Multi-Period Multi-Dimensional Knapsack Problem: Combinatorial Structure and Meta-Heuristics

Hoong Chuin LAU and Min Kwang LIM
School of Computing
National University of Singapore,
3 Science Drive 2, Singapore 117543.
lauhc,limmk@comp.nus.edu.sg

Yuyue SONG
School of Management,
McGill University, Canada.
yuyue.song@mcgill.ca

Abstract

This paper is concerned with the multi-period multi-dimensional knapsack problem (MPMKP), which is motivated by a recent trend in logistics planning and scheduling, called Available-to-Promise. We first provide some properties for a special case, the multi-period single-dimensional knapsack problem. By using the insights obtained from these properties, we propose an efficient two-phase heuristics for solving the multi-dimensional problem. Next, we propose a novel Ant Colony Optimization algorithm that is adept at solving scheduling problems like the MPMKP. The quality of the solutions generated by our heuristics is verified through experiments where we demonstrate that the computational time is far superior compared with an integer programming approach in order to achieve solutions that are within a small percentage of the upper bound.

Keywords: Multi-dimensional knapsack problem, Logistics, Tabu search, Ant colony.

Introduction

The motivation of this work arises from a real-life e-logistics application known as the available-to-promise capability arising typically in an e-Commerce ordering system or an e-Market place. The key requirement for available-to-promise is that suppliers can "promise" to match real-time customer requests with the available inventories for all items (in the warehouse) over a certain planning horizon made up of multiple periods. Each request from customers consists of a collection of items, the quantity of each item, and a specific period when this request is due to be fulfilled. A request may be fulfilled at some period later (but not earlier) than the prescribed period. A late penalty cost will be incurred and it is period-dependent (usually, the later the fulfilment the higher the penalty). Hence the net profit, defined as the total revenue minus the total penalty cost, is also period-dependent.

We model the problem as a generalization of the multi-dimensional knapsack problem (MKP), which we call the multi-period multi-dimensional knapsack problem (MPMKP). MPMKP can be seen as a generalization of the well-studied dynamic capacitated lot-sizing problem in logistics, where there is a single customer (or retailer) and a single item to supply. This problem has been extended in different ways, including the work of Lau and Song (2002) which consider extension to multiple-retailers and time windows.

In a practical available-to-promise application, it is often useful for vendors to be able to make quick decisions for the availability to promise to their customers. Global optimality usually takes too long, given the NP-hard nature of the problem. Instead, the purpose of this paper is to propose and compare two heuristic approaches to get a near-optimal solution fast, to allow vendor to make quick decisions in offering availability-to-promise capability.

In this paper, we assume that all requests from customers and the quantity of each item arriving at the warehouse at the beginning of each period are known and deterministic. Our objective is to maximize the net profit by fulfilling a subset of all requests over the discrete planning horizon subject to item availability constraints in the warehouse. This problem can be modeled as an MPMKP, where any request can be regarded as an object to be added to an MKP of certain period while the available quantities of all items at the beginning of this period can be regarded as the capacity constraint of this knapsack. Each period's MKP is related to the one of the next period in that the remaining inventory of any item at the end of this period is carried forward to the next period given that this period is not the last one of the discrete planning horizon.

This paper is organized as follows. In the next section, we present a brief literature review and notations used in the paper. We begin by studying a special case, the multi-period single-dimensional knapsack problem. Some combinatorial properties and a heuristic with performance bound for this special case are presented. Next, we propose two meta-heuristic approaches for the MPMKP: an efficient Tabu Search, and a novel Ant Colony Optimization algorithm that handles the notion of time explicitly. The quality of the solutions obtained is verified through extensive numerical experiments on

extended benchmark problems.

We assume that the reader is familiar with the general concepts and terminology used in tabu search and ant colony optimation, and will not explain them in this paper.

Preliminaries

In this section, we first review the literature, then provide a mathematical formulation of the MPMKP. While many research works are available in the literature for the MKP, to our best knowledge, there is no published research work for the MPMKP. We review the literature for MKP by first considering exact algorithms followed by heuristic algorithms.

Shih (1979) presented a branch and bound algorithm for the MKP. Another branch and bound algorithm was developed by Gavish and Pirkul (1985) where various relaxations of the problem were used, their algorithm was compared with the exact algorithm of Shih (1979) and was found to be faster by at least one order of magnitude. Other previous exact algorithms, with only limited success reported, include the dynamic programming based methods.

