Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

THE MODAL-SHIFT TRANSPORTATION PLANNING PROBLEM AND ITS FAST STEEPEST DESCENT ALGORITHM

Masami Amano
Takayuki Yoshizumi
Hiroyuki Okano

IBM Research, Tokyo Research Laboratory
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, JAPAN

ABSTRACT

The Modal-Shift Transportation Planning Problem (MSTPP) is the problem that finds a feasible schedule for carriers with the minimum total cost when sets of facilities, delivery orders, and carriers are given. In this paper, we propose a fast steepest descent algorithm to solve the MSTPP. Our solution generates a set of candidate routes for each delivery order as a preprocess. Then, it finds a schedule by iteratively updating selections of the candidate routes in descent directions, while computing a configuration of carrier movements at each iteration by a greedy algorithm. Intensive numerical study using artificial data modeled from the manufacturing industry in Japan is also presented.

1 INTRODUCTION

In the manufacturing and distribution industries, the transformation of logistics is attracting attention due to the recent requirements triggered by a new management trend called supply chain management, which aims to supply products with the minimum total cost. As supply chains are spanning a variety of parts suppliers, plants, and distributors all over the world, cost reduction for the transportation is becoming a major concern. For example, the options for transportation modes (carriers) are expanding from trucks and trains to ships and airplanes, and the best choices of the modes, i.e., *modal shifts*, have a great impact on the total cost.

In this paper, a transportation problem between facilities in a supply chain, referred to as the *Modal-Shift Transportation Planning Problem (MSTPP)*, is described, and a fast solution for the problem is proposed. By "planning," we mean that the problem is to find movements of carriers considering practical constraints, such as their time windows or diagram, their capacities, and the number of available carriers, such that the total cost for the carriers used is minimized. The present problem also deals with movements of goods between facilities in a supply chain, such as plants, depots, and cross-docking points. Each cargo of goods is associated with a delivery order which specifies the starting and destination facilities, and also the release and due dates at these facilities. (Hereafter, *orders* are used to mean delivery orders and the associated goods.) The two facilities, the starting point and the destination, may not be consecutive, which means a direct carrier may not be available between them. In such a case, the problem seeks to choose intermediate facilities (cross-docking points) which the order should go through, and the route for the order is composed of two or more legs. Conversely, multiple types of carriers may be available between arbitrary pairs of facilities. In such a case, the problem seeks to choose one type of carrier. This problem, therefore, involves decisions of modal shifts and cross-dockings (Fig. 1).

When the problem is for designing the transportation network, meaning that no time window is considered for orders and carriers, the problem can be simplified by aggre-

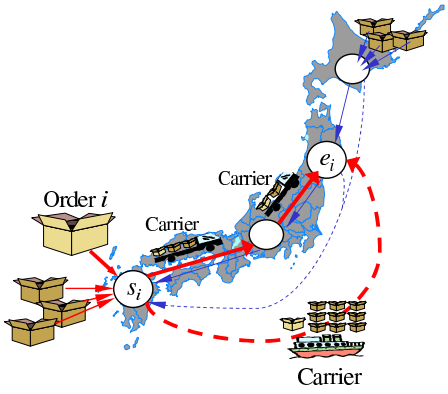

Figure 1: Schematic Figure of Modal-Shift Transportation Planning

gating the orders up to carrier capacities (truckloads). This simplified problem is called the Less-Than-Truckload (LTL) Network Design problem, and has been discussed by Powell and other researchers (Crainic and Roy 1992; Hoppe et al. 1999; Katayama and Yurimoto 2002; Powell and Sheffi 1989). The present problem, the MSTPP, includes the LTL as a special case, and therefore is more difficult. Note that the MSTPP includes the bin-packing problem, and thus it is an NP-hard combinatorial optimization problem.

The solution for the MSTPP proposed in this paper consists of two steps: 1) generation of a set of candidate routes for each order, and 2) selection of a candidate route for each order. The latter step involves the planning of carrier movements. The two-step solution allows input of specific candidate routes, and insures that it is easy to incorporate business requirements without expanding the problem. For the candidate routes generation step, an algorithm using the k-shortest path algorithm is proposed. For the candidate routes selection step, a steepest descent algorithm is proposed.

