Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 183 (2007) 618–632
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Discrete Optimization

A pegging approach to the precedence-constrained knapsack problem

Byungjun You [1], Takeo Yamada *

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

Received 6 May 2005; accepted 10 October 2006
Available online 13 December 2006

Abstract

The knapsack problem (KP) is generalized to the case where items are partially ordered through a set of precedence relations. As in ordinary KPs, each item is associated with profit and weight, the knapsack has a fixed capacity, and the problem is to determine the set of items to be packed in the knapsack. However, each item can be accepted only when all the preceding items have been included in the knapsack. The knapsack problem with these additional constraints is referred to as the precedence-constrained knapsack problem (PCKP). To solve PCKP exactly, we present a pegging approach, where the size of the original problem is reduced by applying the Lagrangian relaxation followed by a pegging test. Through this approach, we are able to solve PCKPs with thousands of items within a few minutes on an ordinary workstation.

Keywords: Combinational optimization; Pegging test; Knapsack problem; Precedence constraints

1. Introduction

Let $G = (V, E)$ be a *directed graph* [10] with vertex set $V = \{1,2,\ldots,n\}$ and edge set $E \subseteq V \times V$. Here, V represents the set of items that can be included into a knapsack of capacity c. Associated with each $j \in V$ are its weight w_j and profit p_j. Without much loss of generality we assume that parameters w_j, p_j $(j = 1,\ldots,n)$ and c are all positive integers. The set of edges E represents *precedence relations* between the items. That is, $(i,j) \in E$ implies that item j can be accepted only when item i has been included in the knapsack. Throughout the paper, we put $m := |E|$, and assume that G is *acyclic*, i.e., no directed cycle is included in G, since otherwise the precedence relations are not well defined.

* Corresponding author.
E-mail address: yamada@nda.ac.jp (T. Yamada).
[1] Current address: The Republic of Korea Navy, Republic of Korea.

0377-2217/$ - see front matter
doi:10.1016/j.ejor.2006.10.031

The problem can be formulated mathematically as a 0–1 programming problem. Let x_j be a variable such that

$$x_j = \begin{cases} 1, & \text{if item } j \text{ is accepted,} \\ 0, & \text{otherwise.} \end{cases}$$

Then, we have the following *precedence-constrained knapsack problem* [21].

PCKP:

$$\text{maximize} \quad z(x) := \sum_{j=1}^{n} p_j x_j \tag{1}$$

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

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

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

Here, without loss of generality, we assume that

$$w_j \leqslant c \quad (\forall j \in V), \quad \sum_{j=1}^{n} w_j > c$$

since otherwise the problem is trivial.

Precedence relations arise naturally as a consequence of logical/physical requirements among items. For example, in a project management activities are usually arranged in the form of a flow chart or a network, and each activity can be initiated only when all the preceding activities have been finished. Or, in open-pit mining [2] we can remove a block only when all the blocks lying immediately above have been removed. Mathematically, these relations are represented in the form of inequality (3). Then, if we wish to complete as many projects as possible, or excavate as many blocks as possible within a fixed time limit, we need to solve PCKP.

PCKP is $\mathcal{NP}$-hard [9]; because without precedence constraint (3), it reduces to the *knapsack problem* (KP, [17,15]), which is already $\mathcal{NP}$-hard. An important subclass of PCKP is the *tree-knapsack problem* (TKP, [5,22,14,12]), where G is a directed tree rooted at node 1. Hirabayashi et al. [11] formulated a tool-module design problem as a PCKP on a bipartite graph. Moriyama et al. [18] generalized this into a PCKP under the name of partially-ordered knapsack problem, and developed a branch-and-bound algorithm with some numerical experiments. Samphaiboon et al. [21] presented a dynamic programming algorithm to solve this problem. If the size of the problem is not so large, it may be solved by commercial or free IP solvers [8]. Using NUOPT [20], a popular IP solver in Japan, we were able to solve most of the randomly generated PCKPs with up to $n = 2000$, but for larger problems we often encountered difficulties in obtaining exact solutions.

In this paper, we propose a novel approach to solve larger PCKPs exactly as follows. First, we eliminate constraints (3) by applying the Lagrangian relaxation, and together with the continuous relaxation of (4), the result is a *continuous knapsack problem*. Then, the pegging test for ordinary KPs can be applied, and if the Lagrangian multipliers are well tuned up, we obtain a PCKP of substantially reduced size. With the precedence constraint (3), we can further derive an improved *block* pegging test, and the reduced PCKPs are often solved by commercial IP solvers. We implement these algorithms, and evaluate the developed method through a series of computational experiments.

2. Upper and lower bounds

This section derives an *upper bound* by applying the *Lagrangian relaxation* [19,23] to PCKP. We also present a *local search* [1] algorithm to obtain a good approximate solution quickly, which gives a *lower bound* to PCKP.

2.1. Lagrangian relaxation

Given a nonnegative multiplier $\lambda := (\lambda_{ij}) \in R^m$, the Lagrangian relaxation of PCKP is **LPCKP**(λ):

$$\text{maximize} \quad L := \sum_{j=1}^{n} p_j x_j + \sum_{(i,j)\in E} \lambda_{ij}(x_i - x_j) \tag{5}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leqslant c, \tag{6}$$

$$0 \leqslant x_j \leqslant 1, \quad \forall j \in V. \tag{7}$$

Here λ_{ij} is the Lagrangian multiplier associated with constraint (3). With 0–1 requirement (4) relaxed to continuous inequality (7), LPCKP(λ) is actually a *continuous* Lagrangian relaxation of PCKP.