Loulou and Michaelides (1979) presented a greedy-like method based on Toyoda's primal heuristic (1975). Balas and Martin (1980) used linear programming by relaxing the integrality constraints and heuristically setting the fractional solution to become integral while maintaining feasibility. Pirkul (1987) presented a heuristic algorithm which makes use of surrogate duality. Freville and Plateau (1994) presented an efficient preprocessing algorithm for the MKP. Freville and Plateau (1997) presented a heuristic for the special case, the bidimensional knapsack problem, their heuristic incorporated a number of components including problem reduction, a bound based surrogate relaxation and partial enumeration. A number of papers involving the use of tabu search to solve the MKP have appeared such as Dammeyer and Voss (1993) and Aboudi and Jornsten (1994).

Before we introduce the mathematical formulation of the MPMKP, let us introduce some notation:

- T : number of periods in the planning horizon
- N : total number of requests over the planning horizon
- M : total number of items
- t^j : the prescribed period where request j ($1 \leq j \leq N$) is due to be filled
- a_{ij} : order size for item i ($1 \leq i \leq M$) in request j ($1 \leq j \leq N$)
- b_{it} : incoming quantity of item i ($1 \leq i \leq M$) at the beginning of period t ($1 \leq t \leq T$)
- p_{jt} : net profit if request j ($1 \leq j \leq N$) is filled in period t ($1 \leq t \leq T$).

In this paper, we assume that the profit function p_{jt} of any request j is a concave function with respect to t ($1 \leq t \leq T$) and partial delivery is not allowed. The MPMKP becomes a linear program (which is easy to solve) if partial deliveries are allowed. The maximum profit of the request j will be attained at period t^j and the profit decreases monotonically from period t^j onwards with increasing time. Without loss of generality, we assume that for any request j, $p_{j;t^j} > p_{j;t^j+1}$ if $t^j < T$. Since we assume that the inventory of any item can be carried forward at no additional cost and there is no inventory capacity constraint at the warehouse, there is no need to consider early fulfillment, and hence we assume $p_{jt} = 0$ for all $1 \leq t < t^j$ if $t^j > 1$. The MPMKP can be mathematically formulated as:

$$\max Z = \sum_{j=1}^{N} \sum_{t=1}^{T} p_{jt} x_{jt} \quad \text{s.t.}$$

$$\sum_{j=1}^{N} a_{ij} x_{jt} + s_{it} = b_{it} + s_{i;t-1} \quad 1 \leq i \leq M; 1 \leq t \leq T \quad (1)$$

$$\sum_{t=1}^{T} x_{jt} \leq 1 \quad 1 \leq j \leq N \quad (2)$$

where $x_{jt} \in \{0, 1\}$, $s_{i0} = 0$, $s_{it} \geq 0$. If $x_{jt} = 1$, it means that request j is filled at period t, otherwise, it is not filled at period t. The s_{it} is a slack variable, i.e., the remaining inventory level at the end of period t for item i. Without loss of generality, we also assume that the initial inventory level at the beginning of the planning horizon is zero for all M items, i.e., $s_{i0} = 0$. Constraints (1) are inventory balance constraints. Constraints (2) mean that each request can be filled at no more than one period.

From the above formulation, it is obvious that if $T = 1$, the MPMKP is reduced to an MKP. If $T = 1$ and $M = 1$, it is reduced to the well known 0-1 knapsack problem (KP). Since the KP is NP-hard, the MPMKP is NP-hard.

Multi-period single-dimensional knapsack problem

In this section, we will study a special case, the multi-period single-dimensional knapsack problem (MPKP). We first provide some basic properties of the continuous relaxation in subsection 3.1. By using these properties, we propose a heuristic with performance bound for solving MPKP in subsection 3.2.

Continuous relaxation

Consider the continuous relaxation of the MPKP (i.e. x_{jt} is a real number and $0 \leq x_{jt} \leq 1$), which is denoted by C-MPKP. In the following, we will characterize the structure of an optimal solution for C-MPKP.

