MODELS AND METHODS FOR MERGE-IN-TRANSIT OPERATIONS

Keely L. Croxton
Fisher College of Business
The Ohio State University

Bernard Gendron
Département d'informatique
et de recherche opérationnelle
and
Centre de recherche sur les transports
Université de Montréal

Thomas L. Magnanti
School of Engineering and Sloan School of Management
Massachusetts Institute of Technology

September 2000

Abstract

We develop integer programming formulations and solution methods for addressing operational issues in merge-in-transit distribution systems. The models account for various complex problem features including the integration of inventory and transportation decisions, the dynamic and multimodal components of the application, and the non-convex piecewise linear structure of the cost functions. To accurately model the cost functions, we introduce disaggregation techniques that allow us to derive a hierarchy of linear programming relaxations. To solve these relaxations, we propose a cutting-plane procedure that combines constraint and variable generation with rounding and branch-and-bound heuristics. We demonstrate the effectiveness of this approach on a large set of test problems with instances with up to almost 500,000 integer variables derived from actual data from the computer industry.

Key words : Merge-in-transit distribution systems, logistics, transportation, integer programming, disaggregation, cutting-plane method.

Résumé

Nous présentons des modèles de programmation en nombres entiers et des méthodes de résolution permettant de résoudre un problème d'optimisation en planification des opérations d'un système de distribution avec fusion. Les modèles tiennent compte de plusieurs aspects complexes tels l'intégration des décisions relatives à l'inventaire et au transport, les composantes dynamique et multimodale inhérentes au problème, de même que la structure des fonctions de coût, qui sont linéaires par morceaux et non-convexes. Pour modéliser ces fonctions, nous introduisons des méthodes de désagrégation qui nous permettent de définir une hiérarchie de relaxations linéaires. Pour résoudre ces relaxations, nous proposons une méthode de coupes qui combine génération de contraintes et de variables avec des heuristiques basées sur des techniques d'arrondissement et de branch-and-bound. Nous démontrons l'efficacité de cette méthode lorsqu'elle est appliquée à un grand ensemble de problèmes tests tirés de données provenant de l'industrie informatique, incluant des instances ayant approximativement 500,000 variables entières.

Mots-clés : Systèmes de distribution avec fusion, logistique, transport, programmation en nombres entiers, désagrégation, méthode de coupes.

1 Introduction

In recent years, stimulated by lasting improvements in computer capabilities and new approaches to managing interfirm relationships and activities, supply chain management has increasingly drawn the attention of both practitioners and researchers (for a survey, see Vidal and Goetschalckx, 1997). In some applications, supply chain management is complicated by products that are composed of several components, manufactured at geographically dispersed locations. For example, in the computer industry, a customer's order for a given type of computer might translate into an order for a monitor, a CPU unit, a printer and a keyboard – all produced at different sources. The customer does not want to receive four shipments. It wants one shipment to arrive on its dock on the requested day.

The manufacturer can meet this need in several ways. It can maintain large warehouses from which it can draw inventories of components to create a single product to meet any customer order. As an alternative, it can use merge centers that act as in-transit consolidation points. It can ship components from plants into a merge center, where they are consolidated into the final product requested by the customer. These merge-in-transit (MIT) centers are not intended to hold inventory. Therefore, the firm needs to coordinate shipments so that they arrive simultaneously (or nearly so) and can be bundled and shipped immediately to the final customer for arrival on the due date.

Some companies have found MIT centers to be an appropriate way to meet customers' needs. It is, however, a complex system to design and manage. In this paper, we describe models and methods that address relevant operational issues. In Section 2, we discuss these issues and present an optimization problem that captures the essential features of MIT operations. Section 3 provides an overview of the relevant literature. Section 4 proposes and analyzes several integer programming formulations of the problem, based on different model disaggregations. These formulations form the bases of our solution method, which we present in Section 5. This cutting-plane approach includes constraint and variable generation strategies, combined with rounding and branch-and-bound techniques, to generate lower and upper bounds on the optimal objective value. Section 6 describes computational results obtained by this method on data instances based on an actual application arising in the computer industry. These results demonstrate that the method is very effective in identifying near-optimal solutions to large-scale realistic instances. For example, the largest problem instances that we examine have 9 sources, 10 components, 10 merge centers, and 73 customers, translating into integer programming formulations with approximately 500,000 integer variables and as many constraints. We are able to solve most of these large instances to within 5% of optimality. In a concluding section, we summarize our work and propose extensions.

2 Problem Description

As illustrated in Figure 1, an MIT system is a two-echelon distribution system. Components move from sources to merge centers, where they are consolidated into products

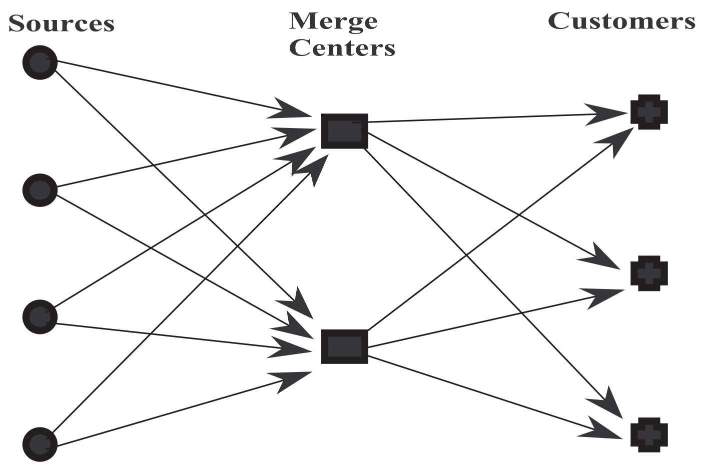

Figure 1: A Typical Merge-in-Transit Network

and shipped to customers. In general, a firm might source multiple components from one plant, or source one component from multiple plants. In the computer industry, for example, manufacturers usually produce a component at a single plant. Each product is composed of several components and is described by a "recipe" of its components. Each time a customer orders a product, the corresponding components must arrive at one of the merge centers before the entire product can be shipped to the customer. The merge centers are not intended to carry large inventories, but they may hold short term inventory when it is cost effective to do so. To operate the merge centers in this fashion, we must coordinate the shipments so that they arrive nearly simultaneously and can be shipped to the customer with minimal delay.

