数理解析研究所講究録
第1734巻 2011年 93-100

Shift-and-Merge Technique for the DP Solution of the Time-Constrained Backpacker Problem

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

Byungjun You, Takeo Yamada
{g48095, yamada}@nda.ac.jp

1 Introduction

We are concerned with a variation of the *knapsack problem* (KP, [10, 11]), as well as of the *2-dimensional KP* (2KP, [12, 17]), where a 'backpacker' travels from a origin to a destination on a *directed acyclic graph* (DAG, [1, 6]). He collects items *en route* within the capacity of his knapsack and within a fixed time limit.

To formulate this problem, let $G = (V, E)$ be a DAG with node set $V = \{v_1, v_2, \cdots, v_n\}$ and arc set $E = \{e_1, e_2, \cdots, e_m\} \subseteq V \times V$. Node v_1 is the *origin*, and v_n is the *destination*, and we assume that there exists at least one path from v_1 to v_n. Each node $v_j \in V$ is associated with an item j of weight w_j and profit p_j, the capacity of the backpacker's knapsack is W, and the time limit is T. For $e = (v, v') \in E$, we write $\partial^+ e = v$ and $\partial^- e = v'$, and for $v \in V$ define the sets of *incoming* and *outgoing* arcs as $E^{\pm}(v) = \{e \in E | \partial^{\pm} e = v\}$ respectively. Each arc $e \in E$ is associated with non-negative time t_e to traverse, and the backpacker wishes to travel from v_1 to v_n within the time limit of T. Let us introduce decision variables as follows: $x_j = 1$ if the backpacker accepts item j, and $x_j = 0$ otherwise. Similarly, $y_e = 1$, if he takes a path that includes arc e, and $y_e = 0$ otherwise.

Then, the *time-constrained backpacker problem* (TCBP) is formulated mathematically as follows.

TCBP:

$$\text{Maximize} \sum_{j \in V} p_j x_j \tag{1}$$

$$\text{subject to} \sum_{j \in V} w_j x_j \leq W, \tag{2}$$

$$\sum_{e \in E} t_e y_e \leq T, \tag{3}$$

$$\sum_{e \in E^+(v_j)} y_e - \sum_{e \in E^-(v_j)} y_e = \begin{cases} 1, & \text{if } j = 1 \\ -1, & \text{if } j = n, \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$x_j \leq \sum_{e \in E^-(v_j)} y_e, \forall j \in V \setminus \{1\}, \tag{5}$$

$$x_j, y_e \in \{0, 1\}, \forall j \in V, \forall e \in E. \tag{6}$$

Here, (1) and (2) are as in ordinary KP, and (3) is the time constraint. Together with the flow conservation law (4), (6) determines a 0-1 vector that corresponds 1 to 1 to a path from v_1 to v_n. Inequality (5) means that only items on the path can be accepted. Without much loss of generality, we assume that problem data w_j, p_j $(\forall j \in V), t_e$ $(\forall e \in E)$, W and T are all positive integers, $w_j \leq W$ $(\forall j \in V)$, $\sum_{j \in V} w_j > W$, and $t_e \leq T$ $(\forall e \in E)$, $\sum_{e \in E} t_e > T$, since otherwise the problem is trivial. TCBP is $\mathcal{NP}$-hard, since it includes KP which is already $\mathcal{NP}$-hard [8].

Remark 1 *Node $v_j \in V$ is maximal if $E^-(v_j) = \emptyset$, and minimal if $E^+(v_j) = \emptyset$. We assume that v_1 is the only one maximal node, and v_n is the only one minimal node. That is, for all intermediate nodes we have $E^{\pm}(v_j) \neq \emptyset$ $(j = 2, \cdots, n-1)$, since otherwise no path exists from v_1 to v_n via v_j.* ∎

KP on a directed graph has been studied as the *precedence-constrained KP* (PCKP, [19]), or a *tree KP* (TKP, [5, 9]). Here, (partial) order relations are assumed among items, and there is a wide range of applications for these problems [15, 16]. However, in the backpacker problem we need to determine a set of items to be accepted, as well as a feasible path from v_1 to v_n. To our knowledge, no previous literature treated these two aspects simultaneously.

