An exact algorithm for the Knapsack Problem with Setup

Yanchun Yang* and Robert L. Bulfin

Department of Industrial and System Engineering,
Auburn University,
3301 Shelby Center, Auburn, AL 36832, USA
Fax: 334-844-1381
E-mail: yanchun.yang@gmail.com
E-mail: bulfirl@auburn.edu
*Corresponding author

Abstract: In this paper we studies a 0-1 Knapsack Problem with Setup (KPS). One set of 0-1 variables represent a family setup and serve as an Upper Bound (UB) on another set of 0-1 variables representing production of a job in a family. We present a branch-and-bound algorithm to find an optimal solution to KPS. The algorithm uses a two-stage branching strategy and chooses the next candidate problem to explore in a non-traditional way. We verify the efficacy of the algorithm through computational testing. This is the first time that an exact algorithm given to KPS with 10,000 integer variables. Computational experiments show that this algorithm is especially effective when objective and constraint coefficients are uncorrelated.

Keywords: Knapsack problem; setup; branch-and-bound; UB; upper bound.

Reference to this paper should be made as follows: Yang, Y. and Bulfin, R.L. (2009) 'An exact algorithm for the Knapsack Problem with Setup', *Int. J. Operational Research*, Vol. 5, No. 3, pp.280–291.

Biographical notes: Yanchun Yang received her PhD in the Department of Industrial and Systems Engineering at Auburn University on August, 2006. She also received her MS Degree in Management Engineering and BE Degree in Industrial Engineering from Northeastern University, China in 1998 and 2001 respectively. Her research interests focus on combinational optimisation, schedule, and supply chain. Currently she works for American Century Home, Inc. as Senior Operation Research Analyst.

Robert L. Bulfin is Thomas Walter Professor of Technology Management in the Department of Systems and Industrial Engineering, Auburn University. Previously, he was a faculty member in the Systems and Industrial Engineering Department, University of Arizona and an Industrial Engineer with Celanese Fibers Company. His research is in the area of discrete optimisation with particular emphasis on scheduling applications. NSF, DOD, DOT, USAID have supported his research and it has been published in several journals, including *IIE Transactions, Operations Research, Bell Journal of Economics, European Journal of Operations Research, Management Science, International Journal of Production Research, and Computers and Operations Research*.

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 1000 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 more time and labour. 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: number of families

n_i: number of jobs in family i

c_{ij}: profit of job j in family i

a_{ij}: time to process job j in family i

f_i: setup cost for family i ($f_i < 0$)

d_i: setup time for family i

b: time available for processing.

The decision variables are:

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.

The model, which we call KPS, is:

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

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

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

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.

Besides order-acceptance, this model can also be applied to project selection or it may appear as a sub-problem of other complex model such as the capacitated multi-item lot sizing problem.

In the following section we give a brief literature review and discuss background for the solution methodology. In Section 3, we present exact and heuristic algorithms to solve KPS. Computational results are given in Section 4. Finally, we give concluding remarks.

2 Literature survey

The linear version 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 this problem. It is based on the ratio rule of Dantzig (1957) for the Linear Knapsack Problem (LKP). A special case of KPS, the knapsack problem has been widely investigated since introduced by Dantzig (1957) (see Martello et al., 1999; Pisinger, 1995). The 0-1 knapsack problem is a special case of KPS, so KPS is NP-hard.

Several researchers have investigated variations of the knapsack problem that are related to KPS. Caserta et al. (2006) consider a Knapsack Problem with Setup (KPS), but each variable has its own setup. Kozanidis and Melachrinoudis (2004) studied a mix-integer knapsack problem applied to a transportation problem whose variables are separated into sets. In each set, there are discrete and continuous variables, but there is no relation between the two kinds of variables. The multiple-choice knapsack problem has variables that are also separated into several sets, and only one variable can be selected from each set, but there is no setup variable.

Akinc (2004) derives an algorithm for a special case of KPS with only a setup cost, which he called a fixed-charge knapsack problem. He presented an effective algorithm for the linear relaxation, but it is the same as Bulfin's algorithm. Akinc (2004) 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:

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

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 $y_1 = 1$, $y_2 = 1$, $x_{11} = 1$ and $x_{12} = 1$ has objective 10. Hence, the optimal objective of the knapsack problem when all y are integer in the 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 the 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 solutions 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 is 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.