Given these features, we can state the MIT problem as follows: *Given a set of orders (product type, customer location and delivery date) over a short term planning horizon (typically a few days), find the optimal routing, timing and transportation mode for sending components from sources to merge centers, and products from merge centers to customers to meet demand.*

The MIT problem is further characterized by the following assumptions:

- Each source has unlimited supply.
- Demand is known for the time horizon being considered.
- Orders must arrive at the customer on the delivery day – not early and not late.
- We consider components as arriving to a merge center "simultaneously" if they arrive on the same day.
- A product is available to leave a merge center the same day that all necessary components are available at that merge center.

- Orders must arrive in a *single shipment*, i.e., on the same mode via the same merge center; however, customers can receive different orders from different merge centers on any day, i.e., they do not need to be sourced from a single merge center.

- Each merge center has a volume capacity.

Given that the time horizon is only a few days and that each time period is equivalent to one day, the assumption of known demands is realistic. By the third assumption, we do not schedule early or late arrivals at the customers. We could slightly modify our modeling approach by including penalties to handle such situations. The next two assumptions define the notion of "simultaneity." Then, we ensure that customers receive only one shipment for each order, but permit customers to receive orders from multiple merge centers over time. The last assumption reflects the fact that merge centers are not designed as large warehouses and are intended to hold only a limited amount of inventory.

Several issues complicate the problem. In particular, the decision of which merge center to use might require a tradeoff between what appears best for different components. For example, while one merge center might be "ideal" for a given component (if it is in a direct line between the source and the customer), it might be less costly to merge the order at another merge center that is closer to the source for some of the other components. Other complicating issues are related to economies of scale. For example, we need to consider all orders collectively because it might be cost effective to consolidate freight and send items to a merge center that in isolation might not be optimal. Economies of scale might also drive us to send components earlier than they are needed, therefore resulting in short term inventory.

Another complication arises from the selection of a mode of transportation. We consider four modes: small package (SMP), less-than-truckload (LTL), truckload (TL) and air. SMP and LTL shipments are the most cost effective for small shipments, but are the slowest modes. TL shipments are cost effective for large shipments. Air is the quickest, but most costly. Because air is the most expensive form of transportation, a shipper will use it only when time constraints permit nothing else.

Each of these four modes has its own cost structure. Air shipment costs are independent of distance, but for all other modes, the cost per weight unit increases with distance. On any given lane (i.e., origin, destination and mode), the cost depends on the total weight transported. SMP, LTL and air modes all exhibit price breaks for increased weight. For example, an LTL shipment with weight between 150 and 500 pounds costs \$0.02/pound, while one with weight between 500 and 1000 pounds costs \$0.016/pound. As illustrated by this example, the cost is not necessarily an increasing function of weight: a shipment of 450 pounds costs \$9, while a shipment of 500 pounds costs \$8. The resulting cost, as a function of weight, would be a piecewise linear, discontinuous function. An intelligent shipper would, however, never send a shipment of 450 pounds, but would declare it to be 500 pounds, thereby reducing its cost. Thus, any shipment with weight between 400 and 500 pounds would have a fixed cost of \$8. The resulting cost function would still be piecewise linear, but would then include flat segments (or pieces) with a fixed cost, but no variable cost.

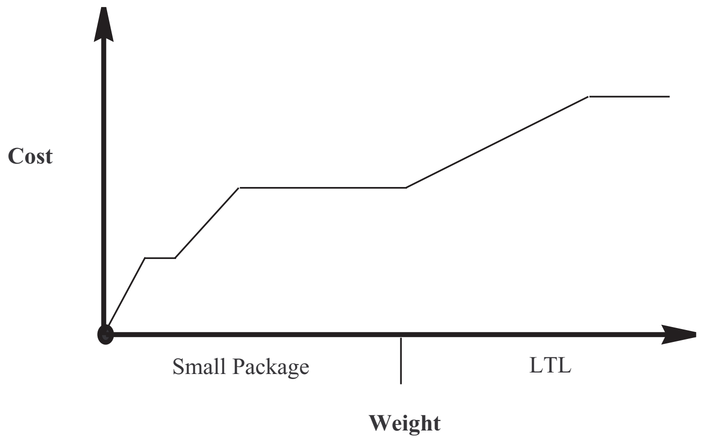

Figure 2: A Typical SMP/LTL Cost Function

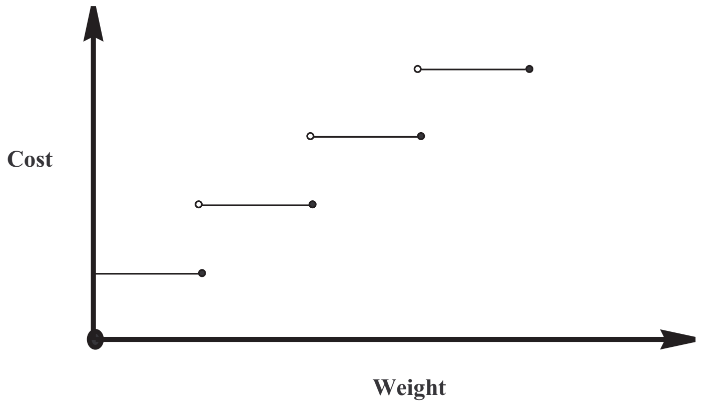

Figure 3: A Typical TL Cost Function

Although they share a similar cost structure, SMP, LTL and air costs also differ in some aspects. In particular, to discourage extremely small shipments, the LTL mode imposes a minimum shipment charge. As a result, shipments under a specified weight should be sent by small package, while those over this weight should be sent by LTL. This observation allows us to represent small package and LTL modes as a single mode, referred to as the SMP/LTL mode in the remainder of the paper. Figure 2 shows a typical cost function for this combined mode. The air mode differs from the SMP/LTL mode by imposing a significant initial fixed cost to discourage small shipments (in this case, the initial segment has no variable cost).

