Annals of University of Craiova, Math. Comp. Sci. Ser.
Volume 30, 2003, Pages 162–168
ISSN: 1223-6934

The packing knapsack problem, and the k-optimal tree in a weighted digraph

Dumitru Lozovanu

Abstract. We consider the packing knapsack problem, which generalizes the classical knapsack problem. A dynamic programming algorithm for solving this problem and its application to find the k-optimal tree in a weighted directed graph are discussed.

2000 Mathematics Subject Classification. 90B10, 90C35, 90C37.
Key words and phrases. packing knapsack problem, k-optimal tree, dynamic programming method.

1. Introduction and problem formulation

We study the packing knapsack problem which arises when finding the k-optimal tree in a weighted graph $[1-4]$. This problem generalizes the well known classical knapsack problem. Its formulation is as follows.

Let a knapsack of size D and a set of items $I = \{1, 2, \ldots, n\}$ be given. For each item $j \in I$, the size d_j and the cost c_j are known. Moreover, the set I is divided into p non-empty disjoint subsets $I_1, I_2, \ldots, I_p$,

$$I = I_1 \cup I_2 \cup \cdots \cup I_p;\ I_i \cap I_j \neq 0,\ i \neq j,$$

and, according to this partition, from each of the subsets I_l, $l = \overline{1,p}$, we include in the knapsack at most one item.

A sequence of items $i_1, i_2, \ldots, i_q$ $(q < p)$ from I is called a packing in the knapsack if the sum of item sizes of the sequence does not exceed D and each of the subsets I_l, $l = \overline{1,p}$, contains at most one item of the sequence.

We consider the problem of finding the packing in the knapsack with maximal sum of items costs.

This problem can be viewed as a Boolean linear programming model, which in the case $p = n$ becomes the classical knapsack problem. Therefore it is NP-complete. We propose a dynamic programming algorithm for solving our problem, which has a pseudo-polynomial estimation. We show that the proposed algorithm can be used for finding the k-optimal tree in a weighted directed graph.

A more general model of the packing knapsack problem is obtained if we consider that from each of the subsets I_l, $l = \overline{1,p}$, we include in the knapsack at most k_l items. The algorithm can be extended for the packing knapsack problem in this general form. Using the general model of the problem we can solve the k-optimal problem when any node of the k-optimal tree contains at most k_j leaving arcs.

Received: 11 December 2002.

2. The dynamic programming method and the algorithm for solving the problem

The general scheme of dynamic programming method for the packing knapsack problem is based on the following Boolean programming model: maximize the object function

$$\mu_I(x) = \sum_{j \in I} c_j x_j$$

subject to

$$\begin{cases} \sum\limits_{j \in I} d_j x_j \le D; \\ \sum\limits_{j \in I_l} x_j \le 1, \quad l = \overline{1,p}; \\ x_j \in \{0,1\}, \quad j \in I, \end{cases}$$

where $x = (x_1, x_2, \ldots, x_m)$. Here $x_j = 1$ if the item j is included in the knapsack; otherwise $x_j = 0$. As we have noted, in the case $p = n$ the problem becomes the classical knapsack problem, therefore it is NP-complete.

We consider the class of problems

$$f_I(z) = \max_{x \in X_z} \mu_I(x), \; z \in [0, D],$$

where X_z is the set of vectors $x = (x_1, x_2, \ldots, x_n)$ which satisfy the conditions:

$$\begin{cases} \sum\limits_{j \in I} d_j x_j \le z; \\ \sum\limits_{j \in I_l} x_j \le 1, \quad l = \overline{1,p}; \\ x_j \in \{0,1\}, \quad j \in I. \end{cases}$$

Let $z_0 = \min d_i$. Then for $z \in [0, z_0)$ the set X_z contains only one element $x = (0, 0, \ldots, 0)$ and $f_I(z) = 0$. Moreover, if $z \in [0, z_0)$, then $f_{I'}(z) = 0$ for any $I' \subseteq I$. If $z \in [z_0, D)$, then for an arbitrary $I' \subseteq I$ the recurrence formula

$$f_{I'}(z) = \max_{j \in I'_z} [c_i + f_{I' \setminus \{j\}}(z - d_j)] \tag{1}$$

holds, where I'_z is a set of items $j \in I$ for which $d_j \le z$.