Since TCBP is a linear 0-1 programming problem, small instances may be solved using *mixed integer programming* (MIP, [18]) solvers such as CPLEX [7] or XPRESS-MP. For larger instances, however, it is often difficult to obtain exact solutions by such an approach. In this article, we first present a *dynamic programming* (DP) algorithm to solve TCBP. This is a pseudo-polynomial time algorithm which can solve only small-sized instances in practice. To solve larger instances, we present an improved 'shift-and-merge' DP algorithm (SMDP). This is an extension of the *list-type* DP [3, 13], which has been successfully applied to solve 1-dimensional KPs, to the 2-dimensional case.

Let TCBP be denoted explicitly as TCBP(W, T). Then, we note that DP and SMDP algorithms solve all TCBP(w, t)'s ($w \leq W, t \leq T$) in *one-pass*. Contrary to this, by using MIP solvers, we need to solve TCBP(w, t) from scratch for each values of $(w, t) \in [0, W] \times [0, T]$.

2 A dynamic programming algorithm

Without loss of generality, the nodes of G are assumed to be *topologically sorted* [6, 14], in the sense that $(v_i, v_j) \in E \Leftrightarrow i < j$. For node $v_i \in V$, we introduce $G_{i,k} = (V_{i,k}, E_{i,k})$ (See Figure 1) as the subgraph of G with the nodes and arcs restricted to the downstream of v_i through the first k arcs of $E^+(v_i)$. More precisely, let $E^+(v_i)$ be explicitly written as $E^+(v_i) = \{e_i^1, e_i^2, \cdots, e_i^{m_i}\}$, where $m_i = |E^+(v_i)|$. We say that $v \in V$ ($e \in E$, respectively) is in the *k-downstream* of v_i if (i) there exists a path from v_i to v (e, resp.), and (ii) the first arc in the path belongs to $\{e_i^1, e_i^2, \cdots, e_i^k\}$. Let $V_{i,k}$ ($E_{i,k}$, resp.) be the set of k-downstream nodes (arcs, resp.), and we define $G_{i,k} = (V_{i,k}, E_{i,k})$. We introduce a subproblem of TCBP on $G_{i,k}$ as follows.

TCBP$_{i,k}(w, t)$:

$$\text{Maximize} \sum_{j \in V_{i,k}} p_j x_j \tag{7}$$

$$\text{subject to} \sum_{j \in V_{i,k}} w_j x_j \leq w, \tag{8}$$

$$\sum_{e \in E_{i,k}} t_e y_e \leq t, \tag{9}$$

$$\sum_{e \in E^+(v_j)} y_e - \sum_{e \in E^-(v_j)} y_e = \begin{cases} 1, & \text{if } j = i \\ -1, & \text{if } j = n, \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

$$x_j \leq \sum_{e \in E^-(v_j)} y_e, \quad \forall j \in V_{i,k} \setminus \{i\}, \tag{11}$$

$$x_j, y_e \in \{0, 1\}, \quad \forall j \in V_{i,k}, \forall e \in E_{i,k}. \tag{12}$$

Here w is the *remaining* knapsack capacity, and t is the *remaining* travelling time for this subproblem. Let $z^\star_{i,k}(w, t)$ be the optimal objective value to TCBP$_{i,k}(w, t)$.

We put

$$z^\star_{i,0}(w, t) \equiv 0, \tag{13}$$

and for simplicity we write

$$z^\star_i(w, t) = z^\star_{i,m_i}(w, t). \tag{14}$$

Then, from the *principle of optimality* [4, 6], we have the following recurrence relation.

$$z^\star_{i,k}(w, t) = \max\{z^\star_{i,k-1}(w, t), z^\star_j(w, t - t_i^k), p_i + z^\star_j(w - w_i, t - t_i^k)\} \tag{15}$$

where $j = \partial^- e_i^k$ and $t_i^k = t_{e_i^k}$.

This means that the optimal objective value to $\mathrm{TCBP}_{i,k}(w,t)$ is given as the maximum of the following three alternatives.

A_1. Do not take e_i^k. In this case an arc in $\{e_i^1, \cdots, e_i^{k-1}\}$ is adopted with the corresponding objective value $z^\star_{i,k-1}(w,t)$.

A_2. Take e_i^k and go to node v_j without accepting item i.

A_3. Accept item i, and take e_i^k to go to node v_j.