Unlike SMP/LTL and air shipments, TL shipments are independent of weight, within the limitations of the truck capacity. On a particular lane, the cost of using one truck is fixed, but if we are to consider the option of using multiple trucks, the cost is a piecewise linear, discontinuous function, with each piece s representing the fixed cost for using s trucks, which is simply s times the cost of using one. Figure 3 illustrates this function.

3 Literature Review

A simple search on the Web using the key words "merge-in-transit" reveals a growing penetration of this concept among logistics businesses. Despite this fact, to the best of our knowledge, there is little literature on MIT systems. In the optimization area, we are aware of only one related work by Cole and Parthasarathy (1998) who develop a model and propose a GIS-based support system that addresses strategic and tactical issues, including the number and location of the merge centers. However, several related problems are well studied, including applications in transportation logistics and assembly systems, and work conducted in network optimization problems with complex nonlinear cost structures.

The body of literature on transportation logistics is large and broad in scope. Bramel and Simchi-Levi (1997), Crainic and Laporte (1997), and Crainic (1999) provide thorough reviews of much of the research in this area and discuss key issues at the strategic, tactical, and operational levels. In particular, network design issues in transportation planning (Magnanti and Wong, 1984) are especially relevant to our work.

The formulation we propose for the MIT problem can be viewed as an integer multicommodity generalized network flow problem. As such, it is closely related to some large-scale applications in assembly systems (Shapiro, 1993). Although most books on network optimization (for example, Ahuja, Magnanti, and Orlin, 1993) discuss solution methods for both single-commodity generalized and multicommodity network flow problems, we are aware of very few references on multicommodity generalized network flow problems, and no research has taken place, to the best of our knowledge, on the integer case (see the recent annotated bibliography by Ahuja, 1998).

Much of the literature on network optimization problems with complex nonlinear costs focuses on concave costs, of which the special case of piecewise linear costs is of particular relevance to our work. Cominetti and Ortega (1997) solve the uncapacitated minimum cost network flow problem with piecewise linear concave costs. Their branch-and-bound algorithm uses sensitivity analysis to improve the cost approximations and to obtain better lower bounds. Chan, Muriel and Simchi-Levi (1997) develop results for the multicommodity version of the same problem with the aim of addressing tactical issues in supply chain management. They derive structural results on a set-partitioning formulation and use them to develop a linear programming-based heuristic.

Balakrishnan and Graves (1989) present a Lagrangian-based heuristic for an arc-based formulation of the uncapacitated minimum cost network flow problem with piecewise linear concave costs. Although they developed their method for the concave case, it is also applicable to non-concave piecewise linear costs. Popken (1994) studies a multicommodity minimum cost network flow problem and develops an algorithm adapted to non-concave continuously differentiable cost functions. Holmberg (1994) develops an algorithm based on convex linearization and Bender's decomposition for the uncapacitated facility location problem with discontinuous piecewise linear costs. Holmberg and Ling (1997) address the same problem, using a Lagrangian-based heuristic to solve it.

Our model differs somewhat from the ones presented in these references. Indeed, the basic constraints in our formulation are similar to those encountered in modeling large-scale assembly systems, but our cost functions are not linear, contrary to what is found in these applications (Shapiro, 1993). Also, most research on network optimization with complex cost structures has focused on uncapacitated problems, while our modeling and solution approach accounts for capacities. Finally, it is worth mentioning that our problem deals with operational issues, while most related formulations found in the literature address strategic and tactical issues. Therefore, our model also integrates dynamic features.

Like Balakrishnan and Graves (1989), our solution method uses an arc-based formulation and applies to any non-convex piecewise linear cost function. However, rather than employing Lagrangian relaxation, we use a simplex-based cutting-plane approach, generating cuts with disaggregation techniques. These techniques are familiar in the integer programming community (Nemhauser and Wolsey, 1988) and have been found to be extremely useful in solving discrete location (Mirchandani and Francis, 1990) and multicommodity network design problems (Balakrishnan, Magnanti and Wong, 1989; Balakrishnan, Magnanti and Mirchandani, 1998; Gendron, Crainic and Frangioni, 1998; Crainic, Frangioni and Gendron, 1998). In this paper, we use them in a decomposition approach that not only implements constraint generation, but also variable generation. These two features, constraint and variable generation, exploit the particular structure of the forcing constraints in our arc-based formulation. In a companion paper (Croxton, Gendron and Magnanti, 2000a), we explore the structural properties of disaggregation approaches applied to minimum cost network flow problems with non-convex piecewise linear cost functions.

4 Problem Formulation

Formulations of the MIT problem need to capture its essential features, but still remain tractable. Modeling the problem must account for several complicating issues such as the integration of inventory and transportation decisions, the dynamic and multimodal aspects of the problem, and the specific nature of the cost functions. Because the model is complex, we have chosen to describe our modeling approach in two parts. We first show how to formulate the problem's constraints, and then we consider modeling of the particular cost functions.

4.1 Modeling Constraints

We use the following notation, which we summarize in Appendix A. The underlying network has a node set N and an arc set A. We subdivide the node set N into three sets: N_S, the set of sources, N_M, the set of merge centers and N_C, the set of customers. We also subdivide the arcs set A into two sets: A_{SM}, the set of arcs from sources to merge centers, and A_{MC}, the set of arcs from merge centers to customers. As in classical network flow problems, we let $N^+(i) = \{j \in N | (i,j) \in A\}$ and $N^-(i) = \{j \in N | (j,i) \in A\}$ denote the sets of outward and inward neighbors of any node $i \in N$. To account for the dynamic aspect of the problem, we introduce a set of discrete time periods $T = \{t_1, ..., t_n\}$ (typically, a time period corresponds to a day). We let the set M denote the transportation modes, the set P the products, and the set K the commodities that flow through the network. For the moment, we assume a commodity is a component; in Section 4.2.2, we specify a different definition of a commodity, which will be useful in better approximating the cost functions.