We propose an algorithm for solving the packing knapsack problem which can be argued by using formula (1).

For a more detailed argumentation of the algorithm we shall use the results from [5] for the classical knapsack problem.

Algorithm 1.

```
1. Set $M_0 = \{(\emptyset, 0)\}$.
2. For $j = 1, 2, \ldots, n$ do steps a), b) and c):
   a) Set $M_j = \emptyset$;
   b) Add each element $(S, c) \in M_{j-1}$ to $M_j$. Then for each $(S, c) \in M_{j-1}$ consider
the element $(S \cup \{j\}, c + c_j)$ if $\sum_{i \in S} d_i + d_j \le D$, $|I_l \cap S| \le 1, l = 1, 2, \ldots, p$, and add
$(S \cup \{j\}, c + c_j)$ to $M_j$;
```

c) Find in M_j elements (S, c) and (S', c') with the same second components. For each pair (S, c) and (S', c) with the same second component, delete (S', c) from M_j if $\sum\limits_{i \in S'} d_i \geq \sum\limits_{i \in S} d_i$; otherwise delete (S, c).

3. Find in M_n the element (S, c) with maximal second component.

Then (S, c) is a solution of the packing knapsack problem.

Algorithm 1 is an extension of the algorithm for solving classical knapsack problem from [5]. Therefore the correctness and the computational complexity of the algorithm can be argued in an analogous way.

Theorem 2.1. *Algorithm 1 finds the optimal solution of the packing knapsack problem in time $O(n^2 c)$, where $c = \sum\limits_{j \in I} c_j$.*

To prove this theorem we need the following result.

Lemma 2.1. *Let be $(S, c) \in M_j$ after running the algorithm. Then*

a) $S \subseteq \{1, 2, \dots, j\}$*;*

b) $\sum\limits_{i \in S} c_i = c$*;*

c) $\sum\limits_{i \in S} d_i \leq D$*;*

d) $|I_l \cap S| \leq 1, l = 1, 2, \dots, p$*;*

e) if $(S', c) \in M_j$ *then* $S' = S$*;*

g) if $S' \subseteq \{1, 2, \dots, j\}$ *and* $\sum\limits_{i \in S'} c_i = c$, $|I_l \cap S'| \leq 1, l = 1, 2, \dots, p$, *then* $\sum\limits_{i \in S} d_i \leq \sum\limits_{i \in S'} d_i$*;*

f) moreover, if $S \subseteq \{1, 2, \dots, j\}$ *and* $\sum\limits_{i \in S'} d_i \leq D$, $\sum\limits_{i \in S} c_i = c$, $|I_l \cap S'| \leq 1$, $l = 1, 2, \dots, p$, *then there exists* $(S', c) \in M_j$.

Proof. We use induction on the iteration number j. The statement for $j = 0$ is evident. Now we consider that $j > 0$ and $(s, c) \in M_j$. There may be two cases:

Case 1. $j \notin S$. Then the element (S, c) has been transferred from M_{j-1} in M_j at step 2(b). Therefore properties a), b), c), d) and e) follow from the induction principle.

Case 2. $j \in S$. Then $(S \backslash \{j\}, c - c_j) \in M_{j-1}$ and properties a), b), c), d) and e) hold too.

Now let us prove f). Suppose that $S \neq S'$ and analyze the following three cases:

Case 1. $j \notin S, S'$. Then (S, c) and $(S', c) \in M_{j-1}$. So, according to the induction principle, $S = S'$.

Case 2. $j \in S, S'$. Then $(S \backslash \{j\}, c - c_j)$, $(S' \backslash \{j\}, c - c_j) \in M_{j-1}$. Therefore $S = S'$.