The objective function (5) can be rewritten as

$$L := \sum_{j=1}^{n} \left(p_j + \sum_{k\in S_j^+} \lambda_{jk} - \sum_{i\in S_j^-} \lambda_{ij} \right) x_j, \tag{8}$$

where S_j^+ (resp. S_j^-) is the set of *terminating* (resp. *originating*) vertices of edges that originate from (resp. terminate in) node j. If we fix $\lambda \geqslant 0$, LPCKP(λ) is the *continuous* KP whose solution is easily found [16]. Let $\bar{x}(\lambda) = (\bar{x}_j)$ denote an optimal solution to LPCKP(λ) with the corresponding optimal value $\bar{z}(\lambda)$, and $z^\star$ is the optimal objective value to PCKP. Then, we have

$$z^\star \leqslant \bar{z}(\lambda), \tag{9}$$

i.e., $\bar{z}(\lambda)$ gives an upper bound to PCKP. As a function of λ, it is known [19,23] that

(i) $\bar{z}(\lambda)$ is a piecewise-linear, convex function of λ,

(ii) if $\bar{z}(\lambda)$ is differentiable at λ,

$$\frac{\partial \bar{z}(\lambda)}{\partial \lambda_{ij}} = \bar{x}_i - \bar{x}_j, \quad \forall (i,j) \in E, \tag{10}$$

(iii) for an arbitrary $\lambda \geqslant 0$, if $\bar{x}(\lambda)$ is feasible to PCKP and

$$\lambda_{ij}(\bar{x}_i - \bar{x}_j) = 0, \quad \forall (i,j) \in E, \tag{11}$$

then $\bar{x}(\lambda)$ is an optimal solution to PCKP.

2.2. Subgradient method

For an arbitrary $\lambda \geqslant 0$, $\bar{z}(\lambda)$ gives an upper bound to PCKP. To make this value as small as possible, we employ the following *subgradient method* [19], where a subgradient is the vector $g = (g_{ij})$ whose element is given by (10) as $g_{ij} := \bar{x}_i - \bar{x}_j$. Then, the direction of search $d = (d_{ij})$ is the projection of $-g$ on the non-negative space, i.e.,

$$d_{ij} := \begin{cases} -g_{ij}, & \text{if } \lambda_{ij} > 0 \text{ or } (\lambda_{ij} = 0 \text{ and } g_{ij} < 0), \\ 0, & \text{otherwise.} \end{cases}$$

Algorithm Subgradient_Method

Step 1. Set $\lambda = 0$.
Step 2. Solve LPCKP(λ).
Step 3. Calculate the subgradient g and the direction of search d.
Step 4. (1-dim search) Find $\alpha^\dagger \geqslant 0$ such that $\bar{z}(\lambda + \alpha d)$ is minimized.
Step 5. If (11) is satisfied or $\bar{z}(\lambda) \cong \bar{z}(\lambda + \alpha^\dagger d)$, stop.
Step 6. Update $\lambda \leftarrow \lambda + \alpha^\dagger d$ and go to *Step 2*.

Let $\lambda^\dagger$ be λ at the termination of the above algorithm. Then, we obtain the 'best' upper bound to PCKP as $\bar{z} := \bar{z}(\lambda^\dagger)$.

2.3. Lower bounds

Let $\bar{x}(\lambda^\dagger) = (\bar{x}_j^\dagger)$ be an optimal solution to LPCKP($\lambda^\dagger$). If this satisfies (3) and (4), this is a feasible solution; hence, the corresponding objective value gives a lower bound to PCKP. If some of the constraints (3) and (4) are violated, we still obtain a feasible solution by modifying $\bar{x}(\lambda^\dagger)$ in the following way: if $0 < \bar{x}_j^\dagger < 1$ for some $j \in V$ or if $\bar{x}_j^\dagger > \bar{x}_i^\dagger$ for some $(i,j) \in E$, we simply put $\bar{x}_j^\dagger \leftarrow 0$. We call the resulting feasible solution the *Lagrangian solution* to PCKP.

To further improve the solution, we employ the *2-opt method* [1], which repeat the following as long as possible.

Procedure 2-opt

(i) In the current solution x find a pair of items i and j, such that $x_i = 1$ and $x_j = 0$, and furthermore these can be swapped as $x_i = 0$ and $x_j = 1$ without destroying feasibility of the resulting solution.
(ii) If this increases the value of the knapsack, carry out this swapping.

We call the resulting solution the *2-opt solution*, with the corresponding lower bound denoted as $\underline{z}$.

3. Pegging tests

A pegging test is well known for the standard 0–1 KP [13,7,6]. By applying this test, many variables are fixed either at 0 or 1, and removing these variables we obtain a problem of (often significantly) reduced size. In this section, we show that the same pegging test can be applied to PCKPs if we introduce the Lagrangian relaxation first.

3.1. Plain pegging test

Assume that we have an optimal Lagrangian multiplier $\lambda^\dagger$, the corresponding upper bound $\bar{z} = \bar{z}(\lambda^\dagger)$, and a lower bound $\underline{z}$ to PCKP, and let us define:

$$\bar{p}_j := p_j + \sum_{k \in S_j^+} \lambda_{jk}^\dagger - \sum_{i \in S_j^-} \lambda_{ij}^\dagger. \tag{12}$$

Then, LPCKP($\lambda^\dagger$) can be rewritten as

$$\text{maximize} \quad \sum_{j=1}^{n} \bar{p}_j x_j \tag{13}$$

$$\text{subject to} \quad (6) \text{ and } (7). \tag{14}$$

For an arbitrary $j = 1,\ldots,n$, let $z^\star(x_j = u)$ denote the optimal objective value to PCKP with an additional constraint $x_j = u$, where u is either 0 or 1. Similarly, $\bar{z}(x_j = u)$ denotes the optimal objective value to LPCKP($\lambda^\dagger$) with $x_j = u$. Then, for $j = 1,\ldots,n$ the followings are obvious.

$$z^\star = \max\{z^\star(x_j = 0), \quad z^\star(x_j = 1)\}, \tag{15}$$

$$z^\star(x_j = u) \leqslant \bar{z}(x_j = u). \tag{16}$$

Then, if

$$\bar{z}(x_j = 0) < \underline{z}, \tag{17}$$

$x_j^\star = 0$ is not possible in any optimal solution $x^\star = (x_k^\star)$ to PCKP, i.e., we necessarily have $x_j^\star = 1$. Similarly, in the case that

$$\bar{z}(x_j = 1) < \underline{z}, \tag{18}$$

$x_j^\star = 0$ must follow. This is the basic idea of the pegging test. To determine (17) and (18) quickly, the following shortcut is taken. First of all, without loss of generality, we assume the following.

$\mathbf{A}_1$: $\bar{p}_j > 0, \quad \forall j \in V$,
$\mathbf{A}_2$: The items are ordered in the non-increasing order of $\bar{p}_j/w_j$.

