From Knapsack to NP = P

T. C. Hu[1], M.T. Shing[2], Leo Landa[3]

Abstract

A Knapsack problem is to select among n types of items of various values and weights to fill a knapsack of weight-carrying capacity b. The problem is NP-complete and has a dynamic programming algorithm of time complexity O(n b). If the given b is large, the optimum solution is periodic with its period equals to the weight of the best item. We introduce a new $O(n^2 w_1 \log (n w_1))$ algorithm that solves all instances b of the problem for a given set of items, where w_1 is the weight of the best item and is usually much smaller than b. We conjecture that there are many other NP-complete number problems whose optimum solutions also become periodic for large instances and hence can be solved by similar algorithms.

Keywords: Knapsack, dynamic programming, shortest-path, NP, pseudo-polynomial

[1] Department of Computer Science and Engineering, UC San Diego, 9500 Gilman Dr., La Jolla, CA 92093-0114. Email: hu@cs.ucsd.edu
[2] Department of Computer Science, Naval Postgraduate School, Code CS/Sh, 833 Dyer Road, Monterey, CA 93943-5118. Email: shing@nps.navy.mil
[3] Department of Computer Science and Engineering, UC San Diego, 9500 Gilman Dr., La Jolla, CA 92093-0114. Email: leo@leolan.com

1. Introduction and Review

A knapsack problem is to select a subset of items among n types so that the total value of the selection is maximized while the total weight does not exceed the knapsack capacity b. Mathematically,

$$\text{Max} \quad x_0 = v_1 x_1 + v_2 x_2 + \ldots + v_n x_n \tag{1.1}$$

$$\text{Subject to} \quad w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \le b$$

$$x_j \ge 0, \text{ integers}$$

where the weights w_j (j = 1, …, n) and the values v_j are given positive integers, and the capacity b could be any integral value 1, 2, …, M, for some integer M.

When $v_n = 0$ and $w_n = 1$, there is always an optimum solution for (1.1) of the form $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = b$. For the rest of the paper, we shall assume that $v_n = 0$ and $w_n = 1$ and rewrite the problem as

$$\text{Max} \quad x_0 = v_1 x_1 + v_2 x_2 + \ldots + v_n x_n \tag{1.2}$$

$$\text{Subject to} \quad w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = b$$

$$x_j \ge 0, \; w_j > 0, \text{ integers}$$

Denote the ratio v_j / w_j as ρ_j (called the density of type j) and adopt the convention that

$$\rho_1 \ge \rho_2 \ge \rho_3 \ge \ldots \ge \rho_n. \tag{1.3}$$

In case of tie, i.e. $v_j / w_j = v_{j+1} / w_{j+1}$, we let $w_j < w_{j+1}$. We call the first type, the best item.

An optimum solution to (1.2) for an arbitrary b will be denoted by an n-component vector

$$x_0(b) = (x_1, x_2, \ldots, x_n). \tag{1.4}$$

When there are two or more optimum solution vectors, we will choose the lexicographically largest vector.

The knapsack problem (1.1) is normally solved by a pseudo-polynomial algorithm as follows.

(1) Define $F_k(y)$ to be the maximum value when the capacity is y (y = 0, 1, 2, … b) and only the first k items can be selected (k = 1, 2, …, n).

(2) From the boundary conditions

$$F_1(y) = v_1 \lfloor y / w_1 \rfloor \text{ for } 1 \le y \le b, \tag{1.5}$$

$$F_k(0) = 0 \text{ for } 1 \le k \le n,$$

and $F_k(y) = -\infty$ for $y < 0$,

we can build a table of n rows and b+1 columns, using the recursive relations (1.6) below.

$$F_1(y) = v_1 \lfloor y / w_1 \rfloor$$

$$F_{k+1}(y) = \max \{ F_k(y), v_{k+1} + F_{k+1}(y - w_{k+1}) \} \text{ for } 1 \le k < n \tag{1.6}$$

When the problem (1.1) has no integer restriction on its variables, it is called a fractional knapsack problem (a linear program with a single constraint). The optimum solution for the fractional knapsack problem is given by

$$x_1 = b / w_1, \tag{1.7}$$

$$x_j = 0 \text{ for } 2 \le j \le n,$$

and $\max x_o = \rho_1 b$.

It is shown in [5-7] that the optimum knapsack solutions are periodic when the capacity b exceeds a critical value (denoted by b**) and the period is w_1, the weight of the best item.

Define the difference function $\theta(b)$ where

$$\theta(b) = \rho_1 b - F_n(b) \tag{1.8}$$

It was shown by Gilmore and Gomory [6] that the difference function $\theta(b)$ is periodic with its period equal to w_1 when b satisfies the sufficient condition (1.9).

$$b \ge \rho_1 w_1 / (\rho_1 - \rho_2) \tag{1.9}$$

The function $\theta(b)$ can be calculated recursively from the boundary condition $\theta(b) = 0$ and

$$\theta(b) = \min_j \{\theta(b - w_j) + (\rho_1 - \rho_j)\, w_j \mid 1 \le j \le n\} \qquad (1.10)$$

When $v_1 = w_1$, we have $\rho_1 = 1$ and the value of $\theta(b) = b - F_n(b)$ can be easily visualized as the unfilled space in the knapsack. Hence, we shall call the value of $\theta(b)$ simply as the gap. Conceptually, it is easier to deal with the case $\rho_1 = 1$; and we can always transform a given knapsack problem into an equivalent one with $\rho_1 = 1$, by replacing the values v_j with v_j', where

$$v_j' = v_j / (v_1 / w_1). \qquad (1.11)$$

For the rest of the paper, we shall assume that

$$v_1 = w_1 \text{ and } \rho_1 = 1 \ge \rho_2 \ge \rho_3 \ge \ldots \ge \rho_n = 0. \qquad (1.12)$$

2. Goals, Concepts and Definitions

Consider the following knapsack problem

$$\text{Max} \quad x_0 = 10\,x_1 + 11\,x_2 + 5\,x_3 + 0\,x_4 \qquad (2.1)$$

$$\text{Subject to} \quad 10\,x_1 + 12\,x_2 + 7\,x_3 + 1\,x_4 = b$$

$$x_j \geq 0, \text{integers}$$

The corresponding densities for the four types of items are

$\rho_1 = (10/10)$, $\rho_2 = (11/12)$, $\rho_3 = (5/7)$ and $\rho_4 = (0/1)$.

We shall first construct a table with 10 rows and 10 columns as shown in Table 2.1.

Track:	0	1	2	3	4	5	6	7	8	9
b	0	1	2	3	4	5	6	7	8	9
	10	11	12	13	14	15	16	17	18	19
	20	21	22	23	24	25	26	27	28	29
	30	31	32	33	34	35	36	37	38	39
	40	41	42	43	44	45	46	47	48	49
	50	51	52	53	54	55	56	57	58	59
	60	61	62	63	64	65	66	67	68	69
	70	71	72	73	74	75	76	77	78	79
	80	81	82	83	84	85	86	87	88	89
	90	91	92	93	94	95	96	97	98	99

Table 2.1

Each column is called a track. The leftmost column is called the 0^{th} track and the rightmost column is called the 9^{th} track. Thus, the numbers 2, 12, 22, …, 92 belong to the 2^{nd} track. There are 10 rows with the entries representing the various values of the capacity b, for b = 0, 1, 2, …, 99.

For a given b, say b = 34, we could fill the knapsack with

$2\,w_1 + 2\,w_3 = 20 + 14 = 34$ with a total value of $20 + 10 = 30$,

or $w_1 + 2\,w_2 = 10 + 24 = 34$ with a total value of $10 + 22 = 32$.

We choose the latter solution since we want the optimum solution for every b.

For the sequence of b = 4, 14, 24, 34, 44, 54, … in the 4th track, we can fill the knapsacks optimally with

$4\ w_4 = 4$ with a total value of 0,
$w_2 + 2\ w_4 = 14$ with a total value of 11,
$2\ w_2 = 24$ with a total value of 22,
$w_1 + 2\ w_2 = 34$ with a total value of 32,
$2\ w_1 + 2\ w_2 = 44$ with a total value of 42,
$3\ w_1 + 2\ w_2 = 54$ with a total value of 52,
etc.

We have two important observations:

(1) Once we find $x_o(24)$, the optimum solution for b = 24, we can fill any knapsack with b > 24 in the 4th track optimally by adding $((b - 24) / w_1)$ copies of w_1 to $x_o(24)$. In other words, the periodic solution starts at 24 for the sequence of b in the 4th track.

(2) The gap, $\theta(b)$, remains constant (with a minimum value 2) for all $b \geq 24$ in the 4th track.

We shall use b*(r) to denote the capacity b where the optimum periodic solution starts for the sequence of values satisfying $b \pmod{w_1} = r$, for r = 1, 2, … , w_1-1. We call b*(r) the critical value of the r^{th} track and define the global critical value, b**, as

$$b^{**} = \max_r b^*(r). \qquad (2.2)$$

We plan to accomplish the following 4 goals in this paper:

(1) Present a better way to find the critical values b*(r) for every track, without using (1.10);

(2) Present a better way to find $x_o(b)$ for all b without using the pseudo polynomial algorithm based on (1.5) and (1.6);

(3) Show that the complexity of the algorithm is polynomial for large instances of b, and exponential for small instances of b;

(4) Present a list of open problems that can further improve our understanding of the knapsack problem, and discuss the implications of the results.

3. Minimum Cost Multi-Track Spanning Tree

The non-negative integers 0, 1, 2, … form a semigroup under the binary operation (addition mod w_1). For the knapsack problem

$$\text{Max} \quad x_0 = 10\,x_1 + 11\,x_2 + 5\,x_3 + 0\,x_4 \tag{3.1}$$

$$\text{Subject to} \quad 10\,x_1 + 12\,x_2 + 7\,x_3 + 1\,x_4 = b$$

$$x_j \geq 0, \text{integers}$$

we can consider

the integer 12 as the group element g_2 with a cost of (12–11) = 1,

the integer 7 as the group element g_7 with a cost of (7–5) = 2,

and the integer 1 as the group element g_1 with a cost of (1–0) = 1.

To find b*(r), for $1 \leq r \leq 9$, we want to solve the congruence relation

$$\text{Min} \quad \text{gap} = 1\,x_2 + 2\,x_3 + 1\,x_4 \tag{3.2}$$

$$\text{Subject to} \quad 12\,x_2 + 7\,x_3 + 1\,x_4 \equiv r \pmod{10}$$

$$x_j \geq 0, \text{integers}$$

When we only use the first type of items ($w_1 = 10$), we can solve all capacities in the 0^{th} track optimally. We say that the 0^{th} track is covered. When we only use the first and second types of items ($w_1 = 10$, $w_2 = 12$), we can generate the group elements g_2, g_4, g_6 and g_8 from g_2. We say that the tracks 0, 2, 4, 6 and 8 are covered. We want to find a linear combination of g_2, g_7 and g_1 to cover all tracks while minimizing the resultant gaps.

Let us formulate this as a graph problem where every capacity b in Table 2.1 is considered as a node of a network. Two nodes b and b' are connected by a directed arc of length w_j (w_j = 12, 7 and 1) from b to b' if the difference $b' - b = w_j$. A directed arc of length w_j is associated with a cost = $(w_j - v_j)$. Here, the arc of length 12 has a cost of 1, the arc of length 7 has a cost of 2, and the arc of length of 1 has a cost of 1. For brevity, we shall call "the arc with length w_j" simply as "the arc w_j".

Each time we use the arc w_2, we will incur a gap cost of 1. Similarly, we will incur a gap cost of 2 for each arc w_3, and a gap cost of 1 for each arc w_4.

Thus, the graph version of (3.1) becomes finding a spanning tree with its root at node 0 and directed paths to nine nodes, each node in a different track, such that the total gap cost of the tree arcs is minimum. Figure 3.1 shows the resultant minimum cost spanning tree for (3.1), with

$b^*(0) = 0$, $b^*(1) = 1$, $b^*(2) = 12$, $b^*(3) = 13$, $b^*(4) = 24$,

$b^*(5) = 25$, $b^*(6) = 36$, $b^*(7) = 7$, $b^*(8) = 8$, $b^*(9) = 19$.

