Journal of Computer Science 3 (3): 138-143, 2007
ISSN 1549-3636
© 2007 Science Publications

An Exact Algorithm for the Unbounded Knapsack Problem with Minimizing Maximum Processing Time

[1]Chanin Srisuwannapa and [2]Peerayuth Charnsethikul
Department of Applied Statistics, King Mongkut's Institue of Technology, Lardkrabang
OR/MS Units, IE Department, Kasetsart University, Bangkok, Thailand

Abstract: We address a variant of the unbounded knapsack problem (UKP) into which the processing time of each item is also put and considered, referred as MMPTUKP. The MMPTUKP is a decision problem of allocating amount of n items, such that the maximum processing time of the selected items is minimized and the total profit is gained as at least as determined without exceeding capacity of knapsack. In this study, we proposed a new exact algorithm for this problem, called MMPTUKP algorithm. This pseudo polynomial time algorithm solves the bounded knapsack problem (BKP) sequentially with the updated bounds until reaching an optimal solution. We present computational experience with various data instances randomly generated to validate our ideas and demonstrate the efficiency of the proposed algorithm.

Key words: Integer programming, bounded and unbounded knapsack problems

INTRODUCTION

One of the most frequently used decision making that operations researchers must deal is to decide which subset of n items or projects should be selected such that the total profit sum of the selected items or projects is maximized, without exceeding the capital budget, referred generally as the knapsack problem (KP). This problem can be formulated as a mathematical model (an integer linear program) and is one of an NP-hard combinatorial optimization problem by which can be solved successfully by various exact algorithms. The commonly used techniques are the dynamic programming and branch-and-bound methods and the branch-and-cut and branch-and-price methods as described in Toth[1]. This kind of problem can be applied and arises in many real world situations. In the following, we will provide a historical overview of this problem and various involved algorithms.

For the 0-1 knapsack problem (KP), it involves with selecting items or projects to maximize total profit without exceeding the existing capital budget or knapsack capacity. Exact algorithms for this problem are mainly based on two approaches: branch-and-bound and dynamic programming. Examples of branch-and-bound algorithms can be found in[2-10] and more recently in[11,12]. Dynamic programming approaches are presented in[5,13,14] and more recently in[15]. A hybrid algorithm, combining the dynamic programming and the branch-and-bound algorithms has been recently proposed by Martello *et al.*[16]. Different hybrid algorithms for the knapsack problem were presented by Plateau and Elkihel[10] and Martello and Toth[17]. An experiment comparison of the most effective exact algorithms for KP is given in[18].

The 0-1 multidimensional knapsack problem (0-1 MKP) is a generalization of the 0-1 knapsack problem and a special case of general 0-1 integer programming. The objective is the same as the 0-1 knapsack problem. Several heuristics or meta-heuristics have been used to solve the 0-1 MKP and updated and comprehensive survey for 0-1 MKP dealing with applications, complexity and heuristics can be found in Freville[19,20].

The multiple-knapsack problem (MKP) is the problem of assigning a subset of n items to m distinct knapsacks to maximize total profits without exceeding the capacity of each of the knapsacks. The problem has several applications in naval, financial management and steel industry[21]. The MKP is NP-hard in the strong sense and thus any dynamic programming approach would result in strictly exponential time bounds. Several branch-and-bound algorithm for MKP have thus been presented during the last two decade in Hung

Corresponding Author: P. Charnsethikul, Operations Research and Management Science Units, Department of Industrial Engineering, Faculty of Engineering, Kasetsart University, Bangkhaen, Bangkok 10900, Thailand

and Fisk[22], Martello and Toth[23], Neebe and Dannenbring[24] and Christofides *et al.*[25], Pisinger[26] and Martello and Toth[27].

The multiple-choice knapsack problem (MCKP) is defined as a 0-1 Knapsack Problem with the addition of disjoined multiple-choice constraints. MCKP is NP-hard as it contains KP as a special case, but it can be solved in pseudo-polynomial time through dynamic programming in Dudzinski and Walukiewicz[28]. The problem has a large range of applications: Capital Budgeting in Nauss[29], Menu Planning in Sinha and Zoltners[30], transforming nonlinear KP to MCKP in Nauss[29], determining which components should be linked in series in order to maximize fault tolerance in Sinha and Zoltners[30] and to accelerate ordinary LP/GUB problems by the dual simplex algorithm in Witzgal[31]. Moreover MCKP appears as Lagrangian relaxation of several integer programming problems in Fisher[32]. Several algorithms for MCKP have been presented during the last two decades: e.g. Nauss[29], Sinha and Zoltners[30], Dyerm Kayal and Walker[33] and Psinger[34].