Let W_j and P_j be, respectively, the *cumulative* weight and profit, i.e.,

$$W_j := \sum_{i=1}^{j} w_i, \quad P_j := \sum_{i=1}^{j} \bar{p}_i, \tag{19}$$

where $W_0 = P_0 = 0$. Then, $\{(W_j, P_j) | j = 0, \ldots, n\}$ gives a piecewise-linear, monotonically non-decreasing, concave function [16].

The intersection of this broken line with the vertical line $W = c$ gives the upper bound $\bar{z}$. The item s satisfying $W_{s-1} < c \leqslant W_s$ is said to be a *critical item*, and we define

$$r_s := \bar{p}_s / w_s.$$

Here, if for any $j < s$ we set $x_j = 0$, it is known [13,7] that

$$\bar{z}(x_j = 0) \leqslant \bar{z} - \theta_j, \tag{20}$$

where we define

$$\theta_j := \bar{p}_j - r_s w_j. \tag{21}$$

Then, if

$$\bar{z} - \underline{z} < \theta_j, \tag{22}$$

from (20) we have $\bar{z}(x_j = 0) < \underline{z}$, and thus $x_j^\star = 1$. By a similar argument, if

$$\bar{z} - \underline{z} < -\theta_j \tag{23}$$

for any $j > s$, we obtain $x_j^\star = 0$. Thus we have the following.

Theorem 1. *For any optimal solution $x^\star = (x_j^\star)$ to PCKP, both of the followings are true.*

(*i*) $\bar{z} - \underline{z} < \theta_j \Rightarrow x_j^\star = 1$,
(*ii*) $\bar{z} - \underline{z} < -\theta_j \Rightarrow x_j^\star = 0$.

Given a pair of upper and lower bounds, by applying this theorem some variables are fixed either at 0 or 1, and removing these variables we obtain a PCKP of (often significantly) reduced size. We call this *reduction* by pegging test, and thus obtain a *reduced problem*.

3.2. Block pegging test

Again, we consider LPCKP($\lambda^\dagger$), and assume $\mathbf{A}_1$ and $\mathbf{A}_2$ as before. Item s stands for the critical item. In the acyclic graph $G = (V, E)$, if there exists a directed path from node i to j, we say that j is a *descendent* of i. By D_j we denote the set of all descendents of j, where we note that this includes node j itself. Furthermore, for an arbitrary node j, we define a subset of D_j by

$$D'_j := \{k \in D_j | k < s\}.$$

Then, by setting $x_j = 0$, we have $x_k = 0$ for all $k \in D'_j$, and the broken line connecting (W_0, P_0), $(W_1, P_1), \ldots, (W_n, P_n)$ will be shifted to the lower leftwards by (w'_j, p'_j), where we put

$$w'_j := \sum_{k \in D'_j} w_k, \quad \bar{p}'_j := \sum_{k \in D'_j} \bar{p}_k. \tag{24}$$

Define

$$\Theta_j := \bar{p}'_j - r_s w'_j. \tag{25}$$

Then, similar to (22) if

$$\bar{z} - \underline{z} < \Theta_j \tag{26}$$

we necessarily obtain $x^\star_j = 1$.

Similarly by considering the case of $j > s$, we obtain the following theorem.

Theorem 2. *For any optimal solution $x^\star = (x^\star_j)$ to PCKP, both of the followings hold.*

(*i*) $\bar{z} - \underline{z} < \Theta_j \Rightarrow x^\star_j = 1$,
(*ii*) $\bar{z} - \underline{z} < -\Theta_j \Rightarrow x^\star_j = 0$.

3.3. Pegging algorithm

Now we can solve PCKP in the following way.

Algorithm Plain_Pegging (resp. Block_Pegging)

Step 1. Compute the upper and lower bounds by the Lagrangian relaxation and 2-opt methods.
Step 2. Reduce the problem size by applying Theorem 1 (resp. Theorem 2).
Step 3. Solve the reduced problem using an appropriate IP solver.

Here, if we use the plain pegging test (Theorem 1) to reduce the problem size, we call this the *plain pegging;* and if Theorem 2 is used, we have the *block pegging* algorithm. *Step 2* can be done by calculating θ_j (or Θ_j, resp.) and checking if (22) (or (26), resp.) is satisfied for $j = 1, \ldots, n$. This takes $O(n)$ time. After this, the reduced problem can be solved using a commercial or free IP solver.

4. Virtual pegging test

The usefulness of the pegging test depends on the gap between the upper and lower bounds. If the gap is not small enough, the effectiveness of our method is limited, since the size of the problem will not be reduced much in such a case. In this section, we introduce the *virtual* pegging test to cope with this problem.

4.1. The principle

In the pegging test, we input a pair of upper and lower bounds $\bar{z}$ and $\underline{z}$ to the pegging algorithm, and partition the original problem into a fixed part and the remaining reduced problem. Here, upper and lower bounds satisfy

$$\underline{z} \leqslant z^\star \leqslant \bar{z}. \tag{27}$$

However, we can try the pegging test using an arbitrary value l within $[\underline{z}, \bar{z}]$ as a hypothetical lower bound. Such an l is referred to as a *trial value*.

Let the set of all the feasible solutions $x = (x_j)$ satisfying (2)–(4) be X. Then, if we carry out the pegging test with $\bar{z}$ and l, some x_j's will be fixed either at 0 or 1. But this pegging is not guaranteed to be correct because l is not necessarily a lower bound. Let the index sets of the variables fixed at 0 (or 1) by the above pegging procedure be $F_0(l)$ ($F_1(l)$, resp.). Then, we have the following *reduced problem*.

$\mathbf{R}(l)$:

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j \tag{28}$$

$$\text{subject to} \quad x \in X, \tag{29}$$

$$x_j = 1, \quad j \in F_1(l), \tag{30}$$

$$x_j = 0, \quad j \in F_0(l). \tag{31}$$

The optimal objective value to this problem will be denoted as $z_l^\star$, and is referred to as the *realization* for the trial value l. If $\mathbf{R}(l)$ is infeasible, we define $z_l^\star := -\infty$. Then, we have

Theorem 3. *For an arbitrary trial value $l \leqslant \bar{z}$ and its realization $z_l^\star$, the followings are true.*

(*i*) $l \leqslant z^\star \Rightarrow z_l^\star = z^\star$,
(*ii*) $l > z^\star \Rightarrow z_l^\star \leqslant z^\star$,
(*iii*) $l \leqslant l' \Rightarrow z_l^\star \geqslant z_{l'}^\star$,
(*iv*) $l \leqslant z_l^\star \Rightarrow z_l^\star = z^\star$.

