

Approximation for Knapsack Problems with Multiple Constraints

ZHANG Li'ang (张立昂) and ZHANG Yin (章 寅)

Department of Computer Science and Technology, Peking University, Beijing 100871, P.R. China

Received July 6, 1998; revised November 24, 1998.

Abstract In this paper, the approximation for four kinds of knapsack problems with multiple constraints is studied: 0/1 Multiple Constraint Knapsack Problem (0/1 MCKP), Integer Multiple Constraint Knapsack Problem (Integer MCKP), 0/1 k -Constraint Knapsack Problem (0/1 k -CKP) and Integer k -Constraint Knapsack Problem (Integer k -CKP). The following results are obtained:

1) Unless $NP = co-R$, no polynomial time algorithm approximates 0/1 MCKP or Integer MCKP within a factor $k^{(1/2)-\sigma}$ for any $\sigma > 0$; unless $NP = P$, no polynomial time algorithm approximates 0/1 MCKP or Integer MCKP within a factor $k^{(1/4)-\sigma}$ for any $\sigma > 0$, where k stands for the number of constraints.

2) For any fixed positive integer k , 0/1 k -CKP has a fully polynomial time approximation scheme (FPTAS).

3) For any fixed positive integer k , Integer k -CKP has a fast FPTAS which has time complexity $O\left(n + \frac{1}{\varepsilon^3} + \frac{1}{\varepsilon^{2k+1-2}}\right)$ and space complexity $O(n + (1/\varepsilon^3))$, and finds an approximate solution to within ε of the optimal solution.

Keywords knapsack problem, approximation algorithm, FPTAS

1 Introduction

The knapsack problem is a well-known combinatorial optimization problem that finds applications to capital budgeting, loading problems, solution of large optimization problems, and computer systems. An extensive literature exists on approximation algorithms for various forms of knapsack problems^[1-8]. In this paper, we study the approximation for four kinds of knapsack problems with multiple constraints: 0/1 Multiple Constraint Knapsack Problem (0/1 MCKP), Integer Multiple Constraint Knapsack Problem (Integer MCKP), 0/1 k -Constraint Knapsack Problem (0/1 k -CKP) and Integer k -Constraint Knapsack Problem (Integer k -CKP).

1.1 Knapsack Problems with Multiple Constraints

A 0/1 Multiple Constraint Knapsack Problem (0/1 MCKP) is defined as follows: Given n pairs of positive integers (v_j, w_j) ($j = 1, 2, \dots, n$), k non-empty sets S_i ($S_i \subseteq \{1, 2, \dots, n\}$, $i = 1, 2, \dots, k$), and k positive integers b_i ($i = 1, 2, \dots, k$), find x_1, x_2, \dots, x_n ($x_j \in \{0, 1\}$, $j = 1, 2, \dots, n$) so as to

$$\begin{aligned} & \max \sum_{j=1}^n v_j x_j \\ & \text{s.t.} \quad \sum_{j \in S_i} w_j x_j \leq b_i, \quad i = 1, 2, \dots, k \end{aligned}$$

This work is supported by The Key Project Fund of the State Ninth Five-Year Plan and the Science Foundation of Peking University.

Without loss of generality, we assume that $\cup_{i=1}^k S_i = \{1, 2, \dots, n\}$. Moreover, we can assume that for any i ($1 \leq i \leq k$) and any j ($j \in S_i$), $w_j \leq b_i$. (Otherwise, item j can be totally eliminated from the problem.) We may think of j as indexing items, with associated values v_j and weights w_j ($1 \leq j \leq n$). The object is to find the most valuable possible selection of items which satisfies the k weight constraints.

In a 0/1 MCKP, if the variables x_j are not restricted to 0, 1 values, but may be non-negative integers, the resulting problem is called an Integer Multiple Constraint Knapsack Problem (Integer MCKP).

Note. In the above two problems, k is not fixed and can be any positive integer.

If the number of constraints in the problems 0/1 MCKP and Integer MCKP, say k , is fixed, the resulting problems are called the 0/1 k -Constraint Knapsack Problem (0/1 k -CKP) and the Integer k -Constraint Knapsack Problem (Integer k -CKP) respectively.

1.2 Main Results

The following results on approximation for knapsack problems with multiple constraints are obtained in this paper:

1) Unless $NP = co - R$, no polynomial time algorithm approximates 0/1 MCKP or Integer MCKP within a factor $k^{(1/2)-\sigma}$ for any $\sigma > 0$; unless $NP = P$, no polynomial time algorithm approximates 0/1 MCKP or Integer MCKP within a factor $k^{(1/4)-\sigma}$ for any $\sigma > 0$, where k stands for the number of constraints.