Lemma 1 Consider an optimal solution for C-MPKP and let $\mathcal{L} = \{t_1, t_2, \ldots, t_k\}$ where $1 \leq t_1 < t_2 < \cdots < t_k \leq T$ denote the set of periods with zero remaining inventory level at the end of the respective periods. Then,

1. Every request partially or fully satisfied at period t, for $t = 1, \ldots, t_1$, was a request which was due at period t.
2. For any $i \geq 2$, any request fully or partially satisfied at period $t_{i-1} + 1$ was due at one of the periods $1, \ldots, t_{i-1} + 1$. Any request satisfied at some period $t_{i-1} + 1 < t \leq t_i$ was due at period t.

Proof. Suppose that $X = \{x_{jt} \mid 1 \leq j \leq N;\ 1 \leq t \leq T\}$ is an optimal solution to C-MPKP, and let $t_1 < t_2 < \cdots < t_k$ be the periods, if any, such that the remaining inventory level at the end of any period t_i is zero and the remaining inventory level at any period $t \neq t_i (1 \leq i \leq k)$ is positive.

If $k = 0$, it is trivial. Hence we assume that $k > 0$.

For t_1, suppose that there is some request j due in period t_a and satisfied at period t_b where $1 \leq t_a < t_b \leq t_1$. Since $p_{j;t_a} > p_{j;t_b}$, and the remaining inventory at the end of period t_a is positive, we could remove some part of the request j from period t_b and fulfill it at period t_a, thus contradicting the optimality of X. Therefore, the result in part (1) is correct. A similar argument proves part (2). }

By using the results in Lemma 1, we propose the following greedy algorithm for solving C-MPKP. Note that each request can be partially or completely satisfied at some period.

Greedy algorithm A: (solving C-MPKP)

while (there are some unsatisfied requests and some inventories) **do**

1. Find request j^* and T_{j^*} such that $\frac{p_{j^*T_{j^*}}}{a_{1j^*}} = \max\{\frac{p_{jt}}{a_{1j}} \mid$ request j can be completely or partially satisfied in period $t\}$.
2. Satisfy request j^* at period T_{j^*} as much as possible.
3. Update the inventory of each item at the beginning of each period and the list of unsatisfied requests.

The feasible solution obtained by using the above algorithm may be not an optimal one for C-MPKP. But, if the profit function is a single point function, i.e., $p_{jt} = 0$ if $t \neq t^j$, we can show in the following that the solution obtained by using Greedy Algorithm A is an optimal one for C-MPKP. In practical terms, this means that late fulfillment does not add any profit. Even for this special profit function, the corresponding MPKP is a generalization of the 0-1 knapsack problem. Although the single point profit function for any request is very special, any solution approach for solving it should also consider the coordination between different periods, i.e., to decide whether the remaining inventory at current period should be used to fulfill some request due in current period or carried forward to fulfill some other request due in later period.

Lemma 2 If the profit function for each request is a single point function, C-MPKP is optimally solved by Greedy Algorithm A.

Proof. Note that the optimal solution structure described in Lemma 1 is further simplified if the profit function for each request is a single point function, i.e., the satisfied requests at each period t are due in at period t. If $k = 0$, it is trivial. In the following, we assume that $k > 0$ and call periods $(1, \ldots, t_1)$ as cycle 1, ..., periods $(t_{k-1} + 1, \ldots, t_k)$ as cycle k, and periods $(t_k + 1, \ldots, T)$ as cycle $k + 1$.

We prove this lemma by induction on T, the total number of periods.

For $T = 1$, the problem becomes the continuous relaxation of the 0-1 knapsack problem. If we replace any satisfied request by any other unsatisfied request, the total profit will not be increase. Hence it is true.

Suppose that for any $T \leq n - 1 (n \geq 2)$ the lemma is true. We shall show that it is also true for $T = n$.