The budgeting problem with bounded multiple choices constraints (BBMC) is a generalization of the multiple choice knapsack problem (MCKP). It has a wide variety of applications in budgeting, where a number of projects from each class has to be selected, such that the overall gain is largest possible and such that the costs demanded for the chosen projects do not exceed a fixed upper limit. It may be applied in sequencing and scheduling problems, strategic production and hospital planning with production constraints in each department, or government budgeting with demands in different sectors. The algorithm was presented in Pisinger[35].

The unbounded knapsack problem (UKP) is a classic NP-hard problem with a wide range of applications[36-39]. The two classic approaches for solving this problem exactly are branch and bound[38] and dynamic programming[36-39]. Another algorithm (dynamic programming revisited) is also presented in Andonov *et al.*[40].

To the best of our knowledge, the unbounded knapsack problem with minimizing maximum processing time, (MMPTUKP) in this study has not been studied in the literature. So the objective of this study is to present a new optimal solution algorithm for this type of problem and to evaluate performance of this algorithm by randomly generated data.

Problem formulation and a new exact algorithm

Problem formulation: The problem studied in this work can be formulated as follows. We are given a knapsack with capacity c and expected profits at least B into which we may put n types of objects. Each object of type i with parameters, the profit, p_i, the weight w_i and the processing time t_i and with n, B and c are all positive integers and we have an unbounded number of copies of each object type. The problem calls for selecting the set of items with minimizing the maximum processing time and must have at least profit of B without exceeding the knapsack capacity (budgeting, c). Mathematically, the problem can be described as the following integer linear programming formulation:

$$\text{Min} \quad T$$

$$\text{subject to} \quad T \geq t_i x_i \quad i = 1,2,..,n$$

$$\sum_{j=1}^{n} p_j x_j \geq B$$

$$\sum_{j=1}^{n} w_j x_j \leq c$$

$x_j \geq 0$ and integer for all j = 1,2,..,n.

An exact algorithm (MMPTUKP algorithm): Here, we propose an exact algorithm for solving MMPTUKP as follows:

1. Solve the following sub-problem, unbounded knapsack problem (UKP), by an exact algorithm such as branch-and-bound or dynamic programming methods.

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

Subject to

$$\sum_{j=1}^{n} w_j x_j \leq c$$

$x_j \geq 0$ and integer for all j = 1, 2,.., n.

After getting an optimal solution, let z* be the optimal objective value and let $x_j^*, j = 1,...,n$, be the optimal solution. Then check whether $z^* \geq B$. If yes, proceed to step 2. Otherwise, it indicates that there is no feasible solution, then stop.

2. Let $T_u = 0, T_l = 0, T = 0$ and let $T_u = \underset{j=1,2,...,n}{\text{Max}} [t_j x_j^*]$,
3. Let $T = [T_u + T_l]/2$, then compute $\lfloor T/t_j \rfloor$ defined as the upper bounds of each item x_j in the next sub-problem, then solve the following sub-problem, bounded knapsack problem(BKP) with these new upper bounds.

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

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \le c$$

$$0 \le x_j \le \lfloor T / t_j \rfloor, \quad j = 1,...,n$$

After getting an optimal solution, let z* be the optimal objective value and $x_j^*, j = 1,...,n$ be the optimal solution for the problem and then check whether $z^* \ge B$ or not. If yes, let $T_u = T$ and then repeat step 3. Otherwise, proceed to step 4.

4. Check whether T is converge or not by using the condition, $|T_u - T_l| < \varepsilon \to 0$, if yes, print the current solution as the optimal solution with $T^* = T_u$ and then stop. Otherwise, proceed to step 5.
5. Let $T_l = T$, go to step 3.

To verify the correctness of the proposed algorithm, the following theorem is proven as follows:

Theorem 1: Algorithm MMPTUKP terminates with a feasible solution under optimality of T.

Proof: Let there exist $T^0 < T^*$ be the optimal value of T with a feasible solution, $x_j^0, \forall j$ and the total profit $Z^0 \ge B$. Then, solve the bounded knapsack problem (BKP) with new upper bound $\lfloor T / t_j \rfloor$, with $T=T^0$ for all x_j. If $Z^*<B$ then T^0 does not exist as claimed. Otherwise, $T_u = T^0$ which leads to the impossible case that $T_u > T_l$. By contradiction, T^* is optimal as stated.

