

## 付加制約のあるナップサック問題の近似解法と厳密解法

セニスカ アミント \* 柳 秉俊 \*

山田 武夫 \*\*

(平成 17 年 1 月 25 日受付; 平成 17 年 4 月 12 日受理)

Heuristic and Exact Algorithms for the Knapsack Problem with Side Constraints

By SENISUKA AMINTO\*, YOU BYUNGJUN\* and YAMADA TAKEO \*\*

**概要:** In this paper we are concerned with some variants of the knapsack problem (KP). 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 an optimal set of items to be packed into the knapsack. However, we are required to satisfy some side constraints as well. Such a problem includes the disjunctively constrained knapsack problem (DCKP), and the precedence constrained knapsack problem (PCKP) among others. We present an approach that combines the Lagrangean relaxation and the pegging test to solve these problems to optimality. Through this method, we are able to solve DCKPs as well as PCKPs with thousands of items within a reasonable computing time.

**Key words :** knapsack problem, combinational optimization, side constraints.

### 1. はじめに

ナップサック問題 (knapsack problem: KP)<sup>5,7)</sup> は OR や情報科学分野の基本問題で, 多数の研究がなされているが, 本稿ではこれにいくつかの制約式が付加された場合を考える. このような問題の例には, 多次元ナップサック問題 (multi-dimensional knapsack problem: MKP)<sup>5)</sup> や, 排他制約付きナップサック問題 (disjunctively constrained knapsack problem: DCKP)<sup>10,13)</sup>, 順序制約付きナップサック問題 (precedence-constrained knapsack problem: PCKP)<sup>8,14)</sup> などがある. 通常のナップサック問題がすでに  $\mathcal{NP}$ -困難<sup>3)</sup> なので, その拡張であるこれらの問題もまた  $\mathcal{NP}$ -困難である.

DCKP は次のように定式化される.

**DCKP:**

$$\text{maximize } z(x) := \sum_{j=1}^n p_j x_j \quad (1.1)$$

$$\text{subject to } \sum_{j=1}^n w_j x_j \leq c, \quad (1.2)$$

$$x_i + x_j \leq 1, \quad \forall (i, j) \in E, \quad (1.3)$$

$$x_j \in \{0, 1\}, \quad \forall j \in V. \quad (1.4)$$

通常のナップサック問題の場合と同様に,  $p_j, w_j$  は商品の利得と重量で,  $c$  はナップサックの容量である. さらに,  $V = \{1, 2, \dots, n\}$  は商品の集合,  $E$  は排他的な商品対の集合を表しており,  $G = (V, E)$  は無向グラフである.

また, PCKP は以下のように定式化される.

**PCKP:**

$$\text{maximize } z(x) := \sum_{j=1}^n p_j x_j \quad (1.5)$$

$$\text{subject to } \sum_{j=1}^n w_j x_j \leq c, \quad (1.6)$$

$$x_i \geq x_j, \quad \forall (i, j) \in E, \quad (1.7)$$

$$x_j \in \{0, 1\}, \quad \forall j \in V. \quad (1.8)$$

この場合,  $E \subseteq V \times V$  は商品間の順序関係を表して

\* 防衛大学校 第 42 期理工学研究科学生 情報数理専攻

\*\* 防衛大学校 情報工学科 教授

いる。順序制約が意味を持つために、 $G = (V, E)$  はサイクルを含まない、無閉路有向グラフとする。

これらの問題は 0-1 整数計画問題<sup>6)</sup>なので、ある程度のサイズの問題までは NUOPT<sup>11)</sup>などの市販の、またはフリーの汎用数理計画ソフトで解くことができる。また、PCKP については Samphaiboon 等<sup>8)</sup>が動的計画法<sup>9)</sup>による解法を示しており、DCKP は Yamada 等<sup>13)</sup>が定式化して陰伏列挙法による解法を提示しているが、いずれも商品数が  $n = 1000 \sim 2000$  程度までの問題しか解けていない。

本稿ではさらに大型の問題を効率よく解くために、ラグランジュ緩和と釘付けテストを併用して問題サイズを縮小し、厳密解を得る新しい方法を示す。すなわち、ラグランジュ緩和の導入によって PCKP や DCKP にも通常のナップサック問題と同様の釘付けテストが適用可能となるが、このような試みは著者らの知る範囲では他になされていない。ほとんどの議論は PCKP, DCKP のいずれにも同様に展開できる<sup>15)</sup>ので、以下では主に DCKP について記述する。排他制約式の数を  $m := |E|$  と記し、通例<sup>7)</sup>に従って次を仮定する。

A<sub>1</sub>  $p_j, w_j, c$  はすべて正の整数。

A<sub>2</sub>  $\sum_{j=1}^n w_j > c, w_j < c, \forall j \in V$ 。

## 2. 上下界値

本節では最初にラグランジュ緩和<sup>6, 7)</sup>により DCKP の上界値を求め、次に局所探索法<sup>12)</sup>をベースとした近似解法により下界値を導く。