If we replace some request fulfilled in the first cycle by some other unfilled request or carry some inventory consumed in the first cycle to the next cycle, the objective value of the new solution will be less than or equal to the objective value of the solution obtained by using Greedy Algorithm A. This can be seen from the procedure of the Greedy Algorithm A and the fact that any fulfilled request j must be fulfilled at period t^j (its due in period). Hence, there will be some optimal solution such that the first cycle is part of this solution. The remaining problem will be reduced to a C-MPKP over a planning horizon of $n - t_1$ periods. By using the assumption for $T \leq n - 1$, we complete the proof. }

A greedy heuristic for MPKP

We can generate two related MPKPs by modifying the profit function p_{jt} of any request $j (1 \leq j \leq N)$. From our assumption, we know that p_{jt} is concave and will attain the maximum at the period t^j. Thus, if we replace p_{jt} by $p^u_{jt} = p_{j;t^j}$, the objective value Z^u of an optimal solution will be an upper bound of Z^*, the optimal objective value of the original MPKP. Similarly, if we replace p_{jt} by the single point function $p^l_{jt} = p_{j;t^j}$ if $t = t^j$ and $p^l_{jt} = 0$ otherwise, the objective value Z^l of an optimal solution will be a lower bound of Z^*.

An upper bound of Z^u can be computed in the following way.

Lemma 3 If the profit function for any request $j (1 \leq j \leq N)$ is a constant function p^u_{jt}, we have $U - \max_{\{1 \leq j \leq N\}} p^u_{jt} \leq Z^u \leq U$ where U is the optimal objective value of the continuous relaxation.

Proof. As the profit of any request is constant with respect to period $t (1 \leq t \leq T)$, we can delay the fulfillment of all requests until the last period and carry forward all the inventories to the last period. Hence, the MPKP is equivalent to the 0-1 knapsack problem. Clearly, the deviation of the optimal value from the upper bound U obtained by continuous relaxation is at most $\max_{\{1 \leq j \leq N\}} p_{j1}$, since there is at most one request that is not completely satisfied in the latter solution, and that request contributes a profit value of at most $\max_{\{1 \leq j \leq N\}} p_{j1}$ to the objective. }

The Greedy Algorithm A can be easily modified to solve the MPKP. Henceforth, if some request cannot be completely satisfied at any period with positive net profit, we say that the best profit for this request is -1, otherwise, the best profit of any unsatisfied request j is defined as $\max\{p_{jt}\}$ request j can be completely satisfied at period t with positive net profit}. The maximizer is denoted as T_j. We say that a request is non-active if the best profit of the request is -1; otherwise, this request is active.

Greedy algorithm B: (solving MPKP)

1. Compute the best profit and T_j for every request j.
2. Set the requests with best profit -1 to non-active.
3. while (there are some active unsatisfied requests) do
   a. Find request j^* such that $\frac{p_{j^*T_{j^*}}}{a_{1j^*}} = \max\{\frac{p_{jT_j}}{a_{1j}}$ request j is active and unsatisfied in period $t\}$.
   b. Satisfy request j^* completely at period T_{j^*}.
   c. Update the inventory for each item at the beginning of each period, the best profit and the active status of any unsatisfied request.

It is obvious that Greedy Algorithm B is the discrete version of Greedy Algorithm A. By using Z^l and Z^u defined above, we can obtain the performance bound of the above Greedy Algorithm B for the MPKP.

Theorem 1 Let Z^* and Z^H be the objective values of the optimal solution and the feasible solution obtained by using Greedy Algorithm B, respectively. Then we have $Z^l \le Z^H \le Z^* \le Z^u$.

Proof. It is obvious that $Z^H \le Z^* \le Z^u$ from the definition of Z^u. About $Z^l \le Z^H$, we only note that the candidates of all active requests at each iteration of Greedy Algorithm B if the profit function of request $j(1 \le j \le N)$ is $p^l_{jt}(1 \le t \le T)$ is a subset of the candidates of all active requests at the same iteration of Greedy Algorithm B if the profit function of request $j(1 \le j \le N)$ is $p_{jt}(1 \le t \le T)$. }

Multi-period multi-dimensional knapsack problem

In this section, we will provide a two-phase solution approach for the MPMKP by using the insights obtained in the previous section. The first phase is a greedy heuristic which is a generalization of Greedy Algorithm B, followed by a local improvement phase via tabu search. We then propose another heuristic based on ant colony optimization, and compare the results obtained in the subsequent section.

A greedy heuristic for MPMKP

The basic idea is first to transform an MPMKP to an MPKP, and then to obtain a feasible solution of the MPMKP by adapting Greedy algorithm B proposed for the MPKP in section 3. This idea is not new, almost all solution approaches for solving MKPs come from this idea: first transform an MKP to a KP by proposing a multiplier for each item and computing the weighted capacity of the knapsack and the weighted weight of each object, then provide a feasible solution for the MKP by using some solution approach for the KP.

