Reduction and Exact Algorithms for the Disjunctively Constrained Knapsack Problem

Aminto Senisuka* Byungjun You† Takeo Yamada‡

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

Abstract *We are concerned with a variation of the knapsack problem (KP), where some items are incompatible with some others. As in ordinary KPs, each item is associated with profit and weight, we have a knapsack of a fixed capacity, and the problem is to determine the set of items to be packed into the knapsack. However, the knapsack is not allowed to include incompatible pairs of items. The knapsack problem with these additional constraints is referred to as the disjunctively constrained knapsack problem (DCKP). We present an algorithm to solve this problem to optimality by combining the Lagrangian relaxation with the pegging test for ordinary KPs. The developed algorithm solves DCKPs with several thousands of items within a reasonable computing time.*

1 Introduction

We are concerned with a variation of the *knapsack problem* (KP) [11, 8], where some items are *incompatible* with some others. As in ordinary KPs, we have n items to be packed into a knapsack of capacity c. Let w_j and p_j denote the weight and profit of the jth item respectively. Without much loss of generality we assume the following.

$\mathbf{A}_1$: Problem data w_j, p_j $(j = 1, \ldots, n)$ and c are all positive integers.

$\mathbf{A}_2$: $\sum_{j=1}^{n} w_j > c$ and $\forall w_j \leq c$.

$\mathbf{A}_3$: Items are arranged in non-increasing order of profit per weight, i.e.,

$$p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n. \tag{1}$$

*Currently with Indonesian Air Force
†Currently with The Republic of Korea Navy
‡Corresponding author: yamada@nda.ac.jp

In addition, let $E \subseteq \{(i,j) \mid 1 \leq i \neq j \leq n\}$ denote the set of incompatible pairs, and $m := |E|$ is the number of such pairs. That is, if $(i,j) \in E$ items i and j are not allowed to be included in the knapsack simultaneously. This relation is assumed to be reflective, i.e., $(i,j) \in E \Leftrightarrow (j,i) \in E$. We call this *disjunctive* constraint, and the knapsack problem with these additional constraints is referred to as the *disjunctively constrained* knapsack problem (DCKP) [15, 6].

The problem is to fill the knapsack with items such that the capacity and disjunctive constraints are all satisfied and the total profit of items in the knapsack is maximized. Let x_j be the decision variable such that $x_j = 1$ if item j is included in the knapsack, and $x_j = 0$ otherwise. Then, mathematically the problem is formulated as the following 0-1 programming problem.

DCKP:

$$\text{maximize} \quad z(x) := \sum_{j=1}^{n} p_j x_j \tag{2}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leq c, \tag{3}$$

$$x_i + x_j \leq 1, \ \forall (i,j) \in E, \tag{4}$$

$$x_j \in \{0,1\}, \ j = 1, \ldots, n. \tag{5}$$

Here by X we denote the set of all feasible solutions, and $z(x)$ is the objective value for $x \in X$. Throughout the paper $x^\star$ denotes an optimal solution with the corresponding optimal objective value $z^\star := z(x^\star)$. DCKP is $\mathcal{NP}$-hard [7], since for $E = \emptyset$ it reduces to KP which is already $\mathcal{NP}$-hard.

DCKP may be solved using any free or commercial integer programming package such as LINDO, CPLEX and so on [4], but to obtain an optimal solution within reasonable computing time, instances must be of a limited size. DCKP was formulated by Yamada *et al.* [15], where they solved the problem with upto 2000 items by an *implicit enumeration* algorithm. In this paper we solve much larger instances by combining the *Lagrangian relaxation* [13, 14, 11] with the *pegging test* for ordinary KPs [2, 3, 7] to reduce the size of the problem significantly. Even if the original problem is difficult to solve by the above mentioned solvers, the reduced problem is often tractable by such a software.

2 Upper and lower bounds

In this section we derive an upper bound to DCKP by applying the *Lagrangian relaxation* [13], and then obtain a lower bound based on a *local search* method [1] to find an approximate solution.

