The Tenth International Symposium on Operations Research and Its Applications (ISORA 2011)
Dunhuang, China, August 28–31, 2011

A Transitive Reduction Approach to the Precedence-Constrained Knapsack Problem

Byungjun You [*1] Takeo Yamada [†1]

[1]Department of Computer Science, The National Defense Academy,
Yokosuka, Kanagawa 239-8686, Japan

Abstract We are concerned with the precedence-constrained knapsack problem (PCKP), which is a knapsack problem defined on a directed acyclic graph (DAG). It is often the case that the original problem includes redundant constraints, and by eliminating these we obtain a PCKP of (much) smaller size. We show that such a reduction can be explained in terms of the 'transitive reduction' of the DAG associated with the original PCKP. Thus, we present a transitive reduction approach to solve PCKP, where PCKP is first reduced in size by applying transitive reduction, and then the reduced problem is solved using MIP solvers. In numerical experiments, we were able to solve PCKPs with thousands of items within a few seconds on an ordinary computing environment.

Keywords Combinatorial optimization, Knapsack problem, Precedence constraints, Transitive reduction, Directed acyclic graph.

1 Introduction

In this article we are concerned with a variation of the standard 0-1 *knapsack problem* (KP, [6, 7]), which we call the *precedence-constrained knapsack problem* (PCKP, [9]) defined as follows. As in ordinary KP, we have n items to be packed into a knapsack of capacity c. Each item j is associated with its weight w_j and profit p_j. In addition, some pairs of items are ordered as follows. If (i, j) is thus ordered, we must have item i included in the knapsack as a prerequisite to accept item j. Let E be the set of these ordered pairs. Then, the problem is:

$$\textbf{PCKP}: \quad \text{maximize} \quad \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \le c, \tag{2}$$

$$x_i \ge x_j, \quad \forall (i, j) \in E \tag{3}$$

$$x_j \in \{0, 1\}, \quad \forall j. \tag{4}$$

Here, x_j is a decision variable such that $x_j = 1$ if item j is accepted, and $x_j = 0$ otherwise. Without much loss of generality we assume that problem data c and w_j, p_j $(j = 1, \ldots, n)$

*byungjun.ryu@gmail.com
†yamada@nda.ac.jp

are all positive integers. We also assume that $w_j \leq c$ $(j = 1, \ldots, n)$, $\sum_{j=1}^{n} w_j > c$, since otherwise the problem is trivial. PCKP is $\mathcal{NP}$-hard, because without the precedence constraints (3), it reduces to KP, which is already $\mathcal{NP}$-hard.

Precedence relations arise naturally as the consequence of some logical/physical requirements. For example, in *project management* activities are usually arranged in the form of flow chart or network, and each activity can be initiated only when all the preceding activities have been finished. Or, in *open-pit mining* [2] we can not remove a block unless all the blocks lying immediately above have been removed. Mathematically, these relations are represented in the form of (3). Then, if we wish to complete as many projects as possible, or excavate as many blocks as possible in a fixed time period, we need to solve PCKP.

PCKP can be described as a knapsack problem on a *directed acyclic graph* (DAG, [2]) $G = (V, E)$ with vertex set $V = \{v_1, v_2, \cdots, v_n\}$ and edge set $E \subseteq V \times V$. Here, V represents the set of items and node $v_j \in V$ is identified with item j. Also, we identify each of the precedence relations with the corresponding arc in G. That is, $(v_i, v_j) \in E$ implies that x_j can take value 1 only when item $x_i = 1$. Thus, throughout the paper we assume that G is acyclic, and G includes neither self-loops nor parallel arcs.

An important subclass of PCKP is the *tree-knapsack problem* (TKP, [3, 8]), where G is a directed tree rooted at node v_1. For PCKP in general, Samphaiboon *et al.* [9] presented a *dynamic programming* algorithm to solve this problem to optimality. You *et al.* [10] gave a *reduction* algorithm.

Small instances of PCKP can be solved using free or commercial MIP (mixed integer programming) solvers such as Xpress-MP or CPLEX [5], since it is a *binary integer programming* (BIP) problem. To solve larger instances, we propose to reduce the size of the problem by identifying and removing *redundant* constraints included in (3). For example, if we have $x_1 \geq x_2$, $x_2 \geq x_3$ and $x_1 \geq x_3$, the last inequality is clearly redundant, and thus can be dropped. In some PCKPs we may have many redundant inequalities, and in such a case by removing all these constraints, we obtain a PCKP of the size much smaller than the original, which we can solve using MIP solvers.

In graph theory, the reduced PCKP corresponds uniquely to the *transitive reduction* [1, 4] of the DAG representation of the original PCKP. We present a criterion to identify redundant constraints, and based on this give an algorithm to reduce the problem. We evaluate this approach on a series of randomly generated instances, and see that for instances with many constraints (as compared to the number of items), reduction approach is quite prospective.