For $i = n$ we have $z_n^\star(w,t) = p_n$ if $b \geq w_n$, and $z_n^\star(w,t) = 0$ otherwise. We compute (15) backward for $i = n-1, n-2, \cdots, 1$. Then, $z_1^\star(W,T)$ gives the optimal objective value to the original TCBP. Thus, TCBP is solved in $O(mWT)$ time and space. We call this **Algorithm** DP.

3 Shift-and-Merge method

In the DP algorithm for the ordinary KP, the optimal objective value $z_i^\star(w)$ is a non-decreasing step-function of w. Then, in list-type DP, instead of computing $z_i^\star(w)$ for all $w \in [0, W]$, we maintain the *list* of discontinuity points of $z_i^\star(w)$, and update this list as we compute backward for $i = n, n-1, \cdots, 1$.

Similarly, it is clear that in TCBP $z^\star_{i,k}(w,t)$ is a non-decreasing 2-dimensional step-function of (w,t). We call this *terrace function* (Figure 1). For a terrace function $z(w,t)$ defined on $[0,W] \times [0,T]$, we call (w,t,p) a *corner point* if $p = z(w,t)$, $z(w-1,t) < p$ and $z(w,t-1) < p$. Then, $z^\star_{i,k}(w,t)$ is completely characterized by the set of corner points of this function. We assume that this set of corner points is *lexicographically ordered* in the non-decreasing order of (w,t), i.e., if $C = (w,t,p)$ and $C' = (w',t',p')$ are corner points, we define $C \preceq C'$ if $t < t'$, or $t = t'$ and $w \leq w'$. By $\mathscr{L}_{i,k}$ we denote the set of corner points of $z^\star_{i,k}(w,t)$ arranged in the non-decreasing order of $\preceq$. Thus, $\mathscr{L}_{i,k}$ is the *list representation* of $z^\star_{i,k}(w,t)$.

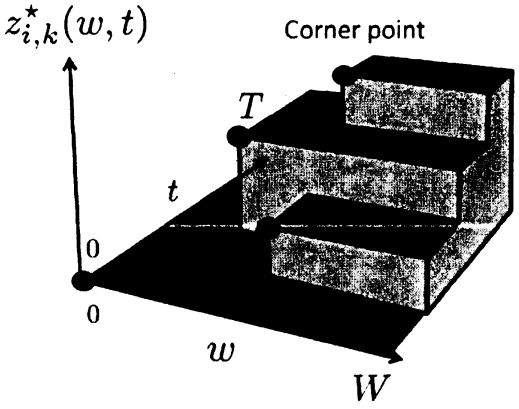


Figure 1: Terrace function and corner points

We now translate (15) in the language of lists as follows. Let $\mathscr{L}^1_{i,k}$ and $\mathscr{L}^2_{i,k}$ be the list representations of $z_j^\star(w, t-t_i^k)$ and $p_i + z_j^\star(w-w_i, t-t_i^k)$, respectively. These can be obtained from $\mathscr{L}_j$, the list representation of $z_j^\star(w,t)$, by applying *shift* operations as follows :

$$\begin{aligned}
\mathscr{L}^1_{i,k} &= \mathscr{L}_j \oplus (0, t_i^k, 0)) \\
&= \{(w, t+t_i^k, p) | (w,t,p) \in \mathscr{L}_j,\ t + t_i^k \leq T\}, \\
\mathscr{L}^2_{i,k} &= \mathscr{L}_j \oplus (w_i, t_i^k, p_i)) \\
&= \{(w+w_i, t+t_i^k, p+p_i) | (w,t,p) \in \mathscr{L}_j,\ t + t_i^k \leq T, w + w_i \leq W\}.
\end{aligned}$$

Next, we *merge* to obtain $\mathscr{L}'_{i,k} = \mathscr{L}_{i,k-1} \cup \mathscr{L}^1_{i,k} \cup \mathscr{L}^2_{i,k}$. However, simple merger does not work, since this may include *dominated* elements, and we have

$$\mathscr{L}_{i,k} = \mathscr{L}'_{i,k} \setminus \mathscr{E}_{i,k}, \tag{16}$$

where $\mathscr{E}_{i,k}$ is the set of *dominated* elements as defined below.

$$\mathscr{E}_{i,k} = \{(w,t,p) \in \mathscr{L}'_{i,k} \mid \exists (w',t',p') \in \mathscr{L}'_{i,k} \\ \text{s.t. } w \geq w', t \geq t', p \leq p',\ (w,t,p) \neq (w',t',p')\}.$$

We compute (16) for $i = n-1, \cdots, 1$ and for all $k \in E^+(i)$: Thus TCBP is solved by the following shift-and-merge DP (SMDP) algorithm.