Case 3. $j \in S, j \notin S'$ or $j \notin S, j \in S'$. Then (S', c) was eliminated at the step 2(c). Therefore f) holds.

Now let us prove property g). We shall use induction on the number $k = \max S$. This property holds for $S = \oslash$. Let us consider $k = \max S > 0$. Then according to the induction principle in M_{k-1} there exists an element $(S \backslash \{k\}, c - c_k) = (S', c - c_k)$. Therefore the element $(S' \cup \{k\}, c)$ has been added to M_k at step 2(b). Then either $(S' \cup \{k\}, c) \in M_j$. So, property f) holds. □

Proof of Theorem 2.1(sketch). The correctness of Algorithm 1 follows from Lemma 2.1. The algorithm has the same computational complexity as algorithm DP.III from [5]. Therefore the algorithm finds the optimal solution in time $O(N^2 c)$. □

3. The algorithm for solving the packing knapsack problem in the general case

Algorithm 1 can be extended to solve the packing knapsack problem in general form if we use the Boolean programming model: maximize the object function

$$\mu(x) = \sum_{j\in I} c_j x_j$$

subject to

$$\begin{cases} \sum\limits_{j\in I} d_j x_j \le D; \\ \sum\limits_{j\in I_l} x_j \le k_l, \quad l = \overline{1,p}; \\ x_j \in \{0,1\}, \ j \in I. \end{cases}$$

Algorithm 2.

```
1. Set $M_0 = \{(\emptyset, 0)\}$.
2. For $j = 1, 2, \ldots, n$ do steps a), b) and c):
   a) Set $M_j = \emptyset$;
   b) Add each element $(S, c) \in M_{j-1}$ to $M_j$. Then for each $(S, c) \in M_{j-1}$ consider
      the element $(S \cup \{j\}, c + c_j)$ if $\sum_{i\in S} d_i + d_j \le D$, $|I_l \cap S| \le K_l, l = 1, 2, \ldots, p$, and add
      $(S \cup \{j\}, c + c_j)$ to $M_j$;
   c) Find in $M_j$ elements $(S, c)$ and $(S', c')$ with the same second components.
      For each pair $(S, c)$ and $(S', c)$ with the same second component delete $(S', c)$ from
      $M_j$ if $\sum_{i\in S'} d_i \ge \sum_{i\in S} d_i$; otherwise delete $(S, c)$.
3. Find in $M_n$ the element $(S, c)$ with maximal second component.
Then $(S, c)$ is a solution of the packing knapsack problem.
```

Algorithm 2 is an extension of the algorithm for splving the classical knapsack problem from [5]. Therefore the correctness and the computational complexity of the algorithm can be argued in an analogous way.

4. The k-optimal tree problem in a weighted digraph and its main properties

In [2–4] the following problem was formulated and studied.

Let $G = (V, E)$ be a directed graph with root vertex $v_0 \in V$. To each arc $e \in E$ a positive real number $c(e)$ is assigned, which we call the cost of e. We should find in G a directed tree $T^* = (V_{T^*}, E_{T^*})$ with root vertex v_0 which contains at most k arcs and has maximal sum of arcs costs, i.e.

$$\sum_{e\in E_{T^*}} c(e) = \max_{T\subset G, |E_T|\le k} \sum_{e\in E_T} c(e).$$

This problem is NP-complete [2, 3]. Some approximation algorithms and some exact polynomial-time algorithms for a special class of k-optimal tree problems can be found in [1–4]. We study this problem in the case when G has the structure of a directed tree.

First we show that the considered problem is closely connected with the classical knapsack problem.