2.1 Lagrangian relaxation

An upper bound to DCKP is obtained by solving the following Lagrangian relaxation problem.

LDCKP(λ):

$$\text{maximize} \quad L := \sum_{j=1}^{n} p_j x_j + \sum_{(i,j)\in E} \lambda_{ij}(1 - x_i - x_j) \tag{6}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leq c, \tag{7}$$

$$0 \leq x_j \leq 1, \ j = 1, ..., n. \tag{8}$$

Here, L is the *Lagrangian* for DCKP, $\lambda = (\lambda_{ij}) \in R_+^m$, and $\lambda_{ij} \geq 0$ is the *Lagrange multiplier* associated with constraint (4). In addition, let $E(j) := \{i \in N | \ (i,j) \in E\}$ denote the set of items which are incompatible with j. Then, the objective function L can be rewritten as

$$L := \sum_{j=1}^{n} (p_j - \sum_{i\in E(j)} \lambda_{ij}) x_j + \sum_{(i,j)\in E} \lambda_{ij} \tag{9}$$

Then for a fixed $\lambda \in R_+^m$, LDCKP(λ) is a *continuous knapsack problem* [11] which is easily solved. Let an optimal solution to this problem be $\bar{x}(\lambda) = (\bar{x}_j)$ with the optimal objective value $\bar{z}(\lambda)$. Then, for an arbitrary $\lambda \geq 0$ we have

$$z^\star \leq \bar{z}(\lambda), \tag{10}$$

namely, $\bar{z}(\lambda)$ is an upper bound. The followings are well known [13, 14].

Proposition 1 $\bar{z}(\lambda)$ is piecewise linear convex function.

Proposition 2 If $z(\lambda)$ is differentiable at λ,

$$\frac{\partial \bar{z}(\lambda)}{\partial \lambda_{ij}} = 1 - x_i - x_j. \tag{11}$$

Proposition 3 If $\bar{x}(\lambda)$ is feasible and

$$\lambda_{ij}(1 - \bar{x}_i - \bar{x}_j) = 0 \quad \forall (i,j) \in E \tag{12}$$

is satisfied, then $\bar{x}(\lambda)$ is an optimal solution.

2.2 Subgradient method

To make the upper bound $\bar{z}(\lambda)$ as small as possible, we employ the *subgradient method* [13, 14] described below. Here *subgradient* is the vector $\partial \bar{z}(\lambda)/\partial \lambda$ whose element is given by (11).

Algorithm Subgradient_Method

Step 1. Set $\lambda := 0$.

Step 2. Solve LDCKP(λ) to obtain $\bar{x}(\lambda)$ and $\bar{z}(\lambda)$.

Step 3. Let the direction of movement be $d := -\partial\bar{z}(\lambda)/\partial\lambda$.

Step 4. (Line search) Find $\alpha \geq 0$ such that $\bar{z}(\lambda + \alpha d)$ is minimized.

Step 5. If (12) is satisfied or $\alpha \cong 0$, stop.

Step 6. Update $\lambda \longleftarrow \lambda + \alpha d$ and go to **Step 2**.

Let $\lambda^\dagger$ be the λ at the termination of the above algorithm, and $\bar{z} := \bar{z}(\lambda^\dagger)$ denotes the *optimal* upper bound to DCKP.

2.3 Lower bounds

Let $x^\dagger = (x_j^\dagger)$ be the solution to LDCKP($\lambda^\dagger$). This satisfies the constraint (3). If, in addition (4) and (5) are also satisfied, $x^\dagger$ gives a feasible solution to DCKP, and thus a *lower bound*. Even if (4) and/or (5) are not satisfied, we may modify the solution to obtain a feasible solution. For example, if we have a non-integer $0x_j^\dagger$, we simply put $x_j^\dagger \leftarrow 0$. If $x_j^\dagger + x_i^\dagger > 1$, again we fix either one of $x_i^\dagger$ or $x_j^\dagger$ at 0. The feasible solution thus obtained is referred to as the *Lagrangian solution*, and the corresponding lower bound is denoted as $\underline{z}_L$.