Proof. (i) If $l \leqslant z^\star$, l is actually a lower bound; thus the pegging test works correctly and finds the optimal value $z^\star$, i.e., $z_l^\star = z^\star$. (ii) Note that for an arbitrary $l \leqslant \bar{z}$, $\mathbf{R}(l)$ is PCKP with the additional constraints (30) and (31). Then, by definition, the optimal objective values satisfy this relation. (iii) In applying Theorem 1 (or Theorem 2) with the gap of $\bar{z} - l$, we note that the larger the trial value l is (and thus the smaller the gap is), the more variables are fixed, i.e., for $l \leqslant l'$ we have $F_u(l) \subseteq F_u(l')$ $(u = 0, 1)$. From this, the above relation is straightforward. (iv) If $l > z^\star$, from (ii) we have $z_l^\star \leqslant z^\star$, and thus $l > z_l^\star$. This is a contradiction. So, we have $l \leqslant z^\star$, and from (i) obtain $z_l^\star = z^\star$. □

As a direct corollary of (iii), if $R(l)$ is infeasible, then $\mathbf{R}(l')$ is also infeasible for all $l' \geqslant l$.

4.2. A virtual pegging algorithm

For an arbitrary trial value l, after carrying out the virtual pegging test and solving the reduced problem $\mathbf{R}(l)$, we obtain the corresponding $z_l^\star$. Then, if (iv) is satisfied in Theorem 3, the problem is solved. In addition, if gap $:= \bar{z} - l$ is small, it is probable that $\mathbf{R}(l)$ is a much smaller problem than the original. Thus, in such a case we obtain an optimal solution by solving this reduced problem. We propose the following algorithm, which includes measures for the case where (iv) is not satisfied.

Algorithm Virtual_Pegging_Test

Step 1. $l \leftarrow \max\{\bar{z} - \delta, \underline{z}\}$.
Step 2. Carry out the pegging test with trial value l, solve $\mathbf{R}(l)$ and obtain $z_l^\star$.
Step 3. If $l \leqslant z_l^\star$, go to *Step 5*.
Step 4. Update $\underline{z} \leftarrow \max\{\underline{z}, z_l^\star\}$ and $l \leftarrow \max\{l - \delta, \underline{z}\}$, and go to *Step 2*.
Step 5. The optimal value is obtained as $z^\star = z_l^\star$.

Here, δ is an arbitrary 'small' margin. We set the trial value initially at $l = \bar{z} - \delta$ unless this is not smaller than $\underline{z}$. Then, if the optimal value is not found in step 3, the trial value is further lowered by δ, and we repeat Steps 2–4 all over again until an optimal solution is found.

5. Numerical experiments

In this section we evaluate the performance of our approach through a series of numerical experiments. We implemented the algorithm in C language and conducted some computation on an IBM RS/6000 Model 270

workstation (CPU: POWER3-II SMP 2way, 375 MHz). The program (including a subprogram to generate random instances) is available from [24] with some explanatory materials.

5.1. Test problems

The number of items n is set between 1000 and 16,000, and the weights and profits are assumed as the following.

- Uncorrelated case (UNCOR)
  w_j: Uniformly random over $[1, 1000]$,
  p_j: Uniformly random over $[1, 1000]$, independent of w_j.
- Weakly correlated case (WEAK)
  $[w_j]$: Uniformly random over $[1, 1000]$,
  $[p_j]$: Uniformly random over $[w_j, w_j + 200]$.

The capacity of the knapsack is fixed at first at $c = 250n$. This means that without precedence constraints approximately a half of all the items can be included in the knapsack, since the average weight of items is approximately 500. The precedence constraints are generated randomly with probability $d/(n-1)$. Hence, the average number of precedence constraints is $nd/2$, where parameter d controls the density of this constraints, and we examined the cases of $d = 0.1$–0.8.

5.2. Upper and lower bounds

Tables 1 and 2 give an overview of the computation of upper and lower bounds for the UNCOR and WEAK cases, respectively. Here shown are d, n, m, the gap between the Lagrangian upper bound $\bar{z}$ and the 2-opt lower bound $\underline{z}$, as well as the computing time in seconds. The column 'ratio' shows the percentage

Table 1
Upper and lower bounds (UNCOR)

d	n	m	Gap			CPU seconds			
			Average	Maximum	Minimum	Average	Maximum	Minimum	Ratio
0.1	1000	55.2	118.3	856	10	0.22	0.37	0.13	0.86
	2000	100.3	35.4	65	3	0.70	1.01	0.52	0.73
	4000	198.6	37.9	58	18	2.62	3.14	1.91	0.71
	8000	402.4	25.6	54	8	8.21	9.69	6.79	0.57
	16,000	807.4	31.5	66	16	29.89	36.77	24.57	0.46
0.2	1000	107.4	59.5	114	19	0.34	0.19	0.67	0.88
	2000	201.2	37.0	63	9	1.00	2.28	0.64	0.82
	4000	395.8	47.3	80	17	3.14	5.04	1.99	0.74
	8000	800.2	23.7	67	4	11.79	21.16	6.04	0.73
	16,000	1609.1	24.5	43	3	40.18	63.18	29.81	0.64
0.4	1000	206.2	85.3	189	43	0.61	1.32	0.33	0.93
	2000	402.9	129.7	542	4	2.11	4.12	0.86	0.92
	4000	807.1	111.3	691	21	8.35	16.82	3.97	0.90
	8000	1593.0	160.4	1131	12	24.14	26.68	17.47	0.85
	16,000	3210.3	78.6	253	3	82.95	108.95	61.95	0.83
0.8	1000	406.0	206.8	618	21	1.46	2.79	0.58	0.97
	2000	802.3	349.5	1051	37	6.65	13.68	3.41	0.97
	4000	1606.3	349.1	1925	22	26.93	35.52	18.35	0.97
	8000	3196.7	227.6	1021	12	76.28	126.77	52.19	0.95
	16,000	6402.6	340.9	1036	19	357.58	466.54	235.50	0.95