We further let α^{kp} denote the number of units of commodity $k \in K$ required to produce one unit of product $p \in P$ and δ_{jt}^{p} denote the demand for product $p \in P$ at customer $j \in N_C$ at time $t \in T$. Also, let $N_C(t)$ represent the set of customers who have a positive demand for at least one product at time $t \in T$. Using this notation, we can impute a derived demand $d_{jt}^{k} = \sum_{p \in P} \alpha^{kp} \delta_{jt}^{p}$ of customer $j \in N_C(t)$ for each commodity $k \in K$ at time $t \in T$.We also let r_{ijm} denote the transport time on arc $(i, j) \in A$ using mode $m \in M$, l^k denote the volume of each commodity $k \in K$ and q_i denote the volumetric capacity of merge center $i \in N_M$.

Finally, we define the following decision variables: u_{it}^{k}, the *inventory variable*, represents the inventory of commodity $k \in K$ at merge center $i \in N_M$ at the beginning of time period $t \in T$; w_{jitm}^{k}, the *flow variable*, is the flow in units of commodity $k \in K$ transported on arc $(j, i) \in A_{SM}$ arriving at time $t \in T$, using mode $m \in M$; v_{ijtm}, the *single shipment variable*, is a binary variable that assumes value 1 if we satisfy the demand of customer $j \in N_C(t)$ at time $t \in T$ by merge center $i \in N_M$ using mode $m \in M$, and is 0 otherwise.

Using this notation, we can model the constraints of our MIT system as follows (all variables and data are nonnegative and integer):

$$\sum_{i \in N_M} \sum_{m \in M} v_{ijtm} = 1, \quad j \in N_C(t), t \in T, \tag{1}$$

$$\sum_{j \in N^-(i)} \sum_{m \in M} w_{jitm}^{k} + u_{it}^{k} = \sum_{j \in N^+(i)} \sum_{m \in M} d_{jt'}^{k} v_{ijt'm} + u_{i(t+1)}^{k}, \tag{2}$$

$$i \in N_M, t \in T, k \in K \ (t' = t + r_{ijm} \leq t_n),$$

$$\sum_{k \in K} \sum_{j \in N^+(i)} \sum_{m \in M} l^k d_{jt'}^{k} v_{ijt'm} + \sum_{k \in K} l^k u_{i(t+1)}^{k} \leq q_i, \quad i \in N_M, t \in T \ (t' = t + r_{ijm} \leq t_n). \tag{3}$$

The first set of constraints (called *single shipment constraints*) ensures that we satisfy demands via a single shipment, while the second set represents *flow balance constraints* at each merge center: the flow arriving on each day plus the inventory at the beginning of that day must equal the flow shipped to customers that day plus the inventory at the beginning of the next day. To complete these flow balance equations, we assume given initial inventories of each commodity. The final set of constraints model the *capacity constraints* at the merge centers.

4.2 Modeling Costs

While it is reasonable to assume linear inventory costs, transportation costs are more difficult to model. The cost functions for SMP/LTL, TL and air modes, although they

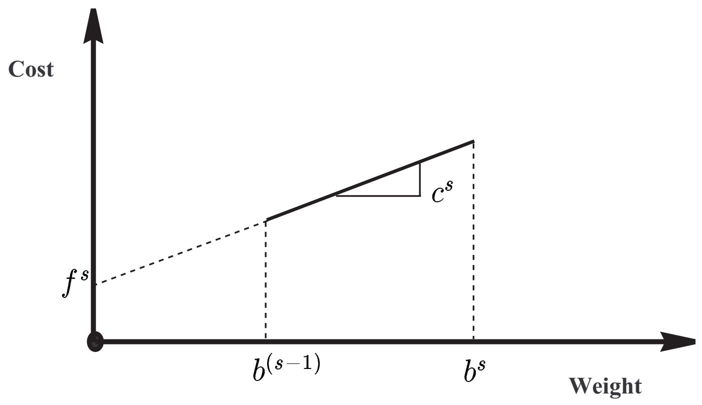

Figure 4: Segment Notation for Modeling Transportation Costs

differ significantly, are all piecewise linear, possibly with fixed costs. Because, in general, these functions are neither convex nor concave, we rely on integer programming (IP) techniques to represent them accurately.

To model the transportation cost on a generic arc in the MIT network, we can use a standard *multiple choice* IP model. As shown in Figure 4, each linear segment s (in the segments set S) of the cost function on the arc is determined by a lower bound $b^{(s-1)}$ ($b^0 = 0$ by convention), an upper bound b^s, a cost c^s per unit weight, and a fixed cost f^s. For each segment $s \in S$, the model contains an *auxiliary variable* x^s that equals the total weight transported, if it lies on the segment, and is 0 otherwise. The binary *indicator variable* y^s indicates whether or not the total weight lies on segment s or not. If w^k and h^k denote the unit flow and the weight per unit of commodity $k \in K$ on the arc, then the model becomes:

$$\sum_{k \in K} h^k w^k = \sum_{s \in S} x^s, \tag{4}$$

$$\sum_{s \in S} y^s \leq 1, \tag{5}$$

$$b^{(s-1)} y^s \leq x^s \leq b^s y^s, \quad s \in S. \tag{6}$$

The total cost on the arc is given by $\sum_{s \in S}(c^s x^s + f^s y^s)$. Equation (4) expresses the relation between flow and auxiliary variables, while the next constraint ensures that we select no more than one segment. We refer to these two constraints as the *basic variable definition constraints*. The *basic forcing constraints* (6) ensure that an auxiliary variable assumes value 0 whenever the corresponding indicator variable has value 0; otherwise, it specifies that the value of the auxiliary variable lies between the lower and upper limits of its corresponding segment.

Other IP models of the cost function are possible, though each of three standard models (the multiple choice model we are using, an incremental model, and a convex combination model) are equivalent in the sense that the LP relaxation of each approximates the cost function by its lower convex envelope (Croxton, Gendron and Magnanti, 2000b). Croxton (1999) argues that the multiple choice model is better suited for our purposes since it is most useful for deriving valid inequalities to improve the LP approximation.

There are two major difficulties in using an IP modeling approach for the MIT problem:

1. The lower convex envelope approximation defined by the LP relaxation typically is a poor approximation of the real cost (the computational results presented in Section 6 further substantiate this observation).