The Lagrangian solution can be improved by the following *greedy* algorithm. Here we examine items, one by one from $j = 1$ to n, if it is not yet included in the knapsack and can be accommodated without violating any constraints. If item j is found acceptable, we simply put it into the knapsack, and repeat this process for $j = 1, \ldots, n$. The resulting solution is the *greedy solution*, and $\underline{z}_G$ denotes the corresponding lower bound.

The greedy solution may further be improve by applying a more sophisticated local search method. An example is the *2-opt* method, which repeat the following until no improvement of solution is possible any further.

Procedure 2-opt

(i) Look for a pair of items, one inside and the other outside the knapsack, and see if it is possible to exchange these items without violating any constraints.

(ii) If this exchange increases the total value of items included in the knapsack, swap these items.

We call the resulting solution the *2-opt solution*, and the corresponding lower bound is denoted as $\underline{z}_2$

3 A Pegging Approach

A pegging test is well known for the ordinary 0-1 KPs [7, 3, 2]. By applying this test, many variables are fixed either at 0 or 1, and the size of such a problem is often reduced significantly. In this section, we shall demonstrate that similar pegging test is applicable to DCKPs as well by introducing the Lagrangian relaxation first.

3.1 Pegging test

Assume that we have the optimal Lagrangian multiplier $\lambda^\dagger$, the corresponding upper bound $\bar{z} = \bar{z}(\lambda^\dagger)$, and a lower bound $\underline{z}$ to DCKP, and let $\bar{p}_j$ be as follows.

$$\bar{p}_j := p_j - \sum_{i \in E(j)} \lambda^\dagger_{ij} \tag{13}$$

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

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} \bar{p}_j x_j + \sum_{(i,j) \in E} \lambda_{ij} \\ \text{subject to} \quad & (7) \text{ and } (8). \end{aligned} \tag{14}$$

For an arbitrary $k = 1, \ldots, n$, let $z^\star(x_k = \delta)$ denote the optimal objective value to DCKP with an additional constraint $x_k = \delta$, where δ is either 0 or 1. Similarly, $\bar{z}(x_k = \delta)$ denote the optimal objective value to LDCKP($\lambda^\dagger$) with an additional constraint $x_k = \delta$. Then, for $k = 1, \ldots, n$ the followings are obvious.

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

$$z^\star(x_k = \delta) \leq \bar{z}(x_k = \delta) \tag{16}$$

Then, if

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

it is not possible that $x^\star_k = 0$ in any optimal solution $x^\star = (x^\star_j)$ to DCKP, i.e., $x^\star_k$ must take $x^\star_k = 1$. Similarly, in the case that

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

$x^\star_k = 0$ must follow. This is the basic idea of the pegging test.

To determine (17) and (18), the following method is used to save the amount of computation. First of all, without loss of generality, we assume the following.

$\mathbf{B}_1$: $\bar{p}_j > 0, \ \forall j$

$\mathbf{B}_2$: The items are ordered in the non-increasing order of $\bar{p}_j / w_j$.

Let W_j and P_j be, respectively the *accumulated* 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, $\{(W_j, P_j) \mid j = 0, \cdots, n\}$ gives a piecewise-linear, monotonically non-decreasing, concave function [10].

The intersection of this graph with the vertical line $W = c$ gives the upper bound $\bar{z}$. The item s satisfying $W_{s-1} < c \leq W_s$ is said to be the *critical item*. Here, if for any $k < s$ we set $x_k = 0$, it is known that

$$\bar{z}(x_k = 0) \leq \bar{z} - \theta_k, \tag{20}$$

where we define

$$\theta_k := \bar{p}_k - r_s w_k. \tag{21}$$

Then, if

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

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

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

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

Theorem 1 For any optimal solution $x^\star = (x_j^\star)$ of DCKP, both of the followings hold true.

(i) $\bar{z} - \underline{z} < \theta_k \Rightarrow x_k^\star = 1$,

(ii) $\bar{z} - \underline{z} < -\theta_k \Rightarrow x_k^\star = 0$.

3.2 A pegging algorithm

Now we can solve DCKPs in the following way.

