KNAPSACK PROBLEMS WITH SETUP

Yanchun Yang

A Dissertation

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Doctor of Philosophy

Auburn, Alabama
August 7, 2006

KNAPSACK PROBLEMS WITH SETUPS

Except where reference in made to the work of others, the work described in this dissertation is my own or was done in collaboration with my advisory committee. This dissertation does not include proprietary or classified information.

Yanchun Yang

Certificate of Approval:

Saeed Maghsoodloo Professor Industrial and Systems Engineering	Robert L. Bulfin, Chairman Technology Management Professor Industrial and Systems Engineering
Jorge Valenzuela Associate Professor Industrial and Systems Engineering	Stephen L. McFarland Dean Graduate School

KNAPSACK PROBLEMS WITH SETUP

Yanchun Yang

Signature of Author

Date of Graduation

VITA

Yanchun Yang, daughter of Jie Yang and Jianmei Zhao, was born on April 14, 1977, in Jiagedaqi Daxinganling, Heilongjiang Province, P.R.China. She graduated with the degrees of Bachelor of Science (Industrial Engineering) in 1998 and Master of Management Engineering in 2001, both from Northeastern University, Shenyang, P.R.China. She entered Graduate School, Auburn University, in January, 2003.

DISSERTATION ABSTRACT

KNAPSACK PROBLEMS WITH SETUP

Yanchun Yang

Doctor of Philosophy, August 7, 2006
(MISE, Northeastern University, China, 2001)
(B.S., Northeastern University, China, 1998)

107 Typed Pages

Directed by Robert L. Bulfin

This research studies three integer programming models which can be applied to order acceptance in make-to-order manufacturing or regional project selection in multiple periods. All three models are the variations of the binary knapsack problems and they are called the knapsack problem with setup (KPS), the multiple knapsacks problem with setup (MKPS) and the multiple-choice knapsack problem with setup (MCKS), respectively. In all three models, jobs belong to different families and some variables represent setup for a family of jobs: if a setup is not done, no jobs in this family can be processed; if two jobs are processed sequentially, no setup is required.

Branch-and-bound algorithms are used to obtain the optimal solutions to all three models. Setup variables are branched on. After all setup variables are fixed, the models are transformed to a (several) knapsack problem(s). For each model, an independent linear knapsack problem is developed to give an upper bound. When a setup variable is fixed during branching, we update certain variables in the linear knapsack problem. The optimal objective of the updated linear knapsack problem is an upper bound on the generated sub-problem. The rounded LP solution of the linear knapsack problem for KPS or MCKS corresponds to an incumbent of KPS or MCKS. A greedy algorithm is developed to obtain a lower bound on MKPS. Computational experiments show the effectiveness of these algorithms.

Style manual or journal used Bibliography conforms to those of European Journal of Operational Research

Computer software used ANSI C, AMPL, Microsoft Office Excel and Microsoft Office Word

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

1.1. Objectives and significance

Make-to-order production is playing an increasingly important role in our economy, partly due to the Internet and manufacturing technology advances. In make-to-order production, price is dictated not only by cost, but also by the customer's expectation as well. Some customers are willing to pay a higher price for a short lead-time while others are willing to wait for their products in exchange for lower prices. Thus prices can be related to a product's delivery date. Price, schedule and the total profit have very complex connections. These connections are of extreme interest to businesses today.

Assume there is a manufacturing company. At time T, they receive some orders (jobs), which belong to N families. Family i, $i = 1,..N$, has n_i jobs. Also assume that these jobs should be produced in the next planning period. The company's manufacturing capacity is fixed and can't be changed in the short term. Setup cost and setup time occurs when manufacturing changes from a job in one family to another job from a different family. There is no setup between jobs of the same family. The company operates with a batch delivery policy; products that are manufactured in the same period have the same shipping date. This is a common scenario in many manufacturing companies. Then the company needs to decide how to choose orders to maximize the total profit. In this case, a single knapsack model with setup is used to solve this problem.

To extend this problem, jobs can be manufactured in T different periods, but a family can only be produced in a single period. Here the price charged for the product many relate to the customer’s desired due date; the price depends on the job’s completion time. The price could be determined by this way: there would be a base price for a job delivered at the customer’s desired due date; there will be “earliness” and “tardiness” penalties for other delivery dates. These prices would depend on the deviation from the desired due date and each customer’s tolerance for this deviation. Sometimes, the price could be increased for urgent jobs; or the price could be decreased if the customer agrees to allow more time for delivery. So in this system, prices are changed based on the product’s actual delivery time. The company might negotiate the price based on customer desires and company capabilities. Before making a production schedule, we should know the prices of jobs as a function of different completion dates.

With the added price variability, this model is more complex than typical scheduling models in make-to-order manufacturing. The company has to consider the marginal profit for different jobs, the current production capacity, and each family’s setup cost and time before choosing orders and deciding the job assignment to maximize its total profit. A multiple knapsack problem with setup (MKPS) model can solve this problem.

In above scenario, if production in T periods need the same non-renewable material and jobs from the same family can be manufactured in multiple periods, then a multiple-choice knapsack problem with setup (MCKS) can model this problem. MCKS is more helpful in an organization’s decision making on a fixed budget to invest a number of projects in multiple areas in multiple periods. In order to do a project in an area, a project office must be set up. The organization would like to decide where to set up offices and which projects to do to maximize net profit subject to this budget restriction.

1.2. Mathematical Model

1.2.1. Order acceptance

In make-to-order, if all orders have to be finished in one time period, a knapsack problem with setup (KPS) can be used to solve the orders acceptance problem. In this situation, a company will decide which jobs will be produced in this period.

Given this model:

$$Max \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i} c_{ij}x_{ij} + \sum_{i=1}^{N} f_i y_i$$

$$s.t.$$

$$\sum_{i=1}^{N}\sum_{j=1}^{n_i} a_{ij}x_{ij} + \sum_{i=1}^{N} d_i y_i \le b \qquad (1)$$

$$x_{ij} \le y_i \quad j=1,n_i; i=1,N \qquad (2)$$

$$x_{ij}, y_i \in \{0,1\} \quad j=1,n_i; i=1,N. \qquad (3)$$

i -is index families,

j -is index jobs,

N -is the number of families,

n_i -is the number of jobs in family i,

c_{ij} -is the profit of job j in family i,

a_{ij} -is the time to process job j in family i,

f_i -is the setup cost for family i ($f_i < 0$),

d_i -is the setup time for family i,

b -is the time available for processing,

x_{ij} -is one if job j in family i is produced, zero otherwise,

y_i -is one if any job in family i is produced, zero otherwise.

Constraint (1) requires that the total time used by jobs and setups cannot exceed the time available for production (resource other than time could also be considered). Constraints (2) prohibit a job from being processed if it belongs to a family that has not been setup.

If jobs can be manufactured in multiple periods, and all items in same family should be manufactured together in one period, then this model could be described as a multiple knapsack problem with setup (MKPS):

$$Max \quad \sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{j=1}^{n_i} c_{ijt}x_{ijt} + \sum_{t=1}^{T}\sum_{i=1}^{N} f_{it}y_{it}$$

$s.t.$

$$\sum_{i=1}^{N}\sum_{j=1}^{n_i} a_{ij}x_{ijt} + \sum_{i=1}^{N} d_i y_{it} \le b_t \quad t=1,..T, \tag{1}$$

$$x_{ijt} \le y_{it} \quad j=1,n_i; i=1,N; t=1,..T, \tag{2}$$

$$\sum_{t=1}^{T} y_{it} \le 1 \quad i=1,..N, \tag{3}$$

$$x,y \in \{0,1\} \quad j=1,..n_i; i=1,..N; t=1,..T. \tag{4}$$

x_{ijt} -is 1 if the job j of family i is arranged into period t, otherwise 0,

y_{it} -is 1 if some job of family i is arranged into period t, otherwise 0,

c_{ijt} -is the profit of job j of family i in period t ($c_{ijt} \ge 0$),

f_{it} -is the setup cost for family i in period t ($f_{it} < 0$),

a_{ij} -is the processing time for job j of family i ($a_{ij} > 0$),

d_i -is the setup time for family i ($d_i > 0$),

b_t -is the available resource for processing in period t ($b_t > 0$).

Constraint (1) requires that the total time used by jobs and setups cannot exceed the time available in each period for production (resource other than time could also be considered). Constraints (2) prohibit a job from being processed if it belongs to a family that has not been setup. Constraints (3) guarantee setup of each family occurs once.

In this model, all jobs belong to N different families. If a job is chosen, then setup time and setup costs must occur. A job may be put into T different periods, but the profit is different in different periods. The objective is to maximize the sum of the profits of accepted jobs.

1.2.2. Regional project selection with a fixed budget

Select projects which can be invested in multiple periods and in different regions to maximize net profit. This model can be described as a multiple-choice knapsack problem with setup.

$$Max \quad \sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{j=1}^{n_i} c_{ijt}x_{ijt} + \sum_{t=1}^{T}\sum_{i=1}^{N} f_{it}y_{it}$$

$s.t.$

$$\sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{j=1}^{n_i} a_{ij}x_{ijt} + \sum_{t=1}^{T}\sum_{i=1}^{N} d_i y_{it} \le b\,, \tag{1}$$

$$x_{ijt} \le y_{it} \quad j=1,..n_i,\ \ i=1,..N;\ \ t=1,..T\,, \tag{2}$$

$$\sum_{t=1}^{T} x_{ijt} \le 1 \quad i=1,...N, \quad j=1,..n_i\,, \tag{3}$$

$$x_{ijt}, y_{it} \in \{0,1\} \quad i=1,..N;\ j=1,..n_i; t=1,..T. \tag{4}$$

c_{ijt} -is the profit of project j in area i in period t ($c_{ijt} \ge 0$),

f_{it} -is the setup cost for opening an office in area i in period t ($f_{it} \le 0$),

a_{ij} -is the investment needed for project j in area i ($a_{ij} > 0$),

d_i -is the investment cost to open an office in area i ($d_i > 0$),

b -is the budge available to invest ($b > 0$),

y_{it} - is one if office is set up in area i in period t, otherwise zero,

x_{ijt} -is one if project j in area i is done in period t, otherwise zero,

N -is the number of areas,

T -is the number of periods.

Constraint (1) requires the total budget used by all projects and setup office can’t exceed the budget available. Constraints (2) prohibit a project done before the office in this area is set up. Constraints (3) guarantee a project in an area only can be invested once. Constraints (4) require the variables to be binary.

1.3. Basic research method

These three models are integer programs (IPs). For integer programming, branch and bound, cutting planes and dynamic programming could be used to optimally solve this class problem.

1.3.1. Cutting Plane

Cutting plane algorithm is an important and well-known approach to solve IPs. It is one of the purest methods in polyhedral description algorithms and an alternative to enumeration. Cutting planes redefines the problem again and again by adding constraints until the problem is solved.

In practice, a successful cutting plane algorithm depends on the relaxation method of the original problem, and the choice of cutting inequalities. There must be a family of valid inequalities, which define any optimal point, and a relaxation that is tractable. In fact when we add valid inequalities to the relaxation, we solve a series of relaxed problems. If this series of problems are easy to solve, that is better. But for these three models, we did not find such an algorithm for the relaxations. Therefore, cutting plane does not appear to be our best choice. For further study of cutting planes, refer to Parker and Rardin (1988).

1.3.2. Dynamic Programming

Dynamic Programming is not a specific algorithm, but we can use dynamic programming theory to design an algorithm for these three models. As the number of jobs increase, that algorithm becomes worse, and storage space will increase exponentially. We do not choose to use dynamic programming.

1.3.3. Branch and Bound

Branch and Bound belongs to the strategy of “partial enumeration”, just like cutting planes belongs to” polyhedral description”. These two strategies are often used to solve IPs. Though they are non-polynomial in the worst case, they can be effective solution procedures for IPs in practice.

In a branch-and-bound algorithm, if a variable x is restricted to be binary, we can separate the problem into two sub-problems: one with $x = 0$ and the other with $x = 1$. Successful applications for B&B need a good algorithm to calculate upper and lower bounds for those sub-problems. The tighter the upper and lower bounds are, the more effective the algorithm is. Only with strong bounds we can expect to fathom candidate problems rapidly enough to avoid being overcome by the exponential growth in the number of potential sub-problems.

Since we design a linear knapsack problem to supply the upper bound for each model and the linear knapsack problem is easy to be solved by Danzig’s algorithm, B&B becomes an attractive method to solve these problems.

1.4 Relaxation Method

1.4.1. Linear Relaxation

Linear programming is, without doubt, the most successful branch of optimization (Parker and Rardin, 1988). Integer programming is usually changed to linear programming by relaxing the integer constraints. Linear programs can be solved easily, and may provide a good upper bound. Therefore, many integer program algorithms use a linear relaxation to get the bound.

In this paper, we relax the integer constraints of job variables for all three models. Linear knapsack problems are designed to give the upper bounds on these relaxations.

1.4.2. Surrogate Relaxation

A surrogate constraint is a linear combination of other constraints. The following is an example of surrogate relaxation:

$$
\begin{aligned}
&Max \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_j x_{ij} \\
&s.t. \\
&\qquad \sum_{j=1}^{n} a_j x_{ij} \le b_i \quad (i = 1, \dots, m), \\
&\qquad x_{ij} \in \{0,1\}
\end{aligned}
$$

Then its surrogate relaxation is:

$$
\begin{aligned}
&Max \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_j x_{ij} \\
&s.t. \\
&\qquad \sum_{i=1}^{m} v_i \sum_{j=1}^{n} a_j x_{ij} \le \sum_{i=1}^{m} v_i b_i \\
&\qquad x_{ij} \in \{0,1\}
\end{aligned}
$$

The original problem’s solution is also a feasible solution to the surrogate relaxation, but the solution of surrogate relaxation is not necessarily feasible to the original problem. The surrogate relaxation has a larger feasible space. The optimal solution to the surrogate is an upper bound of the original problem. In this paper, surrogate relaxation along with linear relaxation will be used in MKPS to obtain a good upper bound.

1.4.3. Lagrangean Relaxation

Lagrangean relaxation is also a common relaxation model. This is an example for Lagrangean relaxation:

Give the model L1

$$
\begin{aligned}
&Max \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i} c_{ij}x_{ij} \\
&s.t. \\
&\qquad \sum_{j=1}^{n_i} a_{ij}x_{ij} \le b_i \quad i = 1,\ldots,N \\
&\qquad x_{ij} \in \{0,1\}
\end{aligned}
$$

Its Lagrangean relaxation, L2, is:

$$
\begin{aligned}
&Max \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i} c_{ij}x_{ij} + \sum_{i=1}^{N} u_i (b_i - \sum_{j=1}^{n_i} a_{ij}x_{ij}) \\
&s.t. \\
&\qquad x_{ij} \in \{0,1\}
\end{aligned}
$$

For each feasible solution of L1, we have

$$
\sum_{i=1}^{N}\sum_{j=1}^{n_i} c_{ij}x_{ij} + \sum_{i=1}^{N} u_i (b_i - \sum_{j=1}^{n_i} a_{ij}x_{ij}) \ge \sum_{i=1}^{N}\sum_{j=1}^{n_i} c_{ij}x_{ij}
$$

and all feasible solutions of L1 must be feasible solutions of L2, but not vice versa.

If we use Lagrangean relaxation, the knapsack problem’s good structure is destroyed. Also experimentation shows the bound is not tight enough. Therefore, Lagrangean relaxation is not used in this paper.

References

Parker, R.G., Rardin, R. L. 1988. Discrete Optimization. Academic Press, Inc. San Diego, CA.

Ⅱ. KNAPSACK PROBLEM WITH SETUP

Abstract

This paper studies a 0-1 knapsack problem with setup (KPS) where one set of variables serves as the upper bound of another set of variables. An efficient algorithm presented by Bulfin (1988) for the linear relaxation of this problem is applied to obtain an upper bound. Branch and bound is used to obtain the optimal solution, and the upper bound variables are branched before the remaining variables so KPS becomes a single knapsack problem. Computational experiments show that this algorithm is effective when objective and constraint coefficients are uncorrelated. This model can be used in order acceptance of single period in make-to-order manufacturing.

2.1. Introduction

A company makes metal door frames based on customer orders. Door frames have different heights, widths, jamb sizes and a number of hinges and lock configurations. An order can be for a single frame or for 1,000 identical frames. To make a particular frame, the production machinery must be set up for the parameters of that door. Some setups, like the height of the door are easily made, while others, like jamb size require much time and labor. The actual cost to produce a frame depends on what other frames are being produced; if many identical frames are made, economies of scale result in a low cost. On the other hand, if a single frame is made, the setup cost dominates and the cost is high.

Thus which orders are accepted, when they are produced and the price charged are critical to profitability.

This scenario describes the basic order acceptance problem faced by all make-to-order manufacturers. Orders consist of jobs, and similar jobs make up a family. Families share a setup; if two jobs from the same family are processed sequentially, no setup is required. The manufacturer plans production for the next period based on orders received. An order can be accepted or rejected for production in this period.

This problem can be formulated as a knapsack with setup. Let

i index families

j index jobs

N be the number of families,

n_i be the number of jobs in family i,

c_{ij} be the profit of job j in family i,

a_{ij} be the time to process job j in family i,

f_i be the setup cost for family i ($f_i < 0$),

d_i be the setup time for family i and

b be the time available for processing.

The decision variables are:

x_{ij} is one if job j in family i is produced, zero otherwise and

y_i is one if any job in family i is produced, zero otherwise.

The model, which we call KPS, is:

$$Max \quad \sum_{i=1}^{N} \sum_{j=1}^{n_i} c_{ij} x_{ij} + \sum_{i=1}^{N} f_i y_i$$

$s.t.$

$$\sum_{i=1}^{N} \sum_{j=1}^{n_i} a_{ij} x_{ij} + \sum_{i=1}^{N} d_i y_i \le b \tag{1}$$

$$x_{ij} \le y_i \quad j = 1, n_i; i = 1, N \tag{2}$$

$$x_{ij}, y_i \in \{0,1\} \quad j = 1, n_i; i = 1, N. \tag{3}$$

Constraint (1) requires that the total time to produce jobs cannot exceed the time available. Constraints (2) ensure a job is processed only if it belongs to a family that has been setup. Constraints (3) require the variables to be binary.

In the following section we give a brief literature review and discuss background used in the solution methodology. In Section 2.3, we present an algorithm to solve KPS. Computational results are given in Section 2.4. Finally, we give concluding remarks.

2.2. Literature survey

This linear relaxation of KPS was first introduced by Ham et al. (1985) as a cell loading problem for a Group Technology production system. Bulfin (1988) developed a polynomial algorithm for the linear relaxation of KPS. It is based on the ratio rule of Dantzig (1957) for the linear knapsack problem.

Akinc (2004) derives an algorithm for a special case of KPS with no setup time, which he called fixed-charge knapsack problem. His algorithm to solve the linear relaxation is the same as the one in Bulfin (1988). He outlined a branch-and-bound algorithm to solve the integer version and used this solution to compare heuristics. No solution times are given for the branch-and-bound algorithm. He states “This problem is solved as an LP. If all y_i are integer, then the optimal solution of P (the fixed-charge knapsack problem) is obtained from the solution of the ordinary 0/1 knapsack problem that optimally allocates the available capacity to all x_{ij} for which $y_i = 1$.” This statement is not true, as seen by the following counter-example:

$$
\begin{aligned}
&Max \quad 6x_{11} + 5x_{12} - y_1 + 5x_{21} + 8x_{22} - y_2 \\
&s.t. \\
&\qquad x_{11} + 3x_{12} + x_{21} + 4x_{22} \le 4 \\
&\qquad x_{11} \le y_1, x_{12} \le y_1 \\
&\qquad x_{21} \le y_2, x_{22} \le y_2 \\
&\qquad x_{11}, x_{12}, x_{21}, x_{22} \in \{0,1\}
\end{aligned}
$$

The LP’s optimal solution is $y_1 = 1, y_2 = 1$, and the objective is 13. Based on Akinc’s claim, solving the integer knapsack with both setups included gives a solution value of 9, with $y_1 = 1, y_2 = 1$, $x_{11} = 1$ and $x_{12} = 1$. But the solution $y_1 = 1, y_2 = 0$, $x_{11} = 1$ and $x_{12} = 1$ has objective 10. Hence, the optimal objective of knapsack problem when all y are integer in LP solution is not necessarily optimal for the integer model. This brings the results of his paper into question.