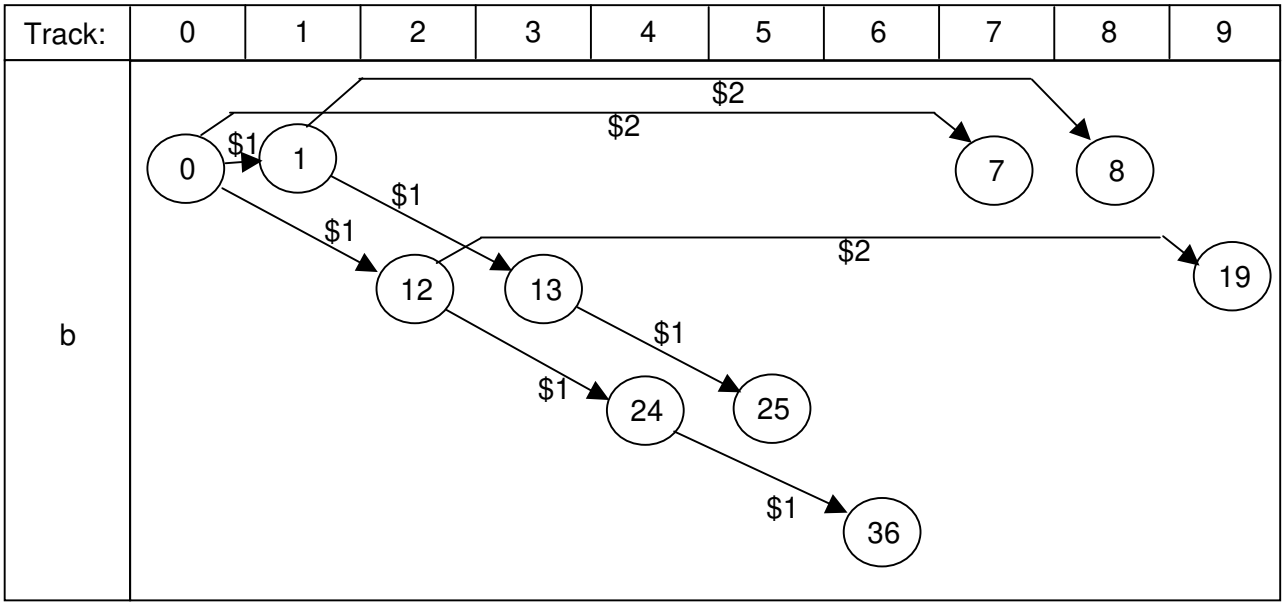

Figure 3.1

4. Numerical Examples

We give a sequence of numerical examples to show the intrinsic details of the knapsack problem before formally introducing the general algorithm.

Example 4.1

Max $x_0 = 10\,x_1 + 13\,x_2 + 17\,x_3 + 0\,x_4$ (4.1)

Subject to $10\,x_1 + 13\,x_2 + 17\,x_3 + 1\,x_4 = b$

$x_j \geq 0$, integers

To find b*(r), for $1 \leq r \leq 9$, we want to solve the congruence relation

Min $gap = 0\,x_2 + 0\,x_3 + 1\,x_4$ (4.2)

Subject to $13\,x_2 + 17\,x_3 + 1\,x_4 \equiv r \pmod{10}$

$x_j \geq 0$, integers

The resultant minimum cost spanning tree is shown in Figure 4.1.

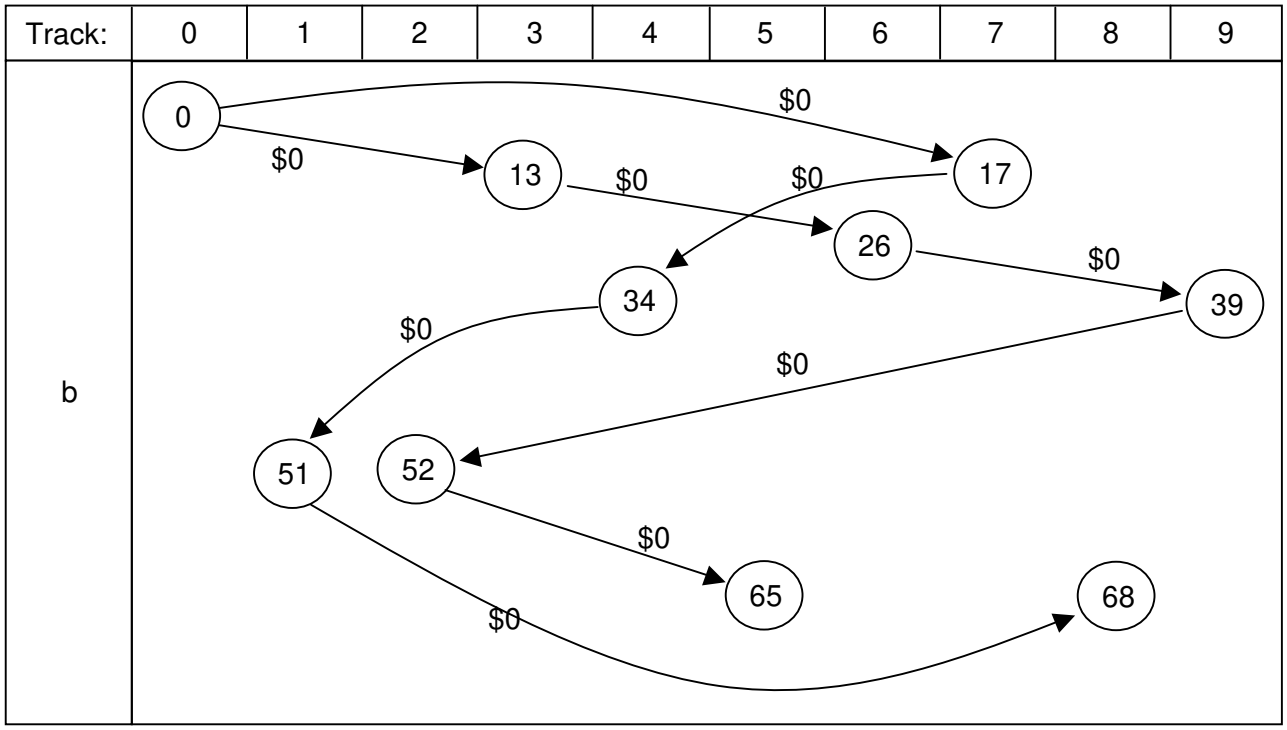

Figure 4.1

Since GCD(w_1, w_2), the greatest common divisor between w_1 and w_2, equals 1 and GCD(w_1, w_3) = 1, we can generate the group elements g_2, g_3, …, g_9 from g_3 or g_7 alone. Moreover, since $\rho_1 = \rho_2 = \rho_3 = 1$, the gap cost = 0 for all tree arcs corresponding to w_2 or w_3. We have

$b^*(0) = 0$, $b^*(1) = 51$, $b^*(2) = 52$, $b^*(3) = 13$, $b^*(4) = 34$,

$b^*(5) = 65$, $b^*(6) = 26$, $b^*(7) = 17$, $b^*(8) = 68$, $b^*(9) = 39$.

Example 4.2

$$\text{Max} \quad x_0 = 10\,x_1 + 12\,x_2 + 15\,x_3 + 0\,x_4 \tag{4.3}$$

$$\text{Subject to} \quad 10\,x_1 + 12\,x_2 + 15\,x_3 + 1\,x_4 = b$$

$$x_j \geq 0, \text{ integers}$$

Here, the group of order 10 is decomposed into two subgroups, one group of order 2 and one group of order 5. To find $b^*(r)$, for $1 \leq r \leq 9$, we want to solve the congruence relation

$$\text{Min} \quad \text{gap} = 0\,x_2 + 0\,x_3 + 1\,x_4 \tag{4.4}$$

$$\text{Subject to} \quad 12\,x_2 + 15\,x_3 + 1\,x_4 \equiv r \pmod{10}$$

$$x_j \geq 0, \text{ integers}$$

Since GCD(w_1, w_2, w_3) = 1, we can generate the group elements g_2, g_3, …, g_9 with a combination of g_2 and g_5. We have

$b^*(0) = 0$, $b^*(1) = 51$, $b^*(2) = 12$, $b^*(3) = 63$, $b^*(4) = 24$,

$b^*(5) = 15$, $b^*(6) = 36$, $b^*(7) = 27$, $b^*(8) = 48$, $b^*(9) = 39$.

The resultant minimum cost spanning tree is shown in Figure 4.2. Again, all tree arcs corresponding to w_2 or w_3 have gap cost = 0 since $\rho_1 = \rho_2 = \rho_3 = 1$.

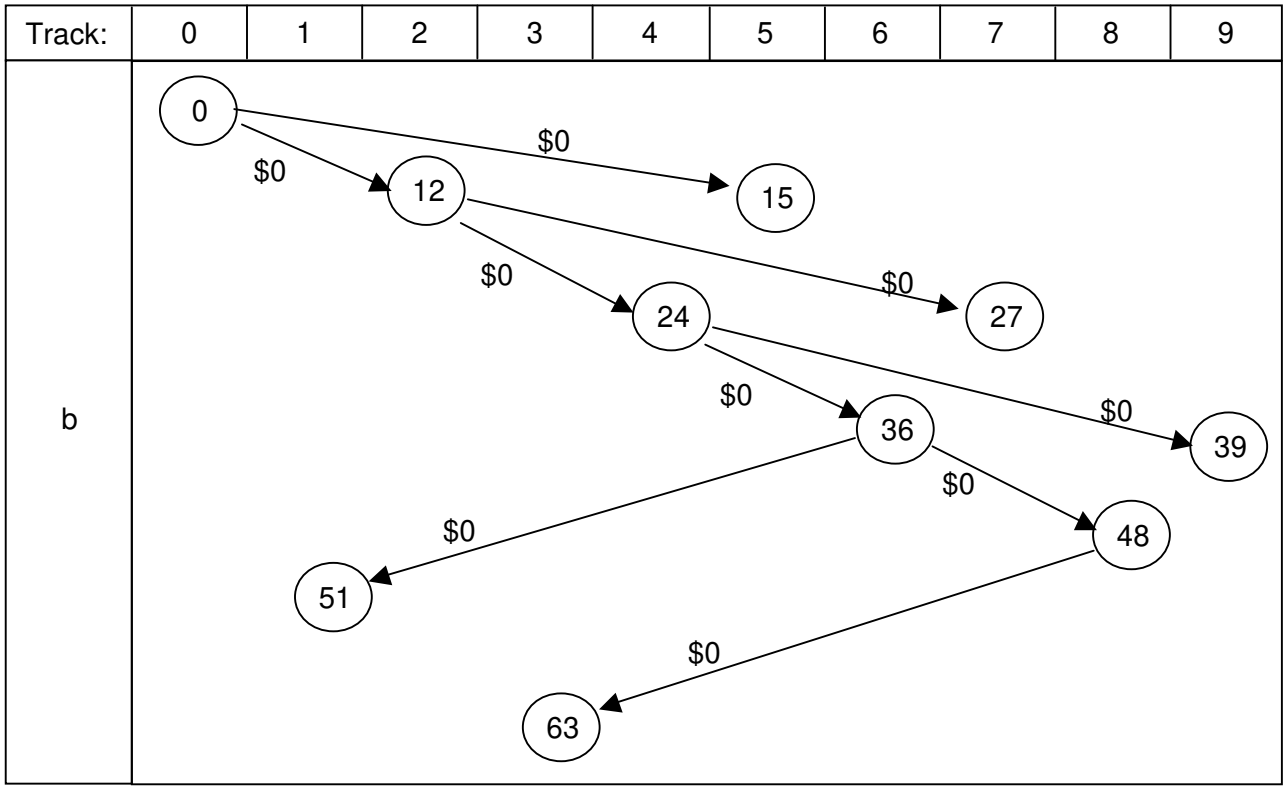

Figure 4.2

Example 4.3

Max $x_0 = 5\,x_1 + 49998\,x_2 + 0\,x_3$ (4.5)

Subject to $5\,x_1 + 49999\,x_2 + 1\,x_3 = b$

$x_j \geq 0$, integers

Here, we want to solve the congruence relation

Min gap $= 1\,x_2 + 1\,x_3$ (4.6)

Subject to $49999\,x_2 + 1\,x_3 \equiv r \pmod 5$

$x_j \geq 0$, integers

for $1 \leq r \leq 4$. Since GCD(w_1, w_2) = 1 and $49999 \equiv 4 \pmod 5$, we can use g_4 to generate the group elements g_2, g_3, …, g_4. However, since the arc w_2 has gap cost = 1, using w_2 to cover tracks 1 and 2 will result in sub-optimum solution. Hence, we have

$b^*(0) = 0$, $b^*(1) = 1$, $b^*(2) = 2$, $b^*(3) = 99998$, $b^*(4) = 49999$.

The resultant minimum cost spanning tree is shown in Figure 4.3.