Algorithm Pegging_Test

Step 1. Compute the upper and lower bounds by the Lagrangian relaxation and local search methods.

Step 2. Reduce the problem size by applying Theorem 1.

Step 3. Solve the reduced problem using an IP solver, e.g., NUOPT [12].

4 Numerical experiments

In this section we evaluate the performance of the pegging approach to DCKP. We implemented the algorithm of the previous section in C language and conducted some numerical experiments on an IBM RS/6000 Model 270 workstation (CPU : POWER3-II SMP 2way, 375MHz).

4.1 Experimental design

The number of items n is set between 1000 and 16000, and the correlation type between weights and profits are assumed as the following.

- UNCOR
  w_j : Uniformly random over [1, 1000],
  p_j : Uniformly random over [1, 1000]; independent of w_j.
- WEAK
  w_j : Uniformly random over [1, 1000],
  p_j : Uniformly random over $[w_j, w_j + 200]$.

The capacity of the knapsack is set to $c = 250n$. This means that approximately a half of all the items can be accommodated into the knapsack, since the average weight of items is approximately 500. ¿From $n(n-1)/2$ possible pairs of items, disjunctive constraints are generated randomly with probability $d/(n-1)$, where d is the parameter that controls the density of constraints in the following way. That is, from this probability, the average number of disjunctive constraints is $nd/2$. We examined the cases of $d = 0.1, 0.2$ and 0.4.

4.2 Upper and lower bounds

Tables 4.1 and 4.2 give an overview of computation of upper and lower bounds. Here shown are d, n, m and the Lagrangian upper bound $\bar{z}_L$ with the corresponding CPU time in seconds. For the columns of 'Greedy LB' and '2-opt LB', 'gap' shows the the gap between $\bar{z}_L$ and respective lower bounds, and each row is the average of 10 independent runs.

From these tables, we observe the followings.

1. Computation time for the Lagrangian upper bound increases with the increase of either n or d, but it is rather insensitive to the correlation type (UNCOR/WEAK) of problems.
2. Gaps remain almost the same in Greedy and 2-opt lower bounds, but in CPU time the former overperforms the latter.
3. In Greedy method, as d increases, CPU times are almost constant but gaps deteriorate significantly. Gaps are usually smaller in WEAK instances.

From this result, we employ the Greedy method for the computation of lower bounds in the later experiments.

Table 4.1 Upper and lower bounds (UNCOR).

d	n	m	Lagrange UB		Greedy LB		2-opt LB	
			$\overline{z}_L$	CPU	gap	CPU	gap	CPU
0.1	1000	48.9	398721.9	0.1	30.4	0.0	28.5	0.1
	2000	96.9	802688.5	0.4	21.3	0.0	18.9	1.1
	4000	199.5	1601207.5	0.9	42.8	0.0	40.9	8.7
	8000	403.8	3196945.2	2.7	8.5	0.1	7.3	71.2
	16000	802.6	6394867.3	8.1	90.8	0.5	90.5	544.6
0.2	1000	99.0	394132.6	0.2	55.4	0.0	49.7	0.1
	2000	197.0	794909.3	0.5	24.6	0.0	22.5	1.2
	4000	394.3	1584789.9	1.5	45.1	0.0	43.1	8.7
	8000	805.0	3162459.6	4.0	34.3	0.1	32.8	71.9
	16000	1598.4	6328402.4	11.3	25.9	0.5	24.8	754.0
0.4	1000	198.9	386437.4	0.3	84.7	0.0	82.9	0.1
	2000	396.5	778653.3	0.8	86.0	0.0	83.3	1.3
	4000	792.1	1553682.9	2.2	85.5	0.0	84.3	7.7
	8000	1605.6	3097011.0	9.8	258.2	0.1	238.2	76.5
	16000	3215.3	6199034.2	30.6	430.4	0.5	429.4	798.6

Table 4.2 Upper and lower bounds (WEAK).