### 2.1 ラグランジュ緩和

DCKP で (1.3) 式に付随するラグランジュ乗数を  $\lambda_{ij} \geq 0$ 、それらの全体を  $\lambda = (\lambda_{ij}) \in R^m$  とした時、

$$\begin{aligned} L(x, \lambda) &:= \sum_{j=1}^n p_j x_j + \sum_{(i,j) \in E} \lambda_{ij} (1 - x_i - x_j) \\ &= \sum_{j=1}^n (p_j - \sum_{i \in E(j)} \lambda_{ij}) x_j + \sum_{(i,j) \in E} \lambda_{ij} \end{aligned} \quad (2.1)$$

をラグランジュ関数という。ここに  $E(j) := \{i \in V | (i, j) \in E\}$  である。 $\lambda \geq 0$  が与えられたとき、連続変数型のラグランジュ緩和問題は次のように定式化される。

LDCKP( $\lambda$ ):

$$\text{maximize } L(x, \lambda) \quad (2.2)$$

$$\text{subject to } \sum_{j=1}^n w_j x_j \leq c, \quad (2.3)$$

$$0 \leq x_j \leq 1. \quad (2.4)$$

LDCKP( $\lambda$ ) は連続型ナップサック問題で、その解は容易に求められる。その最適解を  $\bar{x}(\lambda) = (\bar{x}_j)$ 、対応する最適関数値を  $\bar{z}(\lambda)$  とし、DCKP の最適関数値を  $z^*$  とすると、これらの間には下の関係が成立する。

$$z^* \leq \bar{z}(\lambda) \quad (2.5)$$

すなわち、 $\bar{z}(\lambda)$  は DCKP の一つの上界値を与える。

$\bar{z}(\lambda)$  を  $\lambda$  の関数と考えたとき、これについての基本的事項を以下にまとめておく<sup>6, 7)</sup>。

命題 2.1  $\bar{z}(\lambda)$  は、 $\lambda \geq 0$  において区分的に線形な凸関数である。

命題 2.2  $\lambda$  において  $\bar{z}(\lambda)$  が微分可能なら

$$\frac{\partial \bar{z}(\lambda)}{\partial \lambda_{ij}} = 1 - \bar{x}_i - \bar{x}_j. \quad (2.6)$$

命題 2.3 任意の  $\lambda \geq 0$  において、 $\bar{x}(\lambda)$  が実行可能で (すなわち、式 (1.2) ~ (1.4) を満たし)、

$$\lambda_{ij} (1 - \bar{x}_i - \bar{x}_j) = 0 \quad \forall (i, j) \in E \quad (2.7)$$

ならば、 $\bar{x}(\lambda)$  は DCKP の最適解である。

### 2.2 劣勾配法

任意の  $\lambda \geq 0$  に対し、前に述べたように  $\bar{z}(\lambda)$  は DCKP の上界値を与えるが、この値は出来るだけ小さい方が望ましい。このために、劣勾配法 (subgradient method)<sup>6)</sup>を導入する。以下で、(2.6) を成分とするベクトル  $\partial \bar{z}(\lambda) / \partial \lambda \in R^m$  を劣勾配という。

劣勾配法

- Step 1.  $\lambda = 0$  とする。
- Step 2. LDCKP( $\lambda$ ) を解く。
- Step 3. 劣勾配を計算し、探索方向を  $d := -\partial \bar{z}(\lambda) / \partial \lambda$  により定める。
- Step 4. (1次元探索)  $\bar{z}(\lambda + \alpha d)$  が最小となる  $\alpha \geq 0$  を求める。
- Step 5. (2.7) が成立するか、 $\alpha \cong 0$  の場合、終了。
- Step 6.  $\lambda \leftarrow \lambda + \alpha d$  として Step 2 へ戻る。

上のアルゴリズムが終了した時点での  $\lambda$  を  $\lambda^\dagger$  とする。そのときの  $\bar{z}(\lambda^\dagger)$  を以下では  $\bar{z}$  と記し、DCKP の上界値とする。

### 2.3 ラグランジュ解とグリーディ解

$\lambda^\dagger$  における LDCKP( $\lambda^\dagger$ ) の最適解  $\bar{x}(\lambda^\dagger) = (\bar{x}_j^\dagger)$  は制約条件 (1.2) を満たしているが, (1.3), (1.4) も満たしているならば, これは DCKP の一つの実行可能解なので, その時の目的関数値は DCKP の下界値となる. しかし, 一般にはこれらがすべて満足されるとは限らないので,  $\bar{x}_j^\dagger$  が 0-1 条件を満たさない場合や,  $\bar{x}_i^\dagger + \bar{x}_j^\dagger > 1$  である  $(i, j) \in E$  が見つかった時は,  $\bar{x}_j^\dagger \leftarrow 0$  などとして解を修正する. これによって DCKP の実行可能解が必ず得られるので, これをラグランジュ解と呼ぶ.

次に, この解から出発してさらにナップサックに収容可能な商品を探し, そのような商品が見つかり次第ナップサックに収容して解を改善していく. このようにして得られた解をグリーディ解といい,  $\underline{x}_G$  と記す. また, 対応する下界値を  $\underline{z}_G$  とする.

### 2.4 2-opt 解

グリーディ解をさらに改良するため, 2-opt 解法を考える. DCKP の任意の実行可能解  $x = (x_j)$  について,

- (i) ナップサックに入っていない商品の一つをナップサックに入れる,
- (ii) ナップサックに入っている商品とそうでない商品の一組を入れ替える,

のいずれかの操作によって得られる解  $y$  が実行可能であるとき,  $y$  は  $x$  に隣接するといい, そのような解全体の集合を  $N(x)$  と記して  $x$  の 2-opt 近傍と呼ぶ.