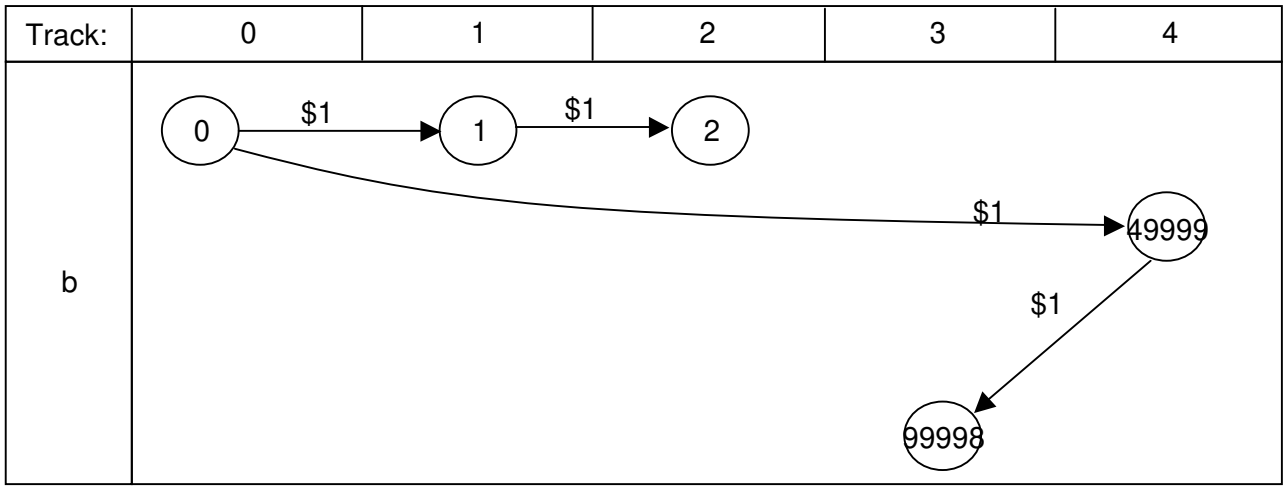

Figure 4.3

Example 4.4

Max $x_0 = 10\,x_1 + 15\,x_2 + 60000\,x_3 + 2\,x_4 + 0\,x_5$ (4.7)

Subject to $10\,x_1 + 15\,x_2 + 69993\,x_3 + 3\,x_4 + 1\,x_5 = b$

$x_j \geq 0$, integers

We want to solve the congruence relation, for $1 \leq r \leq 9$,

Min $gap = 0\,x_2 + 9993\,x_3 + 1\,x_4 + 1\,x_5$ (4.8)

Subject to $15\,x_2 + 69993\,x_3 + 3\,x_4 + 1\,x_5 \equiv r \pmod{10}$

$x_j \geq 0$, integers

Since 69993 (mod 10) = 3 (mod 10) = 3 and $(w_3 - v_3) > (w_4 - v_4)$, we will never use the third item (w_3) to cover any tracks. So, we can eliminate the third item from (4.8) even though $\rho_3 > \rho_4$, resulting in the congruence relations

Min $gap = 0\,x_2 + 1\,x_4 + 1\,x_5$ (4.9)

Subject to $15\,x_2 + 3\,x_4 + 1\,x_5 \equiv r \pmod{10}$

$x_j \geq 0$, integers

The resultant minimum cost spanning tree is shown in Figure 4.4, with

$b^*(0) = 0$, $b^*(1) = 1$, $b^*(2) = 2$, $b^*(3) = 3$, $b^*(4) = 4$,

$b^*(5) = 15$, $b^*(6) = 16$, $b^*(7) = 17$, $b^*(8) = 18$, $b^*(9) = 19$.

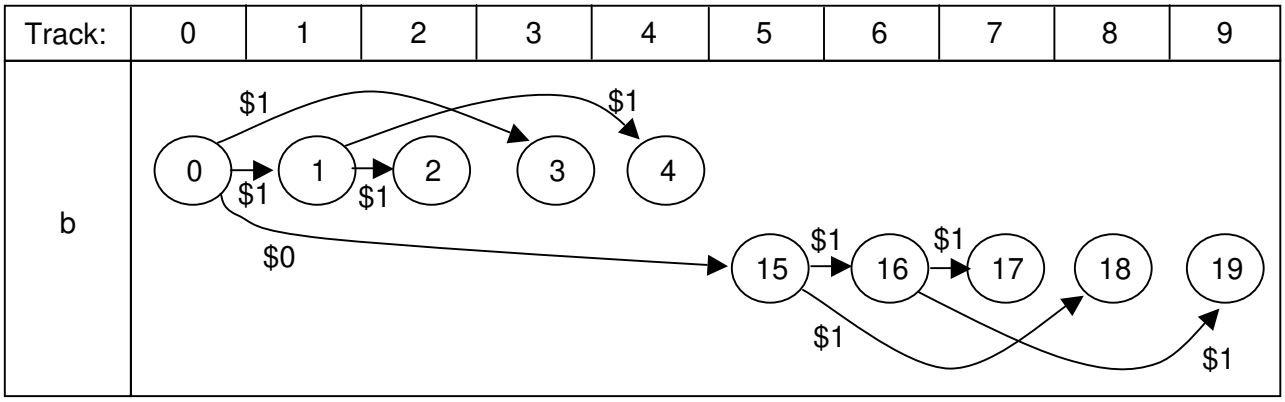

Figure 4.4

Example 4.5

Max $x_0 = 10\,x_1 + 15\,x_2 + 69992\,x_3 + 1\,x_4 + 0\,x_5$ (4.10)

Subject to $10\,x_1 + 15\,x_2 + 69993\,x_3 + 3\,x_4 + 1\,x_5 = b$

$x_j \geq 0$, integers

We want to solve the congruence relation, for $1 \leq r \leq 9$,

Min gap $= 0\,x_2 + 1\,x_3 + 2\,x_4 + 1\,x_5$ (4.11)

Subject to $15\,x_2 + 69993\,x_3 + 3\,x_4 + 1\,x_5 \equiv r \pmod{10}$

$x_j \geq 0$, integers

Since $69993 \equiv 3 \pmod{10}$ and $(w_3 - v_3) < (w_4 - v_4)$, we will never use the fourth item to cover any tracks and hence can be eliminated from (4.11).

The resultant minimum cost spanning tree is shown in Figure 4.5, with

$b^*(0) = 0$, $b^*(1) = 1$, $b^*(2) = 2$, $b^*(3) = 69993$, $b^*(4) = 69994$,

$b^*(5) = 15$, $b^*(6) = 16$, $b^*(7) = 17$, $b^*(8) = 70008$, $b^*(9) = 70009$.

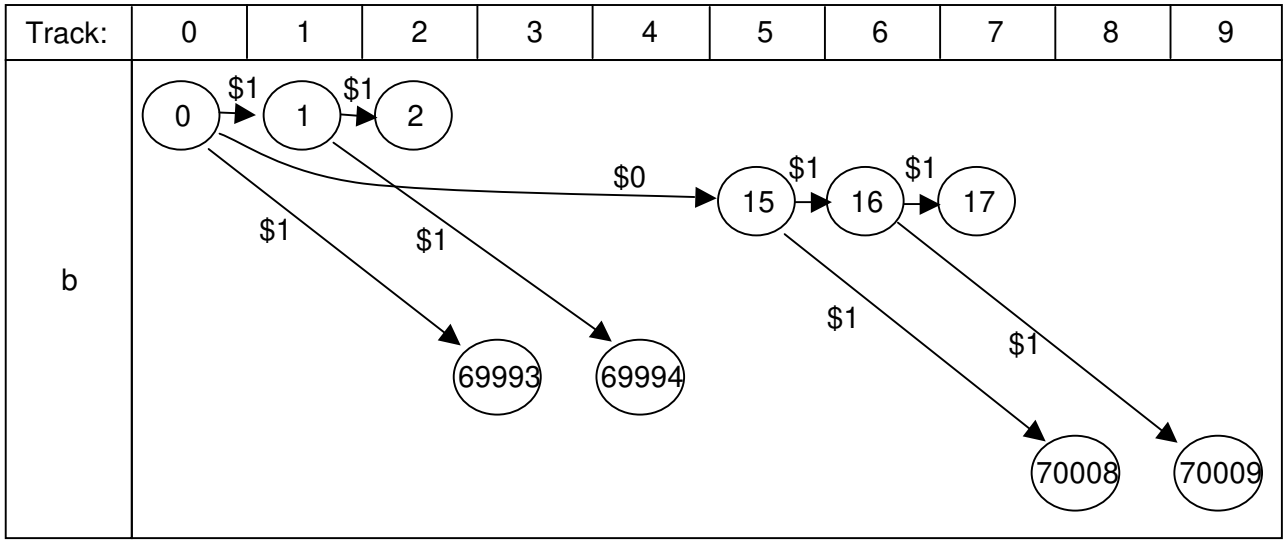

Figure 4.5

Example 4.6

Max $x_0 = 10\,x_1 + 15\,x_2 + 69992\,x_3 + 3\,x_4 + 0\,x_5$ (4.12)

Subject to $10\,x_1 + 15\,x_2 + 69993\,x_3 + 4\,x_4 + 1\,x_5 = b$

$x_j \geq 0$, integers

We want to solve the congruence relation, for $1 \leq r \leq 9$,

Min gap = $0\,x_2 + 1\,x_3 + 1\,x_4 + 1\,x_5$ (4.13)

Subject to $15\,x_2 + 69993\,x_3 + 4\,x_4 + 1\,x_5 \equiv r \pmod{10}$

$x_j \geq 0$, integers

Since the gap cost of the arcs w_3, w_4 and w_5 are all equal to 1, the resultant minimum cost spanning tree shown in Figure 4.6 involves all four kinds of arcs. We have

$b^*(0) = 0$, $b^*(1) = 1$, $b^*(2) = 2$, $b^*(3) = 69993$, $b^*(4) = 4$,

$b^*(5) = 15$, $b^*(6) = 16$, $b^*(7) = 17$, $b^*(8) = 70008$, $b^*(9) = 19$.

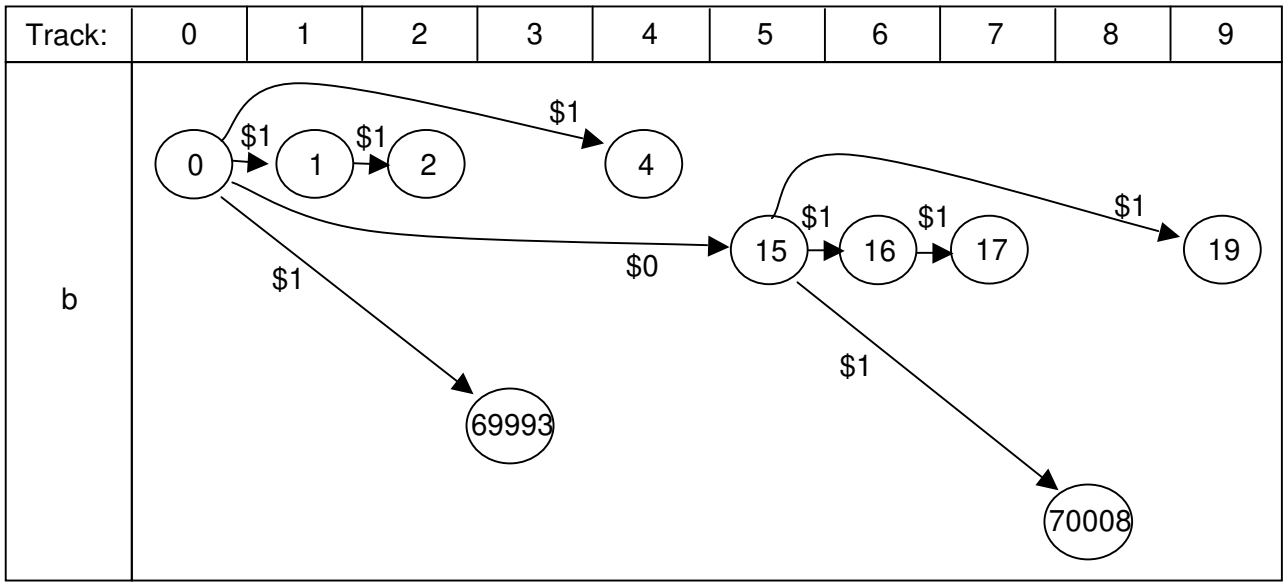

Figure 4.6

Example 4.7

Max $x_0 = 12\,x_1 + 15\,x_2 + 10004\,x_3 + 3\,x_4 + 1\,x_5 + 0\,x_6$ (4.14)

Subject to $12\,x_1 + 15\,x_2 + 10007\,x_3 + 4\,x_4 + 2\,x_5 + 1\,x_6 = b$

$x_j \geq 0$, integers

To find $b^*(r)$, for $1 \leq r \leq 11$, we want to solve the congruence relation

Min $gap = 0\,x_2 + 3\,x_3 + 1\,x_4 + 1\,x_5 + 1\,x_6$ (4.15)

Subject to $15\,x_2 + 10007\,x_3 + 4\,x_4 + 2\,x_5 + 1\,x_6 \equiv r \pmod{12}$

$x_j \geq 0$, integers

Since GCD(12, 15) = 3 and $\rho_1 = \rho_2 = 1$, we can use the second kind of items to cover the tracks 3, 6, 9 with zero gap cost. Moreover, since covering any remaining track with only the second (w_2) and the sixth (w_6) kind of items will result in packings with a gap cost of at most 2, we will never use the third item (w_3) to cover any remaining track and hence can be eliminated from (4.15), because 10007 mod 12 = 11 and we can cover track 11 with a gap cost of 1 by a combination of the second (w_2) and the fifth (w_5) kind of items.