2. The model tends to be very large because the arcs typically have many segments (more than 30 to model some SMP/LTL costs). It also tends to be large depending upon how we define the commodities and what techniques we use to mitigate the poor approximation of the LP relaxation. We will treat this issue both through modeling (by approximating certain arc costs with fewer segments) and through a variable and constraint generation procedure in our algorithms.

4.2.1 Improving the LP Relaxation for Flows Between Sources and Merge Centers

To tighten the LP relaxation of the model, suppose we know a bound a^k on the flow of commodity $k \in K$ on any arc. Then the so-called *strong forcing constraints*

$$w^k \leq a^k \sum_{s \in S} y^s, \quad k \in K, \tag{7}$$

are valid inequalities. They serve the same purpose as the basic forcing constraints by ensuring that no flow circulates on an arc when the corresponding indicator variables equal zero. However, these two types of constraints differ significantly in the sense that the constraints (6) aggregate commodities and the strong forcing constraints disaggregate them, but aggregate segments.

If we could identify forcing constraints that disaggregate both segments and commodities, we could obtain a tighter LP relaxation. To achieve this objective, we define *extended flow variables* that combine a commodity and a segment: w^{ks} is the unit flow of commodity $k \in K$, if the total weight transported on the arc lies on segment $s \in S$, and is 0 otherwise. These new variables are related to the previous ones by the following simple *extended variable definition constraints*:

$$w^k = \sum_{s \in S} w^{ks}, \quad k \in K, \tag{8}$$

$$x^s = \sum_{k \in K} h^k w^{ks}, \quad s \in S. \tag{9}$$

Letting $a^{ks} = \min\left\{a^k, \left\lfloor \frac{b^s}{h^k} \right\rfloor\right\}$, we can combine these variables with the indicator variables to derive the following *extended forcing constraints*:

$$w^{ks} \leq a^{ks} y^s \quad k \in K, s \in S. \tag{10}$$

These inequalities can be stronger than the previous ones; therefore, they can lead to a tighter LP relaxation at the expense of increasing significantly both the number of variables and the number of constraints.

4.2.2 Definition of Commodities

The definition of the underlying commodities can have a significant impact on the LP relaxations. We consider two definitions:

- **Component-based definition.** A commodity k is a component and we can bound its flow from a source to a merge center by a^k, the minimum of (1) the merge center's capacity divided by the component's volume and (2) *the sum of all future demands.*

- **Destination-based definition.** A commodity k is a combination of a component and a destination (i.e., a customer and a time period). We can then bound its flow from a source to a merge center by a^k, the minimum of (1) the merge center's capacity divided by the component's volume and (2) *the demand for that destination.*

The destination-based definition for the commodities creates more variable and constraints (though with the same interpretations). As is easy to see, it provides a tighter LP relaxation when the bounds in the component-based formulation equal the sum of all future demands, which is the case for example when the merge centers have no capacities. This fact is part of the folklore of integer programming. However, when the bound for at least one commodity equals a merge center's capacity divided by a component's volume, then it is easy to construct examples so that the LP relaxation of the component-based formulation dominates the relaxation of the destination-based formulation. Thus in the general capacitated case, neither formulation dominates the other and we create a tighter LP relaxation by combining both commodity interpretations in a single model.

Therefore, we have several alternatives for modeling the transportation costs on the arcs between the source nodes and the merge centers. To classify them, we distinguish: (1) whether they define a commodity as a component, or they integrate in a single formulation this component-based interpretation with a destination-based definition of a commodity; (2) whether they add strong or extended forcing constraints, or no other valid inequalities, to the basic LP relaxation. We denote each of the resulting relaxations by two letters. The first one signals the definition of a commodity: either as a component (C), or as a combined component and destination (D). The second letter indicates the types of forcing constraints included: only the basic forcing constraints (B), both the basic and the strong forcing constraints (S), or, in addition to those two, the extended forcing constraints as well (E). We thus derive five different relaxations:

- CB: This is the basic LP relaxation, i.e., constraints (4) to (6), with no valid inequality added (note that CB and DB are equivalent).

- CS: This is relaxation CB combined with the strong forcing constraints (7).

- CE: This is the relaxation obtained from CS by adding the extended constraints (8) to (10).
- DS: This relaxation is obtained from CE by defining destination-based flow variables (full details are given in Appendix A), and by adding the strong forcing constraints (7) corresponding to these variables.
- DE: This relaxation is obtained from DS by adding the extended constraints (8) to (10) corresponding to destination-based flow variables.

Because we obtain each relaxation from the former one by adding inequalities, if we let $Z(XX)$ denotes the optimal value of relaxation XX, we can rank the lower bounds obtained from these relaxations as follows:

$$Z(CB) \leq Z(CS) \leq Z(CE) \leq Z(DS) \leq Z(DE). \tag{11}$$

4.2.3 Reducing the Number of Segments

Because the cost functions can have a large number of segments, we aim to identify approximations that would significantly reduce this number without sacrificing the effectiveness of the model's representation of the application. We make two assumptions about these approximations: (1) they are piecewise linear; (2) they underestimate the cost. Assumption (1) guarantees that any algorithm based only on the piecewise linear structure of the cost functions can also be used with the approximations. Assumption (2) ensures that any lower bound on the optimal value with respect to the approximation is also a lower bound on the optimal value for the actual cost.

To define this approximation for each arc between a source and a merge center, we propose the following procedure. We assume that the cost function $C(W)$ is continuous everywhere, except at the origin $W = 0$, so the procedure applies to the SMP/LTL and air functions, but not to the discontinuous TL functions. We also assume that each pair of successive segments contains one strictly increasing segment and one flat segment, i.e., with a fixed cost but no variable cost. The procedure attempts to merge pairs of successive segments, with the second segment of each pair being the flat one.

Initially, we attempt to merge segments 1 and 2 for the SMP/LTL functions, and segments 2 and 3 for the air functions (recall that segment 1 is flat for the air functions). When attempting to merge any pair of segments into a single segment, we require two conditions on the resulting tentative approximate cost:

a. The tentative approximate cost does not lie above the actual cost.

b. The relative gap between the tentative approximate cost and the actual cost is less than a specified parameter σ (0.01 worked well in our studies).