2) For any fixed positive integer k , 0/1 k -CKP has a fully polynomial time approximation scheme (FPTAS), i.e. an algorithm which finds an approximate solution to within ε of the optimal solution and operates in time bounded by a polynomial in the length of the encoded input and $1/\varepsilon^{[9]}$.

3) For any fixed positive integer k , Integer k -CKP has a fast FPTAS which has time complexity $O(n + 1/\varepsilon^3 + 1/\varepsilon^{2^{k+1}-2})$ and space complexity $O(n + 1/\varepsilon^3)$, and finds an approximate solution to within ε of the optimal solution.

Comment. Complexity estimates are based on the assumption that computer word length is sufficient to accommodate numbers as large as V^* , b_i ($i = 1, 2, \dots, k$), and n , where V^* is the value of an optimal solution. Arithmetic operations on numbers as large as these are assumed to require constant time. Thus factors such as $\log b_i$ do not appear in time and space bounds obtained in this paper. However, we shall not assume word length on the order of $1/\varepsilon$ or n bits.

1.3 Outline of the Paper

The remainder of the paper is organized as follows. In Section 2, we examine the hardness of approximation for 0/1 MCKP and Integer MCKP. In Section 3, we present a pseudo-polynomial time algorithm for 0/1 k -CKP. This algorithm not only forms the foundation of an FPTAS for 0/1 k -CKP but also provides insights into Integer k -CKP. Based on this algorithm, we develop an FPTAS for 0/1 k -CKP in Section 4. In Section 5, we develop a fast FPTAS for Integer k -CKP. Several efficient techniques are adopted to gain high performance. We end the paper with an open problem.

2 The Hardness of Approximation for 0/1 MCKP and Integer MCKP

The following two results have been presented in [10].

- 1) Unless $NP = co - R$, no polynomial time algorithm approximates Max Clique within a factor $n^{1-\varepsilon}$ for any $\varepsilon > 0$.

2) Unless $NP = P$, no polynomial time algorithm approximates Max Clique within a factor $n^{(1/2)-\varepsilon}$ for any $\varepsilon > 0$, where n stands for the number of vertices in the graph.

We can easily represent a Max Clique with a 0/1 MCKP. Given a graph $G = (V, E)$, where $V = \{1, 2, \dots, n\}$, consider the 0/1 MCKP as follows:

$$\begin{aligned} & \max \sum_{j=1}^n x_j \\ & \text{s.t. } x_i + x_j \leq 1, \text{ if } \{i, j\} \notin E, \quad i, j = 1, 2, \dots, n, i \neq j \\ & x_j \in \{0, 1\}, \quad j = 1, 2, \dots, n \end{aligned} \quad (2.1)$$

It is evident that (x_1, x_2, \dots, x_n) is a feasible solution of the above 0/1 MCKP if and only if $\{j | x_j = 1, j = 1, 2, \dots, n\}$ is a clique of G .

A Max Clique can also be represented with an Integer MCKP. To achieve this, we only need to replace (2.1) with the following constraints:

$$\begin{aligned} & x_j \leq 1, \quad j = 1, 2, \dots, n \\ & x_j \geq 0, \quad \text{integers}, \quad j = 1, 2, \dots, n \end{aligned}$$

Observing that $k \leq n^2$, where k is the number of constraints in the above two knapsack problems with multiple constraints, we have

Theorem 1 (Hardness of Approximation for 0/1 MCKP and Integer MCKP).

1) Unless $NP = co-R$, no polynomial time algorithm approximates 0/1 MCKP or Integer MCKP within a factor $k^{(1/2)-\sigma}$ for any $\sigma > 0$.

2) Unless $NP = P$, no polynomial time algorithm approximates 0/1 MCKP or Integer MCKP within a factor $k^{(1/4)-\sigma}$ for any $\sigma > 0$, where k stands for the number of constraints.

Theorem 1 shows that, unless $NP = P$, neither 0/1 MCKP nor Integer MCKP has a polynomial time approximation algorithm with a constant ratio.

3 A Pseudo-Polynomial Time Algorithm for 0/1 k -CKP

For a 0/1 k -CKP:

$$\begin{aligned} & \max \sum_{j=1}^n v_j x_j \\ & \text{s.t. } \sum_{j \in S_i} w_j x_j \leq b_i, \quad i = 1, 2, \dots, k \\ & x_j \in \{0, 1\}, \quad j = 1, 2, \dots, n \end{aligned} \quad (3.1)$$

3.1 Preparations

Let $S_i^1 = S_i$, $S_i^0 = \overline{S_i} = \{1, 2, \dots, n\} - S_i$, $i = 1, 2, \dots, k$. Let T_1, T_2, \dots, T_l ($l \leq 2^k - 1$) be all the different non-empty sets with the form $\bigcap_{i=1}^k S_i^{\alpha_i}$ ($\alpha_i \in \{0, 1\}$, $i = 1, 2, \dots, k$). The following facts about T_p ($p = 1, \dots, l$) are evident:

- 1) $T_p \cap T_q = \emptyset$, $1 \leq p < q \leq l$.
- 2) $\bigcup_{p=1}^l T_p = \{1, 2, \dots, n\}$.
- 3) Each S_i can be represented as the union of some T_p .