In the steepest descent algorithm, the costs of used carriers are apportioned to relevant legs of the candidate routes, and descent directions of the total cost with respect to the selection of candidate routes are estimated using the apportioned costs. The algorithm repeatedly updates the selection of a candidate route for each of the orders in the descent direction, while a greedy-type algorithm is repeatedly called to obtain the unique configuration of carrier movements that corresponds to the updated selection of the candidate routes. That is, the whole solution space, including all the configurations of carrier movements and all the configurations of candidate selections, is mapped to an abstract solution space which only involves candidate selections. This type of gradient method, in which solutions are mapped to an abstract solution space, was addressed by Okano and Koda (2003) for the traveling salesman problem (TSP). They mapped TSP tours to a real space using a greedy-type algorithm, and proposed a stochastic gradient method. Crawford et al. (1998) mapped a problem of task scheduling to an abstract solution space of task priorities, and discussed success factors of the algorithmic framework for local search. Powell et al. (1995) addressed a gradient descent algorithm for an analogous transportation problem, in which they iteratively call a greedy-type algorithm in each step of descending. The algorithm proposed in this paper focuses on estimating the costs of candidate routes and also on designing a greedy algorithm with which the steepest descent algorithm can converge efficiently.

Intensive numerical study is presented using an artificial data instance, including up to $5,000$ orders, about $300,000$ carriers, and about 400 facilities, modeled from the manufacturing industry in Japan. The present solution can find local optimal solutions in about 50 seconds, which is fast enough to be used for real-time planning.

This paper is organized as follows. In Section 2, the MSTPP is described, and its formulation is shown. In Section 3, a fast steepest descent method for the problem and a cost estimation function for candidate routes are described. The numerical study is presented in Section 4. Finally, conclusions are summarized in Section 5.

2 THE MODAL-SHIFT TRANSPORTATION PLANNING PROBLEM

The Modal-Shift Transportation Planning Problem (MSTPP) is described as follows: Given sets of facilities, orders, and carriers, find a feasible schedule for carriers with the minimum total cost. Each order has a starting facility, a destination facility, an earliest starting time (EST), i.e., a release date and time, and a deadline, i.e., a due date and time. Each carrier has a starting facility, a destination facility, a transportation cost, and so on. A *feasible* schedule means that every order is delivered to its destination facility using the given carriers no later than its deadline, and the load of each carrier does not exceed its capacity.

There may be no carriers that move directly between the starting and the destination facilities of a given order. In such a case, the problem seeks to choose intermediate facilities (cross-docking points) that the order should go through. Conversely, multiple types of carriers may be available between two facilities, with differences such as the capacities. In such a case, the problem seeks to choose one type of carrier. This problem, therefore, involves decisions of cross-dockings and modal shifts.

2.1 The Input Data

The set N of facilities and the set D of orders are given. Each facility $i \in N$ is associated with a handling time h_i, and each order $i \in D$ has the following properties:

- The starting facility : $s_i \in N$,
- The destination facility : $e_i \in N$,
- The earliest starting time : EST_i,
- The deadline : d_i,
- The weight : w_i,
- The penalty coefficient used when the order cannot be delivered : p_i.

The set V of carriers is also given, and each carrier $j \in V$ has the following properties:

- The starting facility : $s_j \in N$,
- The destination facility : $e_j \in N$,
- The travel time between the starting and destination facilities : t_j,
- The earliest starting time : EST_j,
- The latest starting time : LST_j,
- The capacity : q_j,
- The transportation cost : c_j.

The EST_j and LST_j may express the working hours of a truck or a departure time specified by a diagram. The cost c_j is added to the transportation cost if the carrier is used irrespective of the amount of its load, and no cost is added if the carrier is not used.

2.2 The Problem Definition

The MSTPP, in its general setting, is to determine the movements of carriers and orders without any predetermined routes of the orders. The objective is the minimization of the sum of the total transportation cost and the penalty for the orders which cannot be delivered. The constraints are:

1. Time window constraint: Each carrier must start within its time window.
2. Order release and due constraint: Each order is delivered to its destination no later than its due date and time.
3. Temporal order constraint: Each order does not miss its next carrier at cross-docking points.
4. Route constraint: Each order is carried along a feasible route.
5. Indivisible constraint: Each order has to be carried by only one carrier for each of its legs.
6. Load capacity constraint: The load of each carrier does not exceed its capacity.

2.3 A Decomposable Formulation

The original MSTPP described in the last subsection is difficult to handle directly. To make it decomposable into subproblems regarding the orders, we assume that a set R_i of candidate routes is given for each of the orders $i \in D$. The candidate routes may be provided to the problem as an input, or they may be generated as described later. This assumption makes our solution more practical, since specific routes may be provided for some orders in practice.