d	n	m	Lagrange UB		Greedy LB		2-opt LB	
			$\overline{z}_L$	CPU	gap	CPU	gap	CPU
0.1	1000	48.9	329563.6	0.1	21.3	0.0	18.2	1.0
	2000	96.9	660578.6	0.3	18.0	0.0	15.5	8.7
	4000	199.5	1320306.8	0.7	15.6	0.0	11.1	55.9
	8000	403.8	2640457.7	2.0	14.4	0.1	12.2	462.4
	16000	802.6	5280144.6	4.8	8.8	0.5	6.2	20330.0
0.2	1000	99.0	328703.7	0.2	42.0	0.0	28.0	0.9
	2000	197.0	659140.0	0.4	28.2	0.0	24.4	10.7
	4000	394.3	1316941.0	1.1	30.6	0.0	29.3	49.9
	8000	805.0	2633633.5	2.9	29.7	0.1	27.1	440.6
	16000	1598.4	5266806.6	9.9	85.3	0.5	83.2	17081.7
0.4	1000	198.9	327089.9	0.3	53.3	0.0	47.6	1.1
	2000	396.5	655981.1	0.6	43.2	0.0	39.3	8.5
	4000	792.1	1310745.1	1.8	30.6	0.0	28.3	48.8
	8000	1605.6	2620504.0	6.0	88.5	0.1	85.6	417.9
	16000	3215.3	5241053.9	15.1	106.5	0.6	105.0	14454.5

4.3 Exact solution by NUOPT

We now turn to exact methods. First of all, in Table 4.3 we give the result obtained by NUOPT [12], which is a commercial MP solver produced by a Japanese vendor, and is considered competitive to such popular solvers as LINDO, CPLEX, etc [4]. Here shown are, in addition to d and n, the number of subproblems (♯subp) generated by the branch and bound procedure within NUOPT, CPU time in seconds, and the number of solved problems (#solvd) out of 10 randomly generated

UNCOR and WEAK instances. Each row is the average over the solved cases, and dash (-) means that non of the instances were solved to optimality due to time limit of 3600 CPU seconds or insufficient computer memory. From this table we observe that the problem becomes hard to solve by NUOPT for the problem with $n \geq 8000$. Increase of d, and hence the number of disjunctive constraints (m) also makes problem harder.

Table 4.3 Direct solution using NUOPT (UNCOR).

d	n	UNCOR			WEAK		
		#subp	CPU	#solvd	#subp	CPU	#solvd
0.1	1000	2081.3	8.8	10	2956.5	8.9	10
	2000	1718.7	17.5	10	5451.7	36.7	10
	4000	3022.0	57.4	10	9697.4	95.3	10
	8000	2614.6	162.5	6	107476.2	1306.2	9
	16000	4871.2	588.0	7	2299.0	178.4	4
0.2	1000	2014.0	9.6	10	2349.8	8.3	10
	2000	2228.1	25.0	10	5363.5	38.5	10
	4000	3191.8	83.4	9	9399.6	129.7	10
	8000	4717.1	223.9	7	36909.5	615.3	7
	16000	3595.4	600.6	7	68886.6	2427.4	3
0.4	1000	1362.1	9.1	10	4258.2	20.0	9
	2000	2347.4	26.6	10	4178.1	53.3	10
	4000	2161.2	88.1	9	7334.1	142.8	9
	8000	1946.0	179.2	9	38197.4	693.6	7
	16000	2304.6	427.2	3		-	

4.4 Exact solution by the pegging algorithm

Tables 4.4 and 4.5 summerize the result of the pegging approach to UNCOR and WEAK instances respectively. Here shown are the sizes of the reduced problem (n', m'), the rate of reduction (reduc.) defined by

$$\text{reduc.} := \sqrt{n'm'/nm}, \tag{24}$$

and CPU times in seconds. These show, respectively, the computing time for the Lagrangian relaxation + Greedy + pegging (CPU_P), the time to solve the reduced problem using NUOPT (CPU_N), and the total CPU time to solve DCKP completely this way ($\text{CPU}_T = \text{CPU}_P + \text{CPU}_N$). Again, '#solvd' is the number of solved instances out of 10 randomly generated instances, and each row is the average over the solved instances. From these tables, we observe the followings.