Suppose that $S_i = \bigcup_{p \in \Delta_i} T_p$, $\Delta_i \subseteq \{1, 2, \dots, l\}$, $i = 1, 2, \dots, k$. Let $n_p = |T_p|$, we have $\sum_{p=1}^l n_p = n$.

It is evident that the computation of T_p and Δ_i requires $O(n)$ time and space (including input space).

Let $KP_p(y)$ be an ordinary 0/1 knapsack problem as follows:

$$\begin{aligned} & \max \sum_{j \in T_p} v_j x_j \\ & \text{s.t.} \sum_{j \in T_p} w_j x_j \leq y \\ & x_j \in \{0, 1\}, \quad j \in T_p \end{aligned}$$

Let $V_p^*(y)$ be the value of an optimal solution of $KP_p(y)$, $p = 1, 2, \dots, l$.

3.2 Algorithm ZZ1

Step 1. For any p ($p = 1, 2, \dots, l$), find all possible $(V_p(r), Y_p(r), I_p(r))$, $r = 0, 1, \dots, m_p$, where

- 1) $V_p(r) = V_p^*(Y_p(r))$;
- 2) for any $y < Y_p(r)$, $V_p^*(y) < V_p(r)$;
- 3) $I_p(r)$ is an optimal solution of $KP_p(Y_p(r))$;
- 4) $0 = V_p(0) < V_p(1) < \dots < V_p(m_p) \leq \sum_{j \in T_p} v_j \triangleq d_p$, where \triangleq denotes "to be defined as";
- 5) $0 = Y_p(0) < Y_p(1) < \dots < Y_p(m_p)$.

Clearly, $m_p \leq d_p \leq n_p v_{\max}$, where $v_{\max} = \max\{v_j | 1 \leq j \leq n\}$.

Based on the dynamic programming algorithm presented in [5], we can achieve the above computation with $O(n_p^2 d_p)$ time and $O(n_p d_p)$ space for any p in $\{1, 2, \dots, l\}$.

Thus, the computation for $(V_p(r), Y_p(r), I_p(r))$ requires $\sum_{p=1}^l O(n_p^2 d_p)$ time and $\sum_{p=1}^l O(n_p d_p)$ space. Since $d_p \leq n_p v_{\max}$, we have 1) $\sum_{p=1}^l O(n_p^2 d_p) = O(n^3 v_{\max})$ and 2) $\sum_{p=1}^l O(n_p d_p) = O(n^2 v_{\max})$. Therefore, Step 1 requires $O(n^3 v_{\max})$ time and $O(n^2 v_{\max})$ space.

Step 2. Consider the following problem:

$$\begin{aligned} & \max \sum_{p=1}^l V_p(r_p) \\ & \text{s.t.} \sum_{p \in \Delta_i} Y_p(r_p) \leq b_i, \quad i = 1, 2, \dots, k \\ & r_p \in \{0, 1, \dots, m_p\}, \quad p = 1, 2, \dots, l \end{aligned} \quad (3.2)$$

Find an optimal solution of (3.2), say $(r_1^*, r_2^*, \dots, r_l^*)$, by enumerating all possible (r_1, r_2, \dots, r_l) . It is evident that $I^* = \cup_{p=1}^l I_p(r_p^*)$ is an optimal solution of the original 0/1 k -CKP (3.1).

For each (r_1, r_2, \dots, r_l) , we need to compute the value of $\sum_{p \in \Delta_i} Y_p(r_p)$ ($i = 1, 2, \dots, k$) and $\sum_{p=1}^l V_p(r_p)$. This requires $O(kl + l) = O(1)$ time and $O(1)$ space. Altogether, there are $\prod_{p=1}^l (1 + m_p) = O((n v_{\max})^l)$ possible (r_1, r_2, \dots, r_l) . Therefore, Step 2 requires $O((n \cdot v_{\max})^l)$ time. Since at any time we only need to keep the currently optimal (r_1, r_2, \dots, r_l) , the space required in Step 2 is bounded by $2 \cdot O(1) = O(1)$.