Using these two conditions, we implement the procedure for merging segments as follows:

1. If segment 1 is flat, start with segment $j = 2$; otherwise, start with segment $j = 1$.

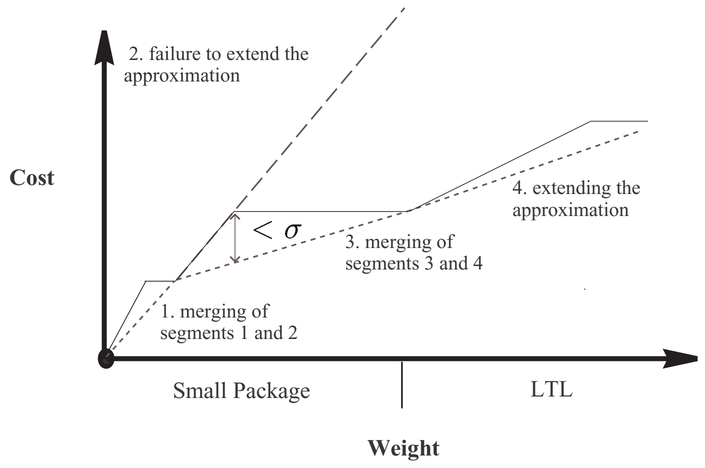

Figure 5: A Typical Approximation for the SMP/LTL Cost Function

2. If $j \geq |S|$, stop.

3. Define the tentative approximate cost $L(W)$ as the segment with extreme points $(b^{j-1}, C(b^{j-1}))$ and $(b^{j+1}, C(b^{j+1}))$ (this definition of $L(W)$ always satisfies condition a).

4. If condition b is *not* satisfied, then use segments j and $j+1$ as part of the approximation and go to step 2 with $j = j+2$ (that is, consider the next pair of segments).

5. Merge segments j and $j+1$ into a single segment defined by $L(W)$ and let $j = j+2$.

6. If $j \geq |S|$, stop.

7. Define the tentative approximate cost $L(W)$ by extending the previous tentative approximate cost to b^{j+1}.

8. If conditions a and b are satisfied, go to step 5; otherwise, go to step 3.

Given the definition of the tentative approximate cost $L(W)$ specified in steps 3 and 7, we implement conditions a and b as follows:

a. $L(b^{j+1}) \leq C(b^{j+1})$.

b. $(C(b^j) - L(b^j))/L(b^j) < \sigma$ (since the largest gap occurs at the intermediate breakpoint b^j between segments j and $j+1$, see Figure 5).

Figure 5 shows a typical approximation obtained by this procedure for the SMP/LTL function. For a typical application, the procedure reduces the number of segments significantly: from 29 to 10, on average, for the SMP/LTL functions, and from 13 to 6 for the air functions.

4.2.4 Modeling Flows Between Merge Centers and Customers

For a generic arc in the time expanded network from a merge center to a customer, the flow $w^k = vd^k$ of commodity k is defined by the commodity's demand d^k and the value v of the corresponding single shipment variable. Although we could use the same approach proposed for modeling the flows from sources to merge centers, this situation is simpler. Since v is a binary variable, we know that the total weight is either $\bar{b} = \sum_{k \in K} h^k d^k$ or 0. Then, the cost is simply given by gv, where g is the cost corresponding to $\bar{b}$. This *linear approximation* clearly provides a valid IP formulation. For most cost functions encountered in our application, the linear approximation is the lower convex envelope. In this case, if we relax the integrality of the single shipment variables, the linear approximation corresponds to the basic LP relaxation defined by constraints (4) to (6). In principle, the linear approximation could be improved by the addition of strong or extended forcing constraints. However, our computational experience, reported in Section 6, shows that the linear approximation is already tight. This could be explained by the fact that we use a tight upper bound on the total weight between any merge center and any customer, based on the demands of the customer (in contrast to the upper bound on the weight between any source and any merge center, which uses the sum of all future demands for components produced at the source). This observation motivates our decision to model the cost functions on arcs from merge centers to customers with their linear approximations, thereby significantly reducing the size of the formulation.

5 Solution Method

To solve the MIT problem, we use a *cut-and-branch* method that proceeds in three steps:

- **Cutting-Plane Procedure.** Because of the large size of the LP relaxations corresponding to the different levels of disaggregation, it is impractical to solve them by using a standard LP code. Therefore, we propose a cutting-plane approach based on constraint and variable generation.

- **Rounding Heuristics.** To generate feasible solutions to the MIT problem, we implement two rounding heuristics that can be invoked at any iteration of the cutting-plane procedure. The v-rounding heuristic takes as input a fractional solution and iteratively fixes binary single shipment variables. The (w, u)-heuristic complements the first one, by taking as input a v-integral solution and a (w, u)-fractional solution, and attempting to obtain an all-integral feasible solution.

- **Branch-and-bound procedure.** After applying the cutting-plane procedure, with its embedded rounding heuristics, we use the final IP formulation in a branch-and-bound procedure, performing branching only on the binary single shipment variables. Every time we obtain a v-integral solution in the enumeration tree, we call the (w, u)-heuristic to obtain an all-integral solution that can serve as an upper bound on the optimal IP objective value. When the branch-and-bound procedure terminates with an optimal solution, we obtain a new lower bound corresponding to the optimal value of our formulation with only the integrality restrictions on the w and u variables relaxed.

We next provide details concerning the implementations of the cutting-plane procedure and the rounding heuristics. We will specify the implementation details of the branch-and-bound procedure in Section 6, since it makes use of a commercial code.

5.1 Cutting-Plane Procedure

Given a set of valid inequalities, any cutting-plane method first solves an initial LP that does not contain all valid inequalities in the set, and then performs the following steps:

1. If the LP solution satisfies all the valid inequalities in the set, stop; otherwise, add at least one violated valid inequality to the LP.
2. Solve the LP and go to step 1.