For the complexity of MMPTUKP, the possible bottleneck computation is due to either solving the UKP in step 1 or solving the corresponding BKP at most $[\text{Log}_2 T_u^*]$ times where T_u^* is the initial upper bound of T obtained from step 2. From theoretical standpoints, there exists at least a pseudo polynomial time algorithm for the proposed MMPTUKP procedure as summarized in the following theorem.

Theorem 2: MMPTUKP procedure can be solved as a pseudo polynomial time algorithm.

Proof: Since the number of iterations in the procedure is logarithmically bounded, a linear complexity is implied. In the procedure, the corresponding UKP and BKPs must be solved. Computationally, both problems can be solved efficiently in a pseudo polynomial bounded algorithm such as dynamic programming (bounded by the complexity of the right hand side and variables upper bound). Therefore, MMPTUKP can also be solved within a number of times bounded by a linear function and each time solved within a complexity of pseudo polynomial time bounded.

Performance evaluation experiments: We have investigated how the MMPTUKP algorithm behaves for many problem-sizes. The algorithm was coded as a C++ program and was complied as an .EXE file to run against solving the problem by direct mixed integer programming (MIP) methods using CPLEX software for small/intermediate scale and large scale problems respectively. All experiments were tested on a PC notebook with CPU GenuineIntel, Pentium(r) III processor 1 Ghz, 256 MB RAM, 20 GB hard disk. Firstly, it is critical to find a highly efficient method to solve the corresponding bounded and unbounded knapsack problems in the proposed algorithm. Though, there are many special algorithms with pseudo polynomial time complexities proposed to solve these problems. Experimentally, we have found that the direct approach using CPLEX can provide quite an acceptable result in term of computational time even when a small computer is used. The basic results are shown Table 1.

Table 1: Solution time in second for solving bounded and unbounded knapsack problems (BKP and UBKP) using CPLEX

Number of variables	(BKP)Time	(UKP)Time
10	0.1	0.06566667
50	0.15	0.13466667
100	0.24	0.21588889
500	0.39	0.30822222
1,000	0.59	0.47077778
5,000	1.211	1.31755556
10,000	2.924	4.807
50,000	138.399	390.942
100,000	659.999 (0.18hrs)	1973.007 (0.54hrs)
500,000	9679.127 (2.688hrs)	27229.564 (7.65hrs)
1,000,000	36851.94 (10.236hrs)	107979.197 (29.99hrs)

The test data in each problem size is randomly generated as p_j =[1,1000], w_j=[1,1000], b_j [1,10] for all j = 1,2,..,n and $C = 0.5 * \sum_{j=1}^{n} w_j b_j$ following[18]. The above results indicate that CPLEX is more efficient when solving the BKP as compared to solving the UKP since the bounded variable constraints in the BKP help reducing alternatives in the search tree during its branch and cut processes. Moreover, these results imply possibilities to solve very large scale BKP and UKP using the current state of the art mixed integer linear programming software especially when utilized

Table 2: Comparative results between using MMPTUKP algorithm (1) and the direct method (2)
*From 5 generated problems

Number of variables	Avg. time of (1) (sec.)	#iterations of (1)	#iterations of (1) from CPLEX	#nodes MIP used of (1)	#nodes of integer solution of (1)	Avg. time of (2) (sec.)	#iterations of (2)	#nodes MIP used of (2)	#nodes of integer solution of (2)
10	1.6364	26	65	79	47	0.081	31	11	8
20	3.405	27	109	152	67	0.1801	76	26	21
30	4.9803	26	111	148	99	0.3213	130	46	45
40	6.5705	28	142	177	118	0.4996	210	66	65
50	8.5975	29	196	253	179	0.7099	294	84	84
60	10.5262	30	148	188	122	1.0004	422	112	110
70	12.4828	30	178	223	158	1.3561	561	144	143
80	14.4877	31	214	264	196	1.7787	704	160	160
90	17.0063	31	333	417	308	2.4206	1145	334	326
100	19.2365	30	305	375	285	3.0125	925	218	211
200	21.9803	33	317	355	281	5.321	2549	449	428
300	24.7051	33	311	330	274	14.953	9474	3205	2935
400	27.5931	33	216	229	184	25.1088	7911	916	674
500	30.8347 (0.008hrs)	36	180	191	123	115.79533 (0.032hrs)	26856	5209	5105
600	34.8417 (0.009hrs)	36	254	272	214	139.055 (0.038hrs)	11788	941	941
700	39.3245 (0.011hrs)	35	140	139	115	>35545.870 (9.87hrs)	>865844	>456900	>447606
800	43.7699	36	91	90	58	-	-	-	-
900	48.698	36	98	96	68	-	-	-	-
1,000	168.4723	38	123	127	92	-	-	-	-
2,000	175.17167	37	26	23	0	-	-	-	-
3,000	186.735	38	80	75	52	-	-	-	-
4,000	203.64267	39	28	24	0	-	-	-	-
5,000	229.677	39	29	22	0	-	-	-	-
6,000	263.7426	39	33	27	0	-	-	-	-
7,000	309.08433	41	31	22	0	-	-	-	-
8,000	365.065	40	34	26	0	-	-	-	-
9,000	435.2993	40	38	27	0	-	-	-	-
10,000	510.2636	41	28	21	0	-	-	-	-
50,000*	9319.9416 (2.59hrs)	43	33	22	0	-	-	-	-
100,000*	43579.1843 (12.10hrs)	45	35	26	0	-	-	-	-