1. Larger instances are solved by this method in considerably smaller computation time than by the direct use of NUOPT.

2. As n increases, pegging becomes less effective and both of the computing time CPU_P and CPU_N increase. The former is smaller than the latter, especially in WEAK case.

3. Reduction rate (reduc.) is better in UNCOR than in WEAK instances, and thus the former is easier to solve. This is in accordance with the general tendency in ordinary KPs.

Table 4.4 Solution by the pegging algorithm (UNCOR).

d	n	n'	m'	reduc.	CPU_P	CPU_N	CPU_T	#solvd
0.1	1000	66.0	2.7	0.05	0.1	0.3	0.5	10
	2000	92.0	3.9	0.06	0.4	0.2	0.6	10
	4000	326.5	13.9	0.14	0.9	2.6	3.5	10
	8000	150.2	6.1	0.05	2.8	1.2	4.0	10
	16000	2432.6	119.3	0.60	8.5	80.2	88.6	10
0.2	1000	115.6	8.5	0.10	0.2	0.5	0.7	10
	2000	105.1	5.8	0.04	0.5	0.6	1.1	10
	4000	385.3	31.0	0.09	1.5	4.1	5.6	10
	8000	566.6	48.6	0.06	4.1	15.6	19.7	10
	16000	885.6	66.8	0.05	11.8	18.2	30.0	10
0.4	1000	164.6	27.2	0.15	0.3	1.6	1.9	10
	2000	302.5	91.7	0.16	0.9	5.5	6.3	10
	4000	696.7	121.1	0.16	2.2	18.4	20.6	10
	8000	3236.2	621.7	0.40	9.9	59.5	69.5	10
	16000	8935.4	1728.7	0.55	31.2	111.4	141.7	6

Table 4.5 Solution by the pegging algorithm (WEAK).

d	n	n'	m'	reduc.	CPU_P	CPU_N	CPU_T	#solvd
0.1	1000	211.4	7.8	0.18	0.1	1.6	1.7	10
	2000	356.4	15.0	0.16	0.3	4.1	4.3	10
	4000	627.1	23.1	0.13	0.8	18.6	19.4	10
	8000	1162.8	48.3	0.13	2.1	196.0	198.1	10
	16000	1158.9	43.0	0.06	5.4	539.2	544.6	7
0.2	1000	391.2	34.2	0.37	0.2	3.2	3.3	10
	2000	537.8	45.6	0.25	0.4	7.0	7.4	10
	4000	1104.0	98.6	0.26	1.2	26.8	27.9	10
	8000	241.6	203.9	0.27	3.2	118.3	121.5	10
	16000	8737.2	857.9	0.54	10.5	135.5	141.8	4
0.4	1000	394.5	68.7	0.43	0.3	6.4	7.2	9
	2000	785.0	137.8	0.37	0.6	8.8	9.3	10
	4000	1150.7	196.7	0.27	1.9	33.5	35.4	10
	8000	5577.5	1101.3	0.69	6.2	185.0	191.1	7
	16000	9308.5	1819.7	0.57	17.3	507.5	524.8	1

5 Virtual Pegging Test

The effect of the pegging test of Section 3depends on the gap between the upper and lower bounds. If the gap is not small enough, the effectiveness of the previous method will be limited, since the size of the problem will not be reduced much in

such a case. In the present section, we introduce the *virtual* pegging test in order to cope with this problem.

5.1 The principle

In the pegging test, we input an upper bound $\bar{z}$, and a lower bound $\underline{z}$ to the reduction algorithm *Pegging Test*, and partition the original problem into a fixed part and the remaining reduced problem. Of course, upper and lower bounds satisfy

$$\underline{z} \ \leq \ z^\star \ \leq \ \bar{z}. \tag{25}$$

However, we may carry out the pegging test with an arbitrary value l within $[\bar{z}, \underline{z}]$ as an assumed lower bound. Such a hypothetical lower bound is referred to as a *trial value*. By carrying out the pegging test using $\bar{z}$ and l, some x_j's will be fixed either at 0 or 1. But it is not guaranteed that this pegging is correct because l is not necessarily a true lower bound. Let the index sets of variables, which are (virtually) fixed at 0 and 1 in this procedure, be $F_0(l)$ and $F_1(l)$ respectively. Then, we have the following *reduced problem*.