2-opt 解法はグリーディ解を最初の解  $x$  とし,  $N(x)$  内に  $x$  より良い解  $y$  が見つければ, これを直ちに新しい  $x$  とするという操作を反復するもので, 具体的には下のように書かれる.

#### アルゴリズム 2-opt

- Step 1.  $x := \underline{x}_G$  とおく.
- Step 2. もし  $\exists y \in N(x)$  で  $z(y) > z(x)$  であれば Step3 へ進む, そうでなければ  $(x, z(x))$  を出力して終了.
- Step 3.  $x := y$  として, Step2 へ戻る.

上のアルゴリズムの出力を  $\underline{x}_2$  と記し, 2-opt 解と呼ぶ. また, 対応する目的関数値を  $\underline{z}_2 := z(\underline{x}_2)$  とする.

### 3. 釘付けテスト

通常の 0-1 ナップサック問題については, 釘付けテスト<sup>1, 2, 4)</sup> が知られており, これによって変数の一部を 0 または 1 に固定して, 問題サイズを (大幅に) 減少することが出来る. 本節では, ラグランジュ緩和を介することによって, この方法が DCKP (や PCKP) にも適用可能となることを示す.

DCKP において, 最適ラグランジュ乗数  $\lambda^\dagger = (\lambda_{ij}^\dagger)$ , 上界値  $\bar{z} = \bar{z}(\lambda^\dagger)$  と (グリーディ解または 2-opt 解に対応する) 下界値  $\underline{z}$  が得られているとする.

$$\bar{p}_j = p_j - \sum_{i \in E(j)} \lambda_{ij}^\dagger \quad (3.1)$$

と置くと, ラグランジュ緩和問題は下のように書ける. LDCKP( $\lambda^\dagger$ ):

$$\text{maximize } \sum_{j=1}^n \bar{p}_j x_j + \sum_{(i,j) \in E} \lambda_{ij}^\dagger \quad (3.2)$$

$$\text{subject to } \sum_{j=1}^n w_j x_j \leq c \quad (3.3)$$

$$0 \leq x_j \leq 1, \quad \forall j \quad (3.4)$$

上で, 変数  $x_i$  を 0 または 1 に制限して得られる問題の最適関数値をそれぞれ  $\bar{z}_i^0, \bar{z}_i^1$  と記す. ここで, もし

$$\bar{z}_i^0 < \underline{z} \quad (3.5)$$

ならば, DCKP の最適解  $x^* = (x_j^*)$  において  $x_i^* = 0$  となることはあり得ない. すなわち,  $x_i^*$  は  $x_i^* = 1$  と固定される. 同様に,

$$\bar{z}_i^1 < \underline{z} \quad (3.6)$$

の場合は  $x_i^* = 0$  となる. 以上が釘付けテストの原理であるが, 以下では (3.5), (3.6) の簡便な判定法を考える.

まず, LDCKP( $\lambda^\dagger$ ) で一般性を失うことなく以下を仮定する.

- A<sub>3</sub> 商品は  $\bar{p}_j/w_j$  の大きいものから順に番号付けられている.

商品  $i$  までの累積重量と累積利得を

$$W_i := \sum_{j=1}^i w_j, \quad P_i := \sum_{j=1}^i \bar{p}_j \quad (3.7)$$

として  $(W_i, P_i)$  をプロットすると、上の仮定より、図 3.1 の実線のような区分的線形で単調増加な凹関数が得られる。この折れ線が、縦軸に平行な直線  $W = c$  と交わる点が上界値  $\bar{z}$  を与えるが、この時の商品 (商品  $s$  とする) を臨界商品と呼ぶ。ここで、 $l < s$  の商品  $l$  を 0 に固定すると、図 3.1 のように商品  $l$  の部分が抜けて、それより右の折れ線が商品  $l$  の分だけ左下に平行移動して破線で示した折れ線を得る。

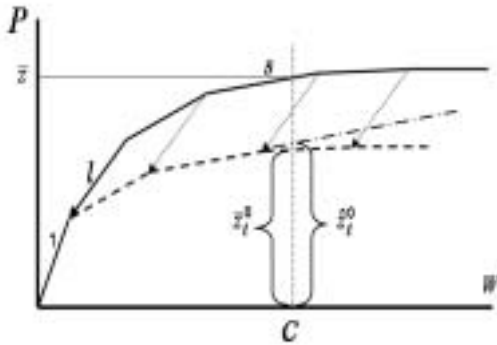


図 3.1 釘付けテスト  
Fig. 3.1 Pegging test.