cooperatively with modern parallel processing technology. Next, the C++ executed program of MMPTUKP algorithm is experimented with a set of randomly generated problems comparing with solving the direct model as described earlier using CPLEX. The obtained results are shown in Table 2.

From Table 2, 10 replications for each problem size are randomly generated except large sizes of 50,000 and 100,000 where 5 replications are used due to their large amount of computation time consumed. Again, we use the ranges according to[18] as follows: $t_j = [1,1000]$, $p_j = [1,1000]$, $w_j = [1,1000]$,

$$C = \sum_{1}^{n} w_j \ , \ B = \sum_{1}^{n} c_j$$

.

The results from Table 2 clearly show that MMPTUKP algorithm is more efficient as compared to solving the problem directly especially when the problem is large

(more than thousand variables). An interesting statistic of the number of nodes with integer solution directly from the branch and bound process illustrates that the use of cutting planes in the branch and cut process utilized in CPLEX plays an important role as the problem size grows because these numbers become none for large scale test problems. Finally, the program is used to solve a set of five very large generated problems with 1,000,000 variables. The average computational time is more than 24 hours with the minimum is below 17 hours and the maximum is over 3 days. Parallel programming might be an alternative approach for improving the algorithm performance in this case.

CONCLUSION

In this study we derived a new exact algorithm, MMPTUKP algorithm, to the problem that the maximum processing time of the selected items is minimized and the total profit is gained as at least as determined without exceeding capacity of knapsack. This algorithm uses CPEX first for solving the sub-problem, the unbounded knapsack problem (UKP), in order to get an initial upper bound and optimal solution of a subset of items, then updating both lower/upper bounds of each item through a binary search approach and solve the corresponding BKP sequentially. The results from computational experiments with various data instances randomly generated validate this idea and demonstrate the efficiency of MMPTUKP algorithm. For comparisons between MMPTUKP results with the direct approach results, they indicate that MMPTUKP algorithm performs better on average especially in large size test problems.

ACKNOWLEDGEMENT

We would like to thank Professor Silvano Martello, University of Bologna, Italy, for many papers and discussions in which contributed significantly for the early version of paper.

REFERENCES

1. Toth, P., 2000. Optimization engineering techniques for the exact solution of NP-hard combination optimization problems. Eur. J. Oper. Res., 125: 222-238.
2. Balas, E. and E. Zemel, 1980. An algorithm for large zero-one knapsack problems. Oper. Res., 28: 1130-1154.
3. Fayard, D. and G. Plateau, 1975. Resolution for the solution of the 0-1 knapsack problem: Comparison of methods. Math. Program., 8: 272-307.
4. Fayard, D. and G. Plateau, 1982. Algorithm for the solution of the 0-1 knapsack problem. Computing, 28: 269-287.
5. Horowitz, E. and S. Sahni, 1974. Computing partitions with applications to the knapsack problem. J. ACM, 21: 277-292.
6. Martello, S. and P. Toth, 1977. An upper bound for the zero-one knapsack problem and a branch and bound algorithm. Eur. J. Oper. Res., 1: 169-175.
7. Martello, S. and P. Toth, 1988. A new algorithm for the 0-1 knapsack problem. Manag. Sci., 34: 633-644.
8. Martello, S. and P. Toth, 1990. Knapsack Problem: Algorithm and Computation Implementations, Wiley, Chichester.
9. Nauss, R.M., 1976. An efficient algorithm for the 0-1 knapsack problem. Manag. Sci., 23: 27-31.
10. Plateau, G. and M. Elkihel, 1985. A hybrid algorithm for the 0-1 knapsack problem. Methods of Oper. Res., 49: 277-293.
11. Martello, S. and P. Toth, 1997. Upper bounds and algorithm for hard 0-1 knapsack problem. Oper. Res., 45: 768-778.
12. Pisinger, D., 1995. An expanding-core algorithm for the exact knapsack problem. Eur. J. Oper. Res., 87: 175-187.
13. Bellman, R.E., 1957. Dynamic Programming. Princeton University Press, Princeton, NJ.
14. Toth, P., 1980. Dynamic programming algorithms for the zero-one knapsack problem. Computing, 25: 29-45.
15. Pisinger, D., 1997. A minimal algorithm for the 0-1 knapsack problem. Oper. Res., 45: 578-767.
16. Martello, S., D. Pisinger and P. Toth, 1999. Dynamic programming and strong bounds for the 0-1 knapsack problem. Manag. Sci., 45: 414-424.
17. Martello, S. and P. Toth, 1984. A mixture of dynamic programming and branch-and-bound for the subset-sum problem. Manag. Sci., 30: 765-771.
18. Martello, D. Pisinger and P. Toth, 2000. New trends in exact algorithm for the 0-1 knapsack problem. Eur. J. Oper. Res., 123: 325-332.
19. Frevill, A., 1991. Contribution a l'Optimization en Nombres Entries, Habilition a Diriger des Recherches. Universite se paris XIII, France.