Chajakis and Guignard (1994) consider the setup knapsack problem which is similar to ours except the setup cost f_i and profit of job c_{ij} can be positive or negative. An extra constraint is added to make sure a setup does not occur if no job in this family is put into knapsack. This is unnecessary in KPS since c_{ij} is positive and f_i is negative. Chajakis and Guignard transform the original problem to an equivalent formulation without setup variables by two methods. Variables y are described by a Boolean union of x variables so that the constraints coupling x and y can be deleted and the problem becomes a "knapsack problem" with a Boolean union of all variables. The second method is to enumerate all non-dominated feasible solution for each family and define a pseudo-variable corresponding to this solution. This transforms the setup knapsack to a multiple-choice knapsack problem and only one pseudo-variable can be one in an optimal solution. Dynamic programming is used to solve the first transformation; branch-and-bound and dynamic programming are both used to solve the multiple-choice knapsack problem in the second transformation. Instances with 5, 10, 20, 50, and 200 families are tested. A maximum of 4000 total variables can be solved.

2.3. Background

The knapsack problem and its many variants are well-studied. For a discussion, see Martello and Toth (1990) and Dudzinski and Walukiewicz (1987). We discuss some basic results that will be used later in this paper. Dantzig (1957) defined the linear knapsack problem as:

$$\begin{aligned}
&Max \;\; \sum_{j=1}^{n} c_j x_j \\
&s.t. \\
&\qquad \sum_{j=1}^{n} a_j x_j \le b \\
&\qquad 0 \le x_j \le 1, \; j = 1,..n
\end{aligned}$$

If the variables are ordered by ${c_1}/{a_1} \ge {c_2}/{a_2} \ge ... \ge {c_n}/{a_n}$, he showed the optimal solution is given by

$$x_j = 1, \; j < t$$

$$x_t = {(b - \sum_{j=1}^{t-1} a_j)} \Big/ {a_t}$$

$$x_j = 0, j > t$$

where $t = \min\{i \mid \sum_{j=1}^{i} a_j > b\}$.

Similarly, we define the linear relaxation of KPS (LKPS), which is given by

$$\begin{aligned}
&Max \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i} c_{ij}x_{ij} + \sum_{i=1}^{N} f_i y_i \\
&s.t. \\
&\qquad \sum_{i=1}^{N}\sum_{j=1}^{n_i} a_{ij}x_{ij} + \sum_{i=1}^{N} d_i y_i \le b, \\
&\qquad x_{ij} \le y_i \quad j = 1, n_i; i = 1, N, \\
&\qquad x_{ij} \ge 0 \quad j = 1, n_i; i = 1, N, \\
&\qquad 0 \le y_i \le 1, \ i = 1,..N.
\end{aligned}$$

Define $r_{ij} = {c_{ij}} \big/ {a_{ij}}$, $i = 1,..N \ \ j = 1,..n_i$. Order jobs so that $r_{i1} \ge r_{i2} \ge r_{i3} ...... \ge r_{i,n_i}$.

Let

$$r_{i0} = \frac{\sum_{j=1}^{t_i} c_{ij} + f_i}{\sum_{j=1}^{t_i} a_{ij} + d_i} = \max\{\frac{\sum_{j=1}^{k} c_{ij} + f_i}{\sum_{j=1}^{k} a_{ij} + d_i} \mid k = 1, 2, ..n_i\} \text{ for } i \in N.$$

Separate the jobs of family i into two sets, $XM_i = \{1 \ldots t_i\}$ and $XT_i = \{t_i + 1 .... n_i\}$. Then

$r_{i,0} \ge r_{i,t+1} \ge r_{i,t+2} \ge ... \ge r_{i,n_i}$; a proof is given in the Appendix A.

For family i, define:

$$c'_{i1} = \sum_{j=1}^{t_i} c_{ij} + f_i$$

$$a'_{i1} = \sum_{j=1}^{t_i} a_{ij} + d_i$$

$$c'_{i,j-t_i+1} = c_{ij} \qquad j = t_i + 1, ..n_i$$

$$a'_{i,j-t_i+1} = a_{ij} \qquad j = t_i + 1, ..n_i$$

$$n'_i = n_i - t_i + 1$$

Then LKPS can be reformulated as a classical linear knapsack problem, which we call LBKP:

$$\begin{aligned} &Max \quad \sum_{i=1}^{N} \sum_{j=1}^{n'_i} c'_{ij} z_{ij} \\ &s.t. \\ &\qquad \sum_{i=1}^{N} \sum_{j=1}^{n'_i} a'_{ij} z_{ij} \le b \\ &\qquad 0 \le z_{ij} \le 1, i = 1, ..N \;\; j = 1, ...n'_i \end{aligned}$$

and solved by Dantzig's ratio rule. If there is no fractional variable, KPS is also solved. We know that, at most, one variable will have a fractional value.

Suppose $z_{ij} = f$, $0 < f < 1$. If $j > t_i$, then job $t_i + j$ in family i will be the only fractional variable KPS and all setup times and costs are considered. On the other hand, if $j = 1$, z_{i1} represents a virtual job composed of setup and jobs 1 through t_i of family i.

Here $y_i = f$ and $x_{ij} = f$, $j = 1,..t_i$ so all are fractional in KPS and the setup time and cost for family i and the processing time and profit of the first t_i jobs are only partially considered. If we round the fractional variable(s) to zero, then the current solution is feasible to KPS, and can be used as a lower bound in the branch-and-bound algorithm.

2.4. Solution algorithm

To develop a branch-and-bound algorithm, we need to make several decisions. These include how to fix variables, calculate bounds, choose the next sub-problem to explore and obtain an initial incumbent solution. We discuss these now.

2.4.1. Fixing variables

We only fix setup variables y_i to be zero or one in our main branch-and-bound scheme. When a sub-problem is created with y_i fixed to one, the right hand side is reduced by d_i and f_i is added to objective directly in the sub-problem. Then all z_{ij}, $j = 1,..n_i$ are replaced by real variables $x_{i1},...x_{in_i}$ of family i. When a sub-problem is created with y_i fixed to zero, z_{ij}, $j = 1,..n_i$ are removed from that sub-problem. Note that if all y_i are binary in the linear relaxation but some x_{ij} is fractional, solving a knapsack problem over the x_{ij} with $y_i = 1$ will not necessarily give the optimal solution as we showed in Section 2. When all y_i are fixed, we solve a knapsack over the remaining variables to obtain the best solution with those variables fixed. If this produces a better solution than the incumbent, it replaces the incumbent.

We order the z_{i1} variables by $r_{10} \le r_{20} \le ... \le r_{N0}$. If a variable has large r_{i0}, it is more likely to be one in an optimal solution, while those with smaller ratios are more likely to be zero. We choose either the first or last variable to fix first and work toward the middle. This tends to keep the number of active branches small.

2.4.2. Bounding

We use LBKP as an upper bound on KPS. It is a linear relaxation which allocates the setup time and cost proportionally. It is initially solved by the ratio rule. When some y_i is fixed, it is easy to resolve the sub-problem. If we fix y_i to one, we delete the pseudo variables $z_{i1},..z_{in}$ and insert the new variables $x_{i1},...x_{in_i}$. This may require taking resource from some free variables, which are chosen by the ratio rule to maintain optimality. Similarly, fixing $y_i = 0$ may free up resource, which is then allocated to free variables using the ratio rule.

2.4.3. Choosing a new sub-problem

When variables are fixed, two sub-problems are created. If a sub-problem's upper bound is no better than an incumbent solution it is discarded. When its bound indicates it could contain a better solution to KPS we store it in a bucket. Each bucket contains sub-problems with bounds that are about the same. Let UB be the best upper bound and INC be the value of the current incumbent solution. If we want K buckets, calculate

$$\Delta = {(UB - INC)}\Big/{K}\,.$$

Then bucket one will contain all sub-problems with upper bounds in the interval $[UB - \Delta, UB]$, bucket two $[UB - 2\Delta, UB - \Delta]$, and bucket K $[INC, INC + \Delta]$. Buckets can be updated as upper bounds or the incumbent change. When we choose a new sub-problem to explore, we take one based on LIFO from the lowest numbered, non-empty bucket. This gives almost a "best-bound" strategy, but without the bookkeeping overhead.

2.4.4. Heuristic

If the fractional valued variable of LBKP is z_{ij}, rounding down z_{ij} to 0 frees $a'_{ij} z_{ij}$ resource. Allocate this resource to variables with processing time less than $a'_{ij} z_{ij}$ and already has its family set up. Variables are chosen by the ratio rule until there are no more variables which can use the remaining resource.

2.5. Computational experiments

Our experiments will be similar to previous experiments on knapsack problems. However KPS has a setup requirement, so setup time and setup cost will be included in this study. We wish to test our algorithm (AKPS) on a variety of problem instances to see what problems can be solved. Instances will be generated by setting four parameters at several levels. The parameters are the number of families, average number of jobs in a family, proportion of setup time/cost relative to totals, and correlation between objective function and constraint coefficients. All data will be integer valued.

The number of families will be fixed at 50 and 100. The number of jobs in family i is a uniformly distributed integer in either [40, 50] or [90,100]. Setup cost and time is given by

$$f_{it} = -e_1\left(\sum_{j=1}^{n_i} c_{ijt}\right)$$

$$d_i = e_2\left(\sum_{j=1}^{n_i} a_{ij}\right)$$

e_1 and e_2 are uniform from [0.05, 0.15], [0.15, 0.25], [0.25, 0.35], and [0.35, 0.45].

We choose a and c two ways. First a_{ij} and c_{ijt} are both chosen uniformly from [10, 10000]; thus they are independent. Next, a_{ij} is chosen uniformly from [10, 10000], while c_{ijt} is chosen uniformly from [a_{ij} -1000, a_{ij} +1000], but if c_{ijt} is less than 10 it is randomly chosen from [10,100]; this introduces some correlation between the two coefficients.

In previous knapsack studies, instances tend to be the hardest when the available resource is roughly one half the sums of the constraint coefficients. Therefore, we choose b uniformly from [$0.4*\sum_{i=1}^{N}\sum_{j=1}^{n_i} a_{ij}$, $0.6*\sum_{i=1}^{N}\sum_{j=1}^{n_i} a_{ij}$].

For each level of the four factors we generate ten instances. AKPS was coded in C and all instances were run on a Dell P.C. with 1.7G Intel processor and 512M bytes of memory. In the following tables, we report the minimum (MIN), average (AVG) and maximum solution time (MAX) in seconds. We also give the average ratio of initial solution (INC) to initial upper bound (UB) and the average ratio of initial incumbent to the optimal solution (OPT).

Table 2.1.
Solution time (seconds) for AKPS

			uncorrelated					correlated				
N	n_i	Setup	LB/UB	LB/OPT	MIN	AVG	MAX	LB/UB	LB/OPT	MIN	AVG	MAX
50	[40,60]	[0.05-0.15]	1.00	1.00	0.03	0.06	0.27	0.98	0.98	8.05	17.46	29.28
		[0.15-0.25]	0.99	0.99	0.06	0.53	1.72	0.97	0.97	2.25	16.63	30.73
		[0.25-0.35]	0.99	0.99	0.03	0.49	1.17	0.97	0.97	1.09	25.69	65.56
		[0.35-0.45]	0.97	0.97	1.25	2.62	4.89	0.98	0.98	12.83	22.97	56.5
50	[90,110]	[0.05-0.15]	1.00	1.00	0.08	0.09	0.12	0.98	0.98	5.69	26.47	63.72
		[0.15-0.25]	0.99	0.99	0.05	0.87	2.94	0.97	0.97	11.30	28.46	55.75
		[0.25-0.35]	0.98	0.98	0.09	2.67	5.28	0.98	0.98	2.77	34.52	82.31
		[0.35-0.45]	0.98	0.98	0.25	4.25	9.30	0.99	0.99	0.91	49.36	101.4
100	[40,60]	[0.05-0.15]	1.00	1.00	0.06	0.16	0.36	0.99	0.99	17.39	153.07	503.38
		[0.15-0.25]	1.00	1.00	0.08	1.43	4.36	0.99	0.99	70.61	124.69	220.53
		[0.25-0.35]	0.99	0.99	0.05	4.96	18.97	0.99	0.99	24.62	175.51	315.67
		[0.35-0.45]	0.99	0.99	2.41	14.34	29.62	0.99	0.99	22.11	131.22	305.85
100	[90,110]	[0.05-0.15]	1.00	1.00	0.14	0.24	0.39	0.99	0.99	121.86	385.44	877.19
		[0.15-0.25]	1.00	1.00	0.28	4.02	7.50	0.99	0.99	58.69	*477.78	877.73
		[0.25-0.35]	0.99	0.99	1.33	11.86	30.48	0.99	0.99	17.55	*468.23	953.29
		[0.35-0.45]	0.99	0.99	1.08	31.26	107.09	0.99	0.99	11.48	*484.35	784.72

Note: “*” shows 3 of these instances ran out of memory; AVG, MAX, and MIN are calculated based on the remaining seven instances.

Our heuristic solution is outstanding. On average, it was less than 2% from the optimal over the entire range of instances tested. Based on the data from Table 2.1, correlated instances are more difficult to solve than uncorrelated instances. The setup proportion has a stronger effect on uncorrelated instances than correlated instances. With the same number of variables, AKPS works better when there are fewer families and the number of jobs per family is large. This makes sense since fewer family variables simplify the first stage of the branching. Instances with 50 families and an average of 100 jobs per family are much easier than instances with 100 families and an average of 50 jobs per family.

Fig. 2.1 shows the solution time of instances with $N = 50$ and an average of 100 jobs per family and instances with $N = 100$ and an average of 50 jobs per family with uncorrelated coefficients. With roughly the same number of variables, instances with larger N are more difficult. Also, solution time increases as setup proportion increases. The incumbent solution gets worse as setup proportion increases. Fig. 2.2 gives the solution time with correlated coefficients. Instances with fewer families still work better than the others but solution time is not changed too much as setup proportion increases. In correlated instances, setup proportion does not have as much effect on the incumbent.

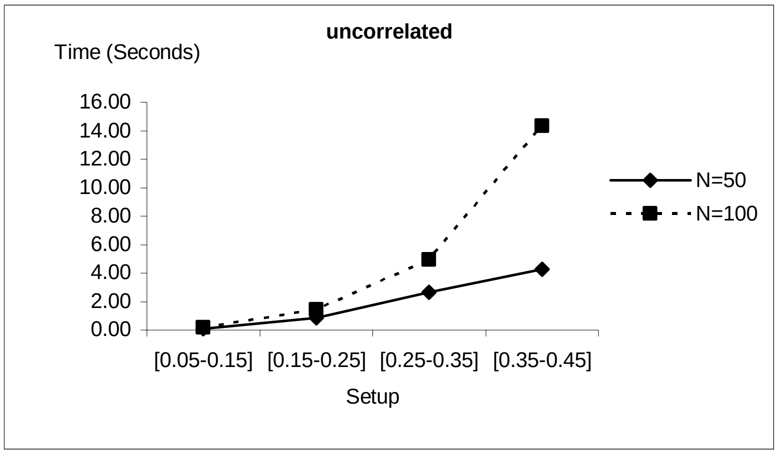

Fig. 2.1. Comparison of uncorrelated instances with similar total variables number

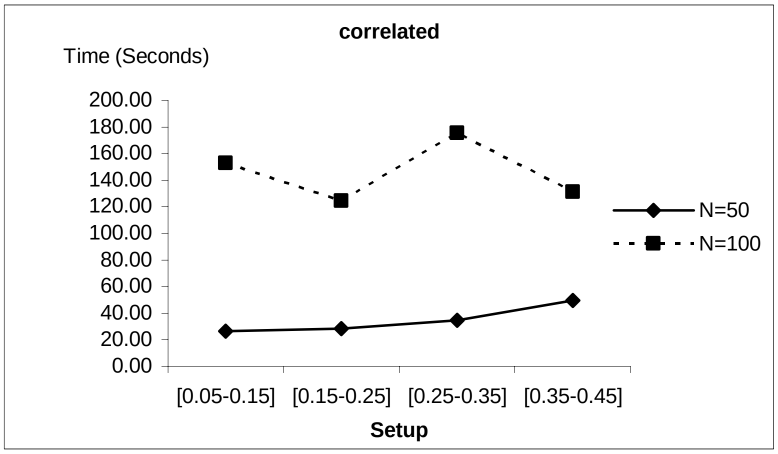

Fig. 2.2. Comparison of correlated instances with similar total variables number

Chajakis and Guignard only test uncorrelated instances with coefficients from a small range. (i.e. one set of instances obtains setup cost, profit from [-100, 100] and setup time, processing time from [1,10]). Since the dynamic programming used in their paper has a pseudo-polynomial worst case complexity, the large coefficients will increase the difficulty of instances and need more storage without doubt. The second approach presented fail in instances with total 4000 variables because of storage used up. The first one can solve the same instances but need over 1000 seconds. They permit job profit negative and setup cost positive in their model, which, to some extent, make instances easier due to parts of variables having fixed to 0 by a preprocessing, which reduce the size of the problem remarkably. The total number of variables after preprocessing is only about 40%-60% of the original one. For instances with 4000 variables, only 2500 variables are left after this preprocessing.

We also compare AKPS with CPLEX 9.1 (called by AMPL). We test instances with 50 families and an average of 100 jobs per family. For each setup, we test five instances. AKPS takes much less time for uncorrelated problems. CPLEX takes from 12 to 96 times longer; as setup proportion increases the difference becomes smaller. When the coefficients are not independent, the difference is much smaller. AKPS is only 3 to 6 times faster on average, and a few instances take less time on CPLEX.

We also compared some instances with 100 families and 50 jobs per family, but do not present the data. CPLEX is better than AKPS when a and c are correlated. But AKPS is better than CPLEX if a and c are uncorrelated for instances with $N = 100, n_i \sim [40, 60]$. Therefore we suggest using AKPS when a and c are uncorrelated; if they are correlated and there are over 50 families CPLEX might be better.

Table 2.2.
Comparing solution time (seconds) of CPLEX and AKPS

	Uncorrelated			Correlated		
SETUP	AKPS	CPLEX	CPLEX/AKPS	AKPS	CPLEX	CPLEX/AKPS
[0.05-0.15]	0.05	1.17	23.40	21.87	13.08	0.60
	0.09	1.92	21.33	13.64	491.73	36.05
	0.05	1.06	21.20	44.06	253.78	5.76
	0.05	1.08	21.60	34.00	3.81	0.11
	0.06	0.86	14.33	37.42	226.00	6.04
AVG			20.37			9.71
[0.15-0.25]	0.05	4.67	93.40	24.58	411.08	16.72
	0.41	26.28	64.10	56.78	929.03	16.36
	0.05	2.31	46.20	40.14	376.75	9.39
	0.11	15.44	140.36	40.22	269.69	6.71
	0.05	6.97	139.40	81.39	215.44	2.65
AVG			96.69			10.36
[0.25-0.35]	4.75	15.52	3.27	23.64	6.67	0.28
	1.95	12.09	6.20	46.01	514.64	11.19
	0.61	16.97	27.82	88.14	5.75	0.07
	2.75	17.39	6.32	7.67	14.86	1.94
	1.55	26.58	17.15	72.03	102.00	1.42
AVG			12.15			2.98
[0.35-0.45]	3.97	11.20	2.82	179.06	7.42	0.04
	0.91	16.36	17.98	6.56	35.95	5.48
	1.41	65.77	46.65	36.91	283.38	7.68
	7.42	2.91	0.39	107.62	265.69	2.47
	4.58	12.78	2.79	22.16	96.05	4.33
AVG			14.13			4.00

2.6. Conclusions

We investigate the knapsack problem with setup. This is an important problem, modeling order acceptance, cell loading, project selection and others. We have developed an exact algorithm for the problem. The first computational tests on exact solutions indicate our algorithm is vastly superior to CPLEX for many instances, superior on others and about the same for the rest. Further, we have determined what parameter values make instances hard for our algorithm. Finally, the proposed heuristic is excellent, being within 2% of optimal for all the problems tested.

Appendix A. r_{i0} is greater than $r_{i,t+1}$.

Proof.

Assume $a, \; b, \; c, \; d > 0$ and $\frac{a}{b} \le \frac{c}{d}$, then $\frac{a+c}{b+d} \ge \frac{a}{b}$. Since $ad \le bc$ from $\frac{a}{b} \le \frac{c}{d}$, then $ab + ad \le bc + ab$, or $a(b+d) \le b(a+c)$, so $\frac{a+c}{b+d} \ge \frac{a}{b}$. Thus if $r_{i,t+1} \ge r_{i0}$, then job $t+1$ should be included in XM_i. Since it is not, then $r_{i,0} > r_{i,t+1}$.

Therefore $r_{i,0} \ge r_{i,t+1} \ge r_{i,t+2} \ge ... \ge r_{i,n_i}$. r_{i0} represents family i 's maximum ability to obtain profit for each unit of resource it consumes.