In our following Greedy Algorithm C, the multiplier of item $m(1 \le m \le M)$ at period t is simply defined as the used inventory r_m for item m if there is at least one request satisfied at period t, otherwise, we simply define the multiplier for each item is 1. Therefore, the multiplier does not depend on the request. It is the same for all requests at period t. Suppose that $u_{mt}(1 \le m \le M;\ 1 \le t \le T)$ are the multipliers for item m in period t. The weighted weight w_{jt} of request j in period t is defined as:

$$w_{jt} = \sum_{i=1}^{M} u_{it} a_{ij}.$$

We call request j is active in period t if request j can be satisfied at period t with positive net profit. Otherwise, we call the request j is non-active in period t. Similarly, if request j can be satisfied in at least one period with positive net profit, we call it active, otherwise, we call it non-active.

Greedy algorithm C: (solving MPMKP)

1. Set the active status for all requests.
2. Compute the multipliers for all resources at each period.
3. while (there are some active unsatisfied requests) do
   a. Find request j^* and period t^* such that $\frac{p_{j^*t^*}}{w_{j^*t^*}} = \max\{\frac{p_{jt}}{w_{jt}}$ request j is active and unsatisfied in period $t\}$.
   b. Satisfy request j^* at period t^*.
   c. Update the inventory for each item, the multipliers for all items at period t^*, and update the active status of each unsatisfied request.

Tabu Search Improvement for MPMKP

By using Greedy algorithm C described in subsection 4.1, we obtain an initial feasible solution for the MPMKP. In the following we propose a tabu search improvement phase. First, we will introduce two operators used in this tabu search approach: the relocate operator and the exchange operator.

Relocate Operator: Suppose request $i(1 \le i \le N)$ is satisfied at period $t_i(1 \le t_i \le T)$. We remove this request from period t_i and satisfy it at period $t' \ne t_i(1 \le t' \le T)$. If this can be done feasibly, we define the profit saving of this relocate operation as $(p_{it'} - p_{it_i})$, otherwise, we define the profit saving of this relocate operation as -1.

Exchange Operator: Suppose there are two requests i and j which are satisfied at periods t_i and $t_j(\ne t_i)$, respectively. We remove request i from period t_i and satisfy it at period t_j, at the same time, we also remove request j from period t_j and satisfy

it at period t_i. If this can be done feasibly, we define the profit saving of this exchange operation as $(p_{i;t_i} + p_{j;t_i}) - (p_{i;t_i} + p_{j;t_j})$, otherwise, we define the profit saving as -1.

In the following, we simply refer to a relocate operation or an exchange operation as a move. The move A is better than the move B if the saving profit of A is greater than the profit saving of B. The traditional greedy local improvement approach has a very obvious drawback: the improving process will stop at a local maximum point. Hence we propose the following simple tabu list and aspiration criteria to avoid this situation.

Tabu List: It is a list of moves executed during the last L steps where L is the length of the list. The status of each move is either tabu or non-tabu. If a move is in the tabu list, the status of this move will be tabu, otherwise, its status is non-tabu.

Aspired Criteria: Each move A is associated with an aspired profit value x_A initialized to zero. When a move is being executed, there are two possibilities. If $x_A = 0$, we call move A aspired and update $x_A = \max\{$current total profit + profit saving of A, $0\}$. If $x_A > 0$, we call move A aspired if current total profit + profit saving of A $> x_A$ and update $x_A = \max\{$current total profit + profit saving of A, $x_A\}$.

Our proposed Tabu Search Algorithm (TSA) can be described as follows. In this algorithm, break is a binary variable to indicate whether the algorithm should stop, time records the total CPU seconds consumed, iterationNo records the total iteration numbers in Step 2, and the stagNo records the total iteration number from the last move executed whose saving profit is negative, respectively. The variables timeLimit, staglimit, and iterationLimit are the time limit, the staged iteration limit, and the total iteration limit, respectively.

Tabu search algorithm (TSA):

1. **Initialization:** break =: false, stagNo =: 0, iterationNo =: 0, time =: 0, timeLimit =: TL, stagLimit =: SL, and iterationLimit =: IL;
2. **while** (break $\neq$ true) **do**

(a). Compute the best move. If the profit saving of the best move is -1, set break =: true and Stop;

(b). Update the aspired profit of the best move in Step 2(a).