In summary, algorithm ZZ1 finds an optimal solution of 0/1 k -CKP (3.1) in $O(n^3 v_{\max}) + O((n v_{\max})^l) = O(n(n v_{\max})^{2k})$ time and $O(n^2 v_{\max}) + O(1) = O(n^2 v_{\max})$ space. Thus for any fixed k , ZZ1 is a pseudo-polynomial algorithm for 0/1 k -CKP.

Comment. In algorithm ZZ1, we change a multiple-constraint problem into a group of single-constraint problems by partitioning $\{1, 2, \dots, n\}$ into T_p ($p = 1, 2, \dots, l$). This technique can not only be applied to 0/1 k -CKP but also be applied to Integer k -CKP.

4 FPTAS for 0/1 k -CKP

Based on algorithm ZZ1 and the scaling technique, we can develop an FPTAS for 0/1 k -CKP now.

Algorithm ZZ2.

Input. Any $\varepsilon > 0$ and any instance of 0/1 k -CKP, say I .

Step 1. Let scale factor $K = \max(\lfloor \varepsilon v_{\max}/n \rfloor, 1)$.

Step 2. Replace v_j with $u_j = \lceil v_j/K \rceil$, $j = 1, 2, \dots, n$. Now we have a new instance of 0/1 k -CKP, say I' .

Step 3. Perform algorithm ZZ1 on I' , take the solution produced by ZZ1 as an approximate solution of I .

Theorem 2. Algorithm ZZ2 has time complexity $O(n(n^2/\varepsilon)^{2k})$ and space complexity $O(n^3/\varepsilon)$, and finds an approximate solution to within ε of the optimal solution. Therefore, ZZ2 is an FPTAS for 0/1 k -CKP.

Proof. Since $K(u_j - 1) < v_j \leq Ku_j$ ($j = 1, 2, \dots, n$), for any $J \subseteq \{1, 2, \dots, n\}$, we have $0 \leq K \sum_{j \in J} u_j - \sum_{j \in J} v_j < K |J| \leq Kn$. Let $(x_1^*, x_2^*, \dots, x_n^*)$ be an optimal solution of I , $\text{OPT}(I) = \sum_{j=1}^n v_j x_j^*$. Let (x_1, x_2, \dots, x_n) be the approximate solution given by ZZ2, $\text{ZZ2}(I) = \sum_{j=1}^n v_j x_j$. (Note: (x_1, x_2, \dots, x_n) is an optimal solution of I' .) Let $J^* = \{j | x_j^* = 1, 1 \leq j \leq n\}$, J be $\{j | x_j = 1, 1 \leq j \leq n\}$. We have:

$$\begin{aligned} \text{OPT}(I) - \text{ZZ2}(I) &= \sum_{j \in J^*} v_j - \sum_{j \in J} v_j \\ &= \left(\sum_{j \in J^*} v_j - K \sum_{j \in J^*} u_j \right) + \left(K \sum_{j \in J^*} u_j - K \sum_{j \in J} u_j \right) + \left(K \sum_{j \in J} u_j - \sum_{j \in J} v_j \right) \\ &\leq K \sum_{j \in J} u_j - \sum_{j \in J} v_j < Kn \end{aligned}$$

For any $\varepsilon > 0$, if $K = 1$, it is evident that $\text{ZZ2}(I) = \text{OPT}(I)$. Otherwise, $K > 1$, observing that $v_{\max} \leq \text{OPT}(I)$, we have

$$\text{OPT}(I) - \text{ZZ2}(I) < Kn \leq \varepsilon v_{\max} \leq \varepsilon \text{OPT}(I)$$

The time complexity of ZZ2 is $O(n(nv_{\max}/K)^{2k}) = O(n(n^2/\varepsilon)^{2k})$. The space complexity is $O(n^2 v_{\max}/K) = O(n^3/\varepsilon)$.

5 Fast FPTAS for Integer k -CKP

In this section, we develop a fast FPTAS for Integer k -CKP. The underlying technique of our algorithm is the same as that in algorithm ZZ1, i.e. to change a multiple-constraint problem into a group of single-constraint problems by partitioning $\{1, 2, \dots, n\}$ into T_p ($p = 1, 2, \dots, l$). In order to gain high performance, we adopt several efficient techniques used in [5] and [6].

5.1 A Greedy Algorithm for Integer k -CKP

For any instance of Integer k -CKP, say I , let l, T_p, n_p ($p = 1, 2, \dots, l$), v_{\max} and Δ_i ($i = 1, 2, \dots, k$) denote the same meanings as in Section 3. Define the value density of item j as v_j/w_j ($j = 1, 2, \dots, n$). For all p ($p = 1, 2, \dots, l$), let h_p be the item with the greatest value density in set T_p .