References

Akinc, U. 2004. Approximate and exact algorithm for the fixed-charge knapsack problem, European Journal of Operational Research 170, 363-375.

Bulfin, R. L. 1988. An algorithm for the continuous variable upper bound knapsack problem, OPSEARCH 25 (2), 119-125.

Chajakis, E.D., Guignard, M. 1994. Exact algorithms for the setup knapsack problem, INFOR 32 (3), 124-142.

Dantzig, G.B. 1957. Discrete variable extremum problems, Operations Research 5, 266-277.

Dudzinski, K., Walukiewicz, S. 1987. Exact methods for the knapsack problem and its generalizations. European Journal of Operational Research 28, 3-21.

Ham, I., Hitomi, K., Yoshida, T. 1985. Group Techonology, Kluwer Nijhoff Publishing, Boston, Massachusetts.

Martello S. and Toth, P. 1990. Knapsack Problems: Algorithms and Computer Implementations, John Wiley and Sons, New York.

III. MULTIPLE KNAPSACK PROBLEM WITH SETUP

Abstract

We present a multiple knapsack problem with setup (MKPS). This problem can be used to model order acceptance and production scheduling of multiple periods in make-to-order manufacturing. Some variables represent setting up production for a family of jobs; if a setup is not done, no jobs in the family can be processed. Further, a family can only be set up in one period of the planning horizon. A linear knapsack problem is designed to give an upper bound on MKPS. A greedy algorithm is developed to obtain a lower bound. Setup variables are branched on; when all set up variables are fixed, MKPS becomes several independent knapsack problems. Computational experiments show this algorithm is effective, especially when resources are tight.

3.1. Introduction

The knapsack problem and its variants are well known problems in integer programming. In this paper, we present a model that we call the multiple knapsack problem with setup (MKPS). In this model, jobs belong to N different families. If a job is processed, then a setup time and a setup cost are incurred. A job can be assigned to T different periods, but only one setup for each family is allowed during the planning horizon, so jobs in the same family must be processed in the same period. The profit for job j of family i processed in period t is c_{ijt}, and varies for different periods, but the processing time a_{ij} stays the same. The objective is to maximize the sum of the profits of processed jobs. Formally, we have:

$$Max \quad \sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{j=1}^{n_i} c_{ijt}x_{ijt} + \sum_{t=1}^{T}\sum_{i=1}^{N} f_{it}y_{it}$$

$s.t.$

$$\sum_{i=1}^{N}\sum_{j=1}^{n_i} a_{ij}x_{ijt} + \sum_{i=1}^{N} d_i y_{it} \le b_t, \quad t = 1,..T \tag{1}$$

$$x_{ijt} \le y_{it,} \quad j = 1, n_i; i = 1, N; t = 1,..T \tag{2}$$

$$\sum_{t=1}^{T} y_{it} \le 1 \quad i = 1,..N \tag{3}$$

$$x, y \in \{0,1\} \quad j = 1,..n_i; i = 1,..N; t = 1,..T \tag{4}$$

x_{ijt} -is one if the j^{th} job of family i is arranged into period t, otherwise zero,

y_{it} -is one if some job of family i is arranged into period t , otherwise zero,

c_{ijt} -is the profit of job j of family i in period t ($c_{ijt} \ge 0$),

f_{it} -is the setup cost for family i in period t ($f_{it} < 0$),

a_{ij} -is the processing time for job j of family i ($a_{ij} > 0$),

d_i -is the setup time for family i ($d_i > 0$),

b_t -is the available resource for processing in period t ($b_t > 0$).

Constraints (1) require that the total resource used by jobs in each period can not exceed the resource available. Constraints (2) prohibit a job from being processed if it belongs to a family that has not been setup. Constraints (3) guarantee jobs in the same family processed in a single period. Constraints (4) require all variables to be binary.

This formulation models order acceptance in make-to-order manufacturers. Assume a manufacturer receives orders for jobs which belong to N different product families. Orders can be manufactured in T periods. Setup time and setup cost occur between jobs of different families. If jobs are accepted, jobs of the same family are done in the same period.

In make-to-order production, price is dictated not only by cost, but also by the customer's expectation as well. Some customers are willing to pay a higher price for a short lead-time, while others are not. Thus prices are related to a product's completion date, and different production schedules could produce different profits. The optimal solution to MKPS gives the maximum profit, which orders to accept, and how to assign jobs to periods.

The multiple knapsack problem assigns a set of items to multiple knapsacks with fixed capacities so that the total profit of selected items is maximal. The multiple knapsack problem is a special case of multiple knapsack problem with setup by ignoring the setup variables and setting $c_{ijt} = c_{ij}$. The multiple knapsack problem has been widely investigated. Martello and Toth (1980, 1981) discussed an upper bound algorithm using Lagrangean relaxation. Pisinger (1999) presented an exact algorithm using a surrogate relaxation to get an upper bound, and dynamic programming to get the optimal solution. The surrogate relaxation of the multiple knapsack problem with identical multipliers is a knapsack problem. Apparently, MKPS can not do in this way not only because each job has the different profit coefficients in periods, but also there are the additional setup variables in the model.

MKPS has multiple-choice constraints like the multiple-choice knapsack problem. An efficient algorithm and two dominance properties exist for the linear multiple-choice knapsack problem. More detail can be found in Pisinger (1995).

The knapsack problem with setup (KPS) is a special case of MKPS when $T = 1$. Bulfin (1988) gave an efficient algorithm for its linear relaxation (LKPS), which is similar to Dantzig's algorithm for the linear knapsack problem. This transforms the LKPS into a knapsack problem by using a modified ratio related to a job set. We state this algorithm in the following section. Akinc (2004) describes algorithms for a fixed-charge knapsack problem, which is a special case of MKPS with a single period and zero setup time.

Though the LP solution is often a good upper bound on integer programs such as knapsack problem and multiple-choice knapsack problem, we do not solve the linear relaxation of MKPS for obtaining an upper bound, but design a linear knapsack problem formulation, whose optimal objective is the upper bound of MKPS. Since MKPS becomes independent knapsack problems if all y_{it} variables are fixed, branching is done in two stages. The first stage is to branch on y_{it} variables. When all y variables are fixed, the second stage solves independent knapsack problems. There are many algorithms available for knapsack problem. We just use a simple branch-and-bound algorithm for knapsack problem.

Our approach (AMKPS) is outlined below:

Step 1. Do surrogate relaxation and linear relaxation for MKPS.

Step 2. Find an initial upper bound for MKPS.

Step 3. Find a feasible solution (incumbent) for MKPS.

Step 4. Determine a branching order for the y variables.

Step 5. Decide which y variable to fix in current node.

Step 6. Generate a new node by solving a sub-problem with y fixed to one; save this node if its bound is better than the incumbent solution. If all y are fixed, then solve a set of knapsack problems and update the incumbent solution if possible.

Step 7. Generate a new node by solving a sub-problem with y fixed to zero; save this node if its bound is better than the incumbent solution. If all y are fixed, then solve a set of knapsack problems and update the incumbent solution if possible.

Step 8. Choose a candidate node. If none exists, stop, the incumbent solution is optimal; else go to Step 5.

The rest of the paper is organized as following: we discuss Steps 1 and 2 in section 3.2; section 3.3 explains the approach used in Step 3 and section 3.4 presents the remaining steps. Computational experiments are discussed in 3.5 and a summary is given in section 3.6.

3.2. Linear knapsack problems and knapsack problem with setup

We use the linear knapsack problem and linear knapsack problem with setup to obtain an upper bound of MKPS. Let us review these two models firstly.

3.2.1. Linear knapsack problem

The linear knapsack problem is a well known integer program:

$$Max \quad \sum_{j=1}^{n} c_j x_j$$

$$s.t.$$

$$\sum_{j=1}^{n} a_j x_j \le b$$

$$0 \le x_j \le 1, \; j = 1,..n$$

All variables are ordered by non-increasing profit-to-process ratio $c_k \big/ a_k$. By Dantzig's algorithm, if variable t is the first one with $\sum_{k=1}^{t} a_k > b$, then

$$x_j = 1, \; j = 1,..t-1$$

$$x_t = \left. b - \sum_{k=1}^{t-1} a_k \right/ a_t$$

$$x_j = 0, \; j = t+1,..n$$

3.2.2. Algorithm for LKPS

Bulfin (1988) shows LKPS can be transformed to a linear knapsack problem. Consider the LKPS:

$$Max \quad \sum_{i=1}^{N} \sum_{j=1}^{n_i} c_{ij} x_{ij} + \sum_{i=1}^{N} f_i y_i$$

$$s.t.$$

$$\sum_{i=1}^{N} \sum_{j=1}^{n_i} a_{ij} x_{ij} + \sum_{i=1}^{N} d_i y_i \le b$$

$$x_{ij} \le y_{i,} \quad j = 1, n_i; i = 1, N$$

$$x_{ij} \ge 0 \quad j = 1, n_i; i = 1, N$$

$$0 \le y_i \le 1, \; i = 1,..N$$

Bulfin's algorithm uses the ratio $[f_i + \sum c_{ij}] \big/ \left[d_i + \sum a_{ij}\right]$ as a criterion to assign the resource.

Define

$$r_{ij} = c_{ij} \big/ a_{ij}, \quad i = 1,..N \quad j = 1,..n_i .$$

Reorder jobs $1\ldots n_i$, so that $r_{i1} \ge r_{i2} \ge r_{i3} ...... \ge r_{i,n_i}$. Let

$$\frac{\sum_{j=1}^{t_i} c_{ij} + f_i}{\sum_{j=1}^{t_i} a_{ij} + d_i} = \max\{\frac{\sum_{j=1}^{k} c_{ij} + f_i}{\sum_{j=1}^{k} a_{ij} + d_i} \mid k = 1,2,..n_i\} \text{ for } i \in N .$$

Then in family i, jobs are separated into two sets: $XM_i = \{1\ldots t_i\}$ and $XT_i = \{t_i + 1 .... n_i\}$.

The jobs in XM_i can be considered as a single job.

Now for family i, define:

$$c'_{i1} = \sum_{j=1}^{t_i} c_{ij} + f_i$$

$$a'_{i1} = \sum_{j=1}^{t_i} a_{ij} + d_i$$

$$c'_{i,j-t_i+1} = c_{ij} \qquad j = t_i + 1,..n_i$$

$$a'_{i,j-t_i+1} = a_{ij} \qquad j = t_i + 1,..n_i$$

$$n'_i = n_i - t_i + 1$$

Then LKPS can be reformulated as:

$$
\begin{aligned}
&Max \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i'} c_{ij}' z_{ij} \\
&s.t. \\
&\qquad \sum_{i=1}^{N}\sum_{j=1}^{n_i'} a_{ij}' z_{ij} \le b \\
&\qquad 0 \le z_{ij} \le 1, i = 1,..N \;\; j = 1,...n_i'
\end{aligned}
$$

Pseudo job z_{i1} is composed of jobs $x_{i1},..x_{it_i}$ along with the setup cost and time, and $z_{ij} = x_{ij+t_i}$, for $j = 2,..n_i - t_i$. Solve this linear knapsack problem. At most one variable can have a fractional value, say f. If $z_{k1} = f$, then $x_{kj} = f$, $j = 1,..t_k$. If $z_{kl} = f$, $l \ne 1$, then $x_{k\ l+t_k} = f$.

3.2.3. An upper bound on MKPS

3.2.3.1. Relaxation

Surrogate relaxation (Pisinger, 1999) and Lagrangian relaxation (Martello and Toth, 1981) have been applied to obtain an upper bound on the multiple knapsack problem. In this paper, surrogate relaxation with identical multipliers on constraints (1) is used. Selecting identical multipliers keeps a_{ij} unrelated to periods after surrogate relaxation. Relaxing integrality of the x variables gives a mix-integer formulation SMKPS:

$$\begin{aligned}
&Max \quad \sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{j=1}^{n_i} c_{ijt}x_{ijt} + \sum_{t=1}^{T}\sum_{i=1}^{N} f_{it}y_{it} \\
&s.t. \\
&\qquad \sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{j=1}^{n_i} a_{ij}x_{ijt} + \sum_{t=1}^{T}\sum_{i=1}^{N} d_i y_{it} \le \sum_{t=1}^{T} b_t \\
&\qquad \sum_{t=1}^{T} y_{it} \le 1 \quad i=1,..N \\
&\qquad x_{ijt} \le y_{it} \quad i=1,..N \quad j=1,..n_i \quad t=1,..T \\
&\qquad 0 \le x_{ijt} \le 1 \quad i=1,..N \quad j=1,..n_i \quad t=1,..T \\
&\qquad y_{it} \in \{0,1\} \quad i=1,..N \quad t=1,..T
\end{aligned}$$

SMKPS gives an upper bound on MKPS since every solution to MKPS is a feasible solution for SMKPS, but not vice versa. Unlike the usual approach, we do not solve SMKPS to obtain an upper bound on MKPS; we design a new knapsack problem based on SMKPS whose optimal solution is an upper bound on MKPS

3.2.3.2. The knapsack problem giving the upper bound of MKPS

Using only the variables of family i in period t of SMKPS, we construct the linear knapsack problem with setup:

$$\begin{aligned}
&Max \quad \sum_{j=1}^{n_i} c_{ijt}x_{ij} + f_{it}y_i \\
&s.t. \\
&\qquad \sum_{j=1}^{n_i} a_{ij}x_{ij} + d_i y_i \le b \\
&\qquad x_{ij} \le y_{i,} \quad j=1,n_i \\
&\qquad x_{ij} \ge 0 \quad j=1,n_i \\
&\qquad 0 \le y_i \le 1,
\end{aligned}$$

Based on Bulfin's algorithm, this formulation can be transformed to a linear knapsack problem with pseudo variables $\bar{z}_1 ,...\bar{z}_{n_{it}}$ and their corresponding profit and processing coefficients $\bar{c}_1 ,...\bar{c}_{n_{it}}$ $\bar{a}_1 ,...\bar{a}_{n_{it}}$. Pseudo variables are ordered by non-increasing ratio $\bar{c}_j / \bar{a}_j$.

We define a set $P_{it} = \{(0,0), (\sum_{j=1}^{t} \bar{a}_j , \sum_{j=1}^{t} \bar{c}_j) \mid t = 1,..n_{it}\}$ from these pseudo coefficients. For the sake of brevity, record these points are $p_0, ...p_{n_{it}}$ with $p_t.x = \sum_{j=1}^{t} \bar{a}_j$ and $p_t.y = \sum_{j=1}^{t} \bar{c}_j$.

We can construct T point sets P_{it} for $t = 1,..T$. Let $P_i^{'} = \bigcup_{t=1}^{T} P_{it}$ and delete any repeated points. Order all points by non-decreasing $p.x$. Apply the following rules to delete points from $P_i^{'}$.

1. If p_r and p_s have $p_r.x \le p_s.x$ and $p_r.y \le p_s.y$, then delete p_s

2. If p_r, p_k, p_s have $p_r.x \le p_k.x \le p_s.x$ and $p_r.y \le p_k.y \le p_s.y$, and

$$\frac{p_k.y - p_r.y}{p_k.x - p_r.x} \le \frac{p_s.y - p_k.y}{p_s.x - p_k.x}$$, then delete p_k .

These two dominance rules are called multiple-choice dominance rules in this paper, and stems from the two dominance rules for the multiple-choice knapsack problem (Sinha and Zoltners, 1979). Assume there are $n_i^{'} + 1$ points $p_0, ...p_{n_i^{'}}$ ($p_0 = (0,0)$) remained, ordered by increasing $p.x$ and $p.y$. We can define $n_i^{'}$ pseudo variables $z_{i1},..z_{in_i^{'}}$ by setting $a_{ij}^{'} = p_j.x - p_{j-1}.x$ and $c_{ij}^{'} = p_j.y - p_{j-1}.y$.

Repeating this process for all N families, we obtain $\sum_{i=1}^{N} n_i^{'}$ pseudo variables and a linear knapsack problem K1 with resource b ($b = \sum_{k=1}^{T} b_k$):

$$
\begin{aligned}
&Max \quad \sum_{i=1}^{N} \sum_{j=1}^{n_i^{'}} c_{ij}^{'} z_{ij} \\
&s.t. \\
&\qquad \sum_{i=1}^{N} \sum_{j=1}^{n_i^{'}} a_{ij}^{'} z_{ij} \le b \\
&\qquad 0 \le z_{ij} \le 1, i = 1,..N \ \ j = 1,..n_i^{'}
\end{aligned}
$$

We prove the optimal objective of K1 is an upper bound on MKPS in Appendix B.

3.3. Feasible solution (lower bound)

A good initial feasible solution can fathom many candidate nodes and reduce the search time. We will use a greedy algorithm to calculate one.

Algorithm $assign(i,t)$ determines a feasible assignment of family i 's jobs to period t when there is b_t resource available. The algorithm returns obj_{it} , the total profit of this assignment and res_{it} , the amount of resource actually used.

Algorithm $assign(i,t)$:

Step 1. Set $b = b_t$, $obj_{it} = 0, res_{it} = 0$, $j = 0$

$$obj_{it} \leftarrow obj_{it} + f_{it} , b \leftarrow b - d_i , res_{it} = d_i .$$

Step 2. $j \leftarrow j+1$; if $j > n_i$, stop.

Step 3. If $b \ge a_{ij}$, then

$$obj_{it} \leftarrow obj_{it} + c_{ijt} , b \leftarrow b - a_{ij} , res_{it} \leftarrow res_{it} + a_{ij} ;$$

If $b > 0$, then go to Step 2; else stop.

else go to Step 2

Algorithm *feas* is used to get the feasible solution. It uses obj_{it} to assign a family to a period and update the available resource. It continues until there is not enough resource left for any job.

Step 1. Set $NN = \{1,..N\}$. Solve $assign(i,t)$ for $i = 1,..N$ $t = 1,..T$

Step 2. Choose $\{r,s\}$ with $obj_{rs} = \max\{obj_{it} \mid i \in NN, t = 1,..T\}$; if $obj_{rs} = 0$, stop.

Step 3. $lb \leftarrow lb + obj_{rs}$, $b_s \leftarrow b_s - res_{rs}$. Delete r from NN. If $NN = \phi$, stop.

Step 4. Solve $assign(i,s)$ $i \in NN$. Go to Step 2.

3.4. Branch-and-bound algorithm

To develop a branch-and-bound algorithm, we need to make several decisions. These include how to fix variables, calculate bounds, choose the next sub-problem to explore and obtain an initial incumbent solution. Also, the order to fix variables has to be decided.

3.4.1. Variable order

Order all y_{it} variables by non-increasing $pro_{it} = \sum_{j=1}^{n_i} c_{ijt} + f_{it}, i = 1,..N, t = 1,..T$. If y_{it} is near the front, then this variable is more likely to be one. Similarly if y_{it} is near the end, it is more likely to be zero. Fixing variables first at the front or rear aid in keeping the number of branches small. We fix y_{it} variables by looking at the beginning and end of this ordered list and working toward the middle. So family i $y_{it}, t = 1,..T$ has a search order: if y_{it} is the k^{th} variable of the T variables related to family i, then set $o(y_{it}) = k$.

In the current node, we decide which variable will be fixed based on all variables fixed. Assume we fix y_{rk} . Since each family is assigned to at most one period, then $y_{rt} = 0$ for $o(y_{rt}) < o(y_{rk})$ $t = 1,..T$.

3.4.2. Fixing y_{rk}

As we proceed through branch-and-bound algorithm, we fix setup variables to zero or one. If y_{rk} is free, family r is represented by pseudo variables z_{rj}, $j = 1,..n'_r$; these variables are never fixed. If y_{rk} is fixed at one, all pseudo variables z_{rj}, $j = 1,..n'_r$ are removed and real variables x_{rjk}, $j = 1,..n_r$ are included in K1; x_{rjk} are always free in the branch-and-bound algorithm. If y_{rk} is fixed at zero, all pseudo variables as well as their coefficients are recalculated, excluding the possibility of family r being setup in period k , and included in K1; again the new z_{rj}, $j = 1,..n'_r$ are always free. When all $y_{it}, i = 1,..N$ $t = 1,..T$ are fixed to either zero or one, a knapsack problem over the appropriate x_{ijt} is solved to determine the optimal solution.

When y_{rk} is fixed to one, the bounding problem K1 changes as follows:

the actual setup cost for family r in period t is added to the objective;

the actual setup time for family r is subtracted from the surrogate constraint;

pseudo variables for family r are removed, and

real variable x_{rjk}, $j = 1,..n_r$ are added

When y_{rk} is fixed to zero, the changes are removing pseudo variables for family r and adding new pseudo variables.