2 Transitive reduction of DAG and PCKP

For a directed graph G, its *adjacency matrix* $A(G) = (a_{ij})$ is the binary matrix with $a_{ij} = 1$ if $(v_i, v_j) \in E$, and $a_{ij} = 0$ otherwise. Similarly, the *reachability matrix* $R(G) = (r_{ij})$ is given by $r_{ij} = 1$ if there exists a directed path from v_i to v_j, and $r_{ij} = 0$ otherwise. We note that $r_{ii} = 1$ for all $v_i \in V$. We say that $(v_i, v_j) \in E$ is *redundant* if there exists an *alternative path* from v_i to v_j, other than the direct (v_i, v_j). Let E^0 be the set of all the redundant arcs in G, and we define $\underline{E}$ as the set of non-redundant arcs. Thus, we have $\underline{E} = E \setminus E^0$, and removing all the redundant arcs from G, we obtain the subgraph $\underline{G} = (V, \underline{E})$. This is the *transitive reduction* [1] of G.

Here, we note that for each redundant arc $(v_i, v_j) \in E^0$, there exists a path from v_i to v_j in $\underline{G}$, and we have $R(\underline{G}) = R(G)$. Indeed, $\underline{G}$ is the *smallest* subgraph of G that keeps reachability unchanged. Then, it is clear that inequality constraints (3) can be replaced with

$$x_i \geq x_j, \quad \forall (v_i, v_j) \in \underline{E}. \tag{5}$$

In fact, $\underline{E}$ is the smallest subset of inequalities we need in solving the original PCKP. To see this, let

$$X(E) = \{(x_j) \in R_+^{|V|} | x_i - x_j \geq 0, \forall (v_i, v_j) \in E\}$$

be the *polytope* defined by (3). $X(\underline{E})$ is similarly defined for $\underline{E}$. Then, we have the following.

Theorem 1. $X(\underline{E}) = X(E)$, and no subset of E smaller than $\underline{E}$ satisfies this equality.

Thus, by eliminating redundant inequalities, PCKP is reduced from the original problem with $|E|+1$ constraints to the one with smaller $|\underline{E}|+1$ constraints. If G is topologically sorted in the sense that $(v_i, v_j) \in E$ implies $i < j$, we can check if (v_i, v_j) is redundant by the following.

Theorem 2. For a topologically sorted DAG, (v_i, v_j) is redundant if and only if

$$\delta_{ij} := \sum_{l=i+1}^{j-1} a_{il} r_{lj} \geq 1. \tag{6}$$

Thus, given the matrices $A(G)$ and $R(G)$, which can be build in $O(|V|^2)$ space and $O(|V|^3)$ time, we can identify the set of redundant inequalities in $O(|V||E|)$ time.

Example 1. For the DAG of Figure 1, we have

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad R = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{7}$$

From the matrices, we have $\delta_{13} = a_{12} \cdot r_{23} = 1, \delta_{25} = a_{23} r_{35} + a_{24} r_{45} = 1$, and all other δ'_{ij}s are 0. Thus we conclude that (v_1, v_3) and (v_2, v_5) are redundant.

3 Numerical experiment

We implemented the transitive reduction algorithm on a Dell Precision T7400 workstation (CPU: Xeon X5482 Quad-Core × 2, 3.20GHz, RAM: 64GB) and conducted numerical experiments for various types and sizes of instances. We compared our results against the direct solution by CPLEX 12.2.

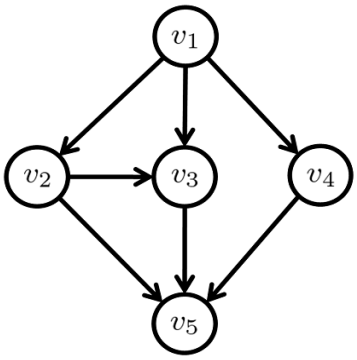

Figure 1: DAG for Example 1

3.1 Design of experiments

Random instances were prepared as follows. Weight w_j and profit $p_j (j = 1, 2, \cdots, n)$ are distributed uniformly and independently over the integer interval $[1, 100]$, and we set the knapsack capacity at $c = 10n$. DAGs are of two types: TYPE 1 and TYPE 2. In TYPE 1, we generate arc (v_i, v_j) with probability α for all pairs of nodes satisfying $i < j$. For TYPE 2, this is limited to the pairs satisfying $i < j \leq i + B$, where B is the *band-width* parameter and in this paper this is fixed at $B = 100$.