Algorithm GA.

Step 1. Solve the following linear programming problem $\text{LP}^*(I)$:

$$\begin{aligned} \max \quad & \sum_{p=1}^l v_{h_p} x_{h_p} \\ \text{s.t.} \quad & \sum_{p \in \Delta_i} w_{h_p} x_{h_p} \leq b_i, \quad i = 1, 2, \dots, k \end{aligned}$$

$$x_{h_p} \geq 0, \quad p = 1, 2, \dots, l$$

LP*(I) has k constraints and l variables. Since k is a constant, and $l \leq 2^k - 1$, we can find an optimal solution of LP*(I), say $(\bar{x}_{h_1}, \bar{x}_{h_2}, \dots, \bar{x}_{h_l})$, with $O(1)$ time and space. Moreover, we can ensure that at most k out of l values are non-zero.

Step 2. If $\sum_{p=1}^l \lfloor \bar{x}_{h_p} \rfloor v_{h_p} \geq v_{\max}$, then output an approximate solution (x_1, x_2, \dots, x_n) of I as:

$$\begin{aligned} x_{h_p} &= \lfloor \bar{x}_{h_p} \rfloor, \quad p = 1, 2, \dots, l \\ x_j &= 0, \quad j \in \{1, 2, \dots, n\} - \{h_1, h_2, \dots, h_l\} \end{aligned}$$

Otherwise, suppose that $v_{j_0} = v_{\max}$, output an approximate solution (x_1, x_2, \dots, x_n) of I as:

$$\begin{aligned} x_{j_0} &= 1 \\ x_j &= 0, \quad j \neq j_0, \quad 1 \leq j \leq n \end{aligned}$$

Clearly, algorithm GA requires $O(1)$ time and space as long as T_p, h_p ($p = 1, 2, \dots, l$), v_{\max} and Δ_i ($i = 1, 2, \dots, k$) have been prepared in advance. As for the approximation performance of GA, we have the following lemma:

Lemma 1. Let $V_0 = GA(I)$, $V^* = OPT(I)$, we have:

$$v_{\max} \leq V_0 \leq V^* < V_0 + k \cdot v_{\max} \leq (1 + k)V_0$$

Proof. We only need to prove that $V^* < V_0 + k \cdot v_{\max}$.

Let LP₀(I) be the following linear programming problem:

$$\begin{aligned} \max \quad & \sum_{j=1}^n v_j x_j \\ \text{s.t.} \quad & \sum_{j \in S_i} w_j x_j \leq b_i, \quad i = 1, 2, \dots, k \\ & x_j \geq 0, \quad j = 1, 2, \dots, n \end{aligned}$$

It is evident that $OPT(LP^*(I)) = OPT(LP_0(I))$, where $OPT(LP_0(I))$ and $OPT(LP^*(I))$ stand for the values of the optimal solutions of LP₀(I) and LP*(I) respectively.

Hence, we have

$$\begin{aligned} V_0 = GA(I) &\geq \sum_{p=1}^l \lfloor \bar{x}_{h_p} \rfloor v_{h_p} > \sum_{p=1}^l \bar{x}_{h_p} v_{h_p} - k \cdot v_{\max} \quad (\text{Note: At most } k \text{ values of } \bar{x}_{h_p} \text{ are non-zero.}) \\ &= OPT(LP^*(I)) - k \cdot v_{\max} = OPT(LP_0(I)) - k \cdot v_{\max} \geq V^* - k \cdot v_{\max} \end{aligned}$$

That is

$$V^* < V_0 + k \cdot v_{\max}$$

5.2 A Fast FPTAS for Integer k -CKP

Algorithm ZZ3.

Input. Any $\varepsilon > 0$ and any instance of Integer k -CKP, say I .

Step 1. Preparation. Find the values of l, T_p, n_p, h_p ($p = 1, 2, \dots, l$), Δ_i ($i = 1, 2, \dots, k$).

Step 2. Perform algorithm GA on I and find $V_0 = GA(I)$. From Lemma 1, we have

$$V_0 \leq V^* \leq (1 + k)V_0, \quad \text{where } V^* = OPT(I).$$

Let scale factor $K = \varepsilon^2 V_0 / (4k)$, value threshold $T = \varepsilon V_0 / (2k)$.

For each p ($p = 1, 2, \dots, l$), let

$$T_p^L = \{j \in T_p | v_j > T\}, \quad n_p^L = |T_p^L|$$

$$T_p^S = T_p - T_p^L = \{j \in T_p | v_j \leq T\}$$

Let q_p be the item with the greatest value density in T_p^S ($p = 1, 2, \dots, l$).