We will use a heuristic procedure to make our solution fast enough for real-time use. To be more explicit, rewrite the problem definition in the last subsection as follows:

$$\begin{array}{ll} \min & \sum_{j\in V} c_j x_j + \sum_{i\in D} p_i(1-z_i) \\ \text{such that} & \text{every order is delivered to its destination} \\ & \text{facility using one of its candidate routes} \\ & \text{no later than its deadline, and the load of} \\ & \text{each carrier does not exceed its capacity,} \end{array} \quad (1)$$

where x_j is a binary decision variable to indicate whether carrier j is used, and z_i is a binary decision variable to indicate whether order i is delivered.

Suppose that the cost of each of the used carriers can be apportioned to the loaded orders according to their contributions to the carrier cost. For example, when order i is loaded onto carrier j, the total weight of the loaded orders is $load(j) = \sum_{i\in D} w_i v_{ij}$, where v_{ij} is a binary decision variable to indicate whether order i is carried by carrier j.

The apportioned cost for the order i may be calculated as $c_j w_i / load(j)$. Further, the cost of a candidate route for the order i may be computed as the sum of the apportioned costs for each of the legs included in the route. Then the objective of (1) can be rewritten as

$$\min \quad \sum_{i\in D} \left(\sum_{k\in R_i} (r_{ik} \sum_l L_{ikl}) + p_i(1-z_i) \right), \quad (2)$$

where r_{ik} is a binary decision variable to indicate whether the candidate route $k \in R_i$ is used for the order i, and L_{ikl} is the apportioned cost of the carrier used for the lth leg of the candidate route k of the order i.

Further rearrangement of (2) gives

$$\sum_{i\in D} \left(\min \sum_{k\in R_i} (r_{ik} \sum_l L_{ikl}) + P \right), \quad (3)$$

where the penalty term $P = p_i(1-z_i)$ is treated as a constant factor, assuming its variation is relatively much smaller than the term for the transportation cost. The objective function (3) can be minimized by local minimization of the route cost for each order, if the apportioned cost of the candidate routes, $\sum_l L_{ikl}$, can be estimated to reflect the overall cost. In Section 3, we also discuss algorithm to estimate the costs of the candidate routes for each order.

3 PROPOSED SOLUTION

In this section, a fast solution for the MSTPP is described. The algorithmic framework and our main approach, i.e., a steepest descent algorithm, are described here, and the details are in the subsequent subsections.

The algorithmic components of our solution are shown in Fig. 2. As described in Section 1, the solution consists of two steps: 1) the generation of a set of candidate routes for each order (*candidate generation*), and 2) the selection of a candidate route for each order and the determination of a configuration of carrier movements (*routing*). Step 2 also involves the allocation of orders to carriers and the determination of a departure time for each carrier. For the candidate generation step, an algorithm using the k-shortest path algorithm generates the candidate routes between each pair of starting and destination facilities. This detailed algorithm is shown in Subsection 3.1. For the routing step, our approach is separated into two stages: stage a) uses a steepest descent algorithm for selection of a candidate route for each order, and b) obtain a configuration of carriers by a greedy algorithm as shown in Fig. 3. The decision variables in this step are quite small, just the selections

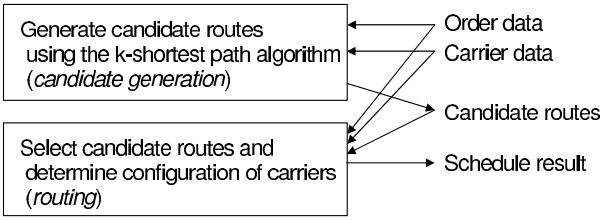

Figure 2: Algorithmic Components and Input and Output Data of Our Solution

of the candidate routes for the orders. The solution space mapped from the original solution space with only such a limited number of variables is called the abstract solution space.

This separation makes the solution space much smaller. Therefore the search space of our steepest descent method shrinks. The whole solution space, including all the configurations of carrier movements and all the configurations of candidates selection, is roughly estimated as

$$((\text{\#candidates/o}) \times (\text{\#carriers/l})^{(\text{\#legs/r})})^{(\text{\#orders})}, \quad (4)$$

where "#candidates/o" denotes the number of candidate routes per order, "#carriers/l" denotes the number of carriers per leg, and "#legs/r" denotes the number of legs per route. Note that, once a route selection and allocation of all the orders to carriers are determined, the departure time for each carrier is scheduled as early as possible. That is why the number of the candidates for the departure times for each carrier was not included in (4). This solution space is huge. For example, suppose that the number of candidate routes per order is 10, the number of carriers per leg is 100, the number of legs per route is 4, and the number of orders is 5,000. Then, the solution space is roughly estimated as $(10 \times 100^4)^{5000} = 10^{45000}$. On the other hand, the abstract solution space, i.e., the selections of candidate routes for each order, is roughly estimated as

$$|\{r\}| = (\text{\#candidates/o})^{(\text{\#orders})}, \quad (5)$$

which is much smaller than the whole solution space. For example, the abstract solution space is 10^{5000} for the above example.