Step 1 requires the solution of the so-called separation problem (Nemhauser and Wolsey, 1988): given a point x^* and a set of inequalities, determine if x^* satisfies all these inequalities or, otherwise, identify at least one that x^* violates. In our case, valid inequalities are forcing constraints of the form (6), (7), or (10). For any set of forcing constraints, it is easy to verify a violated constraint and so to solve the separation problem. Despite this fact, a "naive" cutting-plane approach that generates all variables in the initial LP would fail, because of the formidable dimensions of actual applications. Therefore, we propose two techniques that aim to keep the number of variables and constraints in the current LP as small as possible: *sequential constraint generation* and *dynamic variable generation*. In addition, to further improve the efficiency of the approach, we implement a *cleanup procedure*, which periodically removes redundant forcing constraints and their corresponding variables. We call the procedure when the number of forcing constraints added since the last call to it (or, if it has never been called since the beginning of the execution of the cutting-plane method) exceeds some predetermined threshold C (in our experiments, we use $C = 0.01 \times M$, with M equal to the number of forcing constraints in formulation DE; we discuss the calibration of this parameter in Appendix B). To reduce the size of the LP given to the branch-and-bound algorithm, we also call the cleanup procedure at the end of the cutting-plane step.

5.1.1 Sequential Constraint Generation

Instead of solving the separation problem for all types of forcing constraints simultaneously, our cutting-plane procedure proceeds sequentially as follows. First, it checks for violations of only the basic forcing constraints (6), iterating between the solution of the LP and that of the separation problem restricted to this class of inequalities. Once this process has converged (giving us the optimal solution to relaxation CB), we then proceed by testing violations of the strong forcing constraints (7), iterating between the solution of the LP and that of the separation problem restricted to this set of inequalities. Once the solution satisfies all the strong forcing constraints (7), the procedure returns to checking violations of basic forcing constraints again. If the solution satisfies all these constraints, we have solved the relaxation CS. Otherwise, the procedure solves LPs and separation problems for (6) and (7), alternatively. We can extend this sequential scheme in a similar way, testing violations of extended forcing constraints (10). Eventually, we will have solved all the relaxations presented in Section 4.2.2. It is important to note that, when solving relaxations DS and DE, we must define destination-based flow variables the first time we test violations of strong forcing constraints (7) corresponding to these variables. We can do so by appending to the formulation flow balance equations of type (2), with a commodity defined as a combination of a component and a destination (Appendix A specifies the detailed equations).

5.1.2 Dynamic Variable Generation

The variable generation strategy is based on the following observation: given an LP and its optimal solution, it is easy to derive necessary conditions for a valid inequality involving additional variables (i.e., not defined in the current LP) to be violated. These necessary conditions will restrict the variables to be generated at the current iteration to those that have a chance to be involved in a violated valid inequality; we need not generate all other variables because they cannot belong to an inequality violated by the current LP solution.

Let us illustrate this variable generation strategy applied to the addition of extended forcing constraints of type (10) to an LP that contains all variables and constraints, except extended flow variables w^{ks} and their related defining equations (8) and (9). An obvious necessary condition for a given optimal solution to this LP to violate extended forcing constraints (10) related to a particular arc is the presence of flow on the arc: $\sum_{s \in S} x^s > 0$. Also, because the solution satisfies all strong forcing constraints (7), we can identify two cases when the extended forcing constraints (10) are not necessarily redundant: (1) the y variables for the corresponding arc are split among segments (i.e., $y^{s_1} > 0$ and $y^{s_2} > 0$ for two segments s_1 and s_2); (2) $y^s > 0$ for only one segment s, but for some commodity k, $a^k > \left\lfloor \frac{b^s}{h^k} \right\rfloor$. If the solution satisfies one of these conditions, we know that the addition of extended forcing constraints might improve the solution. Therefore, in this case only, we generate the extended variables w^{ks} and their defining equations, solve the resulting LP to obtain the values of the new variables, and then test again for violations of the extended forcing constraints.

5.1.3 Initial LP

We generate an initial LP using a two-phase procedure. First, for each arc, we define a linear approximation to the cost function using a convex combination of the zero cost at the origin and the cost g at the last breakpoint $\overline{b} = \sum_{k \in K} h^k a^k$. As we noted in Section 4.2.4, we already use this approximation for arcs joining any merge center to a customer location. For arcs from sources to merge centers, we define $h^k g/\overline{b}$ as the flow unit cost for each commodity k and solve the resulting LP subject to constraints (1) to (3). In the second step we generate all the x^s and y^s variables along with their definitional constraints (4) and (5), and modify the objective so that it becomes $\sum_{s \in S}(c^s x^s + f^s y^s)$ for arcs from sources to merge centers (the costs of other variables remain unchanged). For each of these arcs, we add the basic forcing constraint corresponding to the segment where the total weight lies. The resulting LP is the initial LP that we use at the first iteration of the cutting-plane procedure. Note that, instead of generating all the variables x^s and y^s at once, statically, we could also make use of a dynamic variable generation strategy similar to the one described in Section 5.1.2. In this approach, we would generate the variables and modify the cost only if the current solution imposes flow on the arc. However, we conclude from preliminary experiments, reported in Appendix B, that the static approach is more efficient than this dynamic generation strategy.