Table 2
Upper and lower bounds (WEAK)

d	n	m	Gap			CPU seconds			
			Average	Maximum	Minimum	Average	Maximum	Minimum	Ratio
0.1	1000	55.2	21.9	42	7	0.21	0.33	0.13	0.80
	2000	100.3	18.2	54	5	0.61	0.94	0.43	0.70
	4000	198.6	13.6	41	4	1.89	2.75	1.38	0.65
	8000	402.4	36.4	99	3	7.88	10.17	6.50	0.46
	16,000	807.4	40.5	107	10	31.57	49.24	24.07	0.39
0.2	1000	107.4	28.7	73	11	0.25	0.39	0.19	0.84
	2000	201.2	33.7	103	8	0.92	1.64	0.46	0.79
	4000	395.8	20.7	35	6	2.78	4.06	1.68	0.72
	8000	800.2	64.9	268	8	11.93	19.33	6.95	0.67
	16,000	1609.1	66.7	257	6	36.83	46.81	29.06	0.54
0.4	1000	206.2	36.3	110	11	0.50	1.09	0.23	0.91
	2000	402.9	34.0	92	2	1.31	2.55	0.71	0.87
	4000	807.1	70.0	271	2	6.40	10.00	2.78	0.87
	8000	1593.0	73.6	198	11	22.86	39.52	15.04	0.83
	16,000	3210.3	58.2	192	3	75.91	95.95	63.19	0.79
0.8	1000	406.0	54.8	135	18	1.17	1.64	0.43	0.98
	2000	802.3	79.2	310	12	4.65	7.38	2.54	0.97
	4000	1606.3	116.1	252	5	14.90	25.75	9.98	0.97
	8000	3196.7	111.6	360	6	56.85	84.53	36.48	0.94
	16,000	6402.6	358.2	914	69	292.16	530.76	136.33	0.93

of the CPU time spent for the computation of the upper bound. In these tables, each row shows the average, maximum and minimum of these values over 10 randomly generated instances.

From these tables, we observe the followings:

1. The gap between $\bar{z}$ and $\underline{z}$ usually increases with d, and it is often smaller in WEAK than in UNCOR.
2. Computation time for the upper bound increases with n as well as with d, but it is rather insensitive to the correlation type of the instances. CPU time to compute the lower bounds also increases with n, but it stays almost constant with the increase of d or the change of correlation type. As a consequence, with the increase of d the CPU time to compute the upper bounds becomes relatively time consuming (see the column of 'ratio').

5.3. Problem reduction by pegging tests

Tables 3 and 4 summarize the results of computation of the pegging tests. Here the columns PLAIN and BLOCK show the results of plain and block pegging, respectively. The size of the reduced problem is shown by n' (the number of variables) and m' (the number of precedence constraints), and the ratio of reduction is given by 'reduc', which is defined as the 'geometric mean' of n'/n and m'/m, i.e.,

$$\text{reduc} := \sqrt{n'm'/nm}. \tag{32}$$

Computing time for the pegging tests is negligible; indeed, in all cases tested it took at most 0.03 seconds.

The findings from these tables are as follows

1. The reduced problem is always smaller (although slightly) in the block pegging method than in the plain pegging.
2. The pegging method becomes less effective as d increases. Also, correlation between weights and profits makes the pegging less effective.

Table 3
Pegging test (UNCOR)

d	n	PLAIN			BLOCK		
		n'	m'	Reduction	n'	m'	Reduction
0.1	1000	162.4	7.2	0.14	161.5	6.2	0.13
	2000	145.5	3.9	0.05	144.3	2.7	0.04
	4000	305.5	8.9	0.06	304.0	7.4	0.05
	8000	421.1	12.0	0.04	417.9	8.7	0.03
	16,000	1011.3	27.2	0.05	1003.4	19.1	0.04
0.2	1000	118.9	8.1	0.09	117.0	6.0	0.08
	2000	152.2	5.7	0.05	150.7	4.1	0.04
	4000	371.8	17.7	0.06	367.5	13.2	0.06
	8000	396.5	20.1	0.04	392.6	16.1	0.03
	16,000	811.3	38.3	0.03	801.9	28.9	0.03
0.4	1000	167.9	19.4	0.13	163.2	14.4	0.11
	2000	436.6	68.8	0.19	422.7	52.4	0.17
	4000	715.8	108.8	0.15	693.4	82.7	0.13
	8000	1575.1	242.5	0.17	1543.9	205.8	0.16
	16,000	2416.2	310.2	0.12	2335.4	217.0	0.10
0.8	1000	357.5	106.2	0.31	336.5	77.8	0.25
	2000	915.8	308.8	0.42	872.2	243.3	0.36
	4000	1508.4	471.8	0.33	1442.6	383.5	0.29
	8000	2715.3	805.2	0.29	2575.7	615.8	0.25
	16,000	7186.8	2374.8	0.41	6829.8	1855.2	0.35

Table 4
Pegging test (WEAK)

d	n	PLAIN			BLOCK		
		n'	m'	Reduction	n'	m'	Reduction
0.1	1000	210.0	7.6	0.17	207.4	5.0	0.14
	2000	350.4	10.4	0.13	348.4	8.4	0.12
	4000	536.2	12.0	0.09	533.1	8.9	0.08
	8000	2454.6	89.8	0.26	2434.3	68.5	0.23
	16,000	5680.5	211.1	0.30	5629.1	158.1	0.26
0.2	1000	270.2	19.3	0.22	263.4	12.2	0.17
	2000	604.0	38.3	0.24	594.8	28.4	0.20
	4000	816.2	43.7	0.15	805.0	31.6	0.13
	8000	3478.3	283.0	0.39	3428.6	228.4	0.35
	16,000	7282.3	592.5	0.41	7177.4	479.1	0.37
0.4	1000	328.5	46.9	0.27	316.5	33.1	0.23
	2000	613.5	85.0	0.25	592.4	60.5	0.21
	4000	1709.0	292.0	0.39	1666.7	240.6	0.35
	8000	4372.2	714.0	0.49	4235.1	547.7	0.43
	16,000	7124.0	1143.1	0.40	6898.1	873.3	0.34
0.8	1000	464.2	140.5	0.40	433.0	99.4	0.33
	2000	1017.1	327.0	0.45	963.7	250.8	0.39
	4000	2585.4	935.7	0.61	2473.7	764.5	0.54
	8000	4619.7	1628.9	0.54	4424.0	1337.2	0.48
	16,000	14,289.6	5414.3	0.87	13,838.7	4729.6	0.80