この破線の折れ線と  $W = c$  の交点が  $\hat{z}_l^0$  を与えるが、ここでは計算を簡単にするため、臨界商品  $s$  に対応する直線を平行移動した直線 (一点鎖線で表示) が  $W = c$  と交わる点より、 $\hat{z}_l^0$  の上限を

$$\hat{z}_l^0 := \bar{z} - (\bar{p}_l - r_s w_l) \quad (3.8)$$

と評価する (図 3.1 参照)。ここで

$$\theta_l := \bar{p}_l - r_s w_l \quad (3.9)$$

とおくと、(3.5) より、

$$\bar{z} - \underline{z} < \theta_l \quad (3.10)$$

の時、 $x_l^* = 1$  と固定されることがわかる。以下では  $\theta_l$  をしきい値と呼ぶ。

$l > s$  の商品  $l$  を  $x_l = 1$  と固定した場合も同様の議論より、

$$\bar{z} - \underline{z} < -\theta_l \quad (3.11)$$

の時、 $x_l^* = 0$  と固定されることが分かる。以上をまとめると、次の定理を得る (通常の KP の場合は Fayard ら<sup>2)</sup>, Ingargiola ら<sup>4)</sup> を参照)。

定理 1 DCKP の最適解  $x^* = (x_j^*)$  において、

$$(i) \bar{z} - \underline{z} < \theta_j \text{ ならば, } x_j^* = 1$$

$$(ii) \bar{z} - \underline{z} < -\theta_j \text{ ならば, } x_j^* = 0$$

この定理により変数の一部が 0 または 1 に固定される。また、 $(i, j) \in E$  で、 $x_j$  が 1 に固定された場合、 $x_i = 0$  となって、これらの変数と制約式  $x_i + x_j \leq 1$  が除去される。このようにして問題は (大幅に) 縮小され、残った問題は NUOPT<sup>11)</sup> などの市販ソフトで解けることが多い。これにより DCKP の厳密解を得ることが出来る。

#### 4. 仮想釘付けテスト

釘付けテストの効果は上下界値間のギャップに依存する。このギャップが十分小さくない場合、問題はあまり縮小されないで、前節の方法の有効性は限られたものとなる。本節では、これに対処するため仮想釘付けテスト<sup>10, 14)</sup>を導入する。

##### 4.1 仮想釘付けテストの原理

釘付けテストでは、上下界値  $\bar{z}, \underline{z}$  を入力として、問題を固定部分とそれ以外の縮小問題に分割した。上下界値は、当然

$$\underline{z} \leq z^* \leq \bar{z} \quad (4.1)$$

の関係を満たしているが、以下では  $[\underline{z}, \bar{z}]$  間の任意の  $l$  を下界値と見立てて釘付けテストを行うことを考える。この  $l$  を、試行値と呼ぶ。

$\bar{z}$  と  $l$  を用いて釘付けテストを実行すると、一部の  $x_j$  が 1 または 0 に固定されるが、 $l$  が下界値とは限らないので、この釘付けは正しいとは保証されない。しかし、このように (仮に) 釘付けされた変数の添字集合をそれぞれ  $F_1(l), F_0(l)$  とし、次を考える。

縮小問題  $R(l)$  :

$$\text{maximize} \quad (1.1) \quad (4.2)$$

$$\text{subject to} \quad (1.2) \sim (1.4) \quad (4.3)$$

$$x_j = 1, \quad j \in F_1(l) \quad (4.4)$$

$$x_j = 0, \quad j \in F_0(l) \quad (4.5)$$

$R(l)$  の最適関数値を  $z_l^*$  とし、 $l$  に対する実現値と呼ぶ。ただし、 $R(l)$  が実行不可能なら、 $z_l^* := -\infty$  とする。このとき、以下が成立する。

定理 2

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

(ii)  $l > z^* \Rightarrow z_i^* \leq z^*$

(iii)  $l \leq l' \Rightarrow z_i^* \geq z_i^*$ . 特に,  $R(l)$  が実行不可能  $\Rightarrow R(l')$  も実行不可能.

(iv)  $l \leq z_i^* \Rightarrow z_i^* = z^*$

証明: (i) この場合,  $l$  は実際に下界値なので, 釘付けは正しく行われ,  $z_i^* = z^*$  となる.

(ii) DCKP にさらに制約式 (4.4), (4.5) を付加した問題が  $R(l)$  なので, それらの最適値の間には常に  $z_i^* \leq z^*$  の関係がある.

(iii)  $\text{gap} := \bar{z} - l$  を考えると,  $\text{gap}$  が小さい程多くの変数が 0 または 1 に固定され (定理 1 参照), 従って目的関数値は減少する.

(iv) (i), (ii) より,  $l \leq z_i^*$  となるのは  $l \leq z^*$  のときのみで, この場合は再び (i) より,  $z_i^* = z^*$ . ■

#### 4.2 仮想釘付けアルゴリズム

上の定理より, 適当な  $l \leq \bar{z}$  で釘付けテストを行って  $R(l)$  を解いたとき, (iv) が成立していれば問題は解けたことになる. このとき,  $\text{gap} := \bar{z} - l$  が小さいと,  $R(l)$  は元問題よりはるかに小さい問題となっていることが多いので, この解法が有利と期待される. (iv) が成立しない場合の処理を含めて, 以下の解法を提案する.

#### 仮想釘付けテスト

Step 1.  $l \leftarrow \max\{\bar{z} - \alpha, \underline{z}\}$   
 Step 2. 試行値  $l$  で釘付けテストを行い, 縮小問題  $R(l)$  を解いて  $z_i^*$  を求める.  
 Step 3.  $l \leq z_i^*$  なら Step 5 へ, そうでなければ Step 4 へ進む.  
 Step 4.  $l \leftarrow \max\{l - \alpha, z_i^*\}$  として Step 2 へ戻る.  
 Step 5.  $z_i^* = z^*$  で, 最適解が得られた.