Algorithm SMDP

Output: $\mathscr{L}_i$ $(i = 1, 2, \cdots, n)$.

Step 1. Set $i = n$ and $\mathscr{L}_n = \{(0,0,0), (w_n, 0, p_n)\}$.

Step 2. If $i \leq 0$, output $\mathscr{L}_i$ $(i = 1, 2, \cdots, n)$ and stop.

Step 3. Put $\mathscr{L}_{i,0} = \emptyset$.

Step 4. For $k = 1, 2, \cdots, m_i$ compute (16) through the *scanning-wall method.*

Step 5. Put $\mathscr{L}_i = \mathscr{L}_{i,m_i}$, $i = i - 1$, and go to **Step 2.**

To efficiently compute (16) and output the merged result $\mathscr{L}_{i,k}$ in the non-decreasing order of $\leq$ without explicitly applying *sort* operations, we propose the *scanning-wall method* as follows. Let $\mathscr{L}'_{i,k}$ be explicitly written as $\{C^0, C^1, \cdots, C^r\}$, where $C^0 = (0,0,0), C^l = (w^l_C, t^l_C, p^l_C)$ and $C^{l-1} \leq C^l$ $(l = 1, 2, \cdots, r)$. By $\bar{z}_l(w,t)$ we denote the terrace function corresponding to the (non-dominated) set of points $\{C^0, C^1, \cdots, C^{l-1}\}$, and $\hat{z}_l(w) = \bar{z}_l(w, t_l)$ is the *cross-section* of $\bar{z}_l(w,t)$ at the vertical *scanning wall* $t = t_l$ (See Figure 2). Then, $\hat{z}_l(w)$ is a non-decreasing step-function of w, which is completely characterized by the set of discontinuity points $\mathscr{D} = \{D^0, D^1, \cdots, D^s\}$ with $D^h = (w^h_D, p^h_D)$, $0 = w^0_D < w^1_D < \cdots < w^s_D$ and $0 = p^0_D < p^1_D < \cdots < p^s_D$.

We make use of this information to determine if C^l is dominated, and update $\mathscr{D}$ as we move from C^l to C^{l+1} in the following way. That is, if there exists some $D^h \in \mathscr{D}$ such that $w^h_D \leq w^l_C$ and $p^h_D \geq p^l_C$, C^l is dominated, and otherwise C^l is non-dominated. If C^l is non-dominated, we output C^l as an element of $\mathscr{L}_{i,k}$, and insert it into $\mathscr{D}$. At the same time all those elements of $\mathscr{D}$ which are dominated by C^l are removed from $\mathscr{D}$, and we move to the next C^{l+1}. Thus, in Step 4 of SMDP, the scanning-wall method can be written explicitly as follows.

Scanning-wall method

(i) Initialize. Put $\mathscr{D} = \mathscr{L}_{i,k} = \emptyset$.

(ii) For $l = 1$ to $|\mathscr{L}'_{i,k}|$ do

- If $C^l \in \mathscr{L}'_{i,k}$ is dominated by some $D^h \in \mathscr{D}$, skip C^l and go to next l.
- $\mathscr{L}_{i,k} \leftarrow \mathscr{L}_{i,k} \cup \{C^l\}$, $\mathscr{D} \leftarrow \mathscr{D} \cup \{C^l\}$.
- Eliminate all the elements of $\mathscr{D}$ which are dominated by C^l.

The behavior of this scanning-wall method is depicted in Figure 2.