(c). Execute the best move.

(d). Update the tabu list.

(e). Update the values of all the variables: break, stagNo, iterationNo, and time.

Step 1 is an initialization step to set the initial values of all variables. In Step 2(a), we compute the best move among all the possible moves which are either non-tabu or aspired. If the profit saving of the best move in Step 2(a) is greater than the aspired profit of this move, then update the aspired profit of this move to the profit saving of this move in Step 2(b). In Step 2(d), delete the move from the head of the tabu list and add the best move in Step 2(a) to the tail of the tabu list. In Step 2(e), the values of the variables stagNo, iterationNo, and time will be updated accordingly. About the value of break, if stagNo > stagLimit or iterationNO > iterationLimit or time > timeLimit, we set break = true, otherwise, we set break = false.

Ant Colony Optimization for MPMKP

Ant Colony Optimization (ACO) (see (Dorigo and Di Caro (1999)) is a meta-heuristic that is inspired from the observation of nature, in this instance the foraging behavior of a colony of ants. Typically the pheromone trail, a key component in the ACO operation, is specified using a node-to-node matrix, which is not appropriate for scheduling problems involving time. We propose in this paper a new scheme for the pheromone trail that explicitly handle the notion of time to improve the performance of ACO for scheduling problems.

Intuitively, one scheme that exploits the basic operation of ACO is what we call Scheme A, such that in addition to the standard pheromone trail presenting a value from one node to another, we add a third dimension of time to the pheromone trail structure. However, Scheme A suffers an obvious problem in that it is computationally intensive. We notice this supposing a generic ACO implementation using x iterations, y ants, and n nodes. The performance of the algorithm hence is $O(xyn^2)$, since the operation uses a n^2 pheromone trail calculation, which may be improved to $O(xyn \log n)$. By adding the time dimension of size T, the algorithm complexity becomes $O(xyn \log nT)$.

In this paper, we propose the following scheme B for solving problems in the time-period dimension: Modify the pheromone trail from a node-to-node trail to a node-to-period trail. This pheromone trail will specify the attractiveness of placing a node in a time-period. The computational complexity becomes $O(xy \log nT)$. The complexity is reduced, since typically the number of time-periods T is smaller than the number of nodes n. Furthermore, this structure is logically better than in dealing directly with time, since it directly informs the algorithm where the best period to place nodes is.

The computation of the transition probabilities is given as follows:

$$\tau_{ij} = \begin{cases} \frac{p_{ij}}{A_{ij}}; & \text{if } A_{ij} > 0; \\ 0; & \text{otherwise}; \end{cases}$$

$$A_{ij} = \sum_{j=1}^{M} \frac{a_{ij}}{c_j}$$

where τ_{ij} represents the local heuristics for moving from node i to node j; A_{ij} represents the sum tightness of all items ($1 \leq j \leq M$) on request i; and c_j represents the cumulative item left at the end of the last period taking into consideration the current state, i.e., after subtracting all the items that have been allocated.

Our proposed approach ACO Scheme B, using the same variables as TSA, is given as follows:

Ant Colony Optimization algorithm (ACO):

1. **Initialization:** break =: false, stagNo =: 0, iterationNo =: 0, time =: 0, timeLimit =: TL, stagLimit =: SL, **and** iterationLimit =: IL;
2. **while (**break **<>** true**) perform step 3.**
3. **for each ant while solution not complete do:**

(a). Compute transition probability.

(b). Move to next state based on transition probability.

(c). Update local pheremone trail.

4. **Update the values of all the variables:** break, stagNo, iterationNo, **and** time.

Like TSA, if stagNo > stagLimit **or** iterationNO > iterationLimit **or** time > timeLimit, **we set** break = true.

Experimental Results

We now present numerical experiments to verify the algorithms presented in section 4. First, we explain how to generate the test cases of the MPMKP from the benchmark test cases for the MKP in the OR Library (available online at http://www.ms.ic.ac.uk/jeb/pub/). Then, we compare the quality of the solutions generated by our proposed heuristics against those generated using integer programming with CPLEX.