Let $Large = \cup_{p=1}^l T_p^L$, $Small = \cup_{p=1}^l T_p^S$, $v_{max}^S = \max\{v_j | j \in Small\}$

Step 3. Scaling.

For any $j \in T_p^L$, if $v_j \in (2^r T, 2^{r+1} T]$, $0 \leq r \leq \lfloor \log_2(V^*/T) \rfloor$, replace v_j with $u_j = \lfloor v_j / (2^r K) \rfloor 2^r$. Clearly

$$Ku_j \leq v_j < Ku_j + 2^r K = Ku_j + (2^r T)(K/T) < Ku_j + v_j(K/T) = Ku_j + \epsilon v_j / 2$$

i.e.

$$0 \leq v_j - Ku_j \leq (\epsilon/2)v_j$$

Moreover, it is evident that if $v_j \in (2^r T, 2^{r+1} T]$, then $u_j \in v_j(\frac{2}{\epsilon} 2^r, \frac{2}{\epsilon} 2^{r+1}]$ and $2^r |u_j$, i.e. u_j is divisible by 2^r .

Let $\Phi(t_1, t_2, \dots, t_k)$ be the following Integer k -CKP:

$$\max \sum_{j \in Small} v_j x_j$$

$$\text{s.t.} \quad \sum_{j \in Small \cap S_i} w_j x_j \leq t_i, \quad i = 1, 2, \dots, k$$

$$x_j \geq 0, \text{ integers}, j \in Small$$

Step 4. For each $p = 1, 2, \dots, l$, compute all $(U_p(r), Y_p(r))$, $r = 0, 1, \dots, z_p$. Here

1) $0 = U_p(0) < U_p(1) < \dots < U_p(z_p) \leq V^*/K \leq 4k(k+1)/\epsilon^2$

2) There exist non-negative integers x_j^* ($j \in T_p^L$), such that

(i) $\sum_{j \in T_p^L} u_j x_j^* = U_p(r)$

(ii) $\sum_{j \in T_p^L} w_j x_j^* = Y_p(r)$

(iii) $Y_p(r) = \min\{\sum_{j \in T_p^L} w_j x_j | \sum_{j \in T_p^L} u_j x_j = U_p(r), x_j \geq 0, \text{ integers}, j \in T_p^L\}$

Clearly, $z_p \leq 4k(k+1)/\epsilon^2 = O(1/\epsilon^2)$.

It is evident that we need to retain only one item for each u_j value for the computation of $(U_p(r), Y_p(r))$, i.e. one with minimum weight. In order to provide all possible p_j multiplicities of each such item, where $p_j = \lfloor V^*/(K \cdot u_j) \rfloor$, we provide $\lfloor \log_2 p_j \rfloor$ additional copies of the item by doubling. That is, let the i -th copy of the item j be such that

$$u_j^i = 2^i u_j, \quad w_j^i = 2^i w_j$$

Then we only need to retain the smallest-weight item, or the copy of an item, for each u_j value. Since $2^r |u_j$, for any $u_j \in ((2/\epsilon)2^r, (2/\epsilon)2^{r+1}]$, there are at most $O(1/\epsilon)$ distinct u_j on the interval $((2/\epsilon)2^r, (2/\epsilon)2^{r+1}]$. Therefore, there are at most $O((1/\epsilon) \log(1/\epsilon))$ items left for the computation of $(U_p(r), Y_p(r))$. The computation of $(U_p(r), Y_p(r))$ now proceeds by iteration over u_j values (or items), from the largest to the smallest. This is the traditional dynamic programming algorithm and can be carried out in $O(1/\epsilon^3)$ time and $O(1/\epsilon^2)$ space.

Here we only give a brief introduction to the computation of $(U_p(r), Y_p(r))$. Our method is the same as the one presented in [6]. Interested readers can refer to [6] for more details.

Step 5. For all possible (r_1, r_2, \dots, r_l) , $0 \leq r_p \leq z_p$, $p = 1, 2, \dots, l$,

1) examine if (r_1, r_2, \dots, r_l) violates any of the k constraints: $\sum_{p \in \Delta_i} Y_p(r_p) \leq b_i$ ($i = 1, 2, \dots, k$);

2) if none of the above k constraints is violated, then compute $U(r_1, r_2, \dots, r_l) = \sum_{p=1}^l U_p(r_p)$;

3) let $y_i(r_1, r_2, \dots, r_l)$ stand for $\sum_{p \in \Delta_i} Y_p(r_p)$ obtained in 1). Compute $G(r_1, r_2, \dots, r_l)$, where $G(r_1, r_2, \dots, r_l)$ is defined as GA $(\Phi(b_1 - y_1(r_1, r_2, \dots, r_l), \dots, b_k - y_k(r_1, r_2, \dots, r_l)))$.