However, our abstract solution space is still huge. Since we intend to do real-time planning, such as one minute or less of computational time, a fast search method which finds good solutions is needed. As mentioned in Section 2, the overall objective function can be minimized by local minimization of the candidate route for each order. Thus, we propose an iterative algorithm with the above-mentioned two-stage structure that performs the following: Estimate the costs of the candidate routes by an *evaluation function*, select a candidate route for each order on the basis of the estimated costs, and route the carriers. The candidate

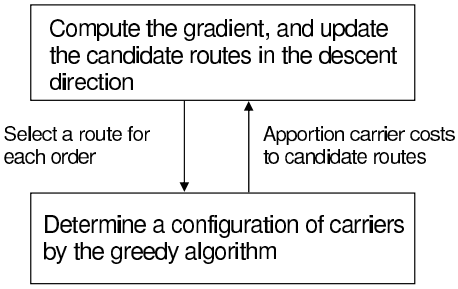

Figure 3: Algorithmic Framework of the Routing Step of Our Solution

cost estimation is redone each time after routing. If all of the selected candidate routes do not differ from those in the previous iteration, the search terminates, and the best solution found so far is output. Figure 4 shows the procedure of our descent algorithm.

Our method changes the solutions drastically iteration by iteration. A candidate route for each order is updated at each iteration either to the same value or to a new value, and thus the number of the neighborhood solutions is $2^{(\text{\#orders})}$. The best solution in the neighborhood is selected uniquely for the next current solution, that is, the decision variables r_i are updated in the steepest descent direction. Note that the actual objective function value cannot be obtained until carrier movements are scheduled, and that it is not known whether the selected move is downhill or uphill. That is, the descending move based on the estimated costs may be an uphill move. Therefore, this method can efficiently escape from bad local optimum and find good solutions. The key success factors of this method are how to define the evaluation function and how to compute the descent direction using the estimated costs. Our approaches to solving these issues are shown in detail in Subsections 3.2 and 3.3.

One more key success factor of our solution is a greedy algorithm to obtain a configuration of carriers when the route for each order is given. The greedy algorithm should be very fast, and should also find a good solution. The greedy algorithm is shown in Subsection 3.4.

3.1 Generating Candidate Routes

Taking modal shift into consideration, we adopt two kinds of cost definition for the leg, the travel time of each carrier and the transportation cost per unit weight for each carrier. (Since a leg may have multiple carriers, the minimum value for those carriers is regarded as the representative value of the leg.)

When the travel time is used as the costs of the legs, calculated routes are likely to include carriers which have a relatively high cost and a short travel time, such as trucks. On the other hand, when the transportation cost per unit weight is used as the costs of the legs, carriers which

have relatively large capacity and low transportation cost per unit weight, such as ships, are selected for candidate routes. For modal shift, therefore, it is desirable to consider that different kinds of candidate routes that have different characteristic.

In particular, the best k routes which have shorter travel times and another best k routes which have lower transportation cost per unit weight are calculated, and then, the union of the two of route sets are defined as candidate routes for an order.

3.2 Evaluating Candidate Routes

As mentioned before, the evaluation function is important since it is used to determine the descent direction. The evaluation value (estimated cost) is initialized first, and updated when the candidate route is used for the current schedule, as shown in Fig. 4.

We use two indices for the evaluation criteria, and compute the weighted sum of each index as the evaluation value for the candidate route. One is the cost apportioned to the order, and the other is the *time hardness* of the candidate route for the order. As described in Section 2, the cost apportioned to the order is calculated as the sum of the apportioned cost of the carrier used in each leg to carry the order. The time hardness of the candidate route for an order i is calculated as the difference between EST_i and the latest starting time by which the order has to depart from the starting facility s_i to meet the deadline d_i at the destination facility e_i when the candidate route is used to carry the order. This value has to be a negative value or zero, and can be calculated in advance for all the pairs of the candidate routes and the orders. The evaluation function $eval(i,k)$ for the candidate route $k \in R_i$ for order i is denoted as

$$eval(i,k) = w_1 \cdot cost(i,k) + w_2 \cdot time(i,k), \qquad (6)$$

Initialize $eval(i,k)$* for all pairs of (i,k), $i \in D$, $k \in R_i$
do
  foreach order $i \in D$
    Select the route $r_i := \arg\min_{k \in R_i} eval(i,k)$.
  Calculate schedule using selected routes.
  foreach order $i \in D$
    Update $eval(i, r_i)$.
while (at least one selection is changed).
Output the best solution found so far.