The resultant minimum cost spanning tree shown in Figure 4.7 with

b*(0) = 0, b*(1) = 1, b*(2) = 2, b*(3) = 15, b*(4) = 4, b*(5) = 17

b*(6) = 30, b*(7) = 19, b*(8) = 32, b*(8) = 45, b*(10) = 34, b*(11) = 47.

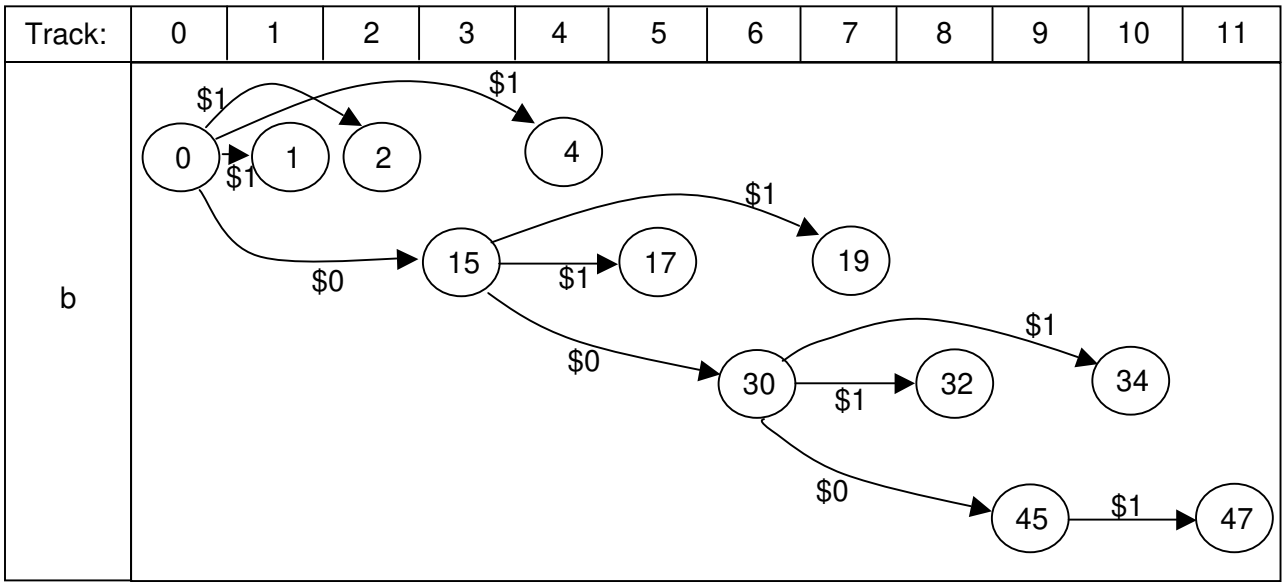

Figure 4.7

5. Algorithms and Complexity

By preprocessing, we can have the input data satisfying

$$1 = \rho_1 \geq \rho_2 \geq \rho_3 \geq \ldots \geq \rho_n = 0 \qquad (5.1)$$

where $v_n = 0$ and $w_n = 1$. We can further partition the data satisfying (5.1) into two categories:

$$1 = \rho_1 = \rho_2 = \ldots = \rho_k > \rho_{k+1} \geq \ldots \geq \rho_n = 0 \qquad (5.2)$$

and

$$1 = \rho_1 > \rho_2 \geq \rho_3 \geq \ldots \geq \rho_n = 0. \qquad (5.3)$$

Let us first consider a special case of (5.2), namely

$$1 = \rho_1 = \rho_2 > \rho_3 \geq \rho_4 \geq \ldots \geq \rho_n = 0. \qquad (5.4)$$

Furthermore, let

$$\text{GCD}(w_1, w_2) = 1 \text{ and } w_2 \,(\text{mod } w_1) = r. \qquad (5.5)$$

Here, we have a cyclic group and we can consider r as the group element g_r. We have $w_2 \,(\text{mod } w_1) = r$. Furthermore, the integers

$$2\, w_2 \,(\text{mod } w_1),\ 3\, w_2 \,(\text{mod } w_1),\ \ldots,\ (w_1 - 1)\, w_2 \,(\text{mod } w_1)$$

will be in different tracks. Each one is the b* value for that particular track and the global critical values b** is given by (5.6).

$$b^{**} = (w_1 - 1)\, w_2. \qquad (5.6)$$

Note that (5.6) is obtained without using the rest of the items and we can compute the $(w_1 - 1)$ critical values directly. This compares favorably with (1.9) and can be used without normalizing ρ_1 and ρ_2 to 1. For example, if we are only given two kinds of items of weights 10 and 13 in (4.1), i.e.

$$\text{Max} \quad x_0 = 10\, x_1 + 13\, x_2 + 0\, x_4 \qquad (5.7)$$

$$\text{Subject to} \quad 10\, x_1 + 13\, x_2 + 1\, x_4 = b$$

$$x_j \geq 0, \text{ integers},$$

then $b^{**} = (w_1 - 1)\, w_2 = (10 - 1) \times 13 = 117$.

When we have three or more items of densities equal to one and their greatest common divisor equals to 1, like {10, 13, 17} in (4.1) or {10, 12, 15} in (4.2), we only have to consider the critical values formed by combinations of these items, resulting in a b** = 68 for (4.1) and 63 for (4.2).

If $\rho_1 = \rho_2 > \rho_3 \geq \rho_4 \geq \ldots \geq \rho_n$ but GCD(w_1, w_2) = d >1, we will not be able to cover all tracks with only w_2. For example, if we are only given two kinds of items of weights 10 and 12 in (4.2), i.e.

$$\text{Max} \quad x_0 = 10\,x_1 + 12\,x_2 + 0\,x_4 \qquad (5.8)$$

$$\text{Subject to} \quad 10\,x_1 + 12\,x_2 + 1\,x_4 = b$$

$$x_j \geq 0, \text{ integers}$$

then we can only cover the even number tracks with w_2 items. The corresponding formula to (5.6), which gives the largest b that can be covered with w_2 items only, is

$$\left(\frac{w_1}{d} - 1\right) \times \frac{w_1}{d} \times d \qquad (5.9)$$

For the odd number tracks r, we have

$$b^*(r) = b^*(r-1) + w_4 = b^*(r-1) + 1.$$

Once we find b*(r) for $0 \leq r \leq w_1 - 1$, we can show that the optimum solutions satisfy

$$x_0(b^*(r)) = (x_1, x_2, \ldots, x_n) \Rightarrow x_0(m\,w_1 + b^*(r)) = (m + x_1, x_2, \ldots, x_n) \qquad (5.10)$$

for any positive integer m. So, all instances of $b \geq b^{**}$ can be solved in polynomial time. We define b*(r) as the smallest integer value of b in the track r such that $x_0(m\,w_1 + b^*(r))$, the optimum solution of the integer value $m\,w_1 + b^*(r)$, $m \geq 0$, is of the form

$$x_0(b^*(r)) + (m, 0, \ldots, 0). \qquad (5.11)$$

Next, let us outline the algorithm for the case

$$1 = \rho_1 \geq \rho_2 \geq \rho_3 \geq \ldots \geq \rho_n = 0.$$

There is a gap cost $w_j - v_j$ for item j ($2 \leq j \leq n$). (For readers of Mathematical Programming background, the gap cost can be viewed as the modified cost

$\overline{c_j}$ associated with a non-basic column.) We will eliminate as many items as possible by various simple tests shown in the numerical examples. Then we need to solve the following congruence relations of the form

$$w_2 x_2 + w_3 x_3 + \ \ldots + w_n x_n \equiv r \pmod{w_1}, \text{ for } 0 \le r \le w_1 - 1 \qquad (5.12)$$

while minimizing the total gap cost for each r.

We want to solve (5.12) with an algorithm that builds a minimum cost Multi-Track (M-T) spanning tree very much like the Dijkstra's single source Shortest-Path tree.

Mentally, we have infinite number of nodes in the underlying graph, but only a subset of these nodes will be considered as candidates for the minimum cost M-T spanning tree. Each node is associated with a b-value like those in Table 2.1 and there is an edge from node b to node b' of length w_j and cost $(w_j - v_j)$ if $(b' - b) = w_j$, $2 \le j \le n$. We say b' is a neighbor of b.

We will start with the node 0 as the root of the spanning tree and grow the tree one node at a time by examining the neighbors of the nodes already in the spanning tree. Only one node in each track will be included in the tree.

Every node, examined by the algorithm, will receive a label with four attributes:

1. The b-value associated with the node.
2. The current total gap cost of all the arcs in the directed path from the root to the node.
3. The track that the node belongs.
4. The parent of the node in the tree. The parent of node 0 is undefined.

Every node is always in one of the three states:

1. Unlabeled (U). Initially, every node, except node 0, is unlabeled.
2. Labeled and Unscanned (LU). Refers to any node that has a label and some of its neighboring nodes have not been labeled. Node 0 is labeled and unscanned initially.

3. Labeled and Scanned (LS). Refers to any node that has a label and all its neighbors have been labeled.

In addition, every track is in one of the two states:

1. Uncovered. Initially, all tracks are uncovered.
2. Covered. A track becomes covered if it has a node that is in the state of labeled and scanned.

For brevity, let p be a node with the same value p as its first attribute, State(p) to denote the state of p, Track(p) to denote the track p belongs, Cost(p) and Parent(p) to denote the total gap cost from node 0 to p and its parent along the path from node 0 to p.
A node p is cheaper than another node q if Cost(p) < Cost(q) or (Cost(p) = Cost(q) and p < q). For brevity, we use the symbol "Cost(p) ≤ Cost(q)" to mean node p is cheaper than another node q.

We now give a high-level English version of the algorithm. (Readers can refer to Appendix 1 for detailed pseudo-code.)

M-T Spanning Tree Algorithm

Input: n pairs of non-negative numbers (v_j, w_j), where all w_j are integers with $w_n = 1$, all v_j are converted into integers with $v_n = 0$, and satisfying

$$\frac{v_j}{w_j} > \frac{v_{j+1}}{w_{j+1}} \quad \text{or} \quad \left(\frac{v_j}{w_j} = \frac{v_{j+1}}{w_{j+1}} \quad \text{and} \quad w_j < w_{j+1} \right)$$

[Comments: The pre-processing of converting v_j into rational numbers such that the density of the best item equals 1, checking to see if two or more densities (expressed as rational numbers) are the same, testing to see if certain items could be eliminated and sorting the densities into non-increasing order all can be done in O(n log n) time.]

Output: A minimum gap cost M-T spanning tree with node 0 as root and $w_1 - 1$ other nodes, one node for every track, corresponding b*(r) for r = 1, … , $w_1 - 1$.

Procedure:

Initially all nodes are unlabeled (U) except the node 0, which is labeled and unscanned (LU). We have two sets of nodes

T: all nodes that are labeled and scanned (LS)

Q: all nodes that are labeled and unscanned (LU).

Both sets are empty initially. Furthermore, there is at most one node from each track in $T \cup Q$.

Step 0. Associate node 0 with its label (0, 0, 0, undefined). State(0) = LU and add it to Q.

Step 1. Let p be the cheapest node in Q. Update the labels of all n−1 neighboring nodes q as follows:

Let $q = p + w_j$ and $q \pmod{w_1} = r$.

Set $Cost(q) = Cost(p) + w_j - v_j$, and Label(q) = (q, Cost(q), r, p).

Add q in Q if there is no node belonging to Track r in $T \cup Q$.