The OR Library has a set of 0-1 MKPs generated with varying size and tightness, which is described by the parameter α. The benchmark problems are divided into 9 sets, each with a given M (the number of items) and n (the number of requests). There are 30 test cases within a set. The first 10 test cases are generated with a tightness ratio with $\alpha = 0.25$, the next 10 test cases are generated with $\alpha = 0.5$, and the last 10 test cases are generated with $\alpha = 0.75$. The parameters used in each test case are generated by using the following formula

$$a_{ij} = U(0, 1000)$$

$$b_i = \alpha \sum_{j=1}^{n} a_{ij}, \alpha \in \{0.25, 0.5, 0.75\}$$

$$p_j = \sum_{i=1}^{m} a_{ij}/m + 500U(0, 1)$$

where b_i is the total inventory for item i ($1 \leq i \leq M$), p_j is the profit of object j ($1 \leq j \leq n$), and a_{ij} is the consumed quantity of item i if object j is fulfilled.

Since this set of benchmark is for 0-1 MKPs, they can only be used as test cases for the single-period multi-dimensional knapsack problem. We generated 6 sets of 7-period MPMKPs (equivalent to 7 days for a week). For example, in the following table, we show how to generate a set of 9 test cases for the MPMKP over a 7-period planning horizon from the 30 test cases in mX ($4 \leq X \leq 9$) test set. The first test case is generated using the first 7 test cases in mX (abbrev. mX). The rest of the test cases are generated by choosing other 7 test cases from mX test set which are described in Table 1.

Table 1: The nine test cases generated from mX

Name	mX test case chosen	α
mX-1-n-M	1,2,3,4,5,6,7	0.25
mX-2-n-M	2,3,4,5,6,7,8	0.25
mX-3-n-M	3,4,5,6,7,8,9	0.25
mX-4-n-M	11,12,13,14,15,16,17	0.50
mX-5-n-M	12,13,14,15,16,17,18	0.50
mX-6-n-M	13,14,15,16,17,18,19	0.50
mX-7-n-M	21,22,23,24,25,26,27	0.75
mX-8-n-M	22,23,24,25,26,27,28	0.75
mX-9-n-M	23,24,25,26,27,28,29	0.75

The main parameters in each group are n, the number of requests in each periods, and M, the number of items. As there are n requests due in each period t ($1 \leq t \leq 7$), the profit for any of these n requests in period t is the same as the OR library. We simply define the profit function $p_{jt'} = 0$ if $t' < t$ and $p_{jt'} = p_{jt} - 0.1(t' - t)p_{jt}$ if $t' \geq t$.

We denote the objective value produced by using CPLEX by Z_A and the objective value of our TS and ACO heuristics by Z_{TS} and Z_{AC} respectively(rounded to the nearest integer). The upper bound produced by CPLEX is denoted by UB. The CPU seconds consumed are respectively denoted by T_A, T_{TS} and T_{AC} respectively(rounded to 2 decimal places). We use the CPLEX integer programming solver to solve all the generated test cases with a maximum CPU time of two hours (7200s) for each test case and stop the solver once the solution is found within 0.5% from the upper bound.

In the interest of space, we will present only one set of large-scale test cases (of size 250).

The results in Table 2 show that CPLEX obtains better solutions compared to our heuristics. This is generally true for all the cases in mX ($1 \leq X \leq 9$). However, Table 3 tabulates the time taken to achieve the results, which is the objective of this experiment, and Table 4 shows the equivalent running time for a larger size set of test cases, m9, with 500 customers per period.

Table 3: CPU time taken for m8 test cases

Name	T_A	T_{TS}	T_{AC}
m8-1-250-30	7200	145	18
m8-2-250-30	7200	140	55
m8-3-250-30	7200	165	19
m8-4-250-30	7200	560	33
m8-5-250-30	7200	599	33
m8-6-250-30	3292	559	34
m8-7-250-30	3455	816	66
m8-8-250-30	666	858	68
m8-9-250-30	749	838	46

Tables 3 and 4 sufficiently show the time efficiency with which the TS and ACO obtains near-optimal solutions (within 6% deviation of the upper bound). In practical