Remark 2 *SMDP computes the optimal objective value $z^\star_i(w,t)$, but produces neither optimal $x^\star_i(w,t)$ nor $y^\star_e(w,t)$. To obtain these, we make elements of each list be of the form $(w,t,p,x^\star,y^\star)$, where $(x^\star,y^\star)$ is the*

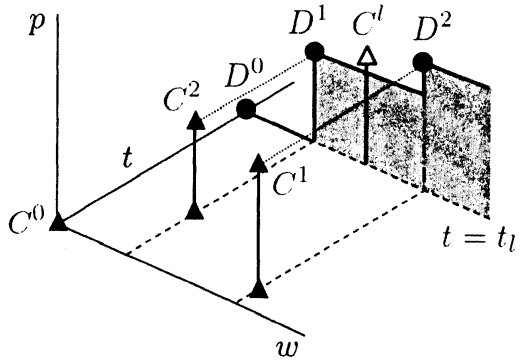


Figure 2: The scanning-wall method.

optimal solution to $TCBP_{i,k}(w,t)$ at the corner point (w,t,p). Then, in computing (16), $(x^\star, y^\star)$ is inherited from $\mathcal{L}_{i,k-1}$, in case of A_1, while this is given as $(0, e_i^k)$ and $(1, e_i^k)$, corresponding to the cases of A_2 and A_3, respectively. ∎

Remark 3 *We may accelerate SMDP by skipping some computation as follows. Let $\underline{t}^i$ be the shortest time from v_1 to v_i. Then, in (ii) of the scanning-wall method if $t_C^l + \underline{t}^i > T$ is satisfied for $C^l = (w_C^l, t_C^l, p_C^l)$, we can skip this point and go to the next C^{l+1}. We call this Shortest-Path test.* ∎

4 Numerical experiments

We implemented SMDP in ANSI-C language and conducted computation on a Dell Precision T7400 workstation (CPU: Xeon X5482 Quad-Core×2, 3.20GHz) for various types and sizes of instances. We also compare the performance of this algorithm against the direct solution by CPLEX [7].

4.1 Design of experiments

The instances tested are WIDE and TALL. These are prepared as follows.

- WIDE. For each pair (i, j) satisfying $1 \le i < j \le n$, we generate an arc (v_i, v_j) randomly with probability $d/(n-1)$.
- TALL. For each pair (i, j) satisfying $1 \le i < j \le \min\{n, i + d \cdot Intvl\}$, we generate an arc (v_i, v_j) randomly with probability $1/Intvl$. Here *Intvl* is a parameter to control the range of arcs as $|j - i| \le d \cdot Intvl$.

Also, d is an integer parameter that controls the number of arcs in G. Since we have $n(n-1)/2$ pairs of nodes, at this stage we have about $m \approx nd/2$ arcs, and the average degree at each node is approximately d for WIDE instances. For TALL instances, this is approximately $m \cong nd$, and thus, we call d the *degree parameter*. Next, for each maximal node $v_i \neq v_1$, we pick up node v_j $(j < i)$ at random and add arc (v_j, v_i) to E. Thus, no maximal nodes remain in G other than v_1. Similarly, we make all nodes other than v_n non-minimal. For a DAG, let *height* and *dist* be the maximum and minimum numbers of steps between v_1 and v_n, respectively. Then, Table 1 gives a summary of the characteristics of WIDE and TALL instances, where *Intvl* is fixed at 100. Thus, the height increases with n in TALL instances, while it remains relatively small in WIDE case, as seen in Figure 6.

We generate the data for items according to the following scheme. The weight w_j is distributed uniformly random (RAND) over the integer interval [1, 100], and profit p_j is related to w_j in the following ways.

- Uncorrelated case (UNCOR): $p_j :=$ RAND [1, 100], independent of w_j.
- Weakly correlated case (WEAK): $p_j := w_j +$ RAND[1, 20].
- Strongly correlated case (STRONG): $p_j := w_j + 20$.

Table 1: Instance characteristics.

d	n	WIDE			TALL		
		m	*height*	*dist*	m	*height*	*dist*
3	2000	4283	26	5	5946	56	8
	8000	17023	32	6	24587	182	10
	32000	68158	39	7	98451	708	10
6	2000	6615	32	5	10506	55	6
	8000	26575	40	5	46349	182	9
	32000	106418	45	6	190242	699	11
9	2000	9377	37	5	14266	62	5
	8000	37739	42	5	67926	200	11
	32000	151182	51	6	284020	716	12

4.2 MIP results

Table 2 summarizes the computation using MIP solver CPLEX 11.1 [7]. We set the time limit at 1800 seconds, and for instances with $n \geq 32000$, the solver frequently failed to produce optimal solutions within this time limit.

Table 2: CPLEX results ($d = 3, W = 500, T = 500, Intvl = 100$) as average over 10 instances.