Replace the existing node x belonging Track r in $T \cup Q$ with q if q is cheaper than x.

Do nothing if the existing node x in Track r is cheaper.

Step 2. After updating all neighboring nodes of p, node p becomes LS. It is removed from Q and added to T. Change the state of Track(p) to covered.
[Comments: The labels of nodes in state LS are bolded in the numerical examples below.]

Step 3. Terminate the algorithm if $|T| = w_1$, otherwise return to Step 1.

We shall use an array to remember which track each node in Q belongs to as we grow the tree. For example, consider the M-T spanning tree for Example 4.4:

1. In the beginning, only node 0 receives a label (0, 0, 0, undefined) and it is in the LU state. Node 0 has 3 neighbors (nodes 1, 3 and 15) and they are in the U state.

Track:	0	1	2	3	4	5	6	7	8	9
Label = (b, cost, track, parent)	(0,0, 0,undef)									

2. We assign them the labels (1, 1, 1, 0), (3, 1, 3, 0) and (15, 0, 5, 0) to nodes 1, 3 and 15 respectively. Node 0 changes its state to LS and its three neighbors are now in the LU state. Track 0 is covered.

Track:	0	1	2	3	4	5	6	7	8	9
Label = (b, cost, track, parent)	**(0,0, 0,undef)**	(1,1, 1,0)		(3,1, 3,0)		(15,0, 5,0)				

3. Among all nodes that are LU, we choose the one with the cheapest total gap cost and break tie by choosing the one with smallest b. So, node 15 is chosen.
4. For each neighbor of newly chosen node that is on an uncovered track,
   (i) assign a new label to the node if it is unlabeled or update its label if it represents a new path with cheaper gap cost or same gap cost but of cheaper length,
   (ii) add the new node to the track if this is the first time a node lands on that track or replace the existing labeled but unscanned node in that track if the new node is cheaper.

   Now, node 15 has 3 neighbors (nodes 16, 18, and 30). We forget node 30 since it belongs to track 0, which is covered. We assign the labels (16, 1, 6, 15) and (18, 1, 8, 15) to nodes 16 and 18 respectively. Node 15 changes its state to LS and its two neighbors are now in the LU state. Track 5 is covered.

Track:	0	1	2	3	4	5	6	7	8	9
Label = (b, cost, track, parent)	**(0,0, 0,undef)**	(1,1, 1,0)		(3,1, 3,0)		**(15,0, 5,0)**	(16,1, 6,15)		(18,1, 8,15)	

5. Now, the gap costs of all nodes that are labeled and unscanned are all equal to 1. We choose node 1 because it has the smallest b.
6. Assign the labels (2, 2, 2, 1) and (4, 2, 4, 1) to nodes 2 and 4 respectively. We do not update the label of node 16 since its cost and length remain the same. Node 1

changes its state to LS and its two neighbors are now in the LU state. Track 1 is covered.

Track:	0	1	2	3	4	5	6	7	8	9
Label = (b, cost, track, parent)	**(0,0, 0,undef)**	**(1,1, 1,0)**	(2,2, 2,1)	(3,1, 3,0)	(4,2, 4,1)	**(15,0, 5,0)**	(16,1, 6,15)		(18,1, 8,15)	

7. Next, we choose node 3. We do not update the labels of its neighbors (nodes 4, 6 and 18) because they do not improve the costs or lengths of the nodes in Q. Node 3 changes its state to LS. Track 3 is covered.

Track:	0	1	2	3	4	5	6	7	8	9
Label = (b, cost, track, parent)	**(0,0, 0,undef)**	**(1,1, 1,0)**	(2,2, 2,1)	**(3,1, 3,0)**	(4,2, 4,1)	**(15,0, 5,0)**	(16,1, 6,15)		(18,1, 8,15)	

8. Next, we choose node 16. Assign the labels (17, 2, 7, 16) and (19, 2, 9, 16) to nodes 17 and 19 respectively. Node 16 changes its state to LS and its two neighbors are now in the LU state. Track 6 is covered.

Track:	0	1	2	3	4	5	6	7	8	9
Label = (b, cost, track, parent)	**(0,0, 0,undef)**	**(1,1, 1,0)**	(2,2, 2,1)	**(3,1, 3,0)**	(4,2, 4,1)	**(15,0, 5,0)**	**(16,1, 6,15)**	(17,2, 7,16)	(18,1, 8,15)	(19,2, 9,16)

9. Next, we choose node 18. We do not update the label of its neighbor (node 19) because it does not improve its cost or length. We forget nodes 21 and 23 because they all belong to covered tracks. Node 18 changes its state to LS. Track 8 is covered.

Track:	0	1	2	3	4	5	6	7	8	9
Label = (b, cost, track, parent)	**(0,0, 0,undef)**	**(1,1, 1,0)**	(2,2, 2,1)	**(3,1, 3,0)**	(4,2, 4,1)	**(15,0, 5,0)**	**(16,1, 6,15)**	(17,2, 7,16)	**(18,1, 8,15)**	(19,2, 9,16)

10. Next, we choose node 2. We do not update the label of its neighbors (nodes 17) because it does not improve its cost or length. We forget nodes 5 and 6 because

they all belong to covered tracks. Node 2 changes its state to LS. Track 2 is covered.

Track:	0	1	2	3	4	5	6	7	8	9
Label = (b, cost, track, parent)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(2,2, 2,1)**	**(3,1, 3,0)**	(4,2, 4,1)	**(15,0, 5,0)**	**(16,1, 6,15)**	(17,2, 7,16)	**(18,1, 8,15)**	(19,2, 9,16)

11. Next, we choose node 4. We do not update the labels of its neighbors (nodes 7 and 19) because they do not improve the costs or lengths of the nodes in Q. We forget node 5 because it belongs to a covered track. Node 4 changes its state to LS. Track 4 is covered.

Track:	0	1	2	3	4	5	6	7	8	9
Label = (b, cost, track, parent)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(2,2, 2,1)**	**(3,1, 3,0)**	**(4,2, 4,1)**	**(15,0, 5,0)**	**(16,1, 6,15)**	(17,2, 7,16)	**(18,1, 8,15)**	(19,2, 9,16)

12. Next, we choose node 17. We do not update the labels of its neighbors (nodes 18, 20 and 32) because they all belong to covered tracks. Node 17 changes its state to LS. Track 7 is covered.

Track:	0	1	2	3	4	5	6	7	8	9
Label = (b, cost, track, parent)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(2,2, 2,1)**	**(3,1, 3,0)**	**(4,2, 4,1)**	**(15,0, 5,0)**	**(16,1, 6,15)**	**(17,2, 7,16)**	**(18,1, 8,15)**	(19,2, 9,16)

13. Next, we choose node 19. We do not update the labels of its neighbors (nodes 20, 22 and 44) because they all belong to covered tracks. Node 19 changes its state to LS. Track 9 is covered.

Track:	0	1	2	3	4	5	6	7	8	9
Label = (b, cost, track, parent)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(2,2, 2,1)**	**(3,1, 3,0)**	**(4,2, 4,1)**	**(15,0, 5,0)**	**(16,1, 6,15)**	**(17,2, 7,16)**	**(18,1, 8,15)**	**(19,2, 9,16)**

14. At the point, all tracks are covered and the algorithm terminates.

The algorithm for building the minimum cost M-T spanning tree is very much like the Dijkstra's algorithm for the shortest-path tree from the root to all other nodes. The

labeled and unscanned nodes are neighbors of at least one nodes in the spanning tree. We successively grow the tree by picking the "cheapest" node among those that are labeled and unscanned, and update the members and their costs of the paths to the set of labeled and unscanned nodes each time the spanning tree T has a new node. The difference is that the neighbor expansion is restricted to uncovered tracks and the terminating condition is when all tracks are covered.

Lemma 5.1 The costs of nodes being added to T are monotonically increasing.
Proof. (By Induction)
Initially, T contains node 0, which is cheaper than any other node in T based on the tie-breaking rules. It follows from the choice of p that the cost of p is always cheaper than all nodes in Q before Step 1. Since all gap costs are non-negative and $w_j > 0$, p is cheaper than all nodes in Q after Step 1. Hence, by induction, p must be cheaper than all nodes that are added to T after p.

Q.E.D.

Lemma 5.2 There is exactly one node in every track in T at the termination of the M-T spanning tree algorithm.
Proof.
It follows from the fact $w_n = 1$ and Step 1 that at least one node from each track is added to Q during the course of the algorithm and at most one node from each track is present in Q at any time. Hence, there is exactly one node in every track in T at the termination of the M-T spanning tree algorithm.

Q.E.D.

Theorem 5.3 The All node in T are b*(r) at termination.
Proof. (By Contradiction)
Let x be the first node being added to T such that $x \equiv r \pmod{w_1}$ but $x \neq b^*(r)$. Then there must exist some node y such that $b^*(r) = y$ and $Cost(y) \leq Cost(x)$. In other words, there exists a path from node 0 to y that makes y cheaper than x. Let Parent(y) be the immediate predecessor of y along this path.

Since all gap costs are non-negative, Parent(y) is cheaper than y and it follows from Lemma 5.1 that Parent(y) is added to T before x. We have the following three cases.

Case 1. Parent(x) = Parent(y) = p

This case can only happen if we have two items w_i and w_j such that

$$y = p + w_i \equiv r \pmod{w_1} \text{ and } x = p + w_j \equiv r \pmod{w_1} \text{ and } Cost(y) \leq Cost(x).$$

It follows from Step 1 that only node y remains in Q after the algorithm scans the neighbors of p. Hence, x can never be added to T, a contradiction.

Case 2. Parent(x) is added to T before Parent(y)

It this case, x will be replaced by y when the algorithm scans the neighbors of Parent(y). Again, x can never be added to T, a contradiction.

Case 3. Parent(y) is added to T before Parent(x)

It this case, x will not be able to replace y in Q when algorithm scans the neighbors of Parent(x). Hence, x can never be added to T, a contradiction.

Q.E.D.

Note that the algorithm depends only on the weights and the relative gap costs of the items. To avoid the extra complexity in handling real numbers and rational numbers, we can eliminate the need to normalize a given knapsack problem by setting the gap cost of w_j to

$$(w_j \times v_1 - v_j \times w_1),\ 2 \leq j \leq n. \qquad (5.13)$$

For example, to find the b*(r) for the knapsack problem

$$\text{Max} \quad x_0 = 11\,x_1 + 11\,x_2 + 8\,x_3 + 0\,x_4 \qquad (5.14)$$

$$\text{Subject to} \quad 10\,x_1 + 11\,x_2 + 9\,x_3 + 1\,x_4 = b$$

$$x_j \geq 0, \text{ integers}$$

we will construct a minimum cost M-T spanning tree with the following gap costs:

$w_2 = 11$, gap-cost(w_2) = $(11 \times 11 - 11 \times 10) = 11$,

$w_3 = 9$, gap-cost(w_3) = $(9\times11 - 8\times10) = 19$,

and $w_4 = 1$, gap-cost(w_4) = $(1\times11 - 0\times10) = 11$

to solve the congruence relation, for $1 \le r \le 9$,

$$\text{Min} \quad \text{gap} = 11\,x_2 + 19\,x_3 + 11\,x_4 \qquad (5.15)$$

$$\text{Subject to} \quad 11\,x_2 + 9\,x_3 + 1\,x_4 \equiv r \pmod{10}$$

$$x_j \ge 0, \text{ integers}$$

Theorem 5.4

The M-T spanning tree algorithm computes a minimum cost M-T spanning tree in time $O(n\, w_1 \log w_1)$.

Proof.

It takes O(n log n) time for the preprocessing to set up the gap costs and order the items according to their relative density.

We shall use a priority queue to maintain the nodes in Q and an array to remember which track each node in Q belongs to. Since at most one node from each track is present in Q at any time, it takes $O(\log w_1)$ to add or remove a node from Q. We use another array to keep track of the status of each track (whether they are covered or uncovered).

In Step 1, it takes O(1) time to find the cheapest node p in Q. Each node p has at most n neighbors. It takes O(1) time to compute the new cost for each neighbor q and compare it against the node belonging to the same track in Q, and $O(\log w_1)$ time to replace the node belonging to Track(q) in Q with q. Hence, it takes a total of $(n - 1) * O(\log w_1) = O(n \log w_1)$ time to scan the neighbors of p.

In Step 2, it take $O(\log w_1)$ time to remove p from Q and O(1) time to add it to T and update the status of Track(p).

Since there are w_1 nodes in T, the algorithm takes a total of $w_1 * O(n \log w_1) = O(n\, w_1 \log w_1)$ time to compute the minimum cost M-T spanning tree.

Q.E.D.