Step 6. Find $(r_1^*, r_2^*, \dots, r_l^*)$ which maximizes $(K \cdot U(r_1, r_2, \dots, r_l) + G(r_1, r_2, \dots, r_l))$.

Based on the backtracing technique presented in [6], we can find non-negative integers x_j^* ($j \in T_p^L$) which satisfy:

$$(i) \sum_{j \in T_p^L} u_j x_j^* = U_p(r_p^*) \quad (ii) \sum_{j \in T_p^L} w_j x_j^* = Y_p(r_p^*)$$

This can be carried out in $O(1/\varepsilon^3)$ time and space. We omit once again the details, which can be found in [6]. Combine x_j^* ($j \in T_p^L, p = 1, 2, \dots, l$) with the approximate solution of $\Phi(b_1 - y_1(r_1^*, r_2^*, \dots, r_l^*), \dots, b_k - y_k(r_1^*, r_2^*, \dots, r_l^*))$ given by GA in Step 5. Output them as an approximate solution of I . It is evident that this can be done in $O((1/\varepsilon) \log(1/\varepsilon))$ time and space.

Theorem 3. Algorithm ZZ3 has time complexity $O(n + (1/\varepsilon^3) + 1/(\varepsilon^{2^{k+1}-2}))$ and space complexity $O(n + (1/\varepsilon^3))$, and finds an approximate solution to within ε of the optimal solution.

Proof. Our proof consists of 3 parts: error analysis, time complexity, and space complexity.

Error analysis. Let x_j^0 ($j = 1, 2, \dots, n$) be an optimal solution of I , and V^* stands for the value of the optimal solution. Let V_1 be the value of the solution given by ZZ3. Let $V_p^* = \sum_{j \in T_p^L} v_j x_j^0$, ($1 \leq p \leq l$). Let $V_{Small}^* = \sum_{j \in Small} v_j x_j^0$. Clearly, $V^* = V_{Small}^* + \sum_{p=1}^l V_p^*$.

Obviously, for any p ($1 \leq p \leq l$), there exists an integer r_p ($0 \leq r_p \leq z_p$), such that $U_p(r_p) = \sum_{j \in T_p^L} u_j x_j^0$. According to the definition of $Y_p(r_p)$, we have $Y_p(r_p) \leq \sum_{j \in T_p^L} w_j x_j^0$. Hence, $\sum_{j \in Small \cap S_i} w_j x_j^0 \leq b_i - \sum_{p \in \Delta_i} \sum_{j \in T_p^L} w_j x_j^0 \leq b_i - \sum_{p \in \Delta_i} Y_p(r_p) = b_i - y_i(r_1, \dots, r_l)$.

Therefore, x_j^0 ($j \in Small$) is a feasible solution of $\Phi(b_1 - y_1, \dots, b_k - y_k)$, where y_i is abbreviated from $y_i(r_1, \dots, r_l)$, $i = 1, 2, \dots, k$. It follows that

$$V_{Small}^* \leq \text{OPT}(\Phi(b_1 - y_1, \dots, b_k - y_k)) < G(r_1, \dots, r_l) + k \max_{j \in Small} v_j \\ \leq G(r_1, \dots, r_l) + k \cdot T = G(r_1, \dots, r_l) + k(\varepsilon \cdot V_0 / (2k)) \leq G(r_1, \dots, r_l) + \varepsilon \cdot V^* / 2$$

We have

$$V^* - V_1 \leq V^* - K \sum_{p=1}^l U_p(r_p) - G(r_1, \dots, r_l) \\ = \sum_{p=1}^l (V_p^* - K \cdot U_p(r_p)) + (V_{Small}^* - G(r_1, \dots, r_l)) \\ = \sum_{p=1}^l \left(\sum_{j \in T_p^L} x_j^0 (v_j - K \cdot u_j) \right) + (V_{Small}^* - G(r_1, \dots, r_l)) \\ < \sum_{p=1}^l \sum_{j \in T_p^L} (x_j^0 (v_j \varepsilon / 2)) + \varepsilon \cdot V^* / 2 = \varepsilon \cdot V^* / 2 + \varepsilon \cdot V^* / 2 = \varepsilon \cdot V^*$$