R(l)**:**

$$\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{n} p_i x_i && (26)\\
\text{subject to} \quad & x \in X, && (27)\\
& x_i = 1, \quad i \in F_1(l), && (28)\\
& x_i = 0, \quad i \in F_0(l) && (29)
\end{aligned}$$

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

Theorem 2

(i) $l \leq z^\star \Rightarrow z_l^\star = z^\star$,

(ii) $l > z^\star \Rightarrow z_l^\star \leq z^\star$,

(iii) $l \leq l' \Rightarrow z_l^\star \geq z_{l'}^\star$,

(iv) $l \leq z_l^\star \Rightarrow z_l^\star = z^\star$.

Proof: (i) If $l \leq z^\star$, l is actually a lower bound; thus the pegging test works correctly and finds the optimal value $z^\star$. (ii) Note that for an arbitrary $l \leq \bar{z}$ R(l) is DCKP with additional constraints (28) and (29). Thus, by definition, the optimal objective values satisfy this relation. (iii) As we apply Theorem 1 with gap $:= \bar{z} - l$, we note that the more variables are fixed either at 1 or 0 as we have the larger l (and thus a smaller gap); i.e., for $l \leq l'$ we have $F_\delta(l) \subseteq F_\delta(l')$ $(\delta = 0, 1)$.

From this, the above relation is straightforward. (iv) This is a consequence of (i) - (iii). ∎

As an immediate consequence of (iii), if R(l) is infeasible and $l \leq l' (\leq \bar{z})$, then R(l') is also infeasible for any $l' \geq l$.

5.2 A virtual pegging algorithm

At an arbitrary trial value l, after carrying out the virtual pegging test and solving the reduced problem R(l), we obtain the realised value $z_l^\star$. Then, if (iv) is satisfied in Theorem 2, the problem is solved. In addition, if gap $:= \bar{z} - l$ is small, it is probable that R(l) is much smaller than the original in size. We may solve the reduced problem easily using, e.g., NUOPT. We propose the following algorithm including the measures for the case where (iv) is not satisfied.

Algorithm Virtual_Pegging_Test

Step 1. $l \leftarrow \max\{\bar{z} - \alpha, \underline{z}\}$

Step 2. Apply *Pegging Test* with the trial value l, solve R(l) and obtain $z_l^\star$

Step 3. If $l \leq z_l^\star$, go to **Step 5**.

Step 4. Update parameters by $\alpha \leftarrow \alpha/2$ and $l \leftarrow l - \alpha$, and go to **Step 2**.

Step 5. An optimal solution is obtained with $z^\star = z_l^\star$.

Here, α is an arbitrary 'small' value. In our numerical experiment, we set this as $\alpha := (\bar{z} - \underline{z})/2$, and the trial value is initially set at $l := \bar{z} - \alpha$. If the optimal value is not obtained in step 3, we halve α and reduce$l$ by α, and repeat Steps 2 - 4 until the optimal solution is found.

5.3 Numerical test

Tables 5.1 and 5.2 show the results of numerical experiment for the virtual pegging method, where $d = 0.05,\ 0.1,\ 0.2$ and $n = 16000,\ 32000,\ 64000$ are tried. Direct application of NUOPT seldom solves problems, and even when successful, it is usually time consuming. By the virtual pegging approach, we are able to solve many instances of this range, usually in much smaller computing time.

In all cases solved, we had to repeat Steps 2 through 4 only once, and the reduction ratio (reduc.) is substantially smaller in these cases than those by plain pegging test shown in Tables 4.4 and 4.5. Thus, the virtual pegging approach works favorably for the instances of this size. Again, WEAK cases are harder to solve than the UNCOR instances.