6. The Extended M-T Spanning Tree

In this section, we show how to modify the M-T Spanning Tree algorithm to solve a knapsack problem where the capacity b is less than $b^*(r)$ for $r = 1, 2, \ldots, w_1-1$. We call the resultant spanning tree an Extended M-T Spanning Tree. Like the original M-T Spanning Tree, an Extended M-T Spanning Tree also has node 0 as root and spans the other w_1-1 tracks. However, it may contain more than one node in each track, where the nodes, say $p_1, p_2, \ldots, p_k$, in each track satisfy the properties:

(1) $p_1 \equiv p_2 \equiv \ldots \equiv p_k \equiv r \pmod{w_1}$ for some r $(1 \le r \le w_1-1)$,

(2) $p_1 < p_2 < \ldots < p_k$,

(3) $Cost(p_1) > Cost(p_2) > \ldots > Cost(p_k)$,

(4) $p_k = b^*(r)$,

(5) for any node q, $(q \equiv r \pmod{w_1}$ and $p_i \le q < p_{i+1}$ $(1 \le i \le k-1))$ implies $Cost(q) = Cost(p_i)$.

We first give a numerical example to show the details of the Extended M-T Spanning Tree algorithm. Consider the knapsack problem

$$\text{Max} \quad x_0 = 10 x_1 + 11 x_2 + 5 x_3 + 0 x_4 \qquad (6.1)$$

$$\text{Subject to} \quad 10 x_1 + 12 x_2 + 7 x_3 + 1 x_4 = b$$

$$x_j \ge 0, \text{integers}$$

To find $b^*(r)$, for $1 \le r \le 9$, we want to solve the congruence relation

$$\text{Min} \quad gap = 1 x_2 + 2 x_3 + 1 x_4 \qquad (6.2)$$

$$\text{Subject to} \quad 12 x_2 + 7 x_3 + 1 x_4 \equiv r \pmod{10}$$

$$x_j \ge 0, \text{integers}$$

We will use a table to keep track of which track each node in Q and T belongs to as we grow the tree. Furthermore, the best “Label and Unscanned” candidate in each track are italicized in Q.

1. In the beginning, only node zero receives a label (0, 0, 0, undefined) and it is in the LU state. Node zero has 3 neighbors (nodes 1, 2 and 7) and they are in the U state.
2. We assign them the labels (1, 1, 1, 0), (12, 1, 2, 0) and (7, 2, 7, 0) to nodes 1, 12 and 7 respectively. Node zero changes its state to LS and its three neighbors are now in the LU state.

Track:	0	1	2	3	4	5	6	7	8	9
Q Label = (b, cost, track, parent)		*(1,1, 1,0)*	*(12,1, 2,0)*					*(7,2, 7,0)*		
T Label = (b, cost, track, parent)	**(0,0, 0,undef)**									

3. Among all nodes that are LU, we choose the one with the cheapest total gap cost and break tie by choosing the one with smallest b. So, node 1 is chosen.
4. Node 1 has 3 neighbors (nodes 2, 13 and 8). We assign them the labels (2, 2, 2, 1), (13, 2, 3, 1) and (8, 3, 8, 1) to nodes 2, 13 and 8 respectively. We keep node 2 even though 12 is cheaper than 2. Node 1 changes its state to LS and its three neighbors are now in the LU state.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates			(2,2, 2,1)							
	Best track candidates			*(12,1, 2,0)*	*(13,2, 3,1)*				*(7,2, 7,0)*	*(8,3, 8,1)*	
T	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**								

5. Next, node 12 is chosen. It has three neighbors (13, 24 and 19). We forgot 13 because it does not improve what is already in track 3, and assign the labels (24, 2, 4, 12) and (19, 3, 9, 12) to nodes 24 and 19 respectively. Node 12 changes its state to LS and its two neighbors are now in the LU state.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Best track candidates			*(2,2, 2,1)*	*(13,2, 3,1)*	*(24,2, 4,12)*			*(7,2, 7,0)*	*(8,3, 8,1)*	*(19,3, 9,12)*
T	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**							

6. Node 2 is chosen (instead of node 7 because 2 is cheaper than 7). It has three neighbors (3, 14 and 9). We assign the labels (3, 3, 3, 2), (14, 3, 4, 2), (9, 4, 9, 2) to nodes 3, 14 and 9. Node 2 changes its state to LS and its neighbors are in the LU state.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates				(3,3, 3,2)	(14,3 4,2)					(9,4, 9,2)
	Best track candidates				*(13,2, 3,1)*	*(24,2, 4,12)*			*(7,2, 7,0)*	*(8,3, 8,1)*	*(19,3, 9,12)*
T	Other critical values			**(2,2, 2,1)**							
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**							

7. Node 7 is chosen. It has three neighbors (8, 19 and 14). We forgot all three neighbors because they do not improve what are already in tracks 8, 9 and 4. Node 7 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates				(3,3, 3,2)	(14,4, 4,7)					(9,4, 9,2)
	Best track candidates				*(13,2, 3,1)*	*(24,2, 4,12)*				*(8,3, 8,1)*	*(19,3, 9,12)*
T	Other critical values			**(2,2, 2,1)**							
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**					**(7,2, 7,0)**		

8. Node 13 is chosen. It has three neighbors (14, 25 and 20). We forgot 20 because it does not improve what is already in track 0, update the label for node 14 to (14, 3, 4, 13) and assign the label (25, 3, 5, 13) to node 25. Node 13 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates					(14,3, 4,13)					(9,4, 9,2)
	Best track candidates				*(3,3, 3,2)*	*(24,2, 4,12)*	*(25,3, 5,13)*			*(8,3, 8,1)*	*(19,3, 9,12)*
T	Other critical values			**(2,2, 2,1)**							
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**				**(7,2, 7,0)**		

9. Node 24 is chosen. It has three neighbors (25, 36 and 31). We forgot 25 and 31 because they do not improve what are already in tracks 5 and 1, and assign the label (36, 3, 6, 24) to node 36. Node 24 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates										(9,4, 9,2)
	Best track candidates				*(3,3, 3,2)*	*(14,3, 4,13)*	*(25,3, 5,13)*	*(36,3, 6,24)*		*(8,3, 8,1)*	*(19,3, 9,12)*
T	Other critical values			**(2,2, 2,1)**							
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**			**(7,2, 7,0)**		

10. Node 3 is chosen. It has three neighbors (4, 17 and 10). We forgot 17 and 10 because they do not improve what are already in tracks 7 and 0, and assign the label (4, 4, 4, 3) to node 4. Node 3 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates					(4,4, 4,3)					(9,4, 9,2)
	Best track candidates					*(14,3, 4,13)*	*(25,3, 5,13)*	*(36,3, 6,24)*		*(8,3, 8,1)*	*(19,3, 9,12)*
T	Other critical values			**(2,2, 2,1)**	**(3,3, 3,2)**						
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**			**(7,2, 7,0)**		

11. Node 8 is chosen. It has three neighbors (9, 20 and 15). We forgot 9 and 20 because they do not improve what is already in tracks 9 and 0, and assign the label (15, 5, 5, 8) to nodes 9 and 15. Node 8 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates					(4,4, 4,3)	(15,5, 5,8)				(9,4, 9,2)
	Best track candidates					*(14,3, 4,13)*	*(25,3, 5,13)*	*(36,3, 6,24)*			*(19,3, 9,12)*
T	Other critical values			**(2,2, 2,1)**	**(3,3, 3,2)**						
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**			**(7,2, 7,0)**	**(8,3, 8,1)**	

12. Node 14 is chosen. It has three neighbors (15, 26 and 21). We forgot 21 because it does not improve what is already in track 1, update the label of 15 to (15, 4, 5, 14), and assign (26, 4, 6, 14) to node 26. Node 14 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates						(15,4, 5,14)	(26,4, 6,14)			(9,4, 9,2)
	Best track candidates					*(4,4, 4,3)*	*(25,3, 5,13)*	*(36,3, 6,24)*			*(19,3, 9,12)*
T	Other critical values			**(2,2, 2,1)**	**(3,3, 3,2)**	**(14,3, 4,13)**					
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**			**(7,2, 7,0)**	**(8,3, 8,1)**	

13. Node 19 is chosen. It has three neighbors (20, 31 and 26). We forgot all three neighbors because they do not improve what are already in tracks 0, 1 and 6. Node 19 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates						(15,4, 5,14)	(26,4, 6,14)			
	Best track candidates					*(4,4, 4,3)*	*(25,3, 5,13)*	*(36,3, 6,24)*			*(9,4, 9,2)*
T	Other critical values			**(2,2, 2,1)**	**(3,3, 3,2)**	**(14,3, 4,13)**					
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**			**(7,2, 7,0)**	**(8,3, 8,1)**	**(19,3, 9,12)**

14. Node 25 is chosen. It has three neighbors (26, 37 and 32). We forgot all three neighbors because they do not improve what are already in tracks 6, 7 and 2. Node 25 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates							(26,4, 6,14)			
	Best track candidates					*(4,4, 4,3)*	*(15,4, 5,14)*	*(36,3, 6,24)*			*(9,4, 9,2)*
T	Other critical values			**(2,2, 2,1)**	**(3,3, 3,2)**	**(14,3, 4,13)**					
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**	**(25,3, 5,13)**		**(7,2, 7,0)**	**(8,3, 8,1)**	**(19,3, 9,12)**

15. Node 36 is chosen. It has three neighbors (37, 48 and 43). We forgot all three neighbors because they do not improve what are already in tracks 7, 8 and 3. Node 36 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates										
	Best track candidates					*(4,4, 4,3)*	*(15,4, 5,14)*	*(26,4, 6,14)*			*(9,4, 9,2)*
T	Other critical values			**(2,2, 2,1)**	**(3,3, 3,2)**	**(14,3, 4,13)**					
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**	**(25,3, 5,13)**	**(36,3, 6,24)**	**(7,2, 7,0)**	**(8,3, 8,1)**	**(19,3, 9,12)**

16. Node 4 is chosen. It has three neighbors (5, 16 and 11). We forgot 11 because it does not improve what is already in track 1, and assign the labels (5, 5, 5, 4) and (16, 5, 6, 4) to nodes 5 and 16. Node 4 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates						(5,5, 5,4)	(16,5, 6,4)			
	Best track candidates						*(15,4, 5,14)*	*(26,4, 6,14)*			*(9,4, 9,2)*
T	Other critical values					**(4,4, 4,3)**					
				(2,2, 2,1)	**(3,3, 3,2)**	**(14,3, 4,13)**	**(25,3, 5,13)**	**(36,3, 6,24)**			
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**	**(25,3, 5,13)**	**(36,3, 6,24)**	**(7,2, 7,0)**	**(8,3, 8,1)**	**(19,3, 9,12)**

17. Node 9 is chosen. It has three neighbors (10, 21 and 16). We forgot all three neighbors because they do not improve what are already in tracks 0, 1 and 6. Node 9 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates						(5,5, 5,4)	(16,5, 6,4)			
	Best track candidates						*(15,4, 5,14)*	*(26,4, 6,14)*			
T	Other critical values					**(4,4, 4,3)**					
				(2,2, 2,1)	**(3,3, 3,2)**	**(14,3, 4,13)**					**(9,4, 9,2)**
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**	**(25,3, 5,13)**	**(36,3, 6,24)**	**(7,2, 7,0)**	**(8,3, 8,1)**	**(19,3, 9,12)**

18. Node 15 is chosen. It has three neighbors (16, 27 and 22). We forgot all three neighbors because they do not improve what are already in tracks 6, 7 and 2. Node 15 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates							(16,5, 6,4)			
	Best track candidates						*(5,5, 5,4)*	*(26,4, 6,14)*			
T	Other critical values					**(4,4, 4,3)**					
				(2,2, 2,1)	**(3,3, 3,2)**	**(14,3, 4,13)**	**(15,4, 5,14)**				**(9,4, 9,2)**
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**	**(25,3, 5,13)**	**(36,3, 6,24)**	**(7,2, 7,0)**	**(8,3, 8,1)**	**(19,3, 9,12)**

19. Node 26 is chosen. It has three neighbors (27, 38 and 33). We forgot all three neighbors because they do not improve what are already in tracks 7, 8 and 3. Node 26 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates										
	Best track candidates						*(5,5, 5,4)*	*(16,5, 6,4)*			
T	Other critical values					**(4,4, 4,3)**					
				(2,2, 2,1)	**(3,3, 3,2)**	**(14,3, 4,13)**	**(15,4, 5,14)**	**(26,4, 6,14)**			**(9,4, 9,2)**
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**	**(25,3, 5,13)**	**(36,3, 6,24)**	**(7,2, 7,0)**	**(8,3, 8,1)**	**(19,3, 9,12)**

20. Node 5 is chosen. It has three neighbors (6, 17 and 12). We forgot node 17 and 12 neighbors because they do not improve what are already in tracks 7 and 2, and assign the label (6,6,6,5) to node 6. Node 5 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Other track candidates							(6,6, 6,5)			
	Best track candidates							*(16,5, 6,4)*			
T	Other critical values					**(4,4, 4,3)**	**(5,5, 5,4)**				
				(2,2, 2,1)	**(3,3, 3,2)**	**(14,3, 4,13)**	**(15,4, 5,14)**	**(26,4, 6,14)**			**(9,4, 9,2)**
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**	**(25,3, 5,13)**	**(36,3, 6,24)**	**(7,2, 7,0)**	**(8,3, 8,1)**	**(19,3, 9,12)**

21. Node 16 is chosen. It has three neighbors (17, 28 and 23). We forgot all three neighbors because they do not improve what are already in tracks 7, 8 and 3. Node 16 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
Q	Best track candidates							*(6,6, 6,5)*			
T	Other critical values					**(4,4, 4,3)**	**(5,5, 5,4)**	**(16,5, 6,4)**			
				(2,2, 2,1)	**(3,3, 3,2)**	**(14,3, 4,13)**	**(15,4, 5,14)**	**(26,4, 6,14)**			**(9,4, 9,8)**
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**	**(25,3, 5,13)**	**(36,3, 6,24)**	**(7,2, 7,0)**	**(8,3, 8,1)**	**(19,3, 9,12)**

22. Node 6 is chosen. It has three neighbors (7, 18 and 13). We forgot all three neighbors because they do not improve what are already in tracks 7, 8 and 3. Node 6 changes its state to LS.