ここで,  $\alpha$  は適当な正整数で, 試行値を最初  $l := \bar{z} - \alpha$  (と  $\underline{z}$  の大きい方) とし, ここで最適解が得られなければ, 試行値をさらに  $\alpha$  だけ下げて,  $l := l - \alpha$  などとすることを意味している.

#### 5. 数値実験

本節では, DCKP にラグランジュ緩和と釘付けテストを適用した場合の計算機実験を行い, その性能を評価する. アルゴリズムは ANSI C 言語により実装し, IBM RS/6000 SP44 Model 270 (CPU

: POWER3-II SMP 2way, 375MHz) 上で実験を行った.

#### 5.1 例題の生成

商品数  $n$  を 1000 ~ 16000 とし,  $w_j$  と  $p_j$  は以下の関連タイプ<sup>7)</sup> によりランダムに発生させた.

- 無相関 (UNCOR)

$p_j$  : [1, 1000] 間の一様乱数,

$w_j$  : [1, 1000] 間の一様乱数で  $p_j$  と独立.

- 弱相関 (WEAK)

$w_j$  : [1, 1000] 間の一様乱数,

$p_j$  :  $w_j + \theta_j$ ; 但し,  $\theta_j$  は [0, 200] 間の一様乱数で  $w_j$  と独立.

ナップサック容量は  $c = 250n$  としたが, これは商品の平均重量が 500.5 なので, 全商品の約半分を収容できることを意味する. また, 排他制約は  $nC_2 = n(n-1)/2$  個の可能な対から, 確率  $d/(n-1)$  でランダムに抽出した. これより, 生成される排他制約式の数は, 平均  $nd/2$  となる.  $d$  は排他制約の生起率を制御するパラメータで, 以下では  $d = 0.1 \sim 0.8$  のケースを実験した.

#### 5.2 上下界値の比較

最初に上下界値の計算を行った. 表 5.1 と表 5.2 に無相関, 弱相関の場合の実験結果をそれぞれ示す. ここで, 'Lagrange UB' の欄はラグランジュ緩和による上界値  $\bar{z}$  とその計算時間を表しており, 'Greedy LB', '2-opt LB' の欄はそれぞれの方法で得られた下界値について,  $\bar{z}$  からのギャップ (gap) と CPU 時間を示している. 各行は, ランダム発生した 10 例題についての平均値である.

これらの表から, 次のことが結論される.

- ラグランジュ緩和は  $n$  の増加にともなって, また排他制約数  $m$  の増加にともなって計算時間が増大する.
- $d$  の増加によって, gap が極めて大きくなる. この傾向は, 無相関の場合に特に大きい.
- グリーディ解と 2-opt 解では, gap はあまり差がないが,  $n$  が大きくなると計算時間では, 2-opt 法が圧倒的に不利となる.
- 無相関の場合よりも弱相関の場合の方が目的関数値, gap とともに小さい.

表 5.1 上下界値の比較 (無相関)

Table 5.1 Upper and lower bounds (UNCOR).

$d$	$n$	$m$	Lagrange UB		Greedy LB		2-opt LB	
			$\bar{z}_L$	CPU sec.	gap	CPU sec.	gap	CPU sec.
0.1	1000	48.9	398721.9	0.12	30.4	0.00	28.5	0.12
	2000	96.9	802688.5	0.38	21.3	0.01	18.9	1.07
	4000	199.5	1601207.5	0.92	42.8	0.02	40.9	8.66
	8000	403.8	3196945.2	2.74	8.5	0.08	7.3	71.15
	16000	802.6	6394867.3	8.05	90.8	0.52	90.5	544.64
0.2	1000	99.0	394132.6	0.19	55.4	0.01	49.7	0.13
	2000	197.0	794909.3	0.50	24.6	0.01	22.5	1.22
	4000	394.3	1584789.9	1.49	45.1	0.02	43.1	8.71
	8000	805.0	3162459.6	4.03	34.3	0.08	32.8	71.94
	16000	1598.4	6328402.4	11.28	25.9	0.53	24.8	754.01
0.4	1000	198.9	386437.4	0.32	84.7	0.00	82.9	0.13
	2000	396.5	778653.3	0.83	86.0	0.01	83.3	1.27
	4000	792.1	1553682.9	2.19	85.5	0.02	84.3	7.69
	8000	1605.6	3097011.0	9.82	258.2	0.09	238.2	76.49
	16000	3215.3	6199034.2	30.62	430.4	0.54	429.4	798.64
0.8	1000	396.9	370545.3	0.63	521.6	0.00	509.8	0.14
	2000	798.6	747467.6	1.97	793.5	0.01	769.9	1.26
	4000	1600.0	1492060.0	5.68	1610.6	0.02	1609.6	6.89
	8000	3221.8	2975262.2	21.10	3253.1	0.09	3252.2	61.41
	16000	6411.0	5947688.4	53.90	6152.4	0.57	6151.2	794.78

表 5.2 上下界値の比較 (弱相関)

Table 5.2 Upper and lower bounds (WEAK).

