the items could be fitted into the knapsack. This type of knapsack problems are notoriously hard, and although using various heuristics one can get a close to optimal solution, proving optimality is essentially impossible. Still, the new formulation proved to be much better than the original one for this problem as well. The optimal value for the LP relaxation of the original formulation is 1200.947. For the new formulation we couldn't solve the LP relaxation to optimality, but even the upper bound we have derived using Dantzig-Wolfe decomposition is better, it is 1185.968. (To prove this bound we ran the code until no more improving columns were found. This took approximately 20 minutes.) The best previously known solution (provided by the matching-based heuristics described in [6]) had value 1142.49 while our method produces a solution of value 1178.163 in just 71 seconds (just to solve the LP relaxation of the original formulation takes 101 seconds). Thus we have been able to close the integrality gap from 4.868% to 0.658%.

5. Conclusion and future work

Our experiment clearly showed that column generation is definitely a viable alternative to traditional, cutting plane based methods. In its original formulation no commercial IP solver was able to do anything even with the smaller problem studied here, they were just branching endlessly.

However, to apply column generation one has to write problem class specific code since a generic IP solver has no way of knowing which columns are admissible to the problem. This is where using BCP proved very advantageous; we had to write only the actual routines that generated the columns, BCP took care of managing them.

On the other hand, column generation presents a different kind of difficulty. As long as one works only in the root node of the search tree everything works well, but it is not obvious how branching should be done. Branching on a variable leads nowhere. On the 0-side (where the variable is fixed to 0) the column generation procedure will be changed to "generate a column of positive reduced cost which is not the column fixed to 0". Branching several times leads to practically unsolvable subproblems. Therefore in a column generation scheme one has to branch on something that not only subdivides the feasible region but in each

region the appropriate restriction can be easily imposed on the column generation subproblems as well. We plan to devise such branching rule and try to solve the *mkc7* problem to optimality.

References

[1] Balas, E. (1975) *Facets of the knapsack polytope*, Mathematical Programming, 8, 146-164.

[2] V. Chvátal (1983) *Linear programming*, W. H. Freeman and Company.

[3] Ferreira, C. E., Martin, A. and Weismantel, R. (1996), *Solving multiple knapsack problems by cutting planes*, SIAM J. Optimization, 3, 859-877.

[4] Gottlieb, E.S. and Rao, M.R. (1990), *The generalized assignment problem: Valid inequalities and facets*, Mathematical Programming, 46, 31-52.

[5] Hammer, P.L., Johnson, E.L. and Peled, U.N. (1975) *Facets of regular 0-1 polytopes*, Mathematical Programming, 8, 179-206.

[6] Kalagnanam, J., Dawande, M., Trumbo, M. and Lee, H.S. (2000) *The surplus inventory matching problem in the process industry*, Operations Research, Vol. 48 (4), 505-516.

[7] S. Martello and P. Toth (1990) *Knapsack problems*, John Wiley & Sons Ltd.

[8] G.L. Nemhauser and L.A. Wolsey (1988) *Integer and Combinatorial Optimization*, John Wiley & Sons Ltd.

[9] Wolsey, L. A. (1975) *Faces of linear inequalities with 0-1 variables*, Mathematical Programming, 8, 165-178.

[10] Wolsey, L. A. (1990) *Valid inequalities for 0-1 knapsacks and MIPS with generalized upper bound constraints*, Discrete Applied Mathematics, 29, 251-261.

[11] Common Optimization INterface for Operations Research home page: http://www.coin-or.org

[12] Deep Computing Institute home page: http://www.research.ibm.com/dci