We also tighten the relaxation by adding the constraint for period k to the bounding problem. Pseudo variables z_{it} will only use the surrogate resource, but x_{ijk} variables will use both the surrogate resource and the resource from period k . Subtracting the setup time will reduce the available surrogate resource and will reduce the resource from period k . Removing pseudo variables may increase the surrogate resource, but will not affect the resource for period k . Thus, the previous optimal solution to the bounding problem may no longer be feasible or optimal. We could re-solve it from scratch, but we will show how to adjust the old solution to obtain the optimal solution to the new bounding problem. First, we introduce some notations.

Let

obj : The current node's upper bound,

$G_t = \{i \mid y_{it} = 1\}$: Family fixed to period t ,

$U = \{i \mid y_{it} \ is \ \ free\}$: Family free,

$b^{'} = b - \sum_{t=1}^{T} \sum_{i \in G_t} d_i$: Available resource for all variables,

$b_t^{'} = b_t - \sum_{i \in G_t} d_i, t = 1,..T$: Available resource for families in period t .

The current node's upper bound is the optimal objective of this formulation, K2. It can be proved by an approach similar to what we used in Appendix A.

$$Max \quad \sum_{i\in U}\sum_{j=1}^{n_i^{'}} c_{ij}^{'} z_{ij} + \sum_{t=1}^{T}\sum_{i\in G_t}\sum_{j=1}^{n_i} c_{ijt} x_{ijt} + \sum_{t=1}^{T}\sum_{i\in G_t} f_{it}$$

$s.t.$

$$\sum_{i\in U}\sum_{j=1}^{n_i^{'}} a_{ij}^{'} z_{ij} + \sum_{t=1}^{T}\sum_{i\in G_t}\sum_{j=1}^{n_i} a_{ij} x_{ijt} \le b^{'}$$

$$\sum_{i\in G_t}\sum_{j=1}^{n_i} a_{ij} x_{ijt} \le b_t^{'}, t = 1,..T$$

When we fix y_{rk} to one, set

$$b_k^{'} = b_k^{'} - d_r$$

$$b^{'} = b^{'} - d_r$$

$$obj = obj + f_{rk}$$

$$G_k = G_k \cup \{r\}$$

$U = U \setminus \{r\}$ and

$$y_{rt} = 0, t = 1,..T\ t \ne k$$

The algorithm to fix y_{rk} to one can be separated into three steps:

*Step 1***.** Delete pseudo variables $z_{r1},...z_{rn_r^{'}}$ from the outer knapsack.

Step 2. Restore feasibility (if necessary).

Step 2.1. Set $b_k^{'} \leftarrow b_k^{'} - d_r$. If $\sum_{i\in G_k}\sum_{j=1}^{n_i} a_{ij} x_{ijk} > b_k^{'}$, find the variable x_{ijk} greater than zero

with smallest ratio. Decrease it until either it is zero or $\sum_{i\in G_k}\sum_{j=1}^{n_i} a_{ij} x_{ijk} = b_k^{'}$.

Repeat until $\sum_{i\in G_k}\sum_{j=1}^{n_i} a_{ij} x_{ijk} = b_k^{'}$ is achieved.

Step 2.2. Set $b^{'} \leftarrow b^{'} - d_r$. If $\sum_{i \in U} \sum_{j=1}^{n_i} a_{ij}^{'} z_{ij} + \sum_{t=1}^{T} \sum_{i \in G_t} \sum_{j=1}^{n_i} a_{ij} x_{ijt} > b^{'}$, repeat the procedure in 2.1, except choose either x_{ijt} or z_{ij} variables greater than zero with smallest ratio.

Step 3. Restore optimality.

Step 3.1. Let $V = \{x_{ijt} \mid x_{ijt} = 0, i \notin U\} \cup \{z_{ij} \mid z_{ij} = 0, i \in U\}$ be the set of all zero-value variables. Find the maximum ratio $r_{\max}$ of all variables in V.

Step 3.2. Set all fractional-valued variables and all variables with value one and ratio less than $r_{\max}$ to zero. Put these variables in V in the proper ratio order. This releases resource for new variables to use. Variables in K2 now have value one only and their ratio is no worse than $r_{\max}$

Step 3.3. Do the following sub-algorithm to obtain the optimal solution of K2.

```
Step a. Set $k = 1$.
Step b. If the $k^{th}$ variable in $V$ is $x_{ijt}$, then go to Step c; else the $k^{th}$ variable
    in $V$ is $z_{ij}$, and go to Step d.
Step c. If $b_t^{'} = 0$, then $k \leftarrow k + 1$;
    else
        If $b_t^{'} \geq a_{ij}$, then $b_t^{'} \leftarrow b_t^{'} - a_{ij}$, $b^{'} \leftarrow b^{'} - a_{ij}$, $obj \leftarrow obj + c_{ijt}$,
            and $x_{ijt} = 1$;
        else $x_{ijt} = b_t^{'} / a_{ij}$, $b^{'} \leftarrow b^{'} - b_t^{'}$, and $obj \leftarrow obj + c_{ijt} x_{ijt}$.
    Go to Step e.
```

Step d. If $b' \geq a'_{ij}$, then $b' \leftarrow b' - a'_{ij}$, $obj \leftarrow obj + c'_{ij}$, and $z_{ij} = 1$.

Set $k \leftarrow k+1$.

else $z_{ij} = b' / a'_{ij}$, $b' = 0$, and $obj \leftarrow obj + c'_{ij} z_{ij}$.

Step e. If $b' > 0$ and $k \leq \|V\|$ go to Step b; else stop.

When we fix y_{rk} to zero, pseudo variables $z_{r1},..z_{rn'_i}$ are deleted from variable set.

Since $y_{rt} = 0, o(y_{rt}) < o(y_{rk}), t = 1,..T$, update $P'_i = \bigcup_{o(y_{rt}) > o(y_{rk})} P_{it}$. Apply the multiple-choice dominance rules to delete dominated points. Use the remaining points to obtain the updated pseudo variables $z_{r1},..z_{rn'_i}$. We can resolve the problem with new variable set to obtain the upper bound of the sub-problem with $y_{rk} = 0$. In this case, only steps 1 and 3 are needed to resolve the problem.

3.4.3. Choosing a new sub-problem

When variables are fixed, two sub-problems are created. If a sub-problem's upper bound is no better than an incumbent solution it is discarded. When its bound indicates it could contain a better solution to MKPS we store it in a bucket. Each bucket contains sub-problems with bounds that are about the same. Let UB be the best upper bound and INC be the value of the current incumbent solution. If we want K buckets, calculate

$$\Delta = (UB - INC) / K.$$

Then bucket one will contain all sub-problems with upper bounds in the interval $[UB - \Delta, UB]$, bucket two $[UB - 2\Delta, UB - \Delta]$, and bucket K $[INC, INC + \Delta]$.

Buckets can be updated as upper bounds or the incumbent change. When we choose a new sub-problem to explore, we take one based on LIFO from the lowest numbered, non-empty bucket. This gives almost a “best-bound” strategy, but without the bookkeeping overhead.

3.5. Computational experiments

We test AMKPS on a variety of problem instances to see what problems can be solved in reasonable time. Instances are generated by setting four parameters at several levels. The parameters are average number of jobs in a family, number of periods, proportion of setup time/cost relative to totals, and resource tightness. The number of families is fixed to ten ($N = 10$). The number of jobs in a family is integer uniformly distributed from three intervals [40, 50], [60, 70] and [80, 90]. The number of periods will be either five or seven, corresponding to a work week. Setup cost and time are determined by

$$f_{it} = -e_1(\sum_{j=1}^{n_i} c_{ijt})$$

$$d_i = e_2(\sum_{j=1}^{n_i} a_{ij})$$

We choose e_1 and e_2 uniformly from [0.15, 0.25], [0.25, 0.35], [0.35, 0.45], and [0.45, 0.55]. Resource availability is determined by $b_t = {(\sum_{i=1}^{N}\sum_{j=1}^{n_i} a_{ij})} \big/ {K}$, where K is 10, 7.5 or 5. Finally, c_{ijt} and a_{ij} are random integers chosen from[10, 10000].

For each level of the four factors we generate ten instances. AMKPS was coded in C and all instances were run on a Dell P.C. with 1.7G Intel processor and 512M bytes of memory. In the following tables, we report the minimum (MIN), average (AVG) and maximum (MAX) solution time in minutes. A zero indicates less than one minute of computational time. We also give the average ratio of initial solution (INC) to initial upper bound (UB) and the average ratio of initial solution to the optimal solution (OPT). Table 3.1 gives results for five period problems and Table 3.2 is for seven periods.

Table 3.1
Solution time (minute) for AMKPS for 5 periods

		[40, 50]					[60 70]					[80, 90]				
Resource	Setup	INC/UB	INC/OPT	MAX	AVG	MIN	INC/UB	INC/OPT	MAX	AVG	MIN	INC/UB	INC/OPT	MAX	AVG	MIN
K=10	[0.15 0.25]	0.84	0.87	0	0	0	0.85	0.88	0	0	0	0.85	0.88	0	0	0
	[0.25 0.35]	0.93	0.98	0	0	0	0.94	0.99	0	0	0	0.95	1.00	0	0	0
	[0.35 0.45]	0.96	0.99	0	0	0	0.97	0.99	0	0	0	0.98	0.99	0	0	0
	[0.45 0.55]	0.92	0.99	0	0	0	0.92	0.99	0	0	0	0.91	0.99	0	0	0
	Average	0.91	0.96	0	0	0	0.92	0.96	0	0	0	0.92	0.97	0	0	0
K=7.5	[0.15 0.25]	0.73	0.74	0	0	0	0.72	0.73	0	0	0	0.71	0.72	0	0	0
	[0.25 0.35]	0.82	0.89	0	0	0	0.81	0.87	1	0	0	0.80	0.85	1	0	0
	[0.35 0.45]	0.89	0.98	0	0	0	0.89	0.99	1	0	0	0.89	0.99	1	0	0
	[0.45 0.55]	0.94	0.99	0	0	0	0.95	0.99	0	0	0	0.96	1.00	0	0	0
	Average	0.85	0.90	0	0	0	0.84	0.90	0	0	0	0.84	0.89	0	0	0
K=5	[0.15 0.25]	0.94	0.95	0	0	0	0.94	0.95	0	0	0	0.94	0.95	0	0	0
	[0.25 0.35]	0.86	0.88	0	0	0	0.89	0.90	0	0	0	0.89	0.90	0	0	0
	[0.35 0.45]	0.76	0.79	0	0	0	0.78	0.80	0	0	0	0.76	0.78	0	0	0
	[0.45 0.55]	0.72	0.82	1	1	0	0.71	0.80	3	1	1	0.70	0.80	8	3	2
	Average	0.82	0.86	0	0	0	0.83	0.86	1	0	0	0.82	0.86	2	1	0

Table 3.2
Solution time (minute) for AMKPS for 7 periods

		[40, 50]					[60 70]					[80, 90]				
Resource	Setup	INC/UB	INC/OPT	MAX	AVG	MIN	INC/UB	INC/OPT	MAX	AVG	MIN	INC/UB	INC/OPT	MAX	AVG	MIN
K=10	[0.15 0.25]	0.87	0.92	2	1	0	0.86	0.91	5	3	1	0.86	0.90	9	5	2
	[0.25 0.35]	0.94	0.99	0	0	0	0.94	0.99	0	0	0	0.95	0.99	1	0	0
	[0.35 0.45]	0.96	0.99	0	0	0	0.96	0.99	0	0	0	0.97	0.99	0	0	0
	[0.45 0.55]	0.88	0.98	0	0	0	0.88	0.99	0	0	0	0.87	0.99	0	0	0
	Average	0.91	0.97	1	0	0	0.91	0.97	1	1	0	0.91	0.97	3	1	0
K=7.5	[0.15 0.25]	0.80	0.86	7	3	1	0.79	0.85	12	6	2	0.78	0.85	26	17	10
	[0.25 0.35]	0.82	0.91	11	4	1	0.83	0.91	22	10	2	0.81	0.90	29	19*	13
	[0.35 0.45]	0.90	0.98	2	1	0	0.91	1.00	12	3	0	0.91	0.99	10	4*	1
	[0.45 0.55]	0.94	0.99	0	0	0	0.95	1.00	0	0	0	0.96	1.00	0	0	0
	Average	0.87	0.94	5	2	0	0.87	0.94	12	5	1	0.87	0.94	16	10	6
K=5	[0.15 0.25]	0.95	0.98	0	0	0	0.95	0.98	0	0	0	0.96	0.98	0	0	0
	[0.25 0.35]	0.90	0.95	1	0	0	0.91	0.96	1	0	0	0.90	0.95	3	1	0
	[0.35 0.45]	0.82	0.91	3	1	0	0.83	0.92	8	3	1	0.82	0.91	11	4	2
	[0.45 0.55]	0.75	0.89	9	4	1	0.74	0.87	16	9	3	0.74	0.87	19	10	5
	Average	0.86	0.93	3	2	0	0.86	0.93	6	3	1	0.86	0.93	8	4	2

Note: "*" means some instances run out of memory, and the value of AVG in the table is the average of the remaining instances, as are Max and Min.

AMKPS performs very well for five period problems, with the hardest taking less than 8 minutes. Seven period instances are harder, but most instances are solved in less than 30 minutes.

Seven of the 720 instances were not solved by AMKPS. These instances had seven periods, and average of 85 jobs per family, resource tightness of $k = 7$ and setup percentage of [0.25, 0.35] or [0.35, 0.45]. These instances used up memory. The Min, Max, and Average are of the problems actually solved. Solution time increases as number of period and number of jobs increase. We ran some instances with different combinations of numbers of periods and jobs and found that the solution time changes in almost the same way as for the test problems.

The relationships between setup and resource tightness are more complex. Fig. 3.1 to 3.6 demonstrate this.

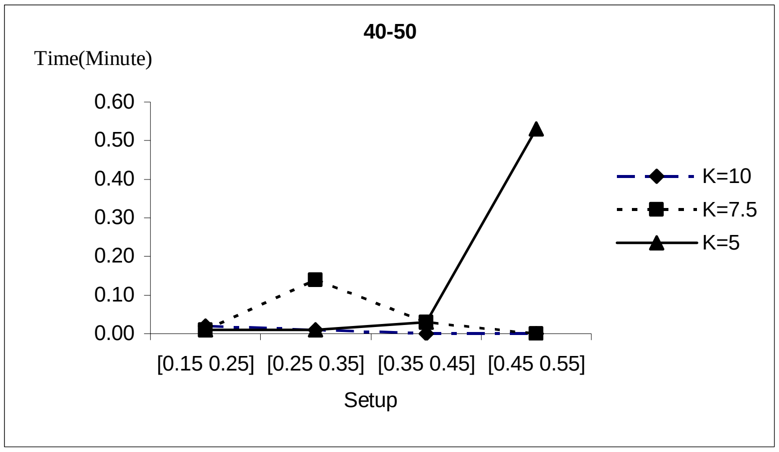

Fig. 3.1. Solution time for average 45 jobs per family and 5 periods

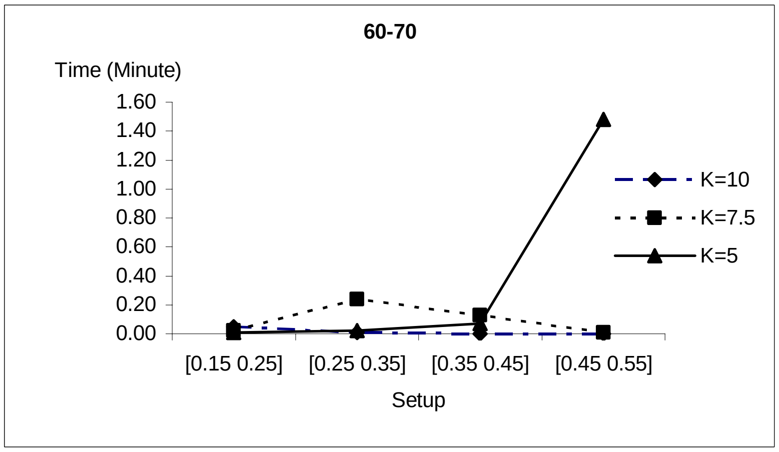

Fig. 3.2. Solution time for average 65 jobs per family and 5 periods

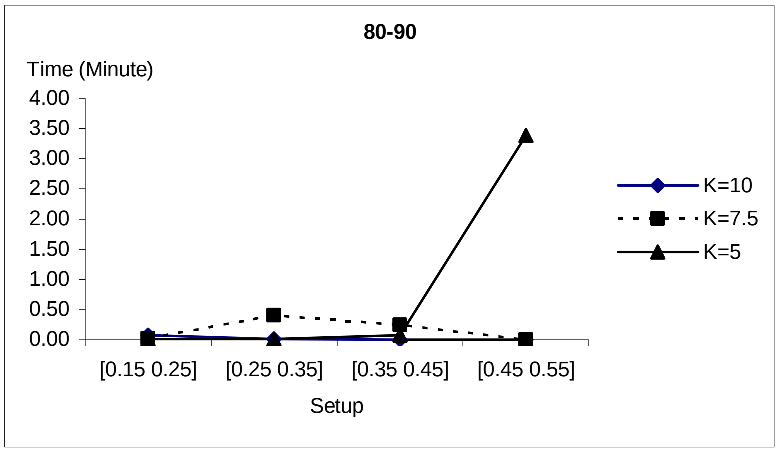

Fig. 3.3. Solution time for average 85 jobs per family and 5 periods

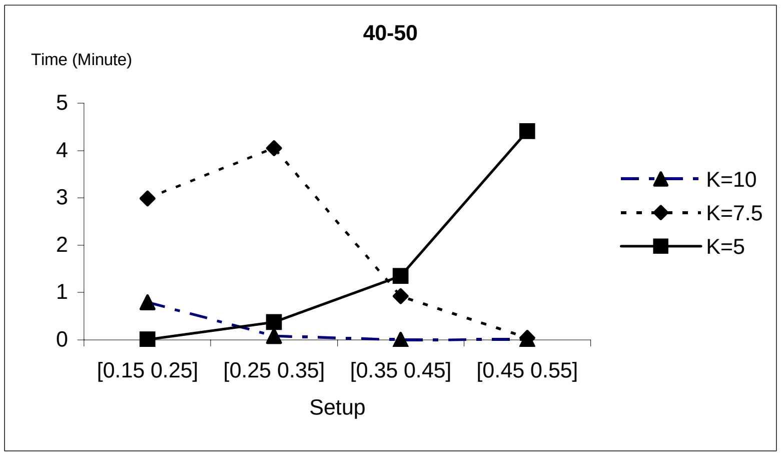

Fig. 3.4. Solution time for average 45 jobs per family and 7 periods

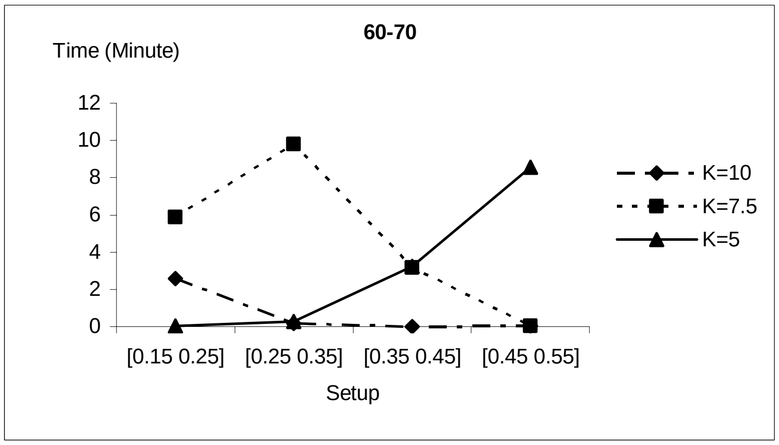

Fig. 3.5. Solution time for average 65 jobs per family and 7 periods

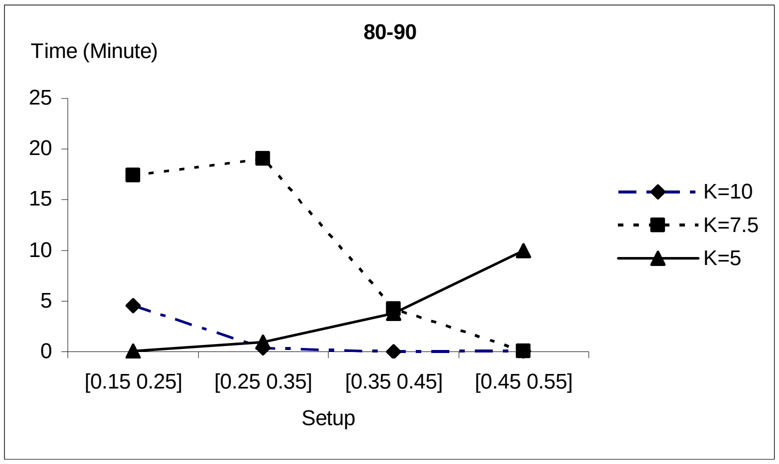

Fig. 3.6. Solution time for average 85 jobs per family and 7 periods

When $K = 10$, the maximum time happens on instances with setup from [0.15, 0.25]; when $K = 7.5$, the maximum time happens on instances with setup from [0.25, 0.35]; when $K = 5$, instances with setup from [0.45, 0.55] use the most time. From these plots, we conclude that problems become difficult when $e * K \sim (2,3)$.