Next, for each *maximal* node $v_j \neq v_1$, we pick up node v_i $(i < j)$ at random and add arc (v_i, v_j) to E. Thus, no maximal nodes remain in G other than v_1. Similarly, we make all nodes other than v_n non-minimal. For a DAG, let its *height* and *distance* be the maximum and minimum numbers of steps between v_1 and v_n, respectively. Then, Table 1 gives a summary of the characteristics of TYPE 1 and TYPE 2 instances. Thus, TYPE 2 is relatively 'tall' as compared to TYPE 1 instances, and TYPE 1 is relatively 'wide.'

Table 1: Instance characteristics.

n	TYPE 1			TYPE 2		
	m	*height*	*distance*	m	*height*	*distance*
1000	5212	37	5	9368	187	11
2000	20266	55	3	19433	336	22
3000	45068	72	4	29525	517	32

3.2 Exact solutions

We solve problems to optimality using CPLEX 12.2 directly (DIRECT), as well as by the transitive reduction method (REDUCTION). In REDUCTION, we first reduce the problem and then solve the reduced problem using CPLEX as well. Table 2 summarizes the result of the experiments for TYPE 1 instances, and Table 3 is for TYPE 2. Here, we show the number of redundant inequalities (#*reduced*) and the CPU times in seconds for

the DIRECT method (DIRECTsec) and REDUCTION method (REDUCTsec). Each row is the average of 10 independent runs. When solved to optimality, both of DIRECT and REDUCTION produced identical values. Thus, REDUCTION solves larger instances much faster than DIRECT. This is especially the case for instances with larger α, where we have larger number of constraints with higher percentage of redundancy.

Table 2: TYPE 1 results ($c = 10n$) as average over 10 instances.

α	n	m	#*reduced*	DIRECTsec	REDUCTsec
0.005	1000	2899	239	0.727	0.613
	2000	10365	2298	6.736	5.206
	3000	22739	9068	37.944	23.825
0.010	1000	5215	1589	3.071	2.181
	2000	20266	11363	26.181	9.070
	3000	45068	30928	145.407	29.538
0.020	1000	9998	6114	6.753	2.530
	2000	39977	31553	64.519	8.755
	3000	89387	76346	245.460	19.431
0.040	1000	19786	16166	14.696	1.880
	2000	79730	72191	146.359	6.424
	3000	179494	168303	547.027	11.352

Table 3: TYPE 2 results ($c = 10n$, $B = 100$) as average over 10 instances.

α	n	m	#*reduced*	DIRECTsec	REDUCTsec
0.05	1000	4739	1493	2.043	1.386
	2000	9799	3096	6.690	4.538
	3000	14817	4768	9.627	6.380
0.10	1000	9368	6528	3.069	1.147
	2000	19433	13457	10.990	2.925
	3000	29525	20624	22.107	4.540
0.20	1000	18933	16625	6.469	0.968
	2000	39083	34338	28.781	1.582
	3000	59139	52041	75.882	3.065
0.40	1000	38054	36317	17.505	0.446
	2000	78326	74811	61.380	1.098
	3000	118431	113089	170.680	2.044

4 Conclusion

We studied the PCKP, presented a criterion to identify redundant constraints, and based on this gave an algorithm to reduce the problem size. The algorithm was implemented in ANSI-C language, and numerical experiments were carried out to evaluate the

performance of the developed method. As a result, we were able to solve PCKPs with up to 3000 items of various types within a few seconds in an ordinary computing environment. This method over-performed the direct solution approach by MIP solvers.

References

[1] A.V. Aho, M.R. Garey, J.D. Ullman, The transitive reduction of a directed graph, SIAM Journal on Computing 1 (1972) 131-137.

[2] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithm, and Applications, Prentice-Hall, 1993.

[3] G. Cho, D. X. Shaw, A depth-first dynamic programming algorithm for the tree knapsack problem, INFORMS Journal on Computing 9 (1997) 431-438.

[4] D. Gries, A.J. Martin, J.A. Snepscheut, J.T. Udding, An algorithm for transitive reduction of an acyclic graph, Science of Computer Programming 12 (1989) 151-155.

[5] IBM ILOG CPLEX 12.2, http://www-01.ibm.com/software/integration /optimization/cplex-optimizer/, 2011.

[6] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer Verlag, 2004.

[7] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, New York, 1990.

[8] D. X. Shaw, G. Cho, The critical item, upper bounds and a branch-and-bound algorithm for the tree knapsack problem, Networks 31 (1998) 205-216.

[9] N. Samphaiboon, T. Yamada, Heuristic and exact algorithms for the precedence-constrained knapsack problem, Journal of Optimization Theory and Application 105 (2002) 659-676.

[10] B.-J. You, T. Yamada, A pegging approach to the precedence-constrained knapsack problem, European Journal of Operational Research 183 (2007) 618-632.