*$eval(i,k)$ denotes the estimated cost of route $k \in R_i$ of order i.

Figure 4: Procedure of the Steepest Descent Algorithm

where $cost(i,k)$ is the cost apportioned to the order i which uses the candidate route k, $time(i,k)$ is the time hardness of the candidate route k for the order i, and w_1 and w_2 are the weight of the above two indices, respectively. Let $cost(i,k,l)$ denote the apportioned cost of the carrier used in the lth leg of route k, and let m be the number of legs of the route k, then L_{ikl} in (2) corresponds to $w_1 \cdot cost(i,k,l) + w_2 \cdot time(i,k)/m$.

The first index, i.e., the cost apportioned to the order, is updated when the candidate route is used for the current schedule. If the order is carried by an efficient carrier whose transportation cost per unit weight is relatively low and the load ratio is high, then the index value will be small and it will be hard for the route to switch to the other candidate routes. Conversely, if the order is carried by an inefficient carrier whose transportation cost per unit weight is relatively high and the load ratio is low, then the index value will be large and it will be easy for the route to switch to one of the others. Therefore, we expect that this index will work well.

The other important issue is to determine how to initialize the first index. It is initialized as the lower bound when the order is carried using the candidate route. The lower bound is calculated as the cost apportioned to the order when the order is carried by the carrier whose transportation cost per unit weight is the lowest in that leg and with the load ratio one, for each of the legs in the candidate route. Therefore, for the first iteration of the search, it is easiest to select a candidate route such that the number of legs is small and each leg has the carrier whose transportation cost per unit weight is small.

The second index, the time hardness, is introduced to incorporate the idea that the probability of orders failing to be delivered is high if the order uses a candidate route whose time constraint is hard. It is not certain at this stage of our research whether or not this index works well. The experiments in Section 4 include tests with and without the time-hardness constraint of the second index.

3.3 Selecting Candidate Routes

The candidate route of each order i is selected as $r_i := \arg\min_{k \in R_i} eval(i,k)$. Suppose that order i has three candidate routes, $R_i = \{1,2,3\}$, and their estimated costs are $eval(i,1) = 5$, $eval(i,2) = 10$, and $eval(i,3) = 15$. Then, Route 1 is selected as the route for the order i, i.e., $r_i = 1$. After the schedule is calculated using Route 1 for the order i, $eval(i,1)$ is updated on the basis of the schedule result. If the updated value, $eval(i,1)$, is equal to or smaller than 10, Route 1 is selected as the route for the order i again for the next iteration. If the updated value is grater than 10, Route 2 is selected as the route for the order i for the next iteration. Note that Route 3 is not selected for the next iteration, because $eval(i,2)$ is smaller than $eval(i,3)$, and

these values are not updated when Route 1 is used as the route for the order i. Thus, r_i is updated at each iteration either to the same route or to the second route. The size of the neighborhood solutions is therefore $2^{(\#\text{orders})}$.

3.4 Calculating a Configuration of Carriers

The configuration of carriers is a set of loaded orders and a departure time for each of the carriers. Given orders and the routes of the orders, the problem to find this configuration is still NP-hard, since it involves the bin-packing problem.

The procedure of our greedy algorithm is shown in Fig. 5. The algorithm has two heaps. One is the heap to store the orders, denoted by H_o, and the other is the heap to store the carriers onto which the orders are loaded, denoted by H_c. Suppose that the earliest starting time and the latest starting time to meet the deadline for the facility where order i is currently located are denoted by $cest(i)$ and $clst(i)$, respectively. The value of $cest(i)$ can be calculated as the sum of the arrival time of the order i at the current facility and the handling time for the current facility. The value of $clst(i)$ is calculated in advance by tracing backward from the destination facility. Note that the order might fail to be delivered even if the order i arrives by $clst(i)$, because the carriers associated with $clst(i)$ might not be available.

Suppose that the available earliest starting time and the available latest starting time for carrier j are denoted by $cest(j)$ and $clst(j)$, respectively. The initial value of $cest(j)$ is EST_j. If order i is loaded onto the carrier j and $cest(i)$ is greater than $cest(j)$, then $cest(j)$ is changed to $cest(i)$. In the same way, the initial value of $clst(j)$ is LST_j, and if the order i is loaded onto the carrier j and $clst(i)$ is smaller than $clst(j)$, then $clst(j)$ is changed to $clst(i)$.