3 Background

The knapsack problem and its variants are well-studied, 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 LKP as:

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

$$\text{s.t.} \quad \sum_{j=1}^{n} a_j x_j \le b \quad 0 \le x_j \le 1, \quad j = 1, \ldots, n.$$

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

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

$$x_t = \frac{\left(b - \sum_{j=1}^{t-1} a_j\right)}{a_t}$$

$$x_j = 0, \quad j > t$$

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

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

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

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

Bulfin (1988) defined this following ratio

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

It separates the jobs of family i into two sets, $XM_i = \{1 \ldots t_i\}$ and $XT_i = \{t_i + 1 \ldots n_i\}$. Then $r_{i,0} \ge r_{i,t+1} \ge r_{i,t+2} \ge \cdots \ge r_{i,n_i}$; a proof, different and simpler than that of Akinc (2004), is given in the Appendix.

For family i, define:

$$\begin{aligned}
& c'_{i1} = \sum_{j=1}^{t_i} c_{ij} + f_i \\
& a'_{i1} = \sum_{j=1}^{t_i} a_{ij} + d_i \\
& c'_{i,j-t_i+1} = c_{ij} \quad j = t_i + 1, \ldots, n_i \\
& a'_{i,j-t_i+1} = a_{ij} \quad j = t_i + 1, \ldots, n_i \\
& n'_i = n_i - t_i + 1.
\end{aligned}$$

Then LKPS can be reformulated as a standard LKP, which we call LBKP:

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

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 $x_{ij} = f$, $j = 1, \ldots, t_i$ 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 is the starting point for our heuristic procedure.

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.

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}, $j = 1, \ldots, n_{ii}$ are replaced by real variables $x_{i1}, \ldots, x_{in}$ of family i. When a sub-problem is created with y_i fixed to zero, z_{ij}, $j = 1, \ldots, n_i$ 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 $x_i = 1$ 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 $r_{10} \leq r_{20} \leq \cdots \leq r_{N0}$. 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.

4.2 Bounding

We use LBKP as an Upper Bound (UB) 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 $z_{i1}, \ldots, z_{in}$ and insert the new variables $x_{i1}, \ldots, x_{in_i}$. 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.

4.3 Choosing a new sub-problem

When variables are fixed two sub-problems are created. If a sub-problem's UB 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 UB and *INC* be the value of the current incumbent solution. If we want K buckets, calculate

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

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

4.4 Heuristic

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

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 sum of the constraint coefficients. Therefore, we choose b uniformly from

$$\left[0.4\times\sum_{i=1}^{N}\sum_{j=1}^{n_i}a_{ij},\ 0.6\times\sum_{i=1}^{N}\sum_{j=1}^{n_i}a_{ij}\right].$$

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 UB and the average ratio of initial incumbent to the optimal solution (OPT). The experiment results are shown in Table 1.

Table 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

*shows three of these instances ran out of memory; AVG, MAX, and MIN are calculated based on the remaining seven instances.

Our heuristic solution is outstanding. Note the LB/UB and LB/OPT ratios are rounded to two decimal places. On average, it was less than 2% from the optimal over the entire range of instances tested. Based on the data from Table 2, 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. 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.

Table 2 Comparing solution time (seconds) of CPLEX and AKPS

Setup	*Uncorrelated*			*Correlated*		
	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

Figure 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. Figure 2 gives the solution time with correlated coefficients. Instances with fewer families still work better than the others but solution time does not change too much as setup proportion increases. In correlated instances, setup proportion does not have as much effect on the incumbent.

Figure 1 Comparison of uncorrelated instances with similar total variables number