$d$	$n$	$m$	Lagrange UB		Greedy LB		2-opt LB	
			$\bar{z}_L$	CPU sec.	gap	CPU sec.	gap	CPU sec.
0.1	1000	48.9	329563.6	0.10	21.3	0.00	18.2	0.94
	2000	96.9	660578.6	0.25	18.0	0.02	15.5	8.70
	4000	199.5	1320306.8	0.74	15.6	0.02	11.1	55.85
	8000	403.8	2640457.7	1.98	14.4	0.08	12.2	462.42
	16000	802.6	5280144.6	4.80	8.8	0.53	6.2	20330.01
0.2	1000	99.0	328703.7	0.16	42.0	0.00	28.0	0.89
	2000	197.0	659140.0	0.37	28.2	0.01	24.4	10.70
	4000	394.3	1316941.0	1.09	30.6	0.02	29.3	49.91
	8000	805.0	2633633.5	2.85	29.7	0.08	27.1	440.63
	16000	1598.4	5266806.6	9.86	85.3	0.53	83.2	17081.67
0.4	1000	198.9	327089.9	0.26	53.3	0.00	47.6	1.07
	2000	396.5	655981.1	0.55	43.2	0.00	39.3	8.49
	4000	792.1	1310745.1	1.83	30.6	0.02	28.3	48.80
	8000	1605.6	2620504.0	5.99	88.5	0.09	85.6	417.87
	16000	3215.3	5241053.9	15.14	106.5	0.55	105.0	14454.51
0.8	1000	396.9	323955.7	0.47	129.5	0.00	122.3	0.97
	2000	798.6	649708.3	1.30	268.6	0.01	260.2	9.34
	4000	1600.0	1298263.9	3.26	485.6	0.02	474.2	60.71
	8000	3221.8	2595308.6	9.00	790.0	0.09	787.1	438.42
	16000	6411.0	5190629.5	18.16	1744.5	0.57	1741.6	13848.40

以上より、以後下界値の計算には専らグリーディ法を使用し、2-opt 法は用いない。すなわち、 $z := z_G$  とする。

### 5.3 NUOPT による厳密解

DCKP を商用ソフト NUOPT で直接解いた時の結果を表 5.3, 表 5.4 に示す。NUOPT は汎用の数値計画ソフトで、DCKP などの 0-1 計画問題を分枝限定法<sup>(6)</sup> で解くことができる。これらの表にはこのときに生成される子問題数 ('#subprobs') と CPU 時間 (秒単位) が示されている。'solved' は、10 回の試行のうち最適解が得られた回数で、各行は計算が成功したのものについての平均値である。計算が不成功に終る理由は、メモリー領域の不足である。

表 5.3 NUOPT による厳密解 (無相関)  
Table 5.3 Direct solution using NUOPT (UNCOR).

$d$	$n$	#subprobs	CPU sec.	solved
0.1	1000	2081.3	8.77	10
	2000	1718.7	17.52	10
	4000	3022.0	57.36	10
	8000	2614.6	162.48	6
	16000	4871.2	587.86	7
0.2	1000	2014.0	9.61	10
	2000	2228.1	24.95	10
	4000	3191.8	83.35	9
	8000	4717.1	223.88	7
	16000	3595.4	600.55	7
0.4	1000	1362.1	9.11	10
	2000	2347.4	26.62	10
	4000	2161.2	88.14	9
	8000	1946.0	179.24	9
	16000	2304.6	427.22	3
0.8	1000	1246.8	14.11	10
	2000	3229.4	76.36	10
	4000	1560.0	92.84	9
	8000	1683.4	161.56	5
	16000	-	-	0

以上の表より、次の所見を得る。

- 無相関、弱相関のいずれでも  $n = 4000$  あたりから解けない問題が出現し、 $n = 16000$  では、ほとんどの問題が困難になる。
- $n, d$  の増加とともに計算時間が増加し、解ける問題数も減少して、問題は困難になる。
- 弱相関の場合、無相関の場合より生成される子問題数がかなり多くなり、計算時間も増える。

表 5.4 NUOPT による厳密解 (弱相関)

Table 5.4 Direct solution using NUOPT (WEAK).

$d$	$n$	#subprobs	CPU sec.	solved
0.1	1000	2956.5	8.94	10
	2000	5451.7	36.73	10
	4000	9697.4	95.32	10
	8000	107476.2	1306.21	9
	16000	2299.0	178.36	4
0.2	1000	2349.8	8.3	10
	2000	5363.5	38.5	10
	4000	9399.6	129.7	10
	8000	36909.5	615.3	7
	16000	68886.6	2427.4	3
0.4	1000	4258.2	20.0	9
	2000	4178.1	53.3	10
	4000	7334.1	142.8	9
	8000	38197.4	693.6	7
	16000	-	-	0
0.8	1000	4056.2	31.15	10
	2000	3115.7	62.01	10
	4000	10094.1	254.23	7
	8000	9849.8	360.49	5
	16000	11699.0	1029.58	2