The heuristic algorithm given in this paper is very effective, especially when the resources are tight. We give the quality (average proportion of lower bound to initial upper bound and to optimal solution) in Table 3.3. The quality decreased as resources increase in each period. For both five and seven period problems, the heuristic is good, typically in the 85%-95% range.

Table 3.3
The lower bound, upper bound and optimal solution

			[40 50]		[60, 70]		[80, 90]	
Period	Resource	Setup	INC/UB	INC/OPT	INC/UB	INC/OPT	INC/UB	INC/OPT
Period 5	K=10	Average	0.91	0.96	0.92	0.96	0.92	0.97
	K=7.5	Average	0.85	0.90	0.84	0.90	0.84	0.89
	K=5	Average	0.82	0.86	0.83	0.86	0.82	0.86
Period 7	K=10	Average	0.91	0.97	0.91	0.97	0.91	0.97
	K=7.5	Average	0.87	0.94	0.87	0.94	0.87	0.94
	K=5	Average	0.86	0.93	0.86	0.93	0.86	0.93

We also compare AMKPS to CPLEX 9.1 (called by AMPL). We choose the hardest instances for AMKPS (7 periods and $n_i \sim [80,90]$) to compare. Trial runs on other instances showed these results are typical. Due to the difficulty of solving with CPLEX, only five instances per level were solved. Table 3.4 shows the clear superiority of AMKPS. CPLEX solved very few problems in less than two hours; we let one solve until the optimal solution is obtained, and it took over 29 hours.

Table 3.4
The comparison of solution time (Minute) between AMKPS and CPLEX

Setup	K=10		K=7.5		K=5	
	CPLEX	AMKPS	CPLEX	AMKPS	CPLEX	AMKPS
	*	2	*	13	26	0
	*	11	*	16	7	0
[0.15,0.25]	*	5	*	13	10	0
	*	4	*	30	9	0
	*	12	*	12	8	0
AVG	*	7	*	17	12	0
	*	1	116	21	36	0
	*	0	*	10	*	1
[0.25, 0.35]	*	0	*	*	77	1
	*	0	*	28	26	0
	*	0	*	*	*	2
AVG	*	0	*	*	*	1
	*	0	*	6	*	2
	*	0	*	2	*	3
[0.35, 0.45]	*	0	*	*	*	4
	*	0	*	6	*	3
	*	0	*	2	*	9
AVG	*	0	*	*	*	4
	*	0	19	0	*	14
	*	0	14	0	*	22
[0.45, 0.55]	*	0	4	0	*	27
	*	0	22	0	*	16
	*	0	7	0	*	11
AVG	*	0	13	0	*	18

Note: * means the instance uses more than 2 hours or uses up memory.

AMKPS use less time than CPLEX for all but three instances when $e * K \sim (2,3)$ and n_i is from [80, 90]. We also do experiments with instances with fewer variables and AMKPS also used less considerably time than CPLEX.

3.6. Conclusions

The MKPS model can be used for order selection in make-to-order manufacturing. In this paper, we use branch-and-bound algorithm to solve MKPS and design a new method to get an upper bound on MKPS. Rather than relaxing constraints of the original models to an upper bound, we propose a new linear knapsack model to obtain an upper bound. We prove the knapsack optimal objective solution is an upper bound on MKPS. In branching, we add a resource constraint whose family has been fixed to that to tighten the relaxation. This prohibits jobs from using more than the period capacity. This knapsack problem can still be solved efficiently. We also give an effective greedy heuristic which supplies a good feasible solution as a lower bound. After all *y* variables are branched on, MKPS is transformed to knapsack problems. The computational experiments show that AMKPS works well with a tight resource limit. Sixty seven-period instances are tested to compare AMKPS with CPLEX: AMKPS solve 57 instances of them in less than 30 minutes but CPLEX fail in 46 instances and need more time than AMKPS for the remaining 14 instances. In this paper, we only use a simple branch-and-bound algorithm for the knapsack problem when all setup variables are fixed. If a better algorithm, e.g. the one developed by Martello et al. (1999) is used, the solution time can be reduced.

Appendix A. The optimal objective of K1 is the upper bound on MKPS

Before proving the proposition, we need the following Lemma:

Lemma 1. A linear knapsack problem can be transformed to a concave piecewise function

Proof.

For knapsack problem

$$
\begin{aligned}
&Max \quad \sum_{j=1}^{n} c_j x_j \\
&s.t. \\
&\qquad \sum_{j=1}^{n} a_j x_j \le b \\
&\qquad 0 \le x_j \le 1,\ j = 1,..n
\end{aligned}
$$

Order all variables by non-increasing ratio c_j / a_j. Define a point set $P = \{(0,0), (\sum_{j=1}^{t} a_j, \sum_{j=1}^{t} c_j) \mid t = 1,..n\}$. Put these points on coordinates and connect the adjacent points, we can obtain a concave piecewise function F and these points are the breaking points of the piecewise function. $F(b)$ is the optimal objective of the linear knapsack problem.

If $b \ge \sum_{j=1}^{n} a_j$, set $F(b) = \sum_{j=1}^{n} c_j$;

else $x_t = 1$ if $\sum_{j=1}^{t} a_j < b$

$$x_t = \left(b - \sum_{j=1}^{t-1} a_j\right) \Big/ a_t \quad \text{if} \quad \sum_{j=1}^{t-1} a_j \le b < \sum_{j=1}^{t} a_j$$

$$x_t = 0 \text{ if } \quad b \le \sum_{j=1}^{t-1} a_j$$

On the verse, if we know $F(b)$, we can construct an equivalent knapsack problem for this piecewise function.

Proposition 1. The optimal objective of K1 is the upper bound on MKPS

Proof.

The coefficients of variables from family i in period t $f_{it}, c_{i1t}, \ldots c_{in_it}, d_i, a_{i1}, \ldots a_{in_i}$ construct a linear knapsack problem with setup, say $LKPS_{it}$:

$$\begin{aligned}
&Max \quad \sum_{j=1}^{n_i} c_{ijt} x_{ij} + f_{it} y_i \\
&s.t. \\
&\qquad \sum_{j=1}^{n_i} a_{ij} x_{ij} + d_i y_i \le b_0 \\
&\qquad x_{ij} \le y_{i,} \quad j = 1, n_i \\
&\qquad x_{ij} \ge 0 \quad j = 1, n_i \\
&\qquad 0 \le y_i \le 1,
\end{aligned}$$

Based on Bulfin's algorithm, this formulation can be transformed to a linear knapsack problem with pseudo variables $z_{i1}, \ldots z_{in_i'}$ and their corresponding profit and processing coefficients c'_{ij} , a'_{ij} $\quad j = 1, ..n'_i$. Pseudo variables are ordered by non-increasing ratio $c'_{ij} \big/ a'_{ij}$.

Then we can obtain a piecewise function F_{it} with its breaking point set P_{it} so that for any available resource b_0, $F_{it}(b_0)$ is the optimal solution of the $LKPS_{it}$.

We define $P_i^{'} = \bigcup_{t=1}^{T} P_{it}$, and delete all dominated points by two multiple-choice dominance rules for linear multiple-choice knapsack problem (Sinha and Zoltners, 1979). Connecting the remaining points, we can obtain another piecewise function F_i , which has $F_i(b_0) \geq F_{it}(b_0),\ b_0 > 0$.

If the optimal solution of MKPS is known, assume $y_{it} = 1$ and the resources and profit from family i are w_i and $profit_i, i = 1,..N$ with solution set $S_i = \{x_{ijt} \mid x_{ijt} = 1\}$. Then S_i is a feasible solution from the following linear knapsack problem ($LKPS_1$) with setup and $profit_i$ is the objective of the feasible solution

$$
\begin{aligned}
&Max \quad \sum_{j=1}^{n_i} c_{ijt} x_{ij} + f_{it} y_i \\
&s.t. \\
&\qquad \sum_{j=1}^{n_i} a_{ij} x_{ij} + d_i y_i \leq w_i \\
&\qquad x_{ij} \leq y_{i,} \quad j = 1, n_i \\
&\qquad x_{ij} \geq 0 \quad j = 1, n_i \\
&\qquad 0 \leq y_i \leq 1,
\end{aligned}
$$

Since $F_{it}(w_i)$ is the optimal objective of $LKPS_1$, thus $F_{it}(w_i) \geq profit_i, i = 1,..N$.

Since $F_i(w_i) \geq F_{it}(w_i)$, then $F_i(w_i) \geq profit_i, i = 1,..N$ and $\sum_{i=1}^{N} F_i(w_i) \geq \sum_{i=1}^{N} profit_i$. Set

$$
\begin{aligned}
&z_{ij} = 1,\ j = 1,..k \ \text{ if} \sum_{j=1}^{k_i} a_{ij}^{'} \leq w_i < \sum_{j=1}^{k_i+1} a_{ij}^{'} \\
&z_{ik_i+1} = \left. w_i - \sum_{j=1}^{k} a_{ij}^{'} \middle/ a_{ik+1}^{'} \right. \ \text{ if } \ \sum_{j=1}^{t-1} a_j \leq b < \sum_{j=1}^{t} a_j \\
&z_{ij} = 0,\ j > k+1
\end{aligned}
$$

Then the solution set z_{ij}, $j = 1,..n_i^{'}, i = 1,..N$ is a feasible solution of K1 since $\sum_{i=1}^{N} w_i \leq b$.

Hence the optimal solution of K1 is greater or equal to $\sum_{i=1}^{N} F_i(w_i)$, and an upper bound on MKPS.

References

Akinc, U. 2004. Approximate and exact algorithm for the fixed-charge knapsack problem, European Journal of Operational Research 170, 363-375.

Bulfin, R. L. 1988. An algorithm for the continous, variable upper bound knapsack problem, OPSEARCH 25 (2), 119-125.

Dantzig, G.B. 1957. Discrete variable extremum problems, Operation Research 5, 266-277.

Martello, S., Pisinger, D., Toth, P. 1999. Dynamic programming and strong bounds for the 0-1 Knapsack Problem. Management Science 45 (3), 414-424.

Martello, S., Toth, P. 1980. Solution of the zero-one multiple knapsack problem, European Journal of Operational Research 4, 276-283.

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

Pisinger, D. 1995. A minimal algorithm for the multiple-choice knapsack problem. European Journal of Operational Research 83, 394-410.

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

Sinha, A., Zoltners, A.A. 1979. The multiple-choice knapsack problem, Operations Research 27, 503-515.

Ⅳ.MULTIPLE-CHOICE KNAPSACK PROBLEM WITH SETUP

Abstract

We present a multiple-choice knapsack problem with setup (MCKS). This model can be applied to regional project selection in multiple periods. In the model, some variables model setups and serve as the upper bound on the remaining ones. A linear knapsack problem is designed to give an upper bound on MCKS, and a branch-and-bound algorithm is used to optimally solve MCKS. Setup variables are branched on; when all are fixed, MCKS becomes a knapsack problem. Computational experiments show this algorithm is effective even for instances CPLEX can not solve in two hours.

4.1. Introduction and literature review

The multiple-choice knapsack problem (MCK) is well known in combinational optimization. In this paper, we present a model we call a multiple-choice knapsack problem with setup (MCKS). This model can be used in regional project selection in multiple periods for an organization (country or company) which has a fixed budget to invest in a number of projects in multiple areas which can be done in multiple periods. To do a project in an area, a project office must be set up. The organization would like to decide where to set up offices and which projects to do to maximize net present value subject to a budget restriction.

Given the formulation of MCKS:

$$Max \quad \sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{j=1}^{n_i} c_{ijt}x_{ijt} + \sum_{t=1}^{T}\sum_{i=1}^{N} f_{it}y_{it}$$

$s.t.$

$$\sum_{t=1}^{T}\sum_{i=1}^{N}\sum_{j=1}^{n_i} a_{ij}x_{ijt} + \sum_{t=1}^{T}\sum_{i=1}^{N} d_i y_{it} \le b\,, \tag{1}$$

$$x_{ijt} \le y_{it} \quad j=1,..n_i,\ \ i=1,..N;\ \ t=1,..T\,, \tag{2}$$

$$\sum_{t=1}^{T} x_{ijt} \le 1 \quad i=1,...N, \quad j=1,..n_i\,, \tag{3}$$

$$x_{ijt}, y_{it} \in \{0,1\} \quad i=1,..N;\ j=1,..n_i;\ t=1,..T. \tag{4}$$

c_{ijt} -is the profit of project j in area i in period t ($c_{ijt} \ge 0$),

f_{it} -is the setup cost for opening an office in area i in period t ($f_{it} \le 0$),

a_{ij} -is the investment needed for project j in area i ($a_{ij} > 0$),

d_i -is the investment cost to open an office in area i ($d_i > 0$),

b -is the budge available to invest ($b > 0$),

y_{it} - is one if office is set up in area i in period t, otherwise zero,

x_{ijt} -is one if project j in area i is done in period t, otherwise zero,

N -is the number of areas,

T -is the number of periods.

Constraint (1) requires the total budget used by all projects and to setup offices can not exceed the budget available. Constraints (2) prohibit a project being done unless the office in this area is set up. Constraints (3) guarantee that a project can only be done once. Constraints (4) require all variables to be binary.

Besides the application of regional development, this model can also be used in order acceptance in multiple periods with a non-renewable resource.

We develop an upper bound and an effective heuristic for MCKS based on the linear knapsack problem with setup and the linear multiple-choice knapsack problem. Following traditional terminology, we call area i family i, and the project j in area i job j of family i. We also call d_i the setup time of family i and f_{it} the setup cost of family i in period t.

For the linear knapsack problem, Dantzig (1957) gave an algorithm which allocates the limited resource to jobs based on the non-increasing profit-to-processing ratio. Without y variables, MCKS becomes a multiple-choice knapsack problem, another well-studied problem. (See Pisinger,1995; Sarin and Karwan, 1989; Armstrong et al, 1983) Two dominance rules for the linear multiple-choice knapsack problem (Sinha and Zoltners, 1979) are used to develop a linear knapsack problem as an upper bound on MCKS.

Without constraint (2), MCKS becomes a knapsack problem with setup. Bulfin (1988) gave an efficient algorithm for its linear relaxation. We explain this algorithm in the following section. Akinc (2004) describes algorithms for a fixed-charge knapsack problem, which is a special case of MCKS; it has a single period and no setup time.

We use a branch-and-bound algorithm to obtain the optimal solution to MCKS. It can be briefly described by two steps. We branch on y variables; when all y variables are fixed, the problem is a knapsack problem in the x variables. We use a simple branch-and-bound algorithm to solve this knapsack problem. To reduce the branches of the tree,

y variables are reordered before branching. The ordering process is as follows: Order y_{it} variables by non-increasing $pro_{it} = \sum_{j=1}^{n_i} c_{ijt} + f_{it}, i = 1,..N, t = 1,..T$. If y_{it} is near the front, then this variable is more likely to be 1. Similarly if y_{it} is near the end, it is more likely to be 0. We fix y_{it} variables by looking at the beginning and end of this ordered list and working toward the middle. Fixing variables first at the front or rear aids in keeping the number of branches small. Renumber variables by this order so that y_{it} will be branched on before y_{it+1}.

The algorithm (AMCKS) for solving MCKS is outlined below:

Step 1. Obtain an upper bound formulation for MCKS and a feasible solution for MCKS.

Step 2. Decide which variable to be fixed in the current node.

Step 2.1. Generate a new node by fixing some *y* to one; save this node if its bound is better than the incumbent solution. If all *y* are fixed, solve a knapsack problem and update the incumbent solution if possible.

Step 2.2. Generate a new node by fixing *y* to zero; save this node if its bound is better than incumbent solution. If all *y* are fixed, then solve a knapsack problem and update the incumbent solution if possible. Delete the current candidate node.

Step 3. Choose a new candidate node. If none exists, stop, the incumbent solution is optimal; else go to Step 2.

In the remaining of the paper, we discuss Step 1 in section 4.2. Section 4.3 explains the algorithms used in Steps 2 and 3. Section 4.4 discusses computational experiments.

4.2. An upper bound and feasible solution

Unlike the usual approaches of relaxing some constraints of a formulation to obtain an upper bound, we design a linear knapsack problem whose optimal objective is an upper bound on MCKS. This approach uses the algorithm presented by Bulfin (1988) for the linear knapsack problem with setup, which transforms a linear knapsack problem with setup to a linear knapsack problem.

4.2.1. Linear knapsack problem

Consider a linear knapsack problem:

$$Max \quad \sum_{j=1}^{n} c_j x_j$$

$$s.t.$$

$$\sum_{j=1}^{n} a_j x_j \le b$$

$$0 \le x_j \le 1$$

All variables are ordered by non-increasing c_j / a_j. By Dantzig's algorithm (1957), if

$$\sum_{j=1}^{k} a_j \le b < \sum_{j=1}^{k+1} a_j \text{ , then}$$

$$x_j = 1,\ j = 1,\ldots k$$

$$x_{k+1} = \left. b - \sum_{j=1}^{k} a_j \middle/ a_{k+1} \right.$$

$$x_j = 0,\ j = k+2,..n$$

A linear knapsack problem corresponds to a concave piecewise function.

Define $P = \{(0,0), (\sum_{j=1}^{k} a_j, \sum_{j=1}^{k} c_j) \mid k = 1,..n\}$, and put these $n+1$ points on coordinates and connect the adjacent points. This defines a concave piecewise function F. This process is independent of resource b. For a given resource b, $F(b)$ is the optimal objective of the linear knapsack problem with resource b (If $b > \sum_{j=1}^{n} a_j$, set $F(b) = \sum_{j=1}^{n} c_j$). For brevity, denote these points as $p_0,..p_n$ with $p_k.x = \sum_{j=1}^{k} a_j$ and $p_k.y = \sum_{j=1}^{k} c_j$ $k = 1,..n$. p_k $k = 1,..n$ are the break points of the piecewise function F. Conversely the n break points of a concave piecewise function F define a linear knapsack problem with some resource b by defining x_j corresponding to $a_j = p_j.x - p_{j-1}.x$ and $c_j = p_j.y - p_{j-1}.y$, $j = 1,..n$.

4.2.2. Transform a linear knapsack problem with setup to a linear knapsack problem

Consider the linear knapsack problem with setup:

$$\begin{aligned}
&Max \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i} c_{ij}x_{ij} + \sum_{i=1}^{N} f_i y_i \\
&s.t. \\
&\qquad \sum_{i=1}^{N}\sum_{j=1}^{n_i} a_{ij}x_{ij} + \sum_{i=1}^{N} d_i y_i \le b, \\
&\qquad x_{ij} \le y_i \quad j = 1, n_i; i = 1, N, \\
&\qquad x_{ij} \ge 0 \quad j = 1, n_i; i = 1, N, \\
&\qquad 0 \le y_i \le 1, \ \ i = 1,..N.
\end{aligned}$$

Bulfin (1988) proposed an efficient algorithm, similar to Dantzig's algorithm for the linear knapsack problem (1957). Reorder all jobs of family i so that $c_{ij}/a_{ij} \geq c_{ij+1}/a_{ij+1}$, $j = 1,..n_i - 1$, $i = 1,..N$. Let

$$\frac{\sum_{j=1}^{t_i} c_{ij} + f_i}{\sum_{j=1}^{t_i} a_{ij} + d_i} = \max\{\frac{\sum_{j=1}^{k} c_{ij} + f_i}{\sum_{j=1}^{k} a_{ij} + d_i} \mid k = 1, 2, ..n_i\} \text{ for } i \in N .$$

Then for family i , jobs can be separated into two sets: XM_i = {1… t_i }and XT_i = { t_i +1.... n_i }. The jobs in XM_i can be considered as a single job.