Table 2: Results of test cases generated from m8

Name	UB	Z_A	Z_{TS}	Z_{AC}
m8-1-250-30	416534	413221	399999	401184
m8-2-250-30	416728	414378	401153	405882
m8-3-250-30	416489	413966	399017	403577
m8-4-250-30	775929	770227	747851	752445
m8-5-250-30	773089	768876	750229	745073
m8-6-250-30	770849	767548	747512	749205
m8-7-250-30	1075402	1070659	1039310	1031131
m8-8-250-30	1078888	1075581	1054980	1045941
m8-9-250-30	1078300	1075166	1052480	1039319

Table 4: CPU time taken for m9 test cases

Name	T_A	T_{TS}	T_{AC}
m9-1-500-30	7200	286	78
m9-2-500-30	7200	288	84
m9-3-500-30	7200	286	80
m9-4-500-30	4009	500	165
m9-5-500-30	7200	507	132
m9-6-500-30	3319	502	133
m9-7-500-30	5875	642	172
m9-8-500-30	6163	641	172
m9-9-500-30	5265	639	170

terms, this means that TSA and ACO generally perform much better than CPLEX in providing a fast estimate for the vendor to make a quick decision on Availability-to-promise, which is an important consideration in e-Business today. Furthermore, the capability of CPLEX to provide a good estimate at a fixed time is apparently random, shown by the variation in T_A. On the other hand, ACO which operates by solution reconstruction of each ant, coupled with an effective scheme, local heuristics and parameters tuning, is even faster in gauging the optimality of the problem overall. Both approaches are also able to provide a constant estimation of the problem optimality at a fixed (early) time, though the random nature of ACO makes it a slightly less suitable candidate in such situations. This slight disadvantage is however easily redeemed by its speed. This overall performance enhances our argument that Scheme B is an effective ACO scheme for hard scheduling problems.

It is also observed that TSA is somewhat dependent on the tightness ratio of the problem, rather than on the problem size. ACO is less sensitive than TSA to the tightness ratio. The disadvantage of ACO, however, is in the intrinsic way it operates using a new solution construction in all iterations, and hence it is more sensitive to problem size. Hence, when the problem size is very large, such as when number of requests is much greater than 500, or the time-period much greater than 7, TSA will prove to be a more effective approach, although in the set of test cases experimented (m9 being the largest problem size), ACO still outperform TSA and CPLEX in terms of the ability to allow for quick decision.

Acknowledgement. The authors like to thank Seet Chong Lua and Roland Yap for contributions on an earlier draft of this paper.

References

Aboudi, R. and K. Jornsten. (1994). Tabu search for general zero-one integer programs using pivot and complement heuristic, ORSA Journal on Computing, 6, 82-93.

Balas, E. and C. H. Martin. (1980). Pivot and complement{A heuristic for 0-1 programming, Management Science, 26, 86-96.

Dammeyer, F. and S. Voss. (1993). Dynamic tabu list management using reverse elimination method, Annals of Operations Research, 41, 31-46.

Dorigo and Di Caro, (1999). Ant Colony Optimization: A New Meta-Heuristic, Proc. 1999 Congress on Evolutionary Computation, 1470-1477.

Freville, A. and G. Plateau. (1994). An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem, Discrete Applied Mathematics, 49, 189-212.

Freville, A. and G. Plateau. (1997). The 0-1 bidimensional knapsack problem: toward an efficient high-level primitive tool, Journal of Heuristics, 2, 147-167.

Gavish, B. and H. Pirkul. (1985). Efficient algorithms for solving multiconstraint Zero-One knapsack problems to optimality, Mathematical Programming, 31, 78-105.

Lau H. C. and Song Y. (2002). Combining Two Heuristics to Solve a Supply Chain Optimization Problem, Proc. 15th European Conference on Artificial Intelligence (ECAI), IOS Press, Amsterdam, 581-585.

Loulou, R. and E. Michaelides. (1979). New greedy-like heuristics for the multidimensional 0-1 knapsack problem, Operations Research, 27, 1101-1114.

Pirkul, H. (1987). A heuristic solution procedure for the multiconstraint Zero-One knapsack problem, Naval Research Logistics, 34, 161-172.

Shih, W. (1979). A branch and bound method for the multiconstraint Zero-One knapsack problem. Journal of the Operational Research Society, 30, 369-378.

Toyoda, Y. (1975). A simplified algorithm for obtaining approximate solutions for zero-one programming problems, Management Science, 21, 1417-1427.