### 5.4 釘付けテストによる厳密解

表 5.5 と表 5.6 に釘付けテストの計算結果を示す。ここで、 $n'$  と  $m'$  は縮小問題のサイズで、それぞれ変数の数と制約式数を表す。'reduction' は次で定義される縮小率を示している。

$$\text{縮小率} = \sqrt{n'm'/nm}$$

また  $\text{CPU}_1$  は上下界値の算出と釘付けテストの計算時間であり、 $\text{CPU}_2$  は縮小問題を NUOPT で解くのに要した時間、 $\text{CPU}_T$  はそれらの和である。'solved' の欄は 10 回の試行中厳密解が得られた回数で、各行はそれらについての平均である。

表 5.5 ~ 表 5.6 より、以下がわかる。

- $d$  の増加とともに、釘付けテストによる問題縮小効果が激減する。
- 弱相関では、無相関に比べ釘付けの効果はかなり落ちる。
- 釘付けテストによる厳密解法により、NUOPT を直接適用したときよりも多くの問題がとけるようになった。
- 計算時間も、数倍から、数十倍速くなっている。

表 5.5 釘付けテストによる解法 (無相関)  
 Table 5.5 Solution by pegging algorithm (UNCOR).

$d$	$n$	$n'$	$m'$	reduction	CPU <sub>1</sub>	CPU <sub>2</sub>	CPU <sub>T</sub>	solved
0.1	1000	66.0	2.7	0.05	0.11	0.33	0.45	10
	2000	92.0	3.9	0.06	0.38	0.24	0.63	10
	4000	326.5	13.9	0.14	0.93	2.60	3.53	10
	8000	150.2	6.1	0.05	2.78	1.19	3.98	10
	16000	2432.6	119.3	0.60	8.45	80.15	88.60	10
0.2	1000	115.6	8.5	0.10	0.18	0.49	0.67	10
	2000	105.1	5.8	0.04	0.51	0.56	1.07	10
	4000	385.3	31.0	0.09	1.53	4.11	5.64	10
	8000	566.6	48.6	0.06	4.12	15.58	19.69	10
	16000	885.6	66.8	0.05	11.79	18.19	29.98	10
0.4	1000	164.6	27.2	0.15	0.32	1.57	1.89	10
	2000	302.5	91.7	0.16	0.85	5.48	6.34	10
	4000	696.7	121.1	0.16	2.22	18.39	20.61	10
	8000	3236.2	621.7	0.40	9.93	59.51	69.45	10
	16000	8935.4	1728.7	0.55	31.24	111.42	141.68	6
0.8	1000	647.9	242.9	0.63	0.63	8.38	9.01	10
	2000	1596.9	625.8	0.79	1.97	63.71	65.67	10
	4000	3899.8	1558.8	0.97	5.72	108.97	114.68	9
	8000	8000.0	3221.8	1.00	20.98	160.98	182.08	5
	16000	16000.0	6411.8	1.00	54.05	-	-	0

表 5.6 釘付けテストによる解法 (弱相関)  
 Table 5.6 Solution by pegging algorithm (WEAK).

$d$	$n$	$n'$	$m'$	reduction	CPU <sub>1</sub>	CPU <sub>2</sub>	CPU <sub>T</sub>	solved
0.1	1000	211.4	7.8	0.18	0.11	1.59	1.70	10
	2000	356.4	15.0	0.16	0.28	4.07	4.34	10
	4000	627.1	23.1	0.13	0.77	18.59	19.36	10
	8000	1162.8	48.3	0.13	2.06	196.04	198.10	10
	16000	1434.8	52.5	0.08	5.47	9104.78	9110.25	10
0.2	1000	391.2	34.2	0.37	0.16	3.15	3.30	10
	2000	537.8	45.6	0.25	0.39	6.97	7.36	10
	4000	1104.0	98.6	0.26	1.15	26.81	27.88	10
	8000	241.6	203.9	0.27	3.15	118.31	121.45	10
	16000	8737.2	857.9	0.54	10.48	135.45	141.84	4
0.4	1000	394.5	68.7	0.43	0.26	6.40	7.21	9
	2000	785.0	137.8	0.37	0.57	8.75	9.32	10
	4000	1150.7	196.7	0.27	1.90	33.47	35.36	10
	8000	5577.5	1101.3	0.69	6.17	185.03	191.11	7
	16000	9308.5	1819.7	0.57	15.42	9101.17	9116.28	3
0.8	1000	789.5	308.1	0.78	0.48	23.61	24.09	10
	2000	1925.1	765.9	0.96	1.32	57.21	58.52	10
	4000	3998.8	1599.5	1.00	3.26	244.94	248.30	6
	8000	8000.0	3221.8	1.00	9.16	357.69	366.86	5
	16000	16000.0	6411.0	1.00	18.90	1029.21	1047.00	2

表 5.7 仮想釘付けによる厳密解法 (無相関)

Table 5.7 Solution by virtual pegging algorithm (UNCOR).

$d$	$n$	$m$	Virtual pegging			NUOPT	
			#rep.	CPU sec.	solved	CPU sec.	solved
0.05	16000	408.5	1.0	10.62	10	688.67	7
	32000	801.2	1.0	30.08	10	1122.33	4
	64000	1582.8	1.0	1676.57	10	1760.51	2
0.1	16000	802.6	1.0	12.6	10	588.62	7
	32000	1605.0	1.4	80.4	10	137.18	1
	64000	3182.3	1.3	1355.3	9	1760.87	2

表 5.8 仮想釘付けによる厳密解 (弱相関)

Table 5.8 Solution by virtual pegging algorithm (WEAK).