Now for family i , define:

$$c'_{i1} = \sum_{j=1}^{t_i} c_{ij} + f_i$$

$$a'_{i1} = \sum_{j=1}^{t_i} a_{ij} + d_i$$

$$c'_{i,j-t_i+1} = c_{ij} \qquad j = t_i + 1,..n_i$$

$$a'_{i,j-t_i+1} = a_{ij} \qquad j = t_i + 1,..n_i$$

$$n'_i = n_i - t_i + 1$$

Then linear knapsack problem with setup can be reformulated as:

$$Max \quad \sum_{i=1}^{N} \sum_{j=1}^{n'_i} c'_{ij} z_{ij}$$

$s.t.$

$$\sum_{i=1}^{N} \sum_{j=1}^{n'_i} a'_{ij} z_{ij} \leq b$$

$$0 \leq z_{ij} \leq 1, i = 1,..N \;\; j = 1,...n'_i$$

Pseudo job z_{i1} is composed of jobs $x_{i1}, x_{i2}, ..x_{it_i}$ and $z_{ij} = x_{ij+t_i}$, for $j = 2,..n_i - t_i$. After solving this linear knapsack problem, at most one variable can have a fractional value, say f. If $z_{k1} = f$, then $y_k = f$ and $x_{kj} = f$, $j = 1,..t_k$. If $z_{kl} = f$, $l \neq 1$, then only $x_{k\,l+t_k} = f$.

4.2.3. The algorithm for the upper bound and feasible solution

Before we explain the approach to obtain an upper bound and a feasible solution for MCKS, let us introduce the period subset's piecewise function of family i.

4.2.3.1. Subset's piecewise function

If the optimal solution of MCKS is known, then $S_i = \{t \mid y_{it} = 1\}$ is the set of periods in which family i is processed. But before solving MCKS, S_i is unknown. We know S_i must be a subset of $\{1,..T\}$. Set $\{1,..T\}$ has total $2^T - 1$ non-empty subsets, which we denote as $S_1,..S_K$, $K = 2^T - 1$ and $\|S_k\|$ is the cardinality of the subset S_k.

For any $S_k \subseteq \{1,..T\}, k = 1,..K$, define

$$c_{ijS_k} = \max\{c_{ijt} \mid t \in S_k\}$$

$$f_{iS_k} = \sum_{t \in S_k} f_{it}$$

$$d_{iS_k} = \|S_k\| d_i .$$

Using pseudo variables x'_{ij}, y'_i, for each S_k, we can formulate a linear knapsack problem with setup:

$$
\begin{aligned}
&Max \quad \sum_{j=1}^{n_i} c_{ijS_k} x'_{ij} + f_{iS_k} y'_i \\
&s.t. \\
&\qquad \sum_{j=1}^{n_i} a_{ij} x'_{ij} + d_{iS_k} y'_i \le b_0, \\
&\qquad x'_{ij} \le y'_i \quad j = 1, n_i, \\
&\qquad x'_{ij} \ge 0 \quad j = 1, n_i, \\
&\qquad 0 \le y'_i \le 1.
\end{aligned}
$$

This problem can be transformed to a linear knapsack problem based on section 4.2.2, the linear knapsack problem defines a concave piecewise function F_{iS_k} with break points set P_{iS_k}. Thus F_{iS_k} is the piecewise function of S_k for family i.

4.2.3.2. Upper bound formulation of MCKS

After obtaining F_{iS_k} and its break points set P_{iS_k} for each subset $S_k, k = 1,..K$ of family i, we define $P'_i = \bigcup_{k=1}^{K} P_{iS_k}$ and delete any repeated points in the set. Apply the following two multiple-choice dominance rules (Sinha and Zoltners, 1979) to P'_i :

Dominance rule 1. If p_r and p_s have $p_r.x \le p_s.x$ and $p_r.y \le p_s.y$, then delete p_s

Dominance rule 2. If p_r, p_k, p_s have $p_r.x \le p_k.x \le p_s.x$, $p_r.y \le p_k.y \le p_s.y$ and $p_k.y - p_r.y \big/ p_k.x - p_r.x \le p_s.y - p_k.y \big/ p_s.x - p_k.x$, then delete p_k.

Call the set of remaining points P_i and put them on coordinates. Connecting the adjacent points in P_i, we obtain a concave piecewise function F_i with break points $p_1, ...p_{n'_i}$. All points in P'_i are below the line of the piecewise function F_i thus

$F_i^{'}(b_0) \geq F_{iS_k}(b_0) \; b_0 \geq 0, k = 1,..K$.

Define $n_i^{'}$ pseudo variables $z_{i1},..z_{in_i^{'}}$ with $a_{ij}^{'} = p_j.x - p_{j-1}.x$ and $c_{ij}^{'} = p_j.y - p_{j-1}.y \;\; j = 1,..n_i'$.

Repeating this process for all N families, we obtain $\sum_{i=1}^{N} n_i^{'}$ pseudo variables and formulate a linear knapsack problem LKP_u with resource b .

$$\begin{aligned} &Max \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i^{'}} c_{ij}^{'} z_{ij} \\ &s.t. \\ &\qquad \sum_{i=1}^{N}\sum_{j=1}^{n_i^{'}} a_{ij}^{'} z_{ij} \leq b \\ &\qquad 0 \leq z_{ij} \leq 1, i = 1,..N \;\; j = 1,..n_i^{'} \end{aligned}$$

We prove that the optimal objective of LKP_u is an upper bound on MCKS. This problem has at most one fractional variable. If we round this fractional variable to zero, then we obtain a feasible integer solution for LKP_u . The integer solution of LKP_u corresponds to a feasible solution of MCKS and its objective is a lower bound on MCKS. (Refer to the Appendix C for their proofs).

This approach is impractical if T is large. We present three dominance rules to reduce the number of subsets considered. (Refer to the Appendix D for their proofs).

Consider two subsets $S_r, S_k \subseteq \{1,..T\}$ of family i : if $F_{iS_r}(b_0) \geq F_{iS_k}(b_0)$ for all $b_0 > 0$, then S_r dominates S_k . All break points of F_{iS_k} are below F_{iS_r} , so $P_{iS_k} \not\subset P_i$ which means P_{iS_k} need not be included in P_i' . Consider $S_1,..S_K$ of family i :

Dominance rule 3. If $S_r \subset S_k$ and $\sum_{j=1}^{n} c_{ijS_r} + f_{iS_r} > \sum_{j=1}^{n} c_{ijS_k} + f_{iS_k}$, then S_r dominates S_k .

Dominance rule 4. Assume $S_r \subset S_k$ and S_r dominates S_k. If there is another subset S_l with $S_l \cap S_k = \phi$, then $S_l \cup S_r$ dominates $S_l \cup S_k$.

Dominance rule 5. Assume $S_r \subset S_k$, $f_{iS_r} = f_{iS_k} = 0$ and $d_{iS_r} = d_{iS_k} = 0$. If there is a subset S_l with $S_l \cap S_k = \phi$, then $S_l \cup S_k$ dominates $S_l \cup S_r$.

With the help of dominance rules, the break points of some period subsets' piecewise functions need not be included into P_i' and thus reduce the effect to determine LKP_u. After finding all non-dominated subsets, we calculate the break points of their piecewise functions. We do not put all break points together and apply dominance rules 1 and 2 once; rather we add these points into P_i' in a specific order and apply dominance rules 1 and 2 total T times.

Define $\overline{S}_{ik}$ to be all non-empty subsets of $\{k,..T\}$ for family i and P_{ik}' be all non-dominated points set from the piecewise functions of all elements in $\overline{S}_{ik}$. With the help of Dominance rule 4, the algorithm to generate P_i is:

```
Step 1. Set $k = T - 1$.
Step 2. While ($k > 0$)
    { Set $\overline{S} = \{k \cup S_j \mid S_j \in \overline{S}_{ik}\}$
      Set $\overline{S}_{ik-1} = \overline{S} \bigcup \overline{S}_{ik}$
      Apply dominance rule 3 to delete dominated sets in $\overline{S}_{ik-1}$
      $k \leftarrow k - 1$ }
Step 3. Calculate $F_{iS_k}$ and $P_{iS_k}$ for $S_k \in \overline{S}_{i1}$.
```

Step 4. Set $k = T$ and $P'_{iT+1} = \phi$.

Step 5. While ($k > 0$)

```
    {   Set $P'_{ik} = P'_{ik+1} \cup \{P_{iS_k} \mid S_k \in \overline{S}_{ik} / \overline{S}_{ik+1}\}$
        Apply rules 1 and 2 to delete dominated points in $P'_i$
        $k \leftarrow k-1$   }
```

When the algorithm ends, P'_{i1} is the P_i we need. After P_i, $i = 1,..N$ are known, LKP_u is obtained.

4.3. Fixing y_{it}

When we fix y_{it} to one or zero, $y_{ik}, k = 1,..t-1$ have been fixed by the variable order. Define $S_i^1 = \{k \mid y_{ik} = 1, k \le t\}$ and $\overline{S}_i^1$ includes all non-empty subsets of S_i^1.

4.3.1. Fixing y_{it} to one

When we fix y_{it} to one, d_i is subtracted from b and f_{it} is added to the objective. Then f_{it} and d_i can be viewed as zero. Coefficients f_{iS_k} and d_{iS_k} related to subset $S_k, t \in S_k, S_k \in \overline{S}_{i1}$ change. Therefore all F and P related to these subsets change, which can cause a change of P_i.

We can calculate the new piecewise functions for all affected S_k and apply the above algorithm to obtain an updated P_i. Then we obtain the new pseudo variables and their processing time and profit coefficients of family i from the updated P_i.

This updating process can be simplified by applying dominance rule 5. Based on this, subset S_i^1 dominates any subset $S_k, S_k \in \overline{S}_i^1$. Therefore, we only need consider subsets are $\{S_i^1 \cup S_j \mid S_j \in \overline{S}_{it+1}\} \bigcup \overline{S}_{it+1}$. The process to update P_i can be described as:

Step 1. Calculate the piecewise function of subset S_i^1.

Step 2. Calculate the piecewise functions of subsets $\{S_i^1 \cup S_j \mid S_j \in \overline{S}_{it+1}\}$.

Step 3. P'_{it+1} corresponding to $\overline{S}_{it+1}$ is known from the calculation of the upper bound on MCKS. Set $P'_i = P'_{it+1} \cup \{P_{iS_i^1}\} \cup \{P_{iS_k} \mid S_k \in \{S_i^{'} \cup S_j \mid S_j \in \overline{S}_{it+1}\}\}$ and apply rules 1 and 2 to delete dominated points in P'_i and obtain the updated P_i.

After we obtain the updated P_i, we can obtain the new pseudo variables of family i.

4.3.2. Fixing y_{it} to zero

If we fix y_{it} to 0, then all subsets including t must delete this period, so their piecewise functions change, resulting in a different P_i.

Assume l is the last period in S_i^1, then S_i^1 stays the same since we fix y_{il} to one. All subsets used to update P_i when we fix y_{il} to one are $\{S_i^1 \cup S_j \mid S_j \in \overline{S}_{il+1}\}$ and $\overline{S}_{il+1}$; all subsets used to update P_i when we fix y_{it} to zero is $\{S_i^1 \cup S_j \mid S_j \in \overline{S}_{it+1}\}$ and $\overline{S}_{it+1}$.

Since $\overline{S}_{il+1} \subset \overline{S}_{it+1}$, then all subsets used when y_{it} zero are part of the subsets for fixing y_{il} to one. We save the P_i when we fix y_{il} to one so we need not calculate the updated P_i.

4.3.3. Bounding

After we obtain the new pseudo jobs $z_{i1},..z_{in_i'}$ from the updated P_i, the node's new upper bound can be obtained by replacing the old pseudo variables $z_{i1},..z_{in_i'}$ of family i with the new ones. For the current node's upper bound formulation:

$$\begin{aligned}
&Max \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i'} c_{ij}' z_{ij} \\
&s.t. \\
&\qquad \sum_{i=1}^{N}\sum_{j=1}^{n_i'} a_{ij}' z_{ij} \le b \\
&\qquad 0 \le z_{ij} \le 1, i = 1,..N \;\; j = 1,..n_i'
\end{aligned}$$

If we fix y_{it} to 1, then reduce available resource b by d_i; delete all old pseudo jobs $z_{i1},..z_{in_i'}$; replace by the new ones and find the new optimal. If we fix y_{it} to 0, then we replace all old pseudo jobs $z_{i1},..z_{in_i'}$ by new ones and find the new optimal. We can prove the new optimal is the upper bound of the current node by an approach similar to what we used in Appendix A.

4.3.4. Choosing a New Sub-problem

When variables are fixed, two sub-problems are created. If a sub-problem's upper bound is no better than an incumbent solution it is discarded. When its bound indicates it could contain a better solution to MCKS we store it in a bucket. Each bucket contains sub-problems with bounds that are about the same. Let UB be the best upper bound and INC be the value of the current incumbent solution. If we want K buckets, calculate

$$\Delta = {(UB - INC)}/{K} .$$

Then bucket one will contain all sub-problems with upper bounds in the interval $[UB-\Delta, UB]$, bucket two $[UB-2\Delta, UB-\Delta]$, and bucket K $[INC, INC+\Delta]$. Buckets can be updated as upper bounds or the incumbent change. When we choose a new sub-problem to explore, we take one based on LIFO from the lowest numbered, non-empty bucket. This gives an almost “best-bound” strategy, but without the bookkeeping overhead.

4.4. Computational experiments

We test AMCKS on a variety of problem instances to see what problems can be solved. Instances will be generated by setting five parameters at several levels. The parameters are number of families, average number of jobs in a family, proportion of setup time/cost relative to total time and cost, number of periods, and relationship between a and c. The number of families will be fixed at 10, 30 and 50. The number of periods will be fixed at 5, 10, 15 and 20. The number of jobs in a family will be integer uniformly distributed from [10, 30], [30, 50] and [50, 70]. Setup cost and time will be determined by

$$f_{it} = -e_1\left(\sum_{j=1}^{n_i} c_{ijt}\right)$$

$$d_i = e_2\left(\sum_{j=1}^{n_i} a_{ij}\right)$$

We will choose e_1 and e_2 uniformly from [0.05, 0.1], [0.1, 0.15], [0.15, 0.2], and [0.2, 0.25].

a and c have three relationships: a_{ij} is uniformly chosen from [10, 10000], and c_{ijt} is also uniformly chosen from [10, 10000] (uncorrelated relationship-U); a_{ij} is uniformly chosen from[10, 10000] and t_{ij} is randomly chosen from[10, 10000], and $c_{ijt} = t_{ij} + e$, e is randomly chosen from [0, 2000] (weak relationship-W); a_{ij} is uniformly chosen from[10, 10000], and c_{ijt} is randomly chosen from[a_{ij} -1000, a_{ij} +1000], if c_{ijt} is less than 10, then it is randomly chosen from [10,100] (strong relationship-S). Resource availability will be uniform from $\left[0.4*\sum_{i=1}^{N}\sum_{j=1}^{n_i}a_{ij}, 0.6*\sum_{i=1}^{N}\sum_{j=1}^{n_i}a_{ij},\right]$.

For each level of the five factors we generate ten instances. AMCKS was coded in C and all instances were run on a Dell P.C. with 1.7G Intel processor and 512M bytes of memory. In the following tables, we report the minimum (MIN), average (AVG) and maximum (MAX) solution time in minutes. We also give the average ratio of initial solution (INC) to initial upper bound (UB) and the average ratio of initial solution to the optimal solution (OPT).

Table 4.1
Solution time (minutes) with $N = 10$ and $n_i \sim [10, 30]$

		U					W					S				
period	Setup	LB/UB	LB/OPT	MIN	AVG	MAX	LB/UB	LB/OPT	MIN	AVG	MAX	LB/UB	LB/OPT	MIN	AVG	MAX
5	[0.05-0.1]	0.977	0.978	0.00	0.00	0.00	0.991	0.991	0.00	0.00	0.00	0.902	0.903	0.00	0.00	0.01
	[0.1-0.15]	0.976	0.977	0.00	0.00	0.00	0.988	0.989	0.00	0.00	0.00	0.843	0.845	0.00	0.00	0.01
	[0.15-0.2]	0.953	0.953	0.00	0.00	0.00	0.991	0.991	0.00	0.00	0.00	0.878	0.883	0.00	0.00	0.01
	[0.2-0.25]	0.912	0.913	0.00	0.00	0.00	0.929	0.930	0.00	0.00	0.00	0.893	0.900	0.00	0.00	0.01
10	[0.05-0.1]	0.992	0.992	0.00	0.00	0.00	0.997	0.997	0.00	0.00	0.00	0.903	0.904	0.00	0.01	0.02
	[0.1-0.15]	0.978	0.978	0.00	0.00	0.01	0.990	0.990	0.00	0.00	0.01	0.894	0.896	0.00	0.01	0.02
	[0.15-0.2]	0.973	0.974	0.00	0.00	0.00	0.997	0.997	0.00	0.00	0.01	0.833	0.838	0.00	0.01	0.02
	[0.2-0.25]	0.947	0.949	0.00	0.00	0.00	0.919	0.921	0.00	0.01	0.01	0.905	0.910	0.01	0.01	0.03
15	[0.05-0.1]	0.993	0.993	0.02	0.02	0.03	0.996	0.996	0.00	0.00	0.01	0.862	0.862	0.00	0.02	0.03
	[0.1-0.15]	0.992	0.993	0.00	0.01	0.01	0.989	0.990	0.00	0.00	0.01	0.893	0.895	0.00	0.02	0.05
	[0.15-0.2]	0.967	0.968	0.00	0.00	0.01	0.996	0.996	0.00	0.01	0.02	0.895	0.899	0.01	0.04	0.07
	[0.2-0.25]	0.935	0.936	0.00	0.00	0.01	0.945	0.946	0.00	0.01	0.03	0.837	0.842	0.03	0.05	0.07
20	[0.05-0.1]	0.994	0.994	0.09	0.10	0.11	0.997	0.998	0.01	0.01	0.01	0.904	0.905	0.02	0.04	0.05
	[0.1-0.15]	0.978	0.979	0.02	0.02	0.03	0.985	0.985	0.00	0.01	0.03	0.907	0.908	0.00	0.04	0.08
	[0.15-0.2]	0.978	0.979	0.01	0.01	0.02	0.997	0.998	0.01	0.02	0.04	0.919	0.923	0.01	0.08	0.19
	[0.2-0.25]	0.958	0.959	0.01	0.01	0.01	0.950	0.952	0.00	0.03	0.12	0.845	0.851	0.05	0.12	0.41

Table 4.2
Solution time (minutes) with $N = 30$ and $n_i \sim [30, 50]$

		U					W					S				
Period	Setup	LB/UB	LB/OPT	MIN	AVG	MAX	LB/UB	LB/OPT	MIN	AVG	MAX	LB/UB	LB/OPT	MIN	AVG	MAX
5	[0.05-0.1]	0.996	0.996	0.00	0.02	0.07	0.999	0.999	0.00	0.02	0.08	0.949	0.950	0.04	0.41	0.76
	[0.1-0.15]	0.991	0.991	0.00	0.02	0.04	0.989	0.989	0.00	0.08	0.25	0.954	0.954	0.04	0.19	0.30
	[0.15-0.2]	0.987	0.987	0.00	0.01	0.04	0.984	0.984	0.00	0.12	0.35	0.958	0.958	0.00	0.22	0.49
	[0.2-0.25]	0.974	0.974	0.00	0.02	0.04	0.977	0.977	0.00	0.11	0.34	0.964	0.965	0.00	0.27	0.63
10	[0.05-0.1]	0.997	0.997	0.01	0.05	0.17	0.999	0.999	0.01	0.03	0.06	0.966	0.966	0.07	0.79	1.57
	[0.1-0.15]	0.999	0.999	0.01	0.01	0.01	0.997	0.997	0.00	0.09	0.46	0.952	0.952	0.08	0.69	1.04
	[0.15-0.2]	0.990	0.990	0.00	0.03	0.05	0.983	0.983	0.00	0.20	0.64	0.961	0.962	0.11	1.08	2.01
	[0.2-0.25]	0.980	0.980	0.00	0.05	0.13	0.983	0.983	0.01	0.29	0.57	0.964	0.965	0.33	1.51	3.67
15	[0.05-0.1]	0.997	0.997	0.06	0.09	0.18	0.999	0.999	0.00	0.05	0.21	0.972	0.972	0.24	0.97	1.92
	[0.1-0.15]	0.996	0.996	0.02	0.04	0.09	0.996	0.996	0.01	0.15	0.44	0.970	0.971	0.12	0.89	1.46
	[0.15-0.2]	0.982	0.982	0.01	0.08	0.18	0.990	0.990	0.01	0.28	1.25	0.962	0.962	0.06	1.69	2.74
	[0.2-0.25]	0.974	0.974	0.05	0.09	0.15	0.983	0.983	0.00	0.88	5.57	0.959	0.959	0.01	2.86	6.17
20	[0.05-0.1]	0.997	0.997	0.87	1.74	3.68	0.999	0.999	0.03	0.10	0.34	0.982	0.982	0.05	0.97	2.10
	[0.1-0.15]	0.999	0.999	0.07	0.08	0.09	0.997	0.997	0.02	0.09	0.16	0.971	0.971	0.12	1.75	5.91
	[0.15-0.2]	0.987	0.987	0.03	0.08	0.17	0.994	0.994	0.02	0.16	0.49	0.975	0.976	0.22	2.18	8.73
	[0.2-0.25]	0.977	0.977	0.03	0.14	0.26	0.977	0.977	0.11	1.28	7.92	0.942	0.943	1.18	12.42	51.60