The second observation above may better be explained in terms of the close relation between the average gap (in Tables 1 and 2) and the ineffectiveness of pegging measured by 'reduc', which can be represented as the following regression functions:

$$\text{UNCOR}: \ \text{reduc} = 0.015 + 0.00097 \cdot \text{gap} \quad (R^2 = 0.970)$$

$$\text{WEAK}: \ \text{reduc} = 0.153 + 0.00230 \cdot \text{gap} \quad (R^2 = 0.815)$$

For example, if we have gap = 100, the expected ratio of reduction would be 0.11 in UNCOR, and 0.38 in WEAK.

Table 5
Exact solutions by NUOPT (UNCOR)

d	*n*	#sol	CPU seconds			BBN		
			Average	Maximum	Minimum	Average	Maximum	Minimum
0.1	1000	10	6.92	14.26	0.63	1185.7	6688	103
	2000	10	22.60	80.03	0.65	3067.5	11,966	14
	4000	7	75.58	180.99	14.84	5196.1	12,888	653
	8000	7	88.47	243.79	12.46	1601.2	5206	27
	16,000	5	425.29	1011.85	29.32	5101.2	16,138	24
0.2	1000	10	8.44	19.84	0.46	1966.1	4529	54
	2000	10	18.47	71.29	2.94	2196.6	9426	319
	4000	6	43.17	71.54	22.10	1681.8	3878	392
	8000	7	200.10	573.94	38.60	4086.0	18,629	230
	16,000	3	284.11	631.02	52.82	998.3	2364	18
0.4	1000	10	11.41	40.53	1.37	1966.5	7303	251
	2000	10	27.91	89.07	3.10	2116.4	8330	144
	4000	6	82.81	159.76	10.73	2247.5	4318	280
	8000	6	426.55	1853.22	15.72	23,800.8	129,326	55
	16,000	3	437.57	526.79	384.81	1210.7	1867	672
0.8	1000	10	16.47	34.96	2.62	1930.0	3521	302
	2000	10	91.52	176.58	4.30	3520.0	6310	167
	4000	7	142.64	300.57	52.84	1984.4	3680	673
	8000	3	180.13	266.85	27.68	1248.0	2611	38
	16,000	3	549.27	879.44	190.37	1475.3	2869	110

Table 6
Exact solutions by NUOPT (WEAK)

d	*n*	#sol	CPU seconds			BBN		
			Average	Maximum	Minimum	Average	Maximum	Minimum
0.1	1000	10	46.87	207.20	1.99	27,310.9	117,046	775
	2000	7	678.33	2739.03	1.42	255,172.3	1,055,088	40
	4000	5	757.12	1832.44	15.78	143,128.8	373,478	366
0.2	1000	10	45.06	172.12	1.19	24,920.5	101,854	210
	2000	7	325.08	937.88	8.95	106,151.4	321,047	605
	4000	3	6.95	8.39	4.23	51.7	84	17
0.4	1000	10	37.89	199.45	3.63	16,031.0	103,943	1116
	2000	10	338.39	1831.27	2.66	117,147.8	692,331	94
	4000	2	348.10	395.39	300.81	40,286.5	44,968	35,605
0.8	1000	10	36.16	162.49	0.44	11,402.0	71,155	185
	2000	8	276.46	1039.92	7.13	64,435.5	271,908	360
	4000	5	311.57	840.26	68.72	33,628.2	124,502	1085

5.4. Exact solutions

We solve the problems to optimality using NUOPT [20] directly, as well as by the block pegging (referred to as BLOCK) and the virtual pegging methods (VIRTUAL). NUOPT is a commercial MP solver developed by a Japanese vendor, and is claimed to be competitive to such popular solvers as LINDO or CPLEX. We use NUOPT to solve the reduced problems in BLOCK and VIRTUAL as well. Tables 5 and 6 show the results of NUOPT, Tables 7 and 8 are those of BLOCK, and Tables 9 and 10 summarize the results of VIRTUAL, all for UNCOR and WEAK cases.

For each value of d and n, the same 10 instances were solved as in Tables 1 and 2 with the limit time of 3600 CPU seconds. Here #sol denotes the number of solved instances within this time limit, and #rep in Tables 9 and 10 is the number of repetitions of Steps 2–4 in Virtual_Pegging_Test. In these tables 'CPU seconds' is the sum of the computation time for the upper and lower bounds, pegging test, and the time to solve the reduced problem using NUOPT. Also 'BBN' is the number of the branch and bound nodes generated by NUOPT to solve respective problems. The average, maximum and minimum of these values over the *solved* (#sol) instances are given in these tables.

In VIRTUAL we fixed δ at $\delta = 10$, since the average of the gap $\bar{z} - z^{\star}$ was approximately 10 in the computation of BLOCK. We observe the followings

1. BLOCK often solves instances that NUOPT cannot solve, and VIRTUAL sometimes solves instances that BLOCK cannot. The converse is very rare. In our experiments, NUOPT was able to solve only one instance (WEAK, $d = 0.2$, $n = 2000$) that BLOCK could not solve. This instance was solved by VIRTUAL.
2. In UNCOR cases, NUOPT frequently fails to solve instances with $n \geqslant 4000$. Contrary, for $n \leqslant 8000$ almost all instances were solved either by BLOCK or VIRTUAL. Even if NUOPT solves these instances, it is usually time consuming.
3. Correlation between weights and profits makes problems much harder. In WEAK, NUOPT sometimes fails for problems with $n = 2000$. With VIRTUAL we are able to solve these problems, but compared to the UNCOR instances, it takes much longer CPU time.