$d$	$n$	$m$	Virtual pegging			NUOPT	
			#rep.	CPU sec.	solved	CPU sec.	solved
0.05	16000	408.5	1.0	907.68	10	1592.42	4
	32000	801.2	1.0	2173.40	9	12162.56	1
	64000	1582.8	1.0	2807.04	8	4257.82	1
0.1	16000	802.6	1.0	3187.75	10	178.46	3
	32000	1605.0	1.0	2103.50	10	1368.96	3
	64000	3182.3	1.0	3665.47	9	-	0

## 5.5 仮想釘付けテストによる厳密解

より大型の問題として、 $n = 16000 \sim 64000$  程度の問題を考える。この場合、排他制約の数が多いとどの方法でも厳密解を得ることは難しいので、 $d$  の値を  $d = 0.05 \sim 0.10$  として実験した。結果を表 5.7 ~ 5.8 に示す。仮想釘付けの欄で '#rep.' とあるのは、4.2 節のアルゴリズムで Step 2 ~ Step 4 を反復した回数であり、'NUOPT' の欄は NUOPT による直接解法を示している。'solved' は、10 例題中それぞれの方法で最適解が得られた回数を示す。いずれの場合も、仮想釘付けテストの方が多くの問題が解け、計算時間でも仮想釘付けがかなり有利になっている。

## 6. まとめ

DCKP についてラグランジュ緩和と釘付けテストを組み合わせるアルゴリズムを開発し、商品数が数千から数万程度の問題を解くことに成功した。PCKP についてもほぼ同様の結果を得ている<sup>15)</sup>。今後、縮小問題を解く部分に専用のアルゴリズムを準備するなどして、より大規模な問題を解くことを試みる予定である。

## 参考文献

- 1) Dembo, R. S. and Hammer, P. L., "A reduction algorithm for knapsack problems", *Methods of Operations Research*, Vol. 36, pp. 49-60, 1980.
- 2) Fayard, D. and Plateau, G., "Resolution of the 0-1 knapsack problem: comparison of methods", *Mathematical Programming*, Vol. 8, pp. 272-307, 1975.
- 3) Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman and Company, 1979.
- 4) Ingargiola, G. P. and Korsh, J. F., "A reduction algorithm for zero-one single knapsack problems", *Management Science*, Vol. 20, pp. 460-463, 1973.
- 5) Kellerer, H., Pferschy, U. and Pisinger, D., *Knapsack Problems*, Springer Verlag, 2004.
- 6) 今野 浩, 鈴木久敏 (編), 整数計画法と組合せ最適化, 日科技連, 1982.
- 7) Martello, S. and Toth, P., *Knapsack Problems: Algorithms and Computer Im-*

- plementations*, Wiley, New York, 1990.
- 8) Samphaiboon, N., Yamada, T., “Heuristic and exact algorithms for the precedence-constrained knapsack problem”, *Journal of Optimization Theory and Applications*, Vol. 105, pp. 659-676, 2002.
  - 9) Sedgewick, R., *Algorithms in C, 3rd Ed.*, Addison-Wesley, Reading, 1998.
  - 10) セニスカ・アミント, “排他制約付きナップサック問題の解法に関する研究”, 修士論文, 防衛大学校理工学研究科, 情報数理専攻, 2005.
  - 11) (株)数理システム, *NUOPT Manual*, <http://www.msi.co.jp/nuopt>, 2002.
  - 12) 柳浦睦憲, 茨木俊秀, 組合せ最適化 – メタ戦略を中心として –, 朝倉書店, 2001.
  - 13) Yamada, T., Kataoka, S. and Watanabe, K., “Heuristic and exact algorithms for disjunctively constrained knapsack problem”, *IPSJ Journal*, Vol. 43, pp. 2864-2870, 2002.
  - 14) 柳 秉俊, “順序制約付きナップサック問題への釘付けアプローチ”, 修士論文, 防衛大学校理工学研究科, 情報数理専攻, 2005.
  - 15) 柳 秉俊, セニスカ・アミント, 山田武夫, “付加制約のあるナップサック問題への釘付けアプローチ”, 研究集会「決定理論と最適化アルゴリズム」, 京都大学数理解析研究所講義録 1409, pp. 101-114, 2005.

# Heuristic and Exact Algorithms for the Knapsack Problem with Side Constraints

By SENISUKA AMINTO\*, YOU BYUNGJUN\* and YAMADA TAKEO

\*\*

(Received January 25, 2005; accepted for publication April 12, 2005)

## Abstract

In this paper we are concerned with some variants of the knapsack problem (KP). 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 an optimal set of items to be packed into the knapsack. However, we are required to satisfy some side constraints as well. Such a problem includes the disjunctively constrained knapsack problem (DCKP), and the precedence constrained knapsack problem (PCKP) among others. We present an approach that combines the Lagrangean relaxation and the pegging test to solve these problems to optimality. Through this method, we are able to solve DCKPs as well as PCKPs with thousands of items within a reasonable computing time.

**Key words :** knapsack problem, combinatorial optimization, side constraints.

---

\* #41 Graduate student, Department of Computer Science, The National Defense Academy

\*\* Professor, Department of Computer Science, The National Defense Academy