Table 4.3
Solution time (minutes) with $N = 50$ and $n_i \sim [50, 70]$

		U					W					S				
Period	Setup	LB/UB	LB/OPT	MIN	AVG	MAX	LB/UB	LB/OPT	MIN	AVG	MAX	LB/UB	LB/OPT	MIN	AVG	MAX
	[0.05-0.1]	0.997	0.997	0.01	0.15	0.82	1.000	1.000	0.00	0.09	0.34	0.979	0.979	0.05	2.12	4.25
	[0.1-0.15]	0.997	0.997	0.00	0.07	0.51	0.995	0.995	0.01	0.39	0.93	0.980	0.980	0.21	1.64	3.70
	[0.15-0.2]	0.991	0.991	0.00	0.14	0.36	1.000	1.000	0.10	1.74	4.49	0.984	0.984	0.03	1.03	3.43
5	[0.2-0.25]	0.987	0.987	0.00	0.13	0.36	0.983	0.983	0.08	0.92	1.76	0.980	0.980	0.30	3.09	5.78
	[0.05-0.1]	0.998	0.998	0.03	0.39	1.88	1.000	1.000	0.02	0.17	0.52	0.988	0.988	0.02	3.10	6.99
	[0.1-0.15]	1.000	1.000	0.01	0.04	0.09	0.999	0.999	0.02	0.29	1.26	0.976	0.976	0.12	4.46	8.44
	[0.15-0.2]	0.991	0.991	0.02	0.16	0.48	1.000	1.000	0.02	2.00	4.56	0.970	0.970	0.86	9.72	19.66
10	[0.2-0.25]	0.985	0.985	0.01	0.42	1.41	0.986	0.986	0.21	2.65	6.05	0.971	0.971	4.23	16.20	28.67
	[0.05-0.1]	0.997	0.997	0.20	0.80	2.10	1.000	1.000	0.02	0.14	0.31	0.984	0.984	0.36	7.29	22.69
	[0.1-0.15]	0.998	0.998	0.05	0.13	0.33	0.997	0.997	0.05	0.95	3.95	0.979	0.979	0.06	8.33	17.03
	[0.15-0.2]	0.993	0.993	0.03	0.35	0.96	1.000	1.000	0.01	2.24	5.83	0.969	0.970	0.06	17.70	35.68
15	[0.2-0.25]	0.987	0.987	0.02	0.43	1.01	0.990	0.990	0.04	1.89	3.75	0.976	0.976	0.07	30.40	82.01
	[0.05-0.1]	0.997	0.997	19.50	29.00	44.47	1.000	1.000	0.08	0.26	0.53	0.983	0.983	0.11	8.88	27.13
	[0.1-0.15]	0.998	0.998	0.17	0.45	1.35	0.998	0.998	0.03	0.41	2.71	0.981	0.981	0.44	13.60	29.35
	[0.15-0.2]	0.991	0.991	0.13	0.52	1.08	1.000	1.000	0.12	2.08	7.84	0.973	0.973	3.15	34.10	125.01
20	[0.2-0.25]	0.990	0.990	0.09	0.64	1.53	0.991	0.991	0.03	5.41	21.96	0.978	0.979	1.34	41.10	95.54

Fig. 4.1 and Fig. 4.2 show when coefficients are uncorrelated, instances with small setup proportion are more difficult than large setup proportion. The difference between 5% setup and 10% setup is apparent, but there is little difference between 10% setup and 15% setup.

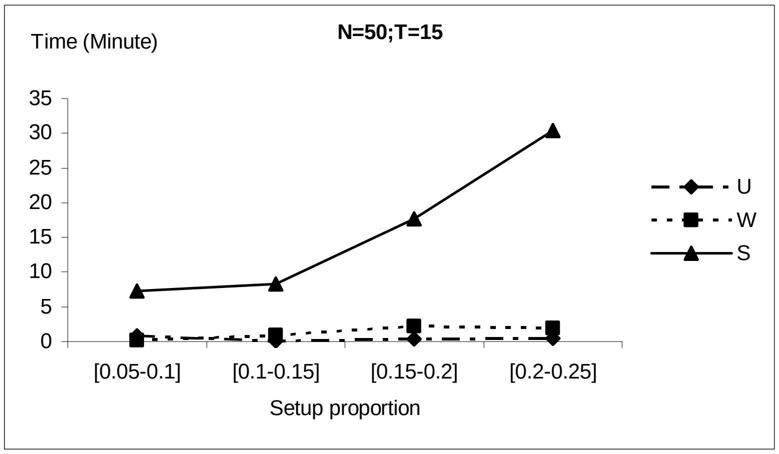

Fig. 4.1. Solution Time for $N = 50, T = 15$ and $n_i \sim [50, 70]$

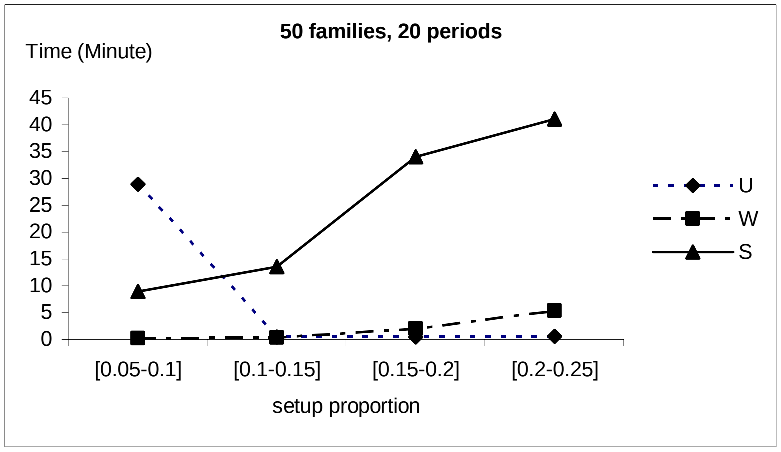

Fig. 4.2. Solution Time for $N = 50, T = 20$ and $n_i \sim [50, 70]$

The dominance rules are more effective when setup proportion is large and when a and c are uncorrelated. If the setup proportion is small, jobs are more often assigned to multiple periods; the dominance rules are not as effective as for instances with large setup proportion. But when c is correlated over different periods, there is not much difference in assigning a job to a particular period. Thus AMCKS can easily solve an instance with small setup proportion.

When setup proportion increases, instances with a, c having weak relationship and strong relationship become harder. By the central limit theorem, the total setup cost and time follow a normal distribution. Under the weak and strong relationships, the correlation of setups in different periods increases. With setup proportion increasing, setup has more effect on the optimal solution and differences in periods decrease. Hence dominance rules are not as effective in this case. The other possible reason is the lower bound. With setup proportion increasing, the lower bound becomes worse so we can not fathom nodes as effectively.

Instances with a and c correlated are more difficult. The piecewise function for different periods become flat, and the computation for the composite piecewise function becomes complex. The knapsack problem when all setup variables fixed is also a hard problem; we did not use a special algorithm to deal with correlation in the knapsack problem. Some improvement can be expected if a special algorithm is used.

We use the rounded solution as a lower bound on MCKS, which is very effective. For instances with $N = 30$ and $N = 50$, the lower bound is at least 95% of the optimal. When $N = 10$, it is worse since there are fewer points on every piecewise function, so the resource is much easier to use up before the first break point of the piecewise function. If the fractional value is the first pseudo variable of some family, all variables corresponded to this pseudo variable are rounded to zero, thus the lower bound is not as good.

We also compare AMCKS to CPLEX 9.1 (called by AMPL). We choose the hardest instances for AMCKS (20 periods, 50 families, $n_i \sim [80, 90]$) to compare. Trial runs on other instances showed these results are typical on CPLEX. Due to the difficulty of solving with CPLEX, only five instances per level were solved. Table 4 shows the clear superiority of AMCKS.

Table 4.4
The solution time (minute) comparison between AMCKS and CPLEX

		U			W			S		
Setup		CPLEX	AMCKS	C/A	CPLEX	AMCKS	C/A	CPLEX	AMCKS	C/A
[0.05-0.1]	1	>120.00	39.82	3.01	4.51	1.31	3.45	5.12	0.77	6.65
	2	>120.00	15.39	7.80	5.53	0.15	37.58	5.90	8.46	0.70
	3	>120.00	31.31	3.83	5.05	0.16	31.06	13.68	15.05	0.91
	4	>120.00	19.23	6.24	4.82	2.58	1.87	4.19	1.60	2.62
	5	>120.00	14.33	8.37	4.42	0.08	58.28	5.07	1.47	3.45
AVG				5.85			26.45			2.87
[0.1-0.15]	1	>120.00	0.21	570.34	7.79	0.04	190.61	5.03	1.79	2.82
	2	>120.00	0.24	506.97	7.82	1.03	7.57	82.07	8.39	9.78
	3	>120.00	0.22	545.99	8.79	0.86	10.19	5.24	1.00	5.22
	4	117.11	0.20	590.17	8.59	5.23	1.64	>120.00	13.56	8.85
	5	>120.00	0.17	725.73	8.53	0.59	14.56	13.23	1.59	8.33
AVG				587.84			44.91			7.00
[0.15-0.2]	1	50.85	0.09	566.01	7.47	0.13	56.14	12.68	17.14	0.74
	2	7.41	0.08	91.82	7.10	0.19	38.35	>120.00	31.10	3.86
	3	15.84	0.12	129.98	9.22	0.32	28.91	>120.00	33.79	3.55
	4	15.52	0.16	97.24	8.92	4.89	1.82	29.80	8.90	3.35
	5	17.36	0.91	19.08	9.18	0.33	27.77	32.76	2.59	12.63
average				180.83			30.60			4.83
[0.2-0.25]	1	8.70	0.32	27.41	6.87	0.15	47.39	>120.00	28.72	4.18
	2	7.59	0.84	8.99	7.11	12.00	0.59	>120.00	17.92	6.70
	3	8.53	2.23	3.82	6.89	0.18	39.29	>120.00	47.09	2.55
	4	7.08	0.20	34.63	6.29	4.88	1.29	>120.00	48.74	2.46
	5	7.44	0.07	103.13	6.87	0.14	47.42	>120.00	26.83	4.47
average				35.60			27.20			4.07

When a and c are uncorrelated, AMCKS is much better than CPLEX. When a and c are correlated, AMCKS is still better than CPLEX except for instances with 5% setup and both solvers take longer for instances with larger setup than those with smaller setup. AMCKS solves problems with 5%-10% setup in about one-third hour; when setup proportion is over 10%, AMCKS takes less than one hour. For problems with 10%, 15%

and 20% setups, CPLEX failed to solve many instances in two hours. Though CPLEX can obtain a near optimal solution, it can't prove it is optimal in two hours, which often happens in many algorithms for integer programming.

4.5. Conclusion

MCKS can be used for project selection for a country or company. In this paper, we use a branch-and-bound algorithm to solve the multiple-choice knapsack problem with setup. A linear knapsack problem is designed to give an upper bound on MCKS. We develop three dominance rules to simplify the process and save time to obtain an upper bound model. The rounded solution of the linear knapsack problem provides a good incumbent for MCKS. For instances with N greater than 30, the heuristic is over 95% of the optimal solution. Computational experiments show the algorithm's effectiveness. Compared to CPLEX, the proposed algorithm obtains the optimal solution in less time for most instances. Setup proportion has more effect on instances with uncorrelated relationship instances than other instances. In this paper, we only use a simple branch-and-bound algorithm for the knapsack problem when all setup variables are fixed. If a better algorithm, e.g. the one developed by Martello et al. (1999) is used, the solution time can be reduced.

Appendix A. The optimal objective of LKP_u is an upper bound on MCKS.

Proof.

If the optimal solution of MCKS is known, then we obtain the sets $S_i = \{t \mid y_{it} = 1\}$, $i = 1,..N$ in the optimal solution of MCKS and the resource taken by family i is w_i as well as contributed profit $profit_i, i = 1,..N$ with $\sum_{i=1}^{N} w_i \le b$ and the optimal objective $\sum_{i=1}^{N} profit_i$.

For the period set S_i , there is $c_{ijS_i} = \max\{c_{ijt}, t \in S_i\}$, $f_{iS_i} = \sum_{t \in S_i} f_{it}$, and $d_{iS_i} = \|S_i\| d_i$. Using pseudo variables x'_{ij} and y'_i , formulate a linear knapsack problem with setup LKP_{S_i} by these coefficients:

$$
\begin{aligned}
&Max \quad \sum_{j=1}^{n_i} c_{ijS_k} x'_{ij} + f_{iS_k} y'_i \\
&s.t. \\
&\qquad \sum_{j=1}^{n_i} a_{ij} x'_{ij} + d_{iS_k} y'_i \le w_i, \\
&\qquad x'_{ij} \le y'_{i,} \quad j = 1,..n_i, \\
&\qquad x'_{ij} \ge 0 \quad j = 1,..n_i, \\
&\qquad 0 \le y'_i \le 1.
\end{aligned}
$$

Since we know the optimal solution of MCKS, set $y'_i = \sum_{t=1}^{T} y_{it}$ and $x'_{ij} = \sum_{t=1}^{T} x_{ijt} \quad j = 1,..n_i$, so that y'_i and x'_{ij} are a feasible solution of LKP_{S_i} . We know

$\sum_{j=1}^{n_i} c_{ijS_i} x'_{ij} = \sum_{j=1}^{n_i} c_{ijS_i} \sum_{t=1}^{T} x_{ijt} \ge \sum_{t=1}^{T} \sum_{j=1}^{n_i} c_{ijt} x_{ijt}$ and $f_{iS_i} y'_i = \sum_{t=1}^{T} f_{it} y_{it}$, thus $\sum_{j=1}^{n_i} c_{ijS_k} x'_{ij} + f_{iS_k} y'_i \ge profit_i$.

Because $F_{iS_i}(w_i)$ is the optimal objective of LKP_{S_i} , then $F_{iS_i}(w_i) \ge profit_i$. The linear knapsack problem obtained from $F_i(w_i)$ is

$$
\begin{aligned}
&Max \quad \sum_{j=1}^{n_i'} c_{ij}' z_{ij} \\
&s.t. \\
&\qquad \sum_{j=1}^{n_i'} a_{ij}' z_{ij} \le w_i \\
&\qquad 0 \le z_{ij} \le 1,\ j = 1..n_i'
\end{aligned}
$$

Define its solution for this problem is Z_i. Repeating this process for all families, $\bigcup_{i=1}^{N} Z_i$ is a feasible solution for LKP_u

$$
\begin{aligned}
&Max \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i'} c_{ij}' z_{ij} \\
&s.t. \\
&\qquad \sum_{i=1}^{N}\sum_{j=1}^{n_i'} a_{ij}' z_{ij} \le b \\
&\qquad 0 \le z_{ij} \le 1, i = 1,..N \ \ j = 1,..n_i'
\end{aligned}
$$

and its objective of this feasible solution is $\sum_{i}^{N} F_i(w_i)$ which is less than the optimal objective of LKP_u. Since $F_i(w_i) \ge F_{iS_i}(w_i) \ge profit_i$, then $\sum_{i=1}^{N} profit_i$ is less than the optimal objective of LKP_u.

Appendix B. The rounded solution of LKP_u corresponds to a feasible solution of MCKS

Proof.

Assume resource taken by family i in the break solution is w_i and the corresponding profit obtained by the family is obj_i. For all pseudo variables $z_{i1},..z_{in_i'}$, there is $c_{ij}' / a_{ij}' \ge c_{ij+1}' / a_{ij+1}'$.

If $z_{ij} = 0$ for all $j = 1,..n_i^{'}$, then assign all variables of family i to zero. If $z_{ik} = 1$ in the rounded solution, and $z_{ij} = 0 \;\; j > k$, then the coefficients of z_{ik} comes from p_k and p_{k-1} in point set P_i by $a'_{ik} = p_k.x - p_{k-1}.x$ and $c'_{ik} = p_k.y - p_{k-1}.y$. On the piecewise function F_i, $p_k.x = w_i$ and $p_k.y = obj_i$. Assume this point p_k comes from the break points of piecewise function F_{iS_k}. F_{iS_k} corresponds to a linear knapsack problem LKP_{S_k}, but this LKP_{S_k} is the transformation of a linear knapsack problem with setup $LKPS_{S_k}$

$$
\begin{aligned}
&Max \quad \sum_{j=1}^{n_i} c_{ijS_k} x'_{ij} + f_{iS_k} y'_i \\
&s.t. \\
&\qquad \sum_{j=1}^{n_i} a_{ij} x'_{ij} + d_{iS_k} y'_i \le b_0, \\
&\qquad x'_{ij} \le y'_{i,} \quad j = 1..n_i, \\
&\qquad x'_{ij} \ge 0 \quad j = 1..n_i, \\
&\qquad 0 \le y'_i \le 1.
\end{aligned}
$$

Based on Bulfin's algorithm for linear knapsack problem with setup, the variables in $LKPS_{S_k}$ can be separated into two set $XM = \{x'_{i1},..x'_{it}\}$ and $XT = \{x'_{it+1},..x'_{in_i}\}$. If z_{ij} is the first break point of F_{iS_k}, then set $x'_{ik} = 1 \;\; k = 1,..t$ and $x'_{ik} = 1 \;\; k = t+1,..n_i$; else set $x'_{ik} = 1 \;\; k = 1,...j + t - 1$. If $x'_{ik} = 1$, let $x_{ijr} = 1$ and $y_{ir} = 0$ if $c_{ijr} = \max\{c_{ijt} \mid t \in S_k\}$; else set $x_{ijr} = 0, r \in S_k$. Then $\sum_{t \in S_k} \sum_{j=1}^{n_i} c_{ijt} x_{ijt} + \sum_{t \in S_k} f_{it} y_{it} = obj_i$ and $\sum_{t \in S_k} \sum_{j=1}^{n_i} a_{ij} x_{ijt} + \sum_{t \in S_k} d_i y_{it} = w_i$. Repeat this process to all families, and we can obtain a feasible integer solution of MCKS with the objective the same as the rounded solution's objective of LKP_u.

Appendix C. Three Dominance rules

Let us introduce two notations:

$F^{'}(b_0) = {p_{j+1}.y - p_j.y}\big/{p_{j+1}.x - p_j.x}$, $b_0 \geq 0$: The derivative function of piecewise function F . $p_j, j = 1,...n$ are n break points of F and $p_0 = (0,0)$. $p_j.x \leq b < p_{j+1}.x$.

If $b_0 > p_n.x$, then $F'(b_0) = 0$.

$J_k = \{ j \mid j = 1,..n_i, {c_{ijS_k}}\big/{a_{ij}} \geq F'_{iS_k}(0)\}$: The job set to form the first line between p_0 and p_1 of F_{iS_k} . $S_k, k = 1,..K$ are all subsets of $\{1,..T\}$ for family i.

Before proving Dominance rule 1, 2 and 3, we need the following Lemma:

Lemma 1. Let S_r, S_k, S_l be three different subsets of $\{1,..T\}$ for family i, and $S_r \cup S_k = S_l$, $S_r \cap S_k = \phi$. If $F'_{iS_r}(0) > F'_{iS_k}(0)$, then $F'_{iS_r}(0) \geq F'_{iS_l}(0)$.

Proof.

Case 1: If there is $j \in J_l \cap \overline{(J_r \cup J_s)}$

(1) If $c_{ijS_l} = c_{ijS_r}$,

then ${c_{ijS_l}}\big/{a_{ij}} \geq F'_{iS_l}(0)$, but ${c_{ijS_l}}\big/{a_{ij}} < F'_{iS_r}(0)$, thus $F'_{iS_r}(0) > F'_{iS_l}(0)$.

(2) If $c_{ijS_l} = c_{ijS_k}$,

then ${c_{ijS_l}}\big/{a_{ij}} \geq F'_{iS_l}(0)$, but ${c_{ijS_k}}\big/{a_{ij}} < F'_{iS_k}(0)$, thus $F'_{iS_l}(0) < F'_{iS_k}(0) < F'_{iS_r}(0)$.