20. Freville, A. and G. Plateau, 1986. Heuristics and reduction methods for multiple constraints 0-1 linear programming. Eur. J. Oper. Res., 24: 206-215.
21. Sinuany-Stern, Z. and I. Winer, 1994. The one dimensional cutting stock problem using two objectives. J. Oper. Res. Soc., 45: 231-236.
22. Hung, M.S. and J.C. Fisk, 1978. An algorithm for zero-one multiple knapsack problem. Naval Res. Log. Quart., 24: 571-579.
23. Martello, S. and P. Toth, 1980. Solution of the zero-one multiple knapsack problem. Eur. J. Oper. Res., 4: 276-283.
24. Neede, A. and D. Dannenbring, 1977. Algorithm for a specialized segregated storage problem. Technical Report 77-5, University of North Carolina, Durham, NC.
25. Christofides, N., A. Mingozzi and P. Toth, 1979.Combinatorial Optimization. N. Christofides, A. Minglzzi, P. Toth, C. Sandi (Eds.) Wiley, Chrichester, pp: 339-369.
26. Pisinger, D., 1999. An exact algorithm for large multiple knapsack problems. Eur. J. Oper. Res., 114: 528-541.
27. Martello and P. Toth, 1981. A branch and bound algorithm for the zero-one knapsack problem. Discrete Appl. Math., 3: 275-288.
28. Dudzinski, K. and S. Walukiewicz, 1987. Exact methods for the knapsack problem and its generalizations. Eur. J. Oper. Res., 28: 3-21.
29. Nuass, R.M., 1978. The 0-1 knapsack problem with multiple choice constraints. Eur. J. Oper. Res., 2: 125-131.
30. Singa, A. and A.A. Zoltners, 1979. The multiple-choice knapsack problem. Oper. Res., 27: 503-515.
31. Witzgal, C., 1997. On one-row linear programs. Applied Mathematics Division, National Bureau of Standards.
32. Fisher, M.L., 1981. The lagrangian relaxation method for solving integer programming problems. Manag. Sci., 27: 1-18.
33. Dyer, M.E., M. Kayal and J. Walker, 1984. A branch and bound algorithm for solving the multiple choice knapsack problem. J. Comput. Appl. Math., 11: 231-249.
34. Pisinger, D., 1995. A minimal algorithm for the multiple-choice knapsack problem. Eur. J. Oper. Res., 83: 394-410.
35. Pisinger, D., 2001. Budgeting with bound multiple-choice constraints. Eur. J. Oper. Res., 129: 471-480.
36. Garfinkel, R. and G. Nemhauser, 1972. Integer Programming. Wiley, New York.
37. Hu, T.C., 1969. Integer Programming and Network Flows. Addison-Wesley, Reading, MA.
38. Martello, S. and P. Toth, 1990. Knapsack Problems: Algorithms and Computer Implementation, Wiley, New York.
39. Nemhauser, G.L. and L.A. Wolsey, 1988. Integer and Combinatorial Optimization. Wiley, New York.
40. Andonov, R., V. Poirriez and S. Rajopadhye, 2000. Unbounded knapsack problem: Dynamic programming revisited. Eur. J. Oper. Res., 123: 394-407.