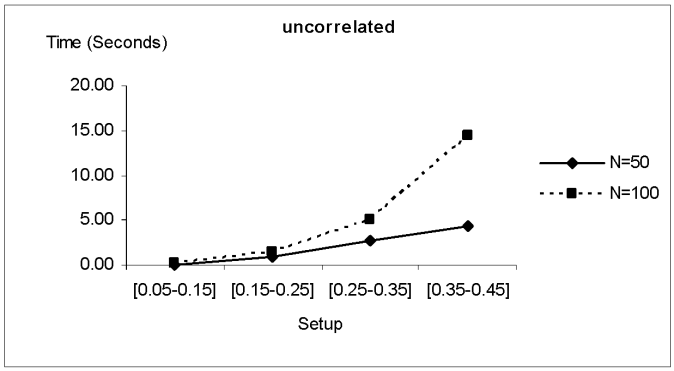
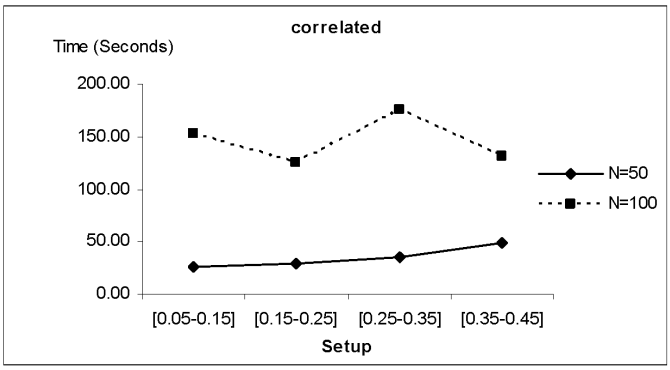


Figure 2 Comparison of correlated instances with similar total variables number



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 require more storage. The second approach presented failed in instances with 4000 variables because storage was used up. The first approach needed over 1000 seconds to solve the same instances. They permit negative job profit and positive setup cost in their model, which make instances easier due to preprocessing, which reduces 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. AKPS was able to solve uncorrelated problems with 10,000 variable in less than 30 seconds.

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 3–6 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.

6 Conclusions

We study the KPS. This is an important problem, modelling order acceptance, cell loading, project selection and others. We derive a new method to explain Bulfin's algorithm for its linear relaxation, which is simpler than Akinc's. We also 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. The proposed heuristic is excellent, being within 2% of optimal for all the problems tested. In this paper, a simple branch-and-bound algorithm is used repeatedly to solve a standard knapsack problem in the process of fixing setup variable; an improved knapsack algorithm would give much better results.

In the future, we can extend the single resource in KPS to multiple resources. This model would have wider application in make-to-order manufacturing.

References

Akinc, U. (2004) 'Approximate and exact algorithm for the fixed-charge knapsack problem', *European Journal of Operational Research*, Vol. 170, pp.363–375.

Bulfin, R.L. (1988) 'An algorithm for the continuous, variable upper bound knapsack problem', *OPSEARCH*, Vol. 25, No. 2, pp.119–125.

Caserta, M., Quinonez Rico, E. and Marquez Uribe, A. (2006) 'A cross entropy algorithm for the knapsack problem with setups', *Computers and Operations Research*, Available online.

Chajakis, E.D. and Guignard, M. (1994) 'Exact algorithms for the setup knapsack problem', *INFOR*, Vol. 32, No. 3, pp.124–142.

Dantzig, G.B. (1957) 'Discrete variable extremum problems', *Operation Research*, Vol. 5, pp.266–277.

Dudzinski, K. and Walukiewicz, S. (1987) 'Exact methods for the knapsack problem and its generalizations', *European Journal of Operational Research*, Vol. 28, pp.3–21.

Kozanidis, G. and Melachrinoudis, E. (2004) 'A branch & bound algorithm for the 0-1 mixed integer knapsack problem with linear multiple choice constraints', *Computers and Operations Research*, Vol. 31, pp.695–711.

Ham, I., Hitomi, K. and Yoshida, T. (1985) *Group Techonology*, Kluwer Nijhoff, Boston.

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

Martello, S., Pisinger, D. and Toth, P. (1999) 'Dynamic programming and strong bounds for the 0-1 knapsack problem', *Management Science*, Vol. 45, No. 3, pp.414–424.

Pisinger, D. (1995) 'A minimal algorithm for the Multiple-Choice Knapsack Problem', *European Journal of Operational Research*, Vol. 83, pp.394–410.

Appendix

Proposition: r_{i0} *is greater than* $r_{i,t+1}$.

Proof: Assume

$$a, b, c, d > 0 \quad \text{and } \frac{a}{b} \le \frac{c}{d}, \quad \text{then } \frac{a+c}{b+d} \ge \frac{a}{b}.$$

Since

$$ad \le bc \quad \text{from } \frac{a}{b} \le \frac{c}{d}, \quad \text{then } ab + ad \le bc + ab,$$

$$\text{or } a(b+d) \le b(a+c), \quad \text{so } \frac{a+c}{b+d} \ge \frac{a}{b}.$$

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