TYPE	n	m	UNCOR		WEAK		STRONG	
			z^*	CPU	z^*	CPU	z^*	CPU
WIDE	4000	8494	866.0	3.08	660.3	7.26	778.8	7.40
	16000	34118	886.0	67.64	670.5	98.43	797.1	54.70
TALL	4000	12200	1041.7	14.91	693.5	53.03	832.0	36.11
	16000	49135	1244.5	213.607	726.7	203.20	872.0	384.40

4.3 SMDP results

Tables 3 and 4 give the result of computation of SMDP for larger instances with $n \leq 64000$. Here we fix base case parameters at $d = 3, Intvl = 100, W = 500$ and $T = 500$, and each row of the tables is average over 10 randomly generated instances. By %accept we mean the percentage of items accepted along the optimal path. When solved to optimality, both of CPLEX and SMDP produced identical optimal values, although the solution obtained were sometimes distinct. Thus, SMDP solves larger instances much faster than CPLEX.

Table 3: SMDP result for WIDE instances.

n	m	UNCOR			WEAK			STRONG		
		z^*	%accept	CPU	z^*	%accept	CPU	z^*	%accept	CPU
2000	4283	907.6	84.71	0.32	661.5	83.07	0.39	776.5	86.08	0.46
4000	8494	866.0	82.58	0.62	660.3	82.75	0.66	778.8	91.13	0.71
8000	17023	929.2	81.84	1.24	677.7	82.36	1.35	801.3	81.59	1.46
16000	34118	886.0	81.91	2.45	670.5	84.84	2.56	797.1	88.91	2.69
32000	68158	945.5	85.71	4.92	677.3	78.76	5.00	788.8	86.23	5.09
64000	136376	941.8	79.62	9.83	667.4	67.96	9.91	795.3	85.54	10.03

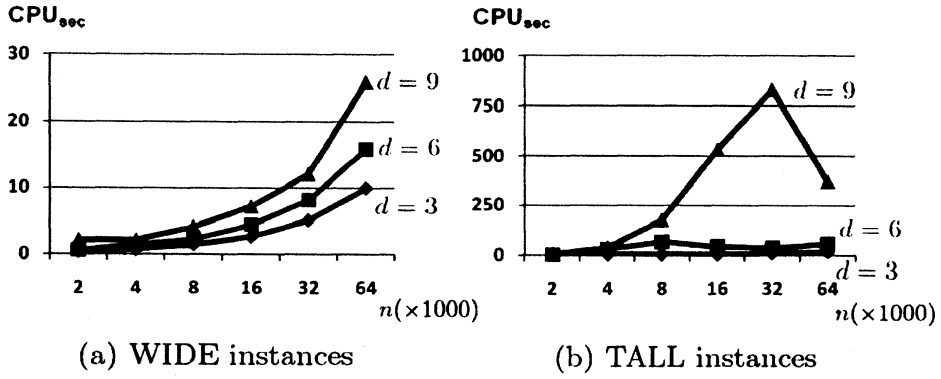
4.4 Sensitivity analysis

We now examine sensitivity of the SMDP algorithm with respect to the parameters W, T, d and the instance types (WIDE/TALL). Figure 3 depicts the CPU time of SMDP for WEAK case with the degree parameter

Table 4: SMDP result for TALL instances.

		UNCOR			WEAK			STRONG		
n	m	z^*	%accept	CPU	z^*	%accept	CPU	z^*	%accept	CPU
2000	5946	1094.7	79.04	0.47	700.0	81.80	0.65	832.0	85.20	0.92
4000	12200	1041.7	75.72	0.89	693.5	73.77	0.94	832.0	77.47	1.02
8000	24587	1025.6	71.64	1.78	696.9	71.05	1.83	833.7	75.48	1.89
16000	49135	1244.5	65.74	3.56	726.7	70.04	3.69	872.0	71.87	3.90
32000	98451	1232.4	64.76	7.18	732.5	69.99	7.54	885.4	72.81	8.03
64000	197706	1119.1	71.03	14.25	706.7	67.33	14.50	865.3	83.17	14.78