Case 2: If $J_l \subseteq J_r \cup J_s$, then define $J_l^r = \{ j \mid j \in J_l, c_{ijS_l} = c_{ijS_r}\}$

and $J_l^k = \{ j \mid j \in J_l, c_{ijS_l} = c_{ijS_k}\}$ then

$$F'_{iS_l}(0) = [(f_{iS_r} + \sum_{j \in J_l^r} c_{ijS_r}) + (f_{iS_k} + \sum_{j \in J_l^k} c_{ijS_k})] \Big/ (d_{iS_r} + d_{iS_k} + \sum_{j \in J_l^r} a_{ij} + \sum_{j \in J_l^k} a_{ij}) \cdot$$

Since $(f_{iS_r} + \sum_{j \in J_l^r} c_{ijS_r}) \Big/ (d_{iS_r} + \sum_{j \in J_l^r} a_{ij}) \le F'_{iS_r}(0)$ in the same way,

$(f_{iS_k} + \sum_{j \in J_l^k} c_{ijS_k}) \Big/ (d_{iS_k} + \sum_{j \in J_l^k} a_{ij}) \le F'_{iS_k}(0)$. Therefore, $F'_{iS_l}(0) \le \max\{F'_{iS_r}(0), F'_{iS_k}(0)\} = F'_{iS_r}(0)$

Lemma 2. If $d_i = 0,\ f_{it} = 0, t = 1,..T$,and there are two sets S_r and S_k with $S_r \subset S_k$,

then $F_{iS_r}(b_0) \le F_{iS_k}(b_0)$ and $F'_{iS_r}(b_0) \le F'_{iS_k}(b_0), 0 \le b_0 \le \sum_{j=1}^{n_i} a_{ij}$.

Proof.

Consider knapsack problem A:

$$\begin{aligned} & Max \quad \sum_{j=1}^{n_i} c_{ijS_r} x_{ijS_r} \\ & s.t. \\ & \qquad \sum_{j=1}^{n} a_{ij} x_{ijS_r} \le b_0, \\ & \qquad 0 \le x_{ijS_r} \le 1, j = 1,..n_i \end{aligned}$$

$F_{iS_r}(b_0)$ is the optimal objective of the linear knapsack problem A with right-hand side b_0.

Consider the knapsack problem B:

$$Max \quad \sum_{j=1}^{n_i} c_{ijS_k} x_{ijS_k}$$

$s.t.$

$$\sum_{j=1}^{n_i} a_{ij} x_{ijS_k} \le b_0$$

$$0 \le x_{ijS_k} \le 1,\ j = 1,..n_i$$

$F_{iS_k}(b_0)$ is the optimal objective of the linear knapsack problem B with right-hand side b_0 .Since $S_r \subseteq S_k$, then $c_{ijS_r} \le c_{ijS_k}$ and A's feasible space is same with B's, thus $F_{iS_r}(b_0) \le F_{iS_k}(b_0)$.

Let

set $J'_r = \{ j \mid j = 1,..n_i, {c_{ijS_r}}/{a_{ij}} \ge F'_{iS_r}(b_0)\}$ and ${c_{ij_r S_r}}/{a_{ij_r}} = \min\{ {c_{ijS_r}}/{a_{ij}}, j \in J'_r\}$ then

$F'_{iS_r}(b_0) = {c_{ij_r S_r}}/{a_{ij_r}}$;

set $J'_k = \{ j \mid j = 1,..n_i, {c_{ijS_k}}/{a_{ij}} \ge F'_{iS_k}(b_0)\}$ and ${c_{ij_k S_k}}/{a_{ij_k}} = \min\{ {c_{ijS_k}}/{a_{ij}}, j \in J'_k\}$ then

$F'_{iS_k}(b_0) = {c_{ij_k S_k}}/{a_{ij_k}}$.

Then $J'_r \not\subset J'_k$.

Case 1: If $J'_k = J'_r$,

if $j_r = j_k$, then since $c_{ij_r S_r} \le c_{ij_k S_k}$, thus $F'_{iS_r}(b_0) \le F'_{iS_k}(b_0)$;

if $j_r \ne j_k$, then ${c_{ij_k S_k}}/{a_{ij_k}} \ge {c_{ij_k S_r}}/{a_{ij_k}} \ge {c_{ij_r S_r}}/{a_{ij_r}}$, thus $F'_{iS_r}(b_0) \le F'_{iS_k}(b_0)$.

Case 2: If $J'_k \ne J'_r$,

there is a job $j \in J_r'$ but $j \notin J_k'$, thus $F_{iS_r}'(b_0) \le {c_{ijS_r}}/{a_{ij}} \le {c_{ijS_k}}/{a_{ij}} \le F_{iS_k}'(b_0)$.

Lemma 3. If $S_r \subset S_k$, let p_r and p_k be the first points except (0, 0) on piecewise functions F_{iS_r}, and F_{iS_k} respectively. Define $b_1 = \max\{p_r.x, p_s.x\}$. Then for $b_0 > b_1$, $F_{iS_r}^{'}(b_0) \le F_{iS_k}^{'}(b_0)$.

Proof.

If $b_0 \ge \|S_k\| * d_i + \sum_{j=1}^{n_i} a_{ij}$, then $F_{iS_r}^{'}(b_0) = F_{iS_k}^{'}(b_0) = 0$.

If $\|S_r\| * d_i + \sum_{j=1}^{n_i} a_{ij} \le b_0 < \|S_k\| * d_i + \sum_{j=1}^{n_i} a_{ij}$, then $F_{iS_r}^{'}(b_0) = 0$, $F_{iS_k}^{'}(b_0) > 0$.

If $b_0 < \|S_r\| * d_i + \sum_{j=1}^{n_i} a_{ij}$, then assume $F_{iS_r}^{'}(b_0) = {c_{ij_r S_r}}/{a_{ij_r}}$, $F_{iS_k}^{'}(b_0) = {c_{ij_k S_k}}/{a_{ij_k}}$.

Since $\|S_r\| * d_i < \|S_k\| * d_i$, then $b_0 - \|S_r\| * d_i > b_0 - \|S_k\| * d_i$. Then for the linear knapsack problem

$$
\begin{aligned}
&Max \quad \sum_{j=1}^{n_i} c_{ijS_r} x_{ijS_r} \\
&s.t. \\
&\qquad \sum_{j=1}^{n} a_{ij} x_{ijS_r} \le b_0 - \|S_r\| * d_i, \\
&\qquad 0 \le x_{ijS_r} \le 1, \; j = 1, ..n_i
\end{aligned}
$$

and its piecewise function $\overline{F}_{iS_r}$, there is $\overline{F}_{iS_r}'(b_0 - \|S_r\| * d_i) = {c_{ij_r S_r}}/{a_{ij_r}}$.

For the linear knapsack problem

$$
\begin{aligned}
&Max \quad \sum_{j=1}^{n_i} c_{ijS_k} x_{ijS_k} \\
&s.t. \\
&\qquad \sum_{j=1}^{n_i} a_{ij} x_{ijS_k} \le b_0 - \|S_k\| * d_i \\
&\qquad 0 \le x_{ijS_k} \le 1,\ j = 1,..n_i
\end{aligned}
$$

and its piecewise function $\bar{F}_{iS_k}$, there is $\bar{F}'_{iS_k}(b_0 - \|S_k\| * d_i) = {c_{ij_k S_r}}/{a_{ij_k}}$.

Based on lemma 2, there is $\bar{F}'_{iS_r}(b_0 - \|S_r\| * d_i) \le \bar{F}_{iS_k}(b_0 - \|S_r\| * d_i)$.

Since piecewise function is concave function, then $\bar{F}_{iS_k}(b_0 - \|S_r\| * d_i) \le \bar{F}_{iS_k}(b_0 - \|S_k\| * d_i)$.

Therefore ${c_{ij_r S_r}}/{a_{ij_r}} \le {c_{ij_k S_k}}/{a_{ij_k}}$, so $F'_{iS_r}(b_0) \le F'_{iS_k}(b_0)$.

Lemma 4. Assume $S_r \subset S_k$. If $F'_{iS_r}(0) > F'_{iS_k}(0)$, then there is at most one intersection of F_{iS_r} and F_{iS_k} except (0,0); if $F'_{iS_r}(0) \le F'_{iS_k}(0)$, then S_k dominates S_r .

Proof.

Case 1: $F'_{iS_r}(0) > F'_{iS_k}(0)$

Assume p_r and p_k are the first points respectively on piecewise function F_{iS_r} and F_{iS_k} . If there are two intersections p_1, p_2 of F_{iS_r} and F_{iS_k} , then there is $F'_{iS_r}(p_1.x) < F'_{iS_k}(p_1.x)$ and $F'_{iS_r}(p_2.x) > F'_{iS_k}(p_2.x)$. Since F_{iS_r} and F_{iS_k} are concave piecewise functions, there is $p_2.x > \max\{p_r.x, p_k.x\}$. But based on lemma 3, $F'_{iS_r}(p_2.x) > F'_{iS_k}(p_2.x)$ can not be true.

Therefore, there is no second intersection p_2 .

Case 2: $F'_{iS_r}(0) < F'_{iS_k}(0)$.

If $p_r.x \le p_k.x$, there is no intersection p between $(0, p_k.x)$. If there is an intersection outside of $(0, p_k.x)$, then there is $F'_{iS_r}(p_1.x) > F'_{iS_k}(p_1.x)$ that is impossible based on lemma 3.

If $p_r.x > p_k.x$:

Define $J^1 = J_r \cap \overline{J}_k$. Since $c_{ijS_r} \le c_{ijS_k}$, $j = 1,..n_i$, then $c_{ijS_r}/a_{ij} \le c_{ijS_k}/a_{ij}$. We have $F'_{iS_k}(p_k.x + \sum_{j\in J^1} a_{ij}) \ge \left(f_{iS_k} + \sum_{j\in J_k \cup J^1} c_{ijS_k}\right) \Big/ \left(d_{iS_k} + \sum_{j\in J_k \cup J^1} a_{ij}\right) \ge F'_{iS_r}(p_k.x)$ and

$p_k x + \sum_{j\in J^1} a_{ij} \ge p_r.x$, thus $F_{iS_k}(b_0) > F_{iS_r}(b_0)$ for $b_0 \in (0, p_r.x)$. If there is an intersection p outside of $(0, p_r.x)$, then $F'_{iS_r}(p.x) > F'_{iS_k}(p.x)$ that is impossible based on lemma 3.

Case 3: $F'_{iS_r}(0) = F'_{iS_k}(0)$

If $p_r.x \le p_k.x$, $F_{iS_r}(b_0) = F_{iS_k}(b_0)$, $b_0 \in (0, p_r.x)$, and

$F_{iS_r}(b_0) < F_{iS_k}(b_0)$, $b_0 \in (p_r.x, p_k.x)$. If there is an intersection p outside of $(0, p_k.x)$, then $F'_{iS_r}(p.x) > F'_{iS_k}(p.x)$ that is impossible based on lemma 3.

$p_r.x > p_k.x$ can not happen since if there is $J^1 = J_r \cap \overline{J}_k$, then

$c_{ijS_k}/a_{ij} \ge c_{ijS_r}/a_{ij} \ge F'_{iS_k}(0)$, $j \in J^1$, thus J^1 should be included into J_k. Then $p_r.x$ can not be larger than $p_k.x$.

Dominance rule 3. If $S_r \subset S_k$, and $\sum_{j=1}^{n_i} c_{ijS_r} + f_{iS_r} > \sum_{j=1}^{n_i} c_{ijS_k} + f_{iS_k}$, then S_r dominates S_k.

Proof.

For $b_0 = \sum_{j=1}^{n_i} a_{ij} + d_{iS_r}$, there is $F_{iS_r}(b_0) > F_{iS_k}(b_0)$, thus $F'_{iS_r}(0) > F'_{iS_k}(0)$ based on case 2 of lemma 4. Based on case 1 of lemma 4, there is at most one intersection p with $F'_{iS_r}(p.x) < F'_{iS_k}(p.x)$, $p.x < b_0$, thus $F_{iS_r}(b_0) > F_{iS_k}(b_0)$ can not be true. Thus there is no intersection of F_{iS_r} and F_{iS_k}, and S_{ir} dominates S_{ik}.

Dominance rule 4. If $S_r \subset S_k$ and S_r dominates S_k, then for another set S_l with $S_r \cap S_l = \phi, S_k \cap S_l = \phi$, $S_r \cup S_l$ dominates $S_k \cup S_l$.

Proof.

Define variable $\Delta c_{ijS_r} = c_{ijS_l} - c_{ijS_r}$, $\Delta c_{ijS_k} = c_{ijS_l} - c_{ijS_k}$. Since $c_{ijS_r} \le c_{ijS_k}$, then $\Delta c_{ijS_r} \ge \Delta c_{ijS_k}$, and $\sum_{j=1}^{n} (\Delta c_{ijS_r} \mid \Delta c_{ijS} > 0) \ge \sum_{j=1}^{n} (\Delta c_{ijS_k} \mid \Delta c_{ijS_k} > 0)$.

We know:

$$\sum_{j=1}^{n} c_{ijS_r \cup S_l} = \sum_{j=1}^{n} c_{ijS_r} + \sum_{j=1}^{n} (\Delta c_{ijS_r} \mid \Delta c_{ijS_r} > 0)$$

$$\sum_{j=1}^{n} c_{ijS_k \cup S_l} = \sum_{j=1}^{n} c_{ijS_k} + \sum_{j=1}^{n} (\Delta c_{ijS_k} \mid \Delta c_{ijS_k} > 0)$$

Since $f_{iS_r \cup S_l} = \sum_{t \in S_r \cup S_l} f_{it} > f_{iS_k \cup S_l} = \sum_{t \in S_k \cup S_l} f_{it}$, we can obtain $\sum_{j=1}^{n} c_{ijS_r \cup S_l} + f_{iS_r \cup S_l} \ge \sum_{j=1}^{n} c_{ijS_k \cup S_l} + f_{iS_k \cup S_l}$. Because $S_r \cup S_l \subseteq S_k \cup S_l$, then S_r dominates S_k by dominance rule 3.

Dominance rule 5. If $S_r \subset S_k$, $f_{iS_r} = f_{iS_k} = 0$, $d_{iS_r} = d_{iS_k} = 0$ then for another period set S_l with $S_l \cap S_r = \phi, S_l \cap S_k = \phi$, $S_l \cup S_k$ dominates $S_r \cup S_l$.

Proof.

Since $S_r \subset S_k$, $f_{iS_r} = f_{iS_k} = 0$, $d_{iS_r} = d_{iS_k} = 0$,

then $f_{iS_r \cup S_l} = f_{iS_k \cup S_l}$, $d_{iS_r \cup S_l} = d_{iS_k \cup S_l}$, a_{ij} keeps same, and $c_{ijS_r \cup S_l} \le c_{ijS_k \cup S_l}$, then for any resource b_0 there is $F_{iS_r \cup S_l}(b_0) \le F_{iS_k \cup S_l}(b_0)$, thus $S_l \cup S_k$ dominates $S_r \cup S_l$.

References

Akinc, U. 2004. Approximate and exact algorithm for the fixed-charge knapsack problem, European Journal of Operational Research 170, 363-375

Armstrong R.D, Kung D.S., Sinha P., Zoltners A.A. 1983. A computation study of a multiple-choice knapsack problem, ACM Transactions on Mathematical Software, 9, 184-198.

Bulfin, R. L. 1988. An algorithm for the continous, variable upper bound knapsack problem, OPSEARCH 25 (2), 119-125.

Dantzig, G.B. 1957. Discrete variable extremum problems, Operation Research 5, 266-277

Martello, S., Pisinger, D., Toth, P. 1999. Dynamic programming and strong bounds for the 0-1 Knapsack Problem. Management Science 45 (3), 414-424.

Pisinger, D. 1995. A minimal algorithm for the multiple-choice knapsack problem. European Journal of Operational Research 83, 394-410.

Sarin S., Karwan MH. 1989. The linear multiple choice knapsack problem", Operations Research Letters, 8, 95-100.

Ⅴ. CONCLUSIONS

This research investigated three integer programming models which can be applied to order acceptance in make-to-order production and regional project selection in multiple periods: the knapsack problem with setup (KPS), the multiple knapsack problem with setup (MKPS) and the multiple-choice knapsack problem with setup (MCKS). The common characteristics of all three models are: jobs belong to different families; setup time and setup costs are incurred if a job is processed; if two jobs from the same family are processed sequentially, no setup is required; resource is limited and some jobs can be selected to be manufactured. The objective is to maximize the sum of profits of processed jobs.

KPS can be used in order acceptance of single period. The model selects the jobs to be processed for maximizing the total profit. MKPS, as an extension of KPS, is used in order acceptance of multiple periods. Besides selecting the jobs to be processed, it also decides the periods which the selected jobs are arranged in. Jobs' coefficients vary in different periods, but the processing time stays the same. In MKPS, jobs' profits affect job's production schedule and the chosen schedules decide the job's profit. The two factors are balanced by maximizing the total profit under a resource limit. MCKS is applied to regional projects selection in multiple periods, and it can also be used in order acceptance of multiple periods with a non-renewable resource.

Branch-and-bound algorithm is used to obtain the optimal solution for all three models. The success of the algorithm relies on the effectiveness of the upper bound and lower bound in branching and the effort to obtain them. Unlike the usual approaches of relaxing some constraints of a formulation to obtain an upper bound, we design a linear knapsack problem for each model, and its LP solution is the upper bound on the model.

As the simplest among the three models, KPS can be viewed as a special case of the other two. Bulfin (1988) gave an algorithm for its linear relaxation, which transforms the linear relaxation to a linear knapsack problem. We show a linear knapsack problem corresponds to a concave piecewise function, and the concave piecewise function defines the variables as well as their coefficients in the linear knapsack problem.

Multiple-choice constraints are on the setup variables in MKPS to guarantee the jobs of the same family be processed in a single period. In MCKS, multiple-choice constraints are on the job variables so that a family's jobs can be processed in multiple periods. Approaches to obtain the linear knapsack problems which give the upper bounds on MKPS and MCKS are similar. We obtain a concave piecewise function for each family with the help of two dominance rules for linear multiple-choice knapsack problem. Pseudo variables as well as their profit and processing coefficients are defined from these piecewise functions. We use these pseudo variables to construct the linear knapsack problem. The process to obtain the concave piecewise functions in MCKS is more complex than those in MKPS. We develop three dominance rules to simplify it.

If the LP solution of the linear knapsack problem for KPS or MCKS is rounded to integers, we obtain an integer solution that corresponds to an incumbent of KPS or MCKS. A greedy algorithm is developed to obtain a lower bound on MKPS.

Branching is done in two stages. The first stage is to branch on setup variables. After all setup variables are fixed, the problem change to a (several) knapsack problem(s). A simple branch-and-bound algorithm is used to solve these knapsack problems. The computational experiments show these algorithms' effectiveness. Compared to CPLEX, the algorithms for all three models arrive at the optimal solution in less time for most instances.

BIBLIOGRAPHY

Akinc, U. 2004. Approximate and exact algorithm for the fixed-charge knapsack problem, European Journal of Operational Research 170, 363-375.

Armstrong R.D., Kung D.S., Sinha P., Zoltners A.A. 1983. A computation study of a multiple-choice knapsack problem, ACM Transactions on Mathematical Software, 9, 184-198.

Bulfin, R. L. 1988. An algorithm for the continuous variable upper bound knapsack problem, OPSEARCH 25 (2), 119-125.

Chajakis, E.D., Guignard, M. 1994. Exact algorithms for the setup knapsack problem, INFOR 32 (3), 124-142.

Dantzig, G.B. 1957. Discrete variable extremum problems, Operation Research 5, 266-277.

Dudzinski, K., Walukiewicz, S. 1987. Exact methods for the knapsack problem and its generalizations. European Journal of Operational Research 28, 3-21.

Ham, I., Hitomi, K., Yoshida, T. 1985 Group Techonology, Kluwer Nijhoff Publishing, Boston, Massachusetts.

Martello, S., Pisinger, D., Toth, P. 1999. Dynamic programming and strong bounds for the 0-1 Knapsack Problem. Management Science 45 (3), 414-424.

Martello, S., Toth, P. 1980. Solution of the zero-one multiple knapsack problem, European Journal of Operational Research 4, 276-283.

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

Martello S., Toth, P. 1990. Knapsack Problems: Algorithms and Computer Implementations, John Wiley and Sons, New York.

Parker, R. G., Rardin, R. L. 1988. Discrete Optimization. Academic Press, Inc. San Diego, CA.

Pisinger, D. 1995. A minimal algorithm for the multiple-choice knapsack problem. European Journal of Operational Research 83, 394-410.

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

Sarin S., Karwan MH. 1989. The linear multiple choice knapsack problem", Operations Research Letters, 8, 95-100

Sinha, A., Zoltners, A.A. 1979. The multiple-choice knapsack problem, Operations Research 27, 503-515.