Track:		0	1	2	3	4	5	6	7	8	9
T	Other critical values							**(6,6, 6,5)**			
						(4,4, 4,3)	**(5,5, 5,4)**	**(16,5, 6,4)**			
				(2,2, 2,1)	**(3,3, 3,2)**	**(14,3, 4,13)**	**(15,4, 5,14)**	**(26,4, 6,14)**			**(9,4, 9,8)**
	b*(r)	**(0,0, 0,undef)**	**(1,1, 1,0)**	**(12,1, 2,0)**	**(13,2, 3,1)**	**(24,2, 4,12)**	**(25,3, 5,13)**	**(36,3, 6,24)**	**(7,2, 7,0)**	**(8,3, 8,1)**	**(19,3, 9,12)**

23. We stop at this point because there are no more nodes that are labeled and unscanned.

Figure 6.1 shows the resultant Extended M-T Spanning Tree, where the arcs of the original M-T Spanning Tree are shown as heavy lines.

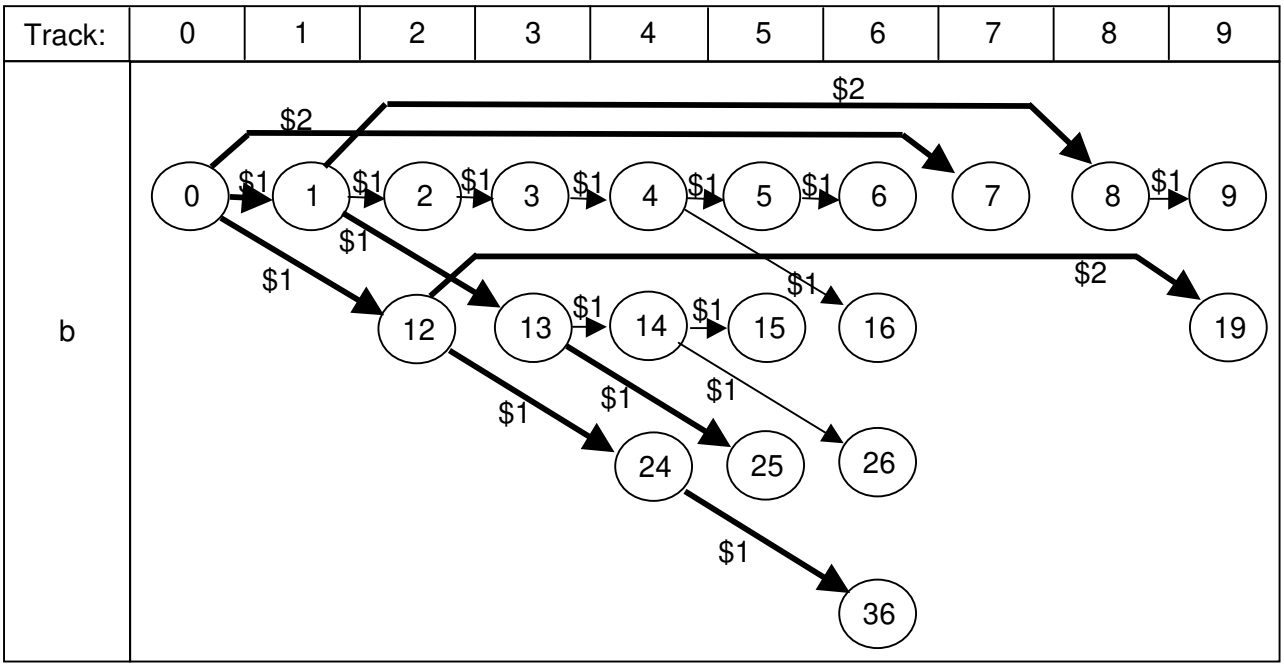

Figure 6.1

The optimum solutions of the nodes in Track 6 are:

b	$x_o(b)$	Gap Cost	Value of x_o
6	$(x_1, x_2, x_3, x_4) = (0,0,0,6)$	\$6	0
16	$(x_1, x_2, x_3, x_4) = (0,1,0,4)$	\$5	11
26	$(x_1, x_2, x_3, x_4) = (0,2,0,2)$	\$4	22
36	$(x_1, x_2, x_3, x_4) = (0,3,0,6)$	\$3	33
$m\ w_1 + 36$	$(x_1, x_2, x_3, x_4) = (m,0,0,0) + x_o(36)$	\$3	$33 + m \times 10$

In general, given an Extended M-T Spanning Tree T, we can obtain the optimum solution for any given knapsack capacity b as follows:

Let $b \equiv r \pmod{w_1}$ for some r, $0 \le r \le w_1 - 1$, and p be the largest node in T such that $p \equiv r \pmod{w_1}$ and $p \le b$. Then optimum solution $x_o(b) = (m,0,\ldots,0) + x_o(p)$ where $b = m\ w_1 + p$.

We now give a high-level description of the Extended M-T Spanning Tree algorithm.

Extended M-T Spanning Tree Algorithm

Goal: To solve the knapsack problem for all instances of b.

Input: n pairs of non-negative numbers (v_j, w_j), where all w_j are integers with $w_n = 1$, all v_j are converted into integers with $v_n = 0$, and satisfying

$$\frac{v_j}{w_j} > \frac{v_{j+1}}{w_{j+1}} \quad \text{or} \quad \left(\frac{v_j}{w_j} = \frac{v_{j+1}}{w_{j+1}} \quad \text{and} \quad w_j < w_{j+1} \right)$$

Output: A minimum gap cost multi-track spanning tree T with node 0 as root and at least one node in each of the other $w_1 - 1$ tracks. Let p and p' be two nodes in T. If $p \pmod{w_1} = p' \pmod{w_1}$ and Cost(p) < Cost(p'), then p > p'. In other words,

the costs of the nodes belonging to the same track in T are decreasing as the b-values of these nodes increase. The node with the largest b-value in a track r is the b*(r) value for that track.

Procedure:

Initially all nodes are unlabeled (U) except the node 0, which is labeled and unscanned (LU). We have two sets of nodes

T: all nodes that are labeled and scanned (LS), $|T| < (n - 1)\ w_1$.

Q: all nodes that are labeled and unscanned (LU).

Both sets are empty initially.

Step 0. Associate node 0 with its label (0, 0, 0, undefined). State(0) = LU and add it to Q.

Step 1. Let p be the cheapest node in Q. Update the labels of all n–1 neighboring nodes q as follows:

Let $q = p + w_j$ and $q \pmod{w_1} = r$.

Set $Cost(q) = Cost(p) + w_j - v_j$, and Label(q) = (q, Cost(q), r, p).

Add q in Q if q satisfies one of the following conditions:

(i) there is no node belonging to Track r in $T \cup Q$, or

(ii) $Cost(q) < Cost(q')$ for all nodes q' in $T \cup Q$ such that $q \geq q'$ and $q' \pmod{w_1} = r$.

and remove any existing node x belonging Track r in $T \cup Q$ if $Cost(q) \leq Cost(x)$ and $q \leq x$.

Do nothing if condition (i) and (ii) are not satisfied.

[Comments: Since $w_j \neq w_1$, no node in a given track can use w_j more than once. Thus, a node q in a given track r with its b-value less than b*(r) can only stay in the track due to (ii). If there are n types of items, there can be at most n-1 nodes in the same track with their costs greater than that of b*(r) and each of these nodes uses a different w_j to reach track r.

Note that condition (ii) also includes the case when q = q', which corresponds to the case when the node q is already in Q but with a cost > Cost(p) + $w_j - v_j$. When this happens, q's label is replaced by (q, Cost(q), r, p).]

Step 2. After updating all neighboring nodes of p, node p becomes LS. It is removed from Q and added to T.

Step 3. Terminate the algorithm if Q is empty, otherwise return to Step 1.

Lemma 6.1 The costs of nodes being added to T are monotonically increasing.

Proof.

Using arguments similar to those in the proof of Lemma 5.1, we can show, by induction, that the node p chosen in Step 1 of the algorithm must be cheaper than all nodes that are added to T after p.

Q.E.D.

Lemma 6.2 There is at least one node in every track in T at the termination of the Extended M-T spanning tree algorithm, and for $1 \le r \le w_1 - 1$, all the nodes $p_1, p_2, \ldots, p_k$ belonging to the track r in T satisfy the property

$r = p_1 < p_2 < \ldots < p_k \Rightarrow \text{Cost}(p_1) > \text{Cost}(p_2) > \ldots > \text{Cost}(p_k)$.

Proof.

It follows from the fact $w_n = 1$ and Step 1 that the node r ($1 \le r \le w_1 - 1$) will be considered at least once as the node q during the course of the algorithm. Since r has the smallest b-value among all nodes in track r, the operations in Step 1 can only update r's label and will not remove r from Q, until r is chosen as p in Step 1 and transfer from Q to T.

It follows from condition (ii) in Step 1 that all the nodes $p_1, p_2, \ldots, p_k$ belonging to the track r in $T \cup Q$ must satisfy the property

$p_1 < p_2 < \ldots < p_k \Rightarrow \text{Cost}(p_1) > \text{Cost}(p_2) > \ldots > \text{Cost}(p_k)$,

and the fact that Q is empty at the termination of the Extended M-T spanning tree algorithm implies that all the nodes $p_1, p_2, \ldots, p_k$ belonging to the track r in T must satisfy the property

$p_1 < p_2 < \ldots < p_k \Rightarrow Cost(p_1) > Cost(p_2) > \ldots > Cost(p_k)$

at the termination of the Extended M-T spanning tree algorithm.

Q.E.D.

Theorem 6.3 Let $b \equiv r \pmod{w_1}$ for some r, $0 \le r \le w_1 - 1$, and p be the largest node in T such that $p \equiv r \pmod{w_1}$ and $p \le b$. Then the gap cost of the optimum solution $x_o(b) = Cost(p)$.

Proof. (By Contradiction)

Let x be the first node being added to T that makes the theorem false and let y be the smallest integer not in T such that $y \pmod{w_1} = x \pmod{w_1} = r$ and x is the largest node in T that is less than y but $Cost(y) < Cost(x)$. Since y is the smallest integer that makes x to violate theorem, $Cost(y) < Cost(y - w_1) = Cost(x)$ and the optimum solution for y cannot contain any w_1. Then there must exist a path from node 0 to y that makes y cheaper than x. Let Parent(y) be the immediate predecessor of y along this path.

Since all gap costs are non-negative, Parent(y) is cheaper than y and it follows from Lemma 6.1 that Parent(y) is added to T before x. Using arguments similar to those in the proof of Theorem 5.3, we can show that x can never be added to T by the Extended M-T Spanning Tree algorithm, a contradiction.

Q.E.D.

Theorem 6.4

The Extended M-T spanning tree algorithm computes a minimum cost Extended M-T spanning tree in time $O(n^2 w_1 \log (n w_1))$.