Let us consider the classical knapsack problem with a given knapsack's size D and a given set of items $I = \{1, 2, \dots, n\}$, where for each item i from I the size d_i and the cost c_i are known. We can formulate this problem as a D-optimal tree problem for the weighted directed tree $G = (V, E)$ with $|V| = \sum\limits_{i \in I} d_i$ vertices, as follows. The tree $G = (V, E)$ is the union of n directed paths $P_1, P_2, \dots, P_n$ which have only one common root vertex v_0 and each path P_i contains d_i vertices. So,

$$G = P_1 \cup P_2 \cup \dots \cup P_m,$$

where

$$\begin{gathered} P_1 = [v_0, e_1^1, v_1^1, e_2^1, v_2^1, \dots, e_{d_1}^1, v_{d_1}^1]; \\ P_2 = [v_0, e_1^2, v_1^2, e_2^2, v_2^2, \dots, e_{d_2}^2, v_{d_2}^2]; \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ P_n = [v_0, e_1^n, v_1^n, e_2^n, v_2^n, \dots, e_{d_n}^n, v_{d_n}^n]. \end{gathered}$$

We define the cost of the arcs $e_j^i \in E_i$ in G as

$$c(e_{d_j}^j) = c_j,\ j = \overline{1, n};\ c(e_i^j) = 0,\ i = \overline{1, d_j - 1},\ j = \overline{1, n}.$$

Further, the directed tree $G = (V, E)$ obtained this way will be called the directed tree of the knapsack problem.

Theorem 4.1. *Let $T^* = (V_{T^*}, E_{T^*})$ be the D-optimal tree for the directed tree $G = (V, E)$ of the knapsack problem. In addition, let $E_{T^*}^0$ represent the set of pendant arcs of T^* and E^0 represent the set of pendant arcs of G. Then the set $S = \{j \in I \mid e_{d_j}^j \in E_{T^*}^0 \cap E^0\}$ is an optimal solution of the knapsack problem.*

Proof. Let $T^* = (V_{T^*}, E_{T^*})$ be the D-optimal tree for the directed tree $G = (V, E)$. Then $\sum\limits_{i \in S} d_i \le D$, i.e. S is an admissible solution for the knapsack problem.

Let us show that S is the optimal solution of the problem. Indeed, if it is not, then there exists a set S', such that

$$\sum_{i \in S'} d_i \le D; \quad \sum_{i \in S'} c_i > \sum_{i \in S} c_i.$$

This means that for the tree $G_S = \bigcup\limits_{i \in S} P_i$ in G the following conditions hold:

1) $G_{S'}$ contains no more than D arcs;
2) $\sum\limits_{e \in E_{G_{S'}}} c(e) > \sum\limits_{e \in E_{T^*}} c(e)$.

This contradicts the D-optimality of the tree T^*. So, S is the optimal solution of the knapsack problem. □

5. Using the packing knapsack problem to find the k-optimal tree in a weighed digraph

In this section we show that the packing knapsack problem can be used to solve the k-optimal tree problem.

So, let us consider that G has the structure of a directed tree with root vertex v_0. Denote by $v_1, v_2, \dots, v_p$ the vertices of G which represent the extremities of the arcs $e_1 = (v_0, v_1)$, $e_2 = (v_0, v_2), \dots, e_p = (v_0, v_p)$ originating in v_0. Let $G^1 = (V^1, E^1)$, $G^2 = (V^2, E^2), \dots, G^p = (V^p, E^p)$ denote the components of the subgraph $G' \in G$, obtained after deleting the edges $e_1, e_2, \dots, e_p$ where

$v_i \in V^i, i = \overline{1,p}$. It is easy to see that each component G^i, $i = \overline{1,p}$, is a directed tree with root vertex v_i and $\overline{G}^i = (V^i \cup \{v_0\}, E^i \cup \{e_i\})$ is a subtree of G with root vertex v_0. We call $\overline{G}^i$, $i = \overline{1,p}$, the branches of the tree G.