Time Complexity. Step 1 requires $O(n)$ time. In Step 2 we need $O(1)$ time to get V_0 and $O(n)$ time to get v_{\max}^S and T_p^L, T_p^S, q_p ($p = 1, 2, \dots, l$). Step 3 requires $O(n)$ time. Step 4 requires $O(1/\varepsilon^3)$ time. In Step 5, for each possible (r_1, \dots, r_l) , we need $O(k \cdot l) = O(1)$ time to examine if (r_1, r_2, \dots, r_l) violates any of the k constraints, and $O(l) = O(1)$ time to compute $U(r_1, r_2, \dots, r_l) = \sum_{p=1}^l U_p(r_p)$. Since Δ_i, T_p^S, q_p and v_{\max}^S have already been prepared in Step 1 and Step 2, we only need $O(1)$ time to compute $G(r_1, r_2, \dots, r_l)$. Altogether there are $\prod_{p=1}^l (1 + z_p) = (O(1/\varepsilon^2))^l = O(1/\varepsilon^{2^{k+1}-2})$ possible (r_1, r_2, \dots, r_l) , so Step 5 requires $O(1/\varepsilon^{2^{k+1}-2})$ time. Step 6 requires $O(1/\varepsilon^3)$ time. In summary, the time complexity for algorithm ZZ3 is bounded by $O(n + (1/\varepsilon^3) + (1/\varepsilon^{2^{k+1}-2}))$.

Space Complexity. Step 1 requires $O(n)$ space (including input space). In Step 2 we need $O(1)$ space to get V_0 and v_{\max}^S , $O(n)$ space to store T_p^L, T_p^S , and q_p ($p = 1, 2, \dots, l$). Step 3 requires $O(n)$ space. Step 4 requires $O(1/\varepsilon^2)$ space. In Step 5, for each possible (r_1, \dots, r_l) , the operation requires $O(1)$ space. Since at any time, we only need to keep the currently optimal (r_1, \dots, r_l) , Step 5 requires $2 \cdot O(1) = O(1)$ space. Step 6 requires $O(1/\varepsilon^3)$ space. In summary, the space complexity for algorithm ZZ3 is bounded by $O(n + 1/\varepsilon^3)$.

Now, we have completed our proof.

Note. For $k = 1$, the time complexity and space complexity of ZZ3 become $O(n + 1/\varepsilon^3)$, which are the same as the results for the unbounded knapsack problem achieved in [6].

6 An Open Problem

In this paper, we have succeeded in developing a fast FPTAS for Integer k -CKP. Here, by "fast", we mean that the algorithm is a linear-time algorithm for any fixed ε . (In fact, the space complexity of our algorithm is also linear to the length of encoded input for any fixed ε .) An open problem is as follows: Can we find a fast FPTAS for 0/1 k -CKP? It seems that the major difficulty here lies in how to find an efficient greedy algorithm for 0/1 k -CKP.

References

- [1] Chandra A K, Hirschberg D S, Wong C K. Approximate algorithms for some generalized knapsack problems. *Theoret. Comput. Sci.*, 1976, 3: 293–304.
- [2] Magazine M J, Oguz O. A fully polynomial approximation algorithm for the 0-1 knapsack problem. *European Journal of Operational Research*, 1981, 8: 270–273.
- [3] Gens G V, Levner E V. Computational complexity of approximation algorithms for combinatorial problems. In *Proc. 8th International Symposium on Mathematical Foundations of Computer Science. Lecture Notes in Computer Science 74*, 1979, pp.292–300.
- [4] Sahni S. Approximate algorithms for the 0/1 knapsack problem. *J. ACM*, 1975, 22: 115–124.
- [5] Ibarra O H, Kim C E. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 1975, 22: 463–468.
- [6] Lawler E L. Fast approximation algorithms for knapsack problems. In *Proc. 18th Annual IEEE Symp. Found. Comp. Science*, 1977, pp.206–213.
- [7] Zhang Li'ang. The complexity of approximation for k -KNAPSACK. In *Proc. International Workshop on Discrete Mathematics and Algorithms*, Su Yunlin (ed.), pp.177–180, Jinan University Press, Guangzhou, 1994.
- [8] Zhang Li'ang, Li Luyang, Huang Xiong. Approximation for multi-knapsack problem. *Chinese Science Bulletin*, 1996, 41: 1042–1045.
- [9] Garey M R, Johnson D S. Strong NP-completeness results: Motivation, examples and implications. *J. ACM*, 1978, 25: 499–508.
- [10] Håstad J. Clique is hard to approximate within $n^{1-\varepsilon}$. In *Proc. 37th Annual IEEE Symp. Found. Comp. Science*, 1996, pp.627–636.

ZHANG Li'ang graduated from Department of Mathematics at Peking University in 1965. He is now a Professor of Department of Computer Science and Technology at Peking University. His research interests include computational complexity and approximation for NP-hard problems.

ZHANG Yin received his B.S. degree from Department of Computer Science and Technology at Peking University in 1997. He is now a Ph.D. candidate in Department of Computer Science, Cornell University.