Proof.

It takes O(n log n) time for the preprocessing to set up the gap costs and order the items according to their relative density.

We shall use $w_1 - 1$ dynamic search trees (such as the AVL-trees), one tree for each track r ($1 \le r \le w_1 - 1$), to keep track of all the nodes belonging to each track in Q, and

use a priority queue to keep track for the cheapest node among the best candidates in the $w_1 - 1$ tracks.

Since $w_j \neq w_1$ $(2 \leq j \leq n)$, no node in a given track can use the j type of item more than once. Hence, there can be at most n–1 nodes in each track in $Q \cup T$ all time, and at most $(w_1-1) \times (n-1)$ nodes can be chosen as p in Step 1 of the algorithm.

It takes O(1) time to find the cheapest node p, $O(\log w_1)$ time to remove it from the priority queue and O(log n) time to update the corresponding array.

Each node p can have at most n neighbors. It takes O(1) time to compute the new cost for each neighbor q, O(log n) to see if q satisfies condition (ii), O(log n) time to insert it into the right place in the array and $O(\log w_1)$ time to replace the node belonging to the same track in priority queue with q. Hence, it takes $O(n \log n + n \log w_1)$ time to check and insert p's n neighbors into the priority queue and the dynamic search trees.

Finally, each node can be removed exactly once from a dynamic search tree in Step 1 and it takes O(log n) to remove a node.

Hence, the algorithm takes a total of $O(n^2 w_1 \log w_1 + n^2 w_1 \log n) = O(n^2 w_1 \log (n w_1))$ time to compute the minimum cost Extended M-T spanning tree.

Q.E.D.

7. Discussions and Miscellaneous Comments

(1) The dynamic algorithm for solving a knapsack problem (1.1) based on (1.5) and (1.6) requires $\theta(n\ b)$ time and $\theta(n\ b)$ space, while our algorithm requires $O(n^2\ w_1 \log (n\ w_1))$ time and $\theta(n\ w_1)$ space, which are independent of b. The Extended M-T Spanning Tree algorithm will result in big savings if one wants to solve many different knapsack problems that involve the same set of items but with different knapsack capacities.

(2) The Extended M-T Spanning Tree approach is an improvement upon previous approaches that find the critical values for the periodic solutions using (1.9) and (1.10). The inequality (1.9) is a sufficient condition and usually provides a high estimate for b**. Moreover, it does not work when $\rho_1 = \rho_2$. It is hard to analyze the time complexity of (1.10) accurately because it depends on checking successive intervals of period w_1.

The formula $b^{**} = (w_1 - 1) \times w_2$ when $\rho_1 = \rho_2$ and $GCD(w_1, w_2) = 1$ is a necessary and sufficient condition. It can provide lower estimates for b** if $\rho_1 = \rho_2 = \ldots = \rho_k$.

(3) In Section 4, we presented several numerical examples to show that certain items can never be present in any optimum knapsack solution. We can further reduce the computational time of the Extended M-T Spanning Tree algorithm by first eliminate these items from the input.

For example, if $b = m\ w_j$ for some positive constant m, then the profit by using the first items only = $\left\lfloor \frac{m w_j}{w_1} \right\rfloor \times v_1$, and the profit by using the j^{th} items only = $m\ v_j$.

Hence, we can eliminate the j^{th} items for $b \geq m\ w_j$ if $\left\lfloor \frac{m w_j}{w_1} \right\rfloor \times v_1 \geq m\ v_j$.

Another way to eliminate the j^{th} items is that if there exists an i^{th} item such that $w_i < w_j$ and $GapCost(w_i) + (w_j - w_i) \times GapCost(w_n) < GapCost(w_j)$. In other words, we will not use the j^{th} item at all if we can get more profit by simply using the i^{th} item and leaving an unfilled capacity of $(w_j - w_i)$.

(4) Nature of the Greedy Algorithms. A greedy algorithm for solving a knapsack problem with $\rho_1 \geq \rho_2 \geq \ldots > \rho_n = 0$ would use the first item repeatedly until $w_1 > b \pmod{w_j} = b'$. Then it would use the second item until $w_2 > b' \pmod{w_j} = b''$, and so on.

A very interesting case is the minimum stamp problem (or the coin-changing problem) where there exists a one-point theorem [14] that defines the condition whether the greedy algorithm can come up with an optimum solution that minimizes the total area of stamps subject to the constraint that postage be met exactly.

While we do not have a similar one-point theorem for the knapsack problem, we can tell whether the greedy algorithms works based on the Extended M-T Spanning Tree.

(5) The knapsack problem belongs to a subset of NP-Complete problems, called the <u>number</u> problems [3]. The Extended M-T Spanning Tree algorithm is said to be a pseudo-polynomial time algorithm because the time complexity $O(n^2 w_1 \log (n w_1))$ becomes polynomial if $w_1 = O(\log |I|)$ where I is a reasonable encoding of the knapsack problem input. In particular, if $w_1 << b$, then the algorithm is polynomial, while the traditional $O(n b)$ dynamic programming algorithm is not.

If $w_1 > b$, the first item is never used in the optimum solutions for b and the second item becomes the best item. We can solve a reduced knapsack problem with one item less.

It will also be useful to find out if the optimum solutions to other NP-complete number problems also become periodic for large instances. Such discovery may result in similar algorithms for solving other NP-complete problems.

8. Conjectures

In this knapsack problem, we can find the b*(r) for r = 0, 1, 2, … , w_1–1, where the optimum solutions begin to be periodic. The total time complexity is O(n w_1 log w_1). On the other hand, to get the optimum solutions for b < b*(r), the total time complexity is O(n^2 w_1 log (n w_1)). In a nutshell, we spend more effort to solve a few small instances, and we spend less effort to solve infinite number of large instances. This phenomenon is not only true in this NP-Complete number problem but also true in many other problems (see examples and discussions below). We would like to state this phenomenon as the *"Large instances are easy, small instances are hard"* conjecture.

(1) Integer programs

We can solve an integer program (IP) by solving its associated linear program (LP) first. If the basic variables of the (LP) are not integers, we can map the non-basic columns and the right-hand side (b) of the LP into the elements of an Abelian group of order D, where D equals to the value of the determinant of the basic column. Gomory [7] showed that the integer program becomes a knapsack problem with one single constraint – the minimum cost representation of the group element corresponding to the column b in terms of the other group elements, and the knapsack problem can be solved by an algorithm of time-complexity O(D^2) developed by Hu [8, 9]. Roughly speaking, the goal is to use the non-basic columns as little as possible. However, the values of the non-basic columns would force the basic variables to become negative if b is very small.

(2) The partition problem

Another interesting problem is to partition a set of positive integers {a_1, a_2, …, a_n} into two subsets N_1 and N_2 so that the difference, $\left| \sum_{i \in N_1} a_i \quad - \quad \sum_{i \in N_2} a_i \right|$ of the two sums is minimum. Karmarker and Karp presented a method to solve the problem by a process that keeps replacing the largest two integers in the set, say a_i and a_j, by a new integer of size $|a_i - a_j|$. The process is repeated on the new set of n –

1 integers until only one item, say δ, remains. Then, through simple backtracking, the method reconstructs the subsets N_1 and N_2 where the difference of the two sums equals to δ [12]. This algorithm produces a near-optimum solution where $\max\left\{\sum_{i\in N_1} a_i, \quad \sum_{i\in N_2} a_i\right\}$ is at most 1/6 more than the larger sum of the two subsets in the optimal partition [2], and the worst instance occurs for a small instance of 5 numbers 3, 3, 2, 2, 2.

(3) Matrix-chain multiplication

Given a matrix chain $M = M_1 \times M_2 \times \ldots \times M_n$ where the total number of multiplications needed depends on the associative order of multiplying the matrices. A simple O(n) algorithm can be used to find an near-optimum order with a worst case error bound of 15% [10, 11]. Again, the worst instance occurs with 5 matrices and percentage of error becomes zero when n, the number of matrices, becomes very large.

Hence, one way to measure the difficulty of a problem is NOT to consider the worst instance of the problem but to consider all instances of the problem, as we have done here with b taking on all positive integer values.

References

1. S.A. Cook, "The Complexity of Theorem-proving Procedures", *Proc. 3rd Annual ACM Symposium of Theory of Computer* (1971), pp. 151-158.
2. M. Fischetti and S. Martello, "Worst-Case Analysis of the Differencing Method for the Partition Problem", Mathematical Programming, 37, (1987) 117-120.
3. M.R. Garey and D.S. Johnson, *Computer and Intractability,* Freeman Co., 1979.
4. P.C. Gilmore and R.E. Gomory, "A linear programming approach to the Cutting Stock Problem, Part I", *J. ORSA* (1961), pp. 849-859.
5. P.C. Gilmore and R.E. Gomory, "A linear programming approach to the Cutting Stock Problem, Part II", *J. ORSA* (1963), pp. 863-887.
6. P.C. Gilmore and R.E. Gomory, "The Theory of Computation of Knapsack Functions", *J. ORSA* (1966), pp. 1045-1074.
7. R.E. Gomory, "Some Polyhedra Related to Combinatorial Problems", *J. Linear Algebra and its Applications* (1969), pp. 451-558.
8. T.C. Hu, "On the asymptotic Integer Algorithm", *J. Linear Algebra and its Applications* (1970), Vol. 3, No. 3, pp. 279-294.
9. T.C. Hu, *Integer Programming and Network Flows,* Addison Wesley, 1982.
10. T.C. Hu and M.T. Shing, *Combinatorial Algorithms (Enlarged Second Edition),* Dover, 2002.
11. T.C. Hu and M. Shing, "An O(n) Algorithm to Find a Near-optimum Partition", *Journal of Algorithms* (1981), Vol. 2, No. 2, pp. 122-138.
12. N. Karmarkar and R. Karp, "The differencing method of set partitioning", Technical Report UCB/CSD 82/113, University of California, Berkeley, 1982.
13. R.M. Karp, "Reducibility Among Combinatorial Problems", in the book *Complexity of Computer Computations,* R.E. Miller and J.W. Thatcher (ed.), Plenum Press, 1972.
14. M. Magazine, G.L. Nemhauser and L.E. Trotter, Jr., "When the Greedy Solution Solves a Class of Knapsack Problems", *J. ORSA* (19756), pp. 207-217.
15. V.N. Shevchenko, "On the Property of Periodicity in the Knapsack Problem", *Modeling of Economic Processes,* Izdat Gorkov University, Gorky, 1981, pp. 36-38 (in Russian).

Appendix 1. Pseudo-code of the M-T Spanning Tree Algorithm

Input: n pairs of non-negative numbers (w_j, v_j), where all w_j are integers, $w_n = 1$, $\frac{v_1}{w_1} = 1$, and for $0 \le j \le n-1$, either $\frac{v_j}{w_j} > \frac{v_{j+1}}{w_{j+1}}$ or $(\frac{v_j}{w_j} = \frac{v_{j+1}}{w_{j+1}}$ and $w_j < w_{j+1})$.

Output: A minimum cost M-T spanning tree

```
Begin
0.  /*  Data Structures:
1.      Let T be the set of nodes that are "labeled and scanned" (LS) and Q
        be the set of nodes that are "labeled and unscanned" (LU).
2.      For 1 ≤ r ≤ n–1, let TempNode(r) return the node p in Q such that
        such that p ≡ r (mod w1).  TempNode(r) = undefined if not such node
        exists in Q.
3.      For 1 ≤ r ≤ n–1, let Cover(r) = true if there exists a node p in T such
        that p is label and scanned and p ≡ r (mod w1).
4.  */
5.  /*  Initialization */
6.      T := empty;
7.      Q := empty;
8.      Add node 0 to Q with Label(0) = (0, 0, 0, undefined) and State(0) =
        LU.
9.  /*  The main loop */
10.     While Q is not empty loop
11.         Remove p from Q such that p is cheaper than q for all node q ≠ p
            in Q;
12.         Cover(p mod w1) := true;
13.         For j from 1 through n–1 loop     /* update the neighbors of p */
14.             q := p + wj;
15.             Cost(q) := Cost(p) + GapCost(wj);
16.             r := q mod w1;
17.             Label(q) := (q, Cost(q), r, p);
18.             State(q) := LU;
```

```
19.             if not Cover(r) and TempNode(r) = undefined then
20.                 Add q to Q;
21.             else
22.                 if not Cover(r) and q is cheaper than TempNode(r) then
23.                     Replace TempNode(r) by q in Q.
24.                 end if;
25.             end if;
26.         end for;
27.         State(p) := LS
28.         Add p to T;
29.     end while;
30.     return(T);
End.
```