The element extracted from the order heap H_o is always the order whose $cest$ is the earliest of all the orders in H_o, and the element extracted from the carrier heap H_c is always the carrier whose $clst$ is the earliest of all the carriers in H_c. The algorithm also uses pre-calculated sorted lists of the available carriers for each leg. The first key for sorting is in the ascending order of the transportation cost per unit weight, and the second key for sorting is in the ascending order of the earliest starting time for each carrier.

```
Initialize
  insert all orders into H_o and H_c = ∅.
while H_o ≠ ∅
  extract order i from H_o and carrier j from H_c.
  if (cest(i) ≤ clst(j) or H_c = ∅)
    then insert j into H_c and find carrier c for i. (5-1)
    insert c into H_c.
  else
    insert i into H_o and
        find carrier d for reloading orders on j. (5-2)
    if (d is not found)
      then j departs and insert orders on j into H_o.
    else insert d into H_c.
```

Figure 5: Procedure of Our Greedy Algorithm

The procedure of our greedy algorithm is described in Fig. 5. Initially, the carrier heap H_c is empty, and all the orders are inserted into the order heap H_o. Then, the loop continues until the order heap H_o becomes empty. The loop starts by extracting the elements from both heaps. Note that the element extracted from the order heap is the order whose $cest$ is the earliest and the element extracted from the carrier heap is the carrier whose $clst$ is the earliest. Suppose that the element extracted from the order heap H_o is order i and the element extracted from the carrier heap H_c is carrier j. If $cest(i)$ is equal to or smaller than $clst(j)$, or the heap H_c is empty, then the carrier to load the order i on is found by the first fit algorithm ((5-1) in Fig. 5). The first fit algorithm searches the sorted list associated with the next leg for the order i from the top of the list and loads it onto the first carrier found without violating $cest(i)$ and $clst(i)$. If a carrier is not found in the sorted list, the order i cannot be delivered. From the characteristics of the sorted list, it is easiest to find the carrier whose transportation cost per unit weight is lowest and whose earliest starting time is earliest. The found carrier is inserted into the carrier heap H_c.

If $clst(j)$ is greater than $cest(i)$, the algorithm tries to find another carrier onto which the orders on the carrier j are reloaded, ((5-2) in Fig. 5), because there is no other order which can be loaded onto the carrier j. If the orders are reloaded onto a carrier whose capacity is smaller or whose latest starting time is later, the total transportation cost might be reduced but the possibility that the orders cannot be delivered will become higher. Our algorithm tries to find the carrier onto which the orders are reloaded by the first fit algorithm from the next carrier of the original carrier in the sorted list as long as the selected carrier can carry all the orders without missing their $clst$. If the carrier is not found, the carrier j departs, the departure time is $cest(j)$, and if the carried orders are not arriving at their destination facilities, the $cest$ and $clst$ of each order carried by the carrier j are updated, and the orders are inserted into the order heap H_o. If the carrier is found, the found carrier is inserted into the carrier heap H_c.

4 NUMERICAL STUDY

4.1 The Instance Data

The instance data was generated for an imaginary manufacturing company in Japan. The total number of facilities is about 400, the number of starting facilities, i.e., production facilities, is about 20, the number of destination facilities,

i.e., stores or dealers, is about 350, and the number of intermediate facilities, i.e., cross-docking points, is about 30. The number of legs which have available carriers is 915, which means some orders must be transported via several intermediate facilities because the number of pairs of starting and destination facilities is about $20 * 350 = 7000$. The handling time for each facility was fixed at 60 minutes.

Three order instances were generated. The three instances involve 5,000 orders whose scheduling horizon is 10 days, 4,000 orders whose scheduling horizon is 8 days, and 3,000 orders whose scheduling horizon is 6 days, respectively.

Each carrier type is defined by the starting facility, the destination facility, the travel time, the EST, the LST, the transportation cost, the capacity, and the amount (e.g., ten 2-ton trucks). Each parameter for each carrier type was generated by supposing two modes of transportation, trucks and ships. To be more specific, the travel times for ships are longer than those for trucks, but the transportation costs per unit weight for ships are lower and the capacities of ships are higher than for trucks. Therefore the more orders transported by one ship, the cheaper the total transportation cost becomes. The ESTs of trucks were set to 9:00, 12:00, and 15:00. All the LSTs of trucks were set to their EST plus two hours. The ESTs and LSTs of ships were all set to 12:00, so the departure time of the ships was fixed.

Two carrier instances were generated. In one instance, some legs are associated with ships and others with trucks. In another instance, all of the legs are associated with trucks. The mixed instance is intended to observe modal shift.