Table 5.1 Virtual pegging algorithm vs. NUOPT (UNCOR).

d	n	m	Virtual Pegging			NUOPT	
			reduc.	CPU	#solvd	CPU	#solvd
0.05	16000	408.5	0.006	9.1	10	688.7	7
	32000	801.2	0.006	24.6	10	1122.3	4
	64000	1582.8	0.005	600.0	9	1760.5	2
0.10	16000	802.6	0.083	50.0	10	588.6	7
	32000	1605.0	0.016	41.2	10	137.2	1
	64000	3182.3	0.030	372.0	8	1760.9	2
0.20	16000	802.6	0.025	21.7	10	607.8	7
	32000	1605.0	0.097	165.1	10	-	
	64000	3182.3	0.016	746.5	7	-	

Table 5.2 Virtual pegging algorithm vs. NUOPT (WEAK).

d	n	m	Virtual Pegging			NUOPT	
			reduc.	CPU	#solvd	CPU	#solvd
0.05	16000	408.5	0.029	299.3	9	1592.4	4
	32000	801.2	0.022	213.2	4	-	
	64000	1582.8	0.006	488.1	1	-	
0.10	16000	802.6	0.032	361.8	7	178.5	3
	32000	1605.0	0.109	707.5	5	1369.0	3
	64000	3182.3	0.060	59.5	1	-	
0.20	16000	802.6	0.091	71.6	4	1781.0	2
	32000	1605.0	0.052	615.9	2	-	
	64000	3182.3	0.128	718.1	1	-	

6 Conclusion

Knapsack problem with disjunctive constraints was solved by applying the pegging test combined with the Lagrangian relaxation. We were able to solve larger problems than using a commercial software directly in considerably smaller computing time.

To solve yet larger problems exactly, we need to explore the methods to obtain more strict upper and lower bounds, as well as the specialized algorithms to solve the reduced problem more efficiently. These are left for future work.

References

[1] Aarts, E. and Lenstra, J. K. (eds.), *Local Search in Combinatorial Optimization*, John Wiley & Sons, Chichester, England (1997).

[2] Dembo, R. S. and Hammer, P. L., "A reduction algorithm for knapsack problems", *Methods of Operations Research*, **36**, 49-60, 1980.

[3] Fayard, D. and Plateau, G., "Resolution of the 0-1 knapsack problem: comparison of methods ", *Mathematical Programming*, **8**, 272-307, 1975.

[4] Fourer, R., "Software Survey: Linear Programming", *OR/MS Today*, **26**, 64-71 (1999).

[5] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman and Company, New York, 1979.

[6] Hifi, M. and Michrafy, M., "A reactive local search-based algorithm for the disjunctively constrained knapsack problem", LaRIA, Univ. Paris, Working paper, 2004.

[7] Ingargiola, G. P and Korsh, J. F., "A reduction algorithm for zero-one single knapsack problem", *Management science*, **20**, 460-463, 1973.

[8] Kellerer, H., Pferschy, U. and Pisinger, D., *Knapsack Problems*, Springer Verlag, Berlin, 2004.

[9] Konno, H. and Suzuki, H. (Eds), *Integer Programming and Combinatorial Optimization* (in Japanese), Nikka Giren, Tokyo, 1982.

[10] Kuno, T., Konno, H. and Zemel, E., "A linear-time algorithm for solving continuous maximin knapsack problems", *Operations Research Letters*, **10**, 23-26 (1991).

[11] Martello, S. and Toth, P., *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, Chichester, 1990.

[12] Mathematical Systems Incorporated, NUOPT manual, 2002. URL: http://www.msi.co.jp/nuopt

[13] Nemhauser, G. L. and Wolsey, L. A., *Integer and Combinatorial Optimization*, John Wiley & Sons, New York, 1988.

[14] Wolsey, L. A., *Integer Programming*, John Wiley & Sons, New York, 1998.

[15] Yamada, T. Kataoka, S. and Watanabe, K., "Heuristic and exact algorithms for disjunctively constrained knapsack problem", *IPS Journal*, **43**, 2864-2870, 2002.