Table 7
Exact solutions by BLOCK (UNCOR)

d	n	#sol	CPU seconds			BBN		
			Average	Maximum	Minimum	Average	Maximum	Minimum
0.1	1000	10	0.72	2.53	0.19	662.5	2555	92
	2000	10	1.38	3.33	0.64	1183.4	2953	14
	4000	10	9.18	19.83	2.85	8999.8	30,966	612
	8000	10	292.40	2753.70	7.50	278,314.1	2,622,402	29
	16,000	8	244.28	1223.88	25.57	217,718.9	1,128,487	20
0.2	1000	10	0.84	1.78	0.38	866.0	2007	54
	2000	10	1.74	4.27	0.69	1492.6	2491	181
	4000	10	17.62	102.24	3.66	22,353.2	169,373	1015
	8000	10	254.35	2366.15	7.39	480,265.1	4,744,547	239
	16,000	8	240.36	827.42	33.09	184,046.6	539,396	18
0.4	1000	10	1.41	3.21	0.51	906.2	2627	244
	2000	10	11.25	49.88	1.30	1530.9	8100	22
	4000	9	11.39	31.84	4.42	2228.0	4714	270
	8000	8	67.38	238.14	24.69	280,91.9	137,069	54
	16,000	6	382.49	1292.05	111.70	551,550.0	2,744,246	667
0.8	1000	10	4.77	24.52	1.09	1379.6	4489	302
	2000	10	43.16	145.19	3.83	2611.3	6525	167
	4000	10	68.44	165.79	27.89	4269.1	13,730	674
	8000	6	138.74	340.02	53.82	15,895.3	72,327	38
	16,000	8	784.37	2782.35	290.37	48,035.8	296,551	92

Table 8
Exact solutions by BLOCK (WEAK)

d	n	#sol	CPU seconds			BBN		
			Average	Maximum	Minimum	Average	Maximum	Minimum
0.1	1000	10	13.49	88.28	0.80	20,105.0	112,990	783
	2000	8	235.46	978.73	0.71	467,400.3	1,972,277	40
	4000	8	370.14	1498.35	1.66	510,953.8	2,635,522	343
0.2	1000	10	14.11	61.45	0.57	23,210.9	99,723	210
	2000	8	261.94	1589.88	0.69	323,718.1	1,961,311	615
	4000	5	674.36	1899.56	1.91	648,243.0	1,690,114	27
0.4	1000	10	10.06	44.83	1.67	16,158.7	99,723	1265
	2000	10	112.29	516.55	1.84	118,568.5	692,357	99
	4000	3	301.02	843.68	2.78	763,563.0	2,249,335	8407
0.8	1000	10	15.29	66.83	1.18	11,355.2	68,881	1
	2000	10	131.78	1008.87	9.36	44,449.5	272,364	378
	4000	6	437.26	1228.72	9.98	397,797.8	2,178,465	1159

Table 9
Exact solutions by VIRTUAL (UNCOR)

d	n	#sol	#rep	CPU seconds			BBN		
				Average	Maximum	Minimum	Average	Maximum	Minimum
0.1	1000	10	1.5	1.04	1.56	0.11	229.5	484	31
	2000	10	1.0	0.88	1.95	0.50	1339.1	6073	14
	4000	10	1.0	4.36	11.15	1.83	6475.4	26,673	417
	8000	10	1.0	130.44	1216.76	4.07	258,468.1	2,491,418	29
	16,000	8	1.0	151.18	737.19	10.80	215,314.9	1,135,948	22
0.2	1000	10	1.4	0.47	0.88	0.20	286.7	656	54
	2000	10	1.2	1.56	2.48	0.62	1117.3	5369	172
	4000	10	1.0	4.30	7.79	2.22	5212.5	16,227	542
	8000	10	1.0	210.14	1992.21	5.54	47,9791.7	4,746,000	230
	16,000	8	1.0	129.66	313.95	20.53	169,248.8	483,664	18
0.4	1000	10	1.4	0.65	1.46	0.32	289.8	649	78
	2000	10	3.6	18.26	27.55	0.82	870.4	4244	128
	4000	10	1.0	8.46	17.18	3.43	1815.1	3614	162
	8000	10	3.8	293.86	788.74	17.50	275,61.6	147,355	54
	16,000	7	1.9	438.59	2189.79	56.22	594,641.6	3,509,838	58
0.8	1000	10	5.0	3.53	5.65	0.64	313.3	946	25
	2000	10	4.0	11.72	23.79	3.46	608.9	1204	350
	4000	10	4.0	68.64	85.94	19.30	2471.7	12,999	22
	8000	9	1.1	77.25	120.58	50.89	8280.4	25,000	38
	16,000	10	1.9	608.24	848.37	241.89	123,548.5	470,867	1228

One reason of the poor performance in WEAK instances is that we are using NUOPT to solve the reduced problem. Even for standard KPs the correlated instances are often hard to solve by general purpose MP solvers.

5.5. Sensitivity analysis

Table 11 gives the result of a sensitivity analysis with respect to the knapsack capacity. We fix $d = 0.2$, and consider the UNCOR case. We compare the CPU time in seconds, with average, maximum and minimum over

Table 10
Exact solutions by VIRTUAL (WEAK)

d	n	#sol	#rep	CPU seconds			BBN		
				Average	Maximum	Minimum	Average	Maximum	Minimum
0.1	1000	10	1.0	9.52	63.69	0.55	23,096.1	147,336	783
	2000	10	1.0	591.24	1282.07	0.77	288,462.3	1,067,322	40
	4000	8	1.0	396.01	1912.41	1.74	585,500.0	2,789,513	348
0.2	1000	10	1.0	9.87	39.65	0.44	25,249.7	98,486	176
	2000	9	1.0	349.69	1716.75	0.89	698,107.2	3,673,265	596
	4000	6	1.0	827.03	2765.84	1.99	1,113,609.0	3,692,637	25
0.4	1000	10	1.0	6.79	43.14	0.99	15,299.0	100,793	1151
	2000	10	1.0	63.63	317.41	1.58	130,041.8	692,126	96
	4000	5	1.0	1080.53	2532.03	11.71	1,382,056.0	3,113,746	318
0.8	1000	10	1.0	4.52	30.14	0.98	7435.6	65,397	1
	2000	10	1.2	29.97	163.93	7.07	420,83.8	270,575	380
	4000	8	1.1	1065.24	3156.37	11.91	1,322,136.0	4,131,502	1817