4.2 Experimental Results

The experiments were conducted using an IBM ThinkPad T21 with a Pentium III 800 MHz CPU and 128 MB of memory. The penalty coefficient, which is used when orders are not delivered, was fixed at 10,000 for all of the orders. The number of candidate routes for each order is ten.

We evaluated our steepest descent algorithm. Two weight settings for the evaluation function of each candidate route were experimented with. They are (1) $w_1 = 1$, $w_2 = 0$ and (2) $w_1 = 1$, $w_2 = 0.01$. Setting (1) ignore the time hardness, and Setting (2) includes it. Table 1 shows the experimental results using these settings. Three order instances whose numbers of orders are 5,000, 4,000, and 3,000 were used with the two carrier instances. The columns "ini", "fin", and "time" show the objective values for the initial solution, the objective value for the final solution, and the calculation time taken by our steepest descent algorithm.

From Table 1, we cannot see any significant differences regarding their solution qualities. Regarding the calculation times, Setting (2) was relatively slower to converge, but the slow convergence did not insure obtaining good final solutions. Comparing the carrier instance with ships to the instance without ships, the instance with ships always gave better objective values. However, as the number of orders gets small, the difference between them became small. This might be caused by the fact that the order instance does not include any order whose time window is longer than the scheduling horizon. Therefore, if the scheduling horizon is short, it is easier for generated orders to have destination facilities that are geographically close to their starting facilities. Such orders do not need to be carried by ships.

Next, we examined how the solution is improved by our steepest descent algorithm. Table 2 shows the improvement ratio using our steepest descent algorithm. The improvement ratio is calculated as $100 *$ (final objective value) / (initial objective value). It is easy to see that the carrier instances with ships are more likely to be improved than the instances without ships. As mentioned in Subsection 3.2, it is easiest for the route which has the lowest transportation cost per unit weight of all the candidate routes to be selected as the initial route for each order. Therefore, it seems that many ships are used with low load ratio. That is why the initial objective values are worse for the carrier instances with ships. Further, it can be observed that for the carrier instances with ships, the improvement ratio of Setting (1) is higher than that of Setting (2), because the initial objective values of Setting (2) are better than those of Setting (1).

Figure 6 shows the variation of the objective values obtained at each iteration by our steepest descent algorithm. The upper figure shows the overall variation and the lower figure shows the variation for the objective values ranging from 850,000 to 1,100,000. The horizontal axis shows the number of iterations and the vertical axis shows the objective values. The results of Setting (1) and Setting (2) for both carrier instances and the order instance with 5,000 orders are plotted. The overall behavior is almost the same for every case. During the early iterations, the objective values fluctuate a lot and the fluctuation gradually gets smaller. The large fluctuations for early iterations might be because the initial evaluation value for each route is set to the ideal value. Thus, even hopeless routes are selected for early iterations. From the figure on the right, it can be observed that the objective value is being improved as the search continues, and finally converges.

5 CONCLUSIONS

In this paper, we have defined the MSTPP, and proposed a fast steepest descent algorithm for this problem. Our approach consists of two steps: candidate generation and routing. The routing is separated into two stages: routes selection by a steepest descent algorithm, and the determination of a configuration of carriers by a greedy algorithm. Our steepest descent algorithm converges very quickly; it finds solutions

Table 1: Result of Our Steepest Descent Algorithm

#order	carrier	(1) ini	(1) fin	(1) time	(2) ini	(2) fin	(2) time
5,000	SHIP	1,061,943	898,637	40	1,016,080	924,756	45
5,000	NO SHIP	1,089,006	1,009,964	46	1,091,364	1,003,879	51
4,000	SHIP	827,207	685,170	22	783,279	711,411	25
4,000	NO SHIP	801,877	749,413	25	809,747	739,996	38
3,000	SHIP	581,742	497,323	13	565,123	487,111	16
3,000	NO SHIP	535,102	499,946	16	539,045	499,409	15

(time scale: seconds)

Table 2: Ratio of Improvement by Steepest Descent Algorithm

#order	carrier	(1) ratio	(2) ratio
5,000	SHIP	84.6%	91.0%
5,000	NO SHIP	92.7%	91.9%
4,000	SHIP	82.8%	90.8%
4,000	NO SHIP	93.5%	91.4%
3,000	SHIP	85.4%	86.2%
3,000	NO SHIP	93.4%	92.6%

(ratio) $=$ 100 $*$ (final obj) / (initial obj).

within one minute for large instances whose solution space is about 10^{45000} in size. Our solution can be used for both daily operations and logistics design, since the MSTPP handles many practical constraints, such as time windows or diagram, capacities, and the amount for available carriers. Further, our fast steepest descent algorithm allows real-time planning.