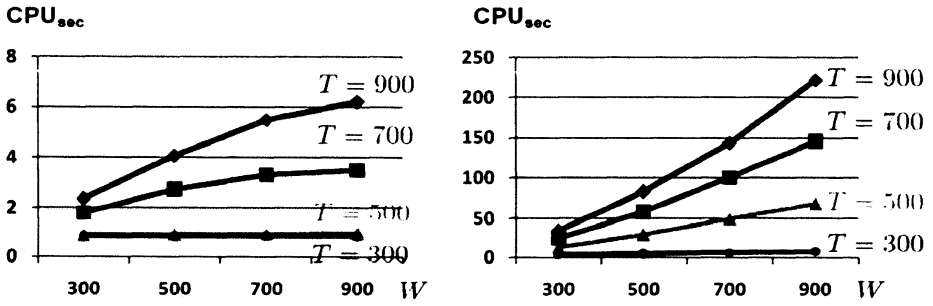
varied from $d = 3$ to $d = 9$. CPU time increases with d, since for larger d we have TCBP of larger m. TALL instances become harder to solve for $d = 9$, and in Figure 3 (b) it took longer CPU time for $n = 32000$ than for $n = 64000$. This is because more corner points were generated in the former case than in the latter.



(a) WIDE instances (b) TALL instances

Figure 3: Sensitivity analysis of parameter d

Finally Figure 4 gives the CPU time of SMDP for WEAK case with $n = 4000$ and $d = 6$ as the function of the knapsack capacity W for various values of T. In this case, CPU time increases moderately both with W and T.



(a) WIDE instances (b) TALL instances

Figure 4: Sensitivity with respect to W and T.

5 Conclusion

We have formulated the TCBP and presented a SMDP algorithm to solve this problem to optimality. This is an extension of the list-type DP, which has been successful for 1-dimensional KPs, to the 2-dimensional

case. The algorithm was implemented in ANSI-C language, and numerical experiments were carried out to evaluate the performance of the developed algorithm. We were able to solve TCBPs with up to 64000 items of various instance types within a few minutes in an ordinary computing environment. Computation was not much influenced by the change of the parameters W, T, d and instance types (WIDE/TALL), and this algorithm over-performed computation by MIP solvers.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms and Applications, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[2] S. Balev, N. Yanev, A. Freville, R. Andonov, A dynamic programming based reduction procedure for the multidimensional 0-1 knapsack problem, European Journal of Operational Research 186 (2008) 63-76.

[3] D. El Baz, M. Elkihel, Load balancing methods and parallel dynamic programming algorithm using dominance technique applied to the 0-1 knapsack problem, Journal of Parallel and Distributed Computing 65 (2005) 74-84.

[4] R. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, 1957.

[5] G. Cho, D.X. Shaw, A depth-first dynamic programming algorithm for the tree knapsack problem, INFORMS Journal on Computing 9 (1997) 431-438.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 2nd Edition, MIT Press, MA, 2001.

[7] CPLEX 11.1, ILOG, http://www.ilog.com/ products/cplex, 2009.

[8] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman and Company, San Francisco, CA, 1979.

[9] D.S. Johnson, K.A. Niemi, On knapsacks, partitions, and a new dynamic programming technique for trees, Mathematics of Operations Research 8 (1983) 1-14.

[10] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer, Berlin, 2004.

[11] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley and Sons, New York, NY, 1990.

[12] S. Martello, P. Toth, An exact algorithm for the two-constraint 0-1 knapsack problem, Operation Research 51(2003) 826-835.

[13] G.L. Nemhauser, Z. Ullmann, Discrete dynamic programming and capital allocation, Management Science 15 (1969) 494-505.

[14] R. Sedgewick, Algorithms in C, 3rd Edition, Addison-Wesley, Reading, 1998.

[15] D.X. Shaw, G. Cho, H. Chang, A depth-first dynamic programming procedure for the extended tree knapsack problem in local access network design. Telecommunication Systems 7 (1997) 29-43.

[16] K.E. Stecke, I. Kim, A study of part type selections approaches for short-term production planning, International Journal of Flexible manufacturing Systems 1 (1988) 7-29.

[17] B. Thiongane, A. Nagih, G. Plateau, Lagrangian heuristics combined with optimization for the 0-1 bidimensional knapsack problem, Discrete Applied Mathematics 154 (2006) 2200-2211.

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

[19] B.-J. You, T. Yamada, A pegging approach to the precedence-constrained knapsack problem, European Journal of Operational Research 183 (2007) 618-632.