Table 11
Sensitivity of the CPU time to the knapsack capacity

n	$c = 125n$			$c = 250n$			$c = 375n$		
	Minimum	Average	Maximum	Minimum	Average	Maximum	Minimum	Average	Maximum
1000	0.24	0.70	1.35	0.38	0.85	1.83	0.12	0.39	1.34
2000	0.83	1.70	3.54	0.70	1.95	4.17	0.36	0.89	1.35
4000	2.92	5.10	8.47	3.61	18.31	101.76	1.70	3.49	7.27
8000	13.05	21.53	44.92	7.39	280.67	2452.00	7.58	15.99	59.46

Table 12
CPU seconds for strongly correlated instances

n	m	NUOPT		BLOCK					
		CPU	#sol	gap	n'	m'	Reduction	CPU	#sol
100	9.1	93.3	9	48.9	58.3	3.2	45.0	89.2	9
200	21.1	455.5	5	55.1	119.5	8.5	47.4	182.5	6

10 random instances shown in the table for the cases of $c = 125n$, $c = 250n$ and $c = 375n$. These represent approximately 1/4, 1/2 and 3/4 of the total weight of items.

From the table we see that PCKP is most difficult when the knapsack capacity is near to 1/2 of the total weight of items, while the problem is easier to solve for the knapsack capacity far apart from this value.

Finally, we solved some 'strongly correlated' instances, where w_j is uniformly distributed over $[1, 1000]$, but p_j is related to w_j by

$$p_j = w_j + 200.$$

We show a result in Table 12. As in ordinary knapsack problems, this type of PCKP is quite difficult to solve. This is because the weakness of the solvers to solve strongly correlated knapsack problems. We were unable to solve PCKPs with $n \geqslant 400$ by any of the methods considered in our work.

6. Conclusion

In this paper, we have shown that the pegging test for the standard 0–1 KPs can be extended to PCKPs by introducing the Lagrangian relaxation first to the precedence constraints. In addition, by proposing the block and virtual pegging tests, we were able to solve PCKPs with up to 16,000 items.

However, the experiments in this paper were limited to PCKPs with 'sparse' precedence relations in the sense that $|E| = O(n)$. Also, these relations were randomly picked up from all the possible pairs. For instances with 'dense' precedence relations such as $|E| = O(n^2)$, or with E generated from some other random mechanisms, the results may be quite different. These 'instance characteristics' of the algorithm are left for future investigations.

To solve yet larger problems exactly we need to explore the methods to obtain more tight upper and lower bounds, as well as the specialized algorithms to solve the reduced problem more efficiently. Analysis of polyhedral structure of PCKP [4,3] may prove useful in this direction.

Acknowledgements

The authors express their appreciation to Mr. S. Aminto (currently with the Indonesian Air Force) for his help in some part of the computation. They are also greatful to anonymous referees for informing us of a polyhedral analysis reference, as well as for their constructive comments.

References

[1] E. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, John Wiley & Sons, Chichester, 1997.
[2] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, Englewood Criffs, 1993.
[3] N. Boland, C. Fricke, G. Floyland, R. Sotirov, Clique-based facets for the precedence constrained knapsack problem, Optimization Online. http://www.optimization-online.org/DB_HTML/01/1322.html, 2006.
[4] E.A. Boyd, Polyhedral results for the precedence-constrained knapsack problem, Discrete Applied Mathematics 41 (1993) 185–201.
[5] G. Cho, D.X. Shaw, A depth-first dynamic programming algorithm for the tree knapsack problem, INFORMS Journal on Computing 9 (1997) 431–438.
[6] R.S. Dembo, P.L. Hammer, A reduction algorithm for knapsack problems, Methods of Operations Research 36 (1980) 49–60.
[7] D. Fayard, G. Plateau, Resolution of the 0–1 knapsack problem: Comparison of methods, Mathematical Programming 8 (1975) 272–307.
[8] R. Fourer, Software survey: Linear programming, OR/MS Today 26 (1999) 64–71.
[9] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman and Company, New York, 1979.
[10] F. Harary, Graph Theory, Addison Wesley, Reading, 1969.
[11] R. Hirabayashi, H. Suzuki, N. Tuchiya, Optimal tool module design problem for NC machine tools, Journal of the Operations Research Society of Japan 27 (1983) 205–229.
[12] D.D. Hochbaum, Selection, provisioning, shared fixed costs, maximum closure, and implications on algorithmic methods today, Management Science 50 (2004) 709–723.
[13] G.P. Ingargiola, J.F. Korsh, A reduction algorithm for zero-one single knapsack problems, Management Science 20 (1973) 460–463.
[14] D.S. Johnson, K.A. Niemi, On knapsacks, partitions, and a new dynamic programming technique for trees, Mathematics of Operations Research 8 (1983) 1–14.
[15] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer Verlag, 2004.
[16] T. Kuno, H. Konno, E. Zemel, A linear-time algorithm for solving continuous maximin knapsack problems, Operations Research Letters 10 (1991) 23–26.
[17] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, New York, 1990.
[18] H. Moriyama, T. Hada, H. Suzuki, A partially ordered knapsack problem and scheduling, in: Proceedings of the Production Scheduling Symposium'96, Japan Industrial Management Association, 1996, pp. 79–84 (in Japanese).
[19] G.L. Nemhauser, L.A. Wolsey, Integer and Combinatorial Optimization, John Wiley & Sons, New York, 1988.
[20] Mathematical Systems Incorporated (in Japanese). NUOPT manual. http://www.msi.co.jp/nuopt, 2002.
[21] N. Samphaiboon, T. Yamada, Heuristic and exact algorithms for the precedence-constrained knapsack problem, Journal of Optimization Theory and Application 105 (2002) 659–676.
[22] D.X. Shaw, G. Cho, The critical-item, upper bounds, and a branch-and-bound algorithm for the tree knapsack problem, Networks 28 (1998) 205–216.
[23] L.A. Wolsey, Integer Programming, John Wiley &Sons, New York, 1998.
[24] T. Yamada. http://www.nda.ac.jp/~yamada/ypublication.html, 2006.