Future work is to make our problem more practical. So far, we have not considered inventory control and delivery lead time in the MSTPP, but they should be considered in real situations and the importance of these metrics will be different for various customers. Therefore, our solution has to be able to accommodate the differences of each customer's policies. Another extension of our work is to consolidate modal-shift transportation with regional routing.

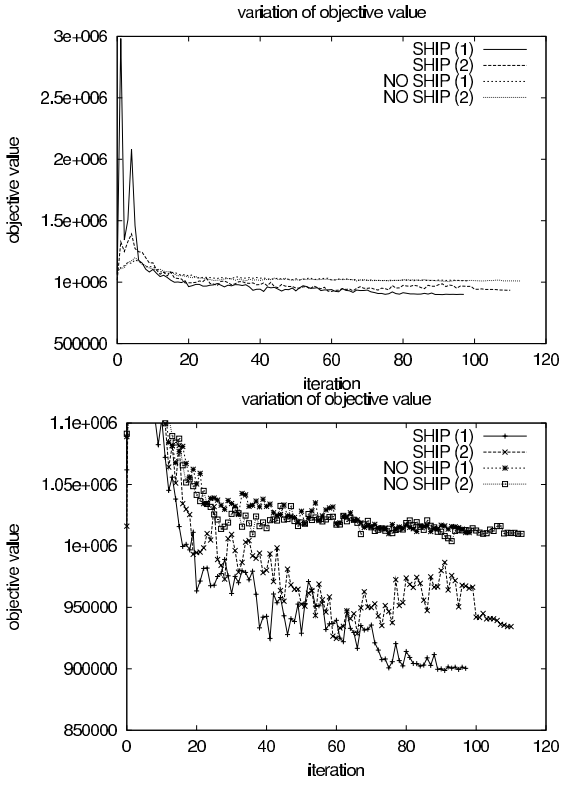

Figure 6: Convergence Behavior of Our Steepest Descent Algorithm

REFERENCES

Crainic, T. G. and J. Roy. 1992. Design of regular intercity driver routes for the LTL motor carrier industry. *Transportation Science*, 26: 280–295.

Crawford, J. M., M. Dalal and J. P. Walser. 1998. Abstract Local Search. In *Proceedings of the AIPS-98 Workshop on Planning as Combinatorial Search*.

Hoppe, B., E. Z. Klampfl, C. McZeal and J. Rich. 1999. *Strategic Load-Planning for Less-Than-Truckload Trucking*. Technical Report CRPC-TR99812-S, Center for Research on Parallel Computation, Rice University.

Katayama, N. and S. Yurimoto. 2002. *The Load Planning Problem for Less-than-Truckload Motor Carriers and a Solution Approach*. In Proceedings of the 7th International Symposium on Logistics, 567–572.

Okano, H. and M. Koda. 2003. An optimization algorithm based on stochastic sensitivity analysis for noisy objective landscapes. *Journal of Reliability Engineering and System Safety*, 79: 245–252.

Powell, W. B. and Y. Sheffi. 1989. Design and implementation of an interactive optimization system for network design in the motor carrier industry. *Operations Research*, 37: 12–29.

Powell, W. B., T. Carvalho, G. Godfrey and H. Simao. 1995. Dynamic fleet management as a logistics queueing network. *Annals of Operations Research*, 6: 165–188.

AUTHOR BIOGRAPHIES

MASAMI AMANO is a researcher of Tokyo Research Laboratory of IBM Japan. He received a B.S. degree (1999) and a M.S. degree (2001) in Informatics from Kyoto University. He joined Tokyo Research Laboratory, IBM Japan, in 2001, and researched on production planning and logistics. His research interests include the supply chain management and the business strategy. His e-mail address is <amanom@jp.ibm.com>.

TAKAYUKI YOSHIZUMI is a researcher of Tokyo Research Laboratory of IBM Japan. He received a B.S. degree (2000) and a M.S. degree (2002) in Informatics from Kyoto University. He joined Tokyo Research Laboratory, IBM Japan, in 2002, and joined operations research group. His research interests include the artificial intelligence. His e-mail address is <yszm@jp.ibm.com>.

HIROYUKI OKANO is a researcher of Tokyo Research Laboratory of IBM Japan. He received a B.S. degree (1988) and a M.S. degree (1990) in Information Science from the Tokyo University of Agriculture and Technology. He joined Tokyo Research Laboratory, IBM Japan, in 1990, and researched on user interface and system software. In 1995, he joined operations research group, and started to research on combinatorial optimization. His research interests include the traveling salesman problem and the vehicle routing problem. His e-mail address is <okanoh@jp.ibm.com>.