Let us consider that for each branch $\overline{G}^i, i = \overline{1,p}$, all t-optimal trees, $t = 1, 2, \dots, r_i$, are known, where $r_i = \max(|V^i \cup \{v_0\}|, k)$. Then for each subtree G^i, $i = \overline{1,p}$, all $(t-1)$-optimal trees, $t = 1, 2, \dots, r_i$, are known because G^i is obtained from $\overline{G}^i$ by deleting the arc e_i. Denote by $c_{t^i} = c_t(\overline{G}^i)$ the cost of the t-optimal tree for a branch G^i, $i = \overline{1,p}$, $t = \overline{1,r_i}$.

Now, let us show that if for each branch $\overline{G}^i$, $i = \overline{1,p}$, all t-optimal trees $t = 1, 2, \dots, r_i$ are known, then we can find the k-optimal tree in G using Algorithm 1.

We consider the packing knapsack problem with the set of items $I = \{1_1, 2_1, \dots \dots, r_1, 1_2, 2_2, \dots, r_2, \dots, 1_p, 2_p, \dots, r_p\}$ and partition $I = I_1 \cup I_2 \cup \dots \cdots \cup I_p$, where $I_i = \{1_i, 2_i, \dots, r_i\}$, $i = \overline{1,p}$. We define the size and the cost of the item $j_i \in I_i$ as $d_{j_i} = j$ and $c_{j_i} = c_j(\overline{G}^i)$ for every $i = \overline{1,p}$. The Boolean programming model for this packing knapsack problem can be written as follows: maximize

$$\mu(x) = \sum_{i=1}^{p} \sum_{j=1}^{r_i} c_{ji} x_{ji} \tag{2}$$

subject to

$$\begin{cases} \sum_{i=1}^{p} \sum_{j=1}^{r_i} j x_{ji} \le k; \\ \sum_{j=1}^{r_i} x_{ji} \le 1, \ i = \overline{1,p}; \\ x_j \in \{0,1\}, \ j = \overline{1,r_i}, \ i = \overline{1,p}. \end{cases} \tag{3}$$

Note that for solving the k-optimal tree problem on a directed tree G we should find all possible t-optimal trees $(t < k)$ with respect to each directed subtree G^i arising from the vertices $v_i \in V$. Therefore, starting from the vertices v_i with the highest level in G we should solve at least $n \cdot k$ problems of type (2), (3).

In the general case when G has the structure of an arbitrary directed graph we can solve the problem in two steps. First, we find the optimal spanning directed tree with root vertex v_i and then we solve the k-optimal tree problem on the optimal spanning tree by using the proposed algorithm. Note that for finding the optimal spanning tree with root vertex in a weighted directed graph, the algorithms from [6, 7] can be used.

References

[1] D. Hochbaum (editor), *Approximation algorithms for NP-hard problems*, PWS Publishing Company, 1997.

[2] D. Lozovanu, A. Zelikovsky, Minimal and bounded tree problems, *Rezumatele lucrărilor prezentate la Congresul XVIII al Academiei Româno-Americane, Chişinău*, 25-26 (1993).

[3] R. Ravi, R. Sundaram, M. Marathe, D. Rozenkrantz, S.S. Ravi, Spanning tree short or small, *The 5 th Ann. ACM-SIAM Symp. on Discrete Algorithms*, 546-555 (1994).

[4] A. Blum, P. Chalasani, A. Vempalo, A constant-factor approximation for the k-MST problem in the plane, *Proceedings of the 27 th Ann. ASM Symp. on Theory of Computing*, 294-302 (1995).

[5] Ch.H. Papadimitriu, K. Steiglitz, *Combinatorial optimization: Algorithm and complexity*, New Jersey, 1982.

[6] D. Fulkerson, Packing rooted directed cuts in a weighted directed graph, *Math. Programming*, **6**, 1-13 (1974).
[7] D. Lozovanu, Optimal subgraphs in a weighted digraph, *Cybernetics*, **2**, 16-19 (1981).

(Dumitru Lozovanu) Institute of Mathematics and Computer Science
Academy of Sciences
Academy str., 5, Kishinev, MD–2028, Moldova
E-mail address: lozovanu@math.md