5.2 Rounding Heuristics

5.2.1 v-Heuristic

The v-heuristic takes as input a fractional solution and either returns a v-integral solution that satisfies the single shipment constraints (1) or else fails to identify a feasible solution. Given $\overline{v}$, some user-supplied value larger than or equal to 0.5, it first attempts to fix to 1 the v variables whose LP values exceed $\overline{v}$ (in our experiments, we use $\overline{v} = 0.5$, as we explain in Appendix B). Any time we attempt to fix a variable, we ensure that we do not violate any of the constraints $\sum_{k \in K} \sum_{j \in N^+(i)} \sum_{m \in M} l^k d^k_{jt} \hat{v}_{ijtm} \leq \theta q_i$, $i \in N_M, t \in T$. In this expression, $\hat{v}$ represents the values (0 or 1) of the fixed variables and θ is a parameter that allows us to maintain some slack in the capacity constraints ($\theta = 0.9$ worked well in our experiments). If we fix a variable to value 1, then we set all other variables related to the same destination to value 0 to satisfy constraints (1). If we have fixed some variables, we solve the resulting restricted LP by the cutting-plane procedure, and repeat the previous step until we can fix no more variables in this way. If we have yet to fix some variables, the procedure examines all destinations that have not yet been assigned to one merge center and one mode. For each of these destinations, the procedure scans the list of related v variables in decreasing order of LP values (ties being broken arbitrarily). It attempts to fix to 1 the first variable in this list unless doing so would violate the corresponding capacity constraint. In this case, it examines the next v variable in the list and repeats the previous step, until it has fixed one variable to value 1. If, for a given destination, we cannot fix any variables without violating the capacity constraints, the heuristic fails to identify a feasible solution. If we have fixed all the v variables, we solve the resulting LP by the cutting-plane procedure, obtaining a v-integral solution, which can still have fractional w and u variables. We then invoke the (w, u)-heuristic, attempting to obtain an all-integral solution.

5.2.2 (w, u)-Heuristic

The (w, u)-heuristic examines each merge center i and each commodity k, trying to satisfy flow balance equations (2) for all time periods. It examines every time period t sequentially from t_1 to t_n (we assume the initial inventory to be integral). Its first step is to round to the nearest integer every incoming flow of commodity k to merge center i at time t (from all modes and all sources). Next, the heuristic rounds the inventory variable $u^k_{i(t+1)}$, so that it will be integral when period $t+1$ will be examined. To round $u^k_{i(t+1)}$, we compare it to $B^k_{it} = \sum_{j\in N^-(i)} \sum_{m\in M} w^k_{jitm} + u^k_{it} - \sum_{j\in N^+(i)} \sum_{m\in M} d^k_{jt'} v_{ijt'm}$ ($t' = t + r_{ijm} \leq t_n$). Note that every term in B^k_{it} is integral. If $B^k_{it} = u^k_{i(t+1)}$, then all variables are integral and the solution satisfies the flow balance equation and the capacity constraint, since we have not changed the total flow passing through node i at time t. Otherwise, there are two cases:

- If $B^k_{it} > u^k_{i(t+1)}$, then we attempt to round up $u^k_{i(t+1)}$ unless doing so violates the capacity constraint for merge center i at time t. In this case, we round down $u^k_{i(t+1)}$. In either case, $u^k_{i(t+1)}$ is now integral, but $B^k_{it} > u^k_{i(t+1)}$ is still possible. To achieve balance, we now reduce the total incoming flow by the maximum possible amount, which is $\min\{\sum_{j\in N^-(i)} \sum_{m\in M} w^k_{jitm}, B^k_{it} - u^k_{i(t+1)}\}$ (we comment below on how to accomplish this in our case). If we succeed in reducing the total incoming flow by $B^k_{it} - u^k_{i(t+1)}$, we have an integral feasible solution. If not, we try again to increase $u^k_{i(t+1)}$ by the quantity $B^k_{it} - u^k_{i(t+1)}$. If we cannot satisfy the capacity constraint for i at time t in this way, the heuristic fails to identify a feasible solution. Otherwise, it has obtained an integral feasible solution.

- If $B^k_{it} < u^k_{i(t+1)}$, then we round down $u^k_{i(t+1)}$ to the nearest integer. Since $B^k_{it} < u^k_{i(t+1)}$ is still possible, we then round up the total incoming flow by $u^k_{i(t+1)} - B^k_{it}$.

When rounding up or down the total incoming flow, we exploit the structure of our particular application with each component produced at a single source. Consequently, we need to consider only one source. The solution could, however, use several modes to transport the same component (for example, when the solution produces several units of the same component in different time periods, but assembles them at the same merge center). If we want to round up the incoming flow, we simply increase the flow first on the SMP/LTL mode, then on the TL mode and, finally, on the air mode. If we want to round down the incoming flow, we do the opposite. Although our implementation of the (w, u)-heuristic is designed for applications with each component produced at a single source, it is easy to extend it to other situations as well.

5.2.3 Combining the Rounding Heuristics and the Cutting-Plane Procedure

There are several ways to combine the rounding heuristics with the cutting-plane procedure. Our implementation calls the v-rounding heuristic whenever the current lower bound has improved significantly over the previous one, i.e., when the gap between the two bounds is larger than some user-supplied parameter ϵ (as reported in Appendix B, we use $\epsilon = 1.0\text{e-}3$ in our experiments). Note that the v-rounding heuristic makes use of the cutting-plane procedure to solve the restricted LPs. Another possibility would have been to solve the restricted LPs without attempting to add any valid inequality to the formulation. However, computationally, we observed that it is often preferable to use the cutting-plane method. Indeed, when we have fixed the v variables, the solutions to the restricted LPs tend to vary significantly from those obtained when solving unrestricted LPs. Therefore, the procedure often adds new sets of valid inequalities to the formulation, which helps to avoid degeneracy problems, and consequently, accelerates the convergence of the cutting-plane method.

6 Computational Results

Our computational experiments were guided by three objectives:

- To identify the most effective formulation among the five different formulations described in Section 4.2.2.
- To measure the quality of the approximation of the cost function described in Section 4.2.3.
- To analyze the performances of the solution method with respect to various problem characteristics, such as the size of the network, the demand pattern, and the number of time periods.

6.1 Set of Instances

We performed our experiments on data obtained from a major American logistics company, which provided us with a typical network from the computer industry, as well as realistic estimates of inventory and transportation costs associated with this application. The network has 9 sources, 10 components (each produced at a single source), 10 merge centers and 73 customers. The air cost function is the same for each arc and has 13 segments, while the SMP/LTL cost function varies by arc and includes, on average, 29 segments. Once we apply the procedure to reduce the number of segments, described in Section 4.2.3, we obtain 6 segments for the air function, and 10 segments, on average, for the SMP/LTL function. We have imposed a realistic limit of four trucks used for any single shipment, so the TL cost function exhibits four segments for each arc.

Based on this realistic data, we have generated several instances by specifying:

- subsets of the sets of nodes and components, i.e., N_S, N_M, N_C and K;
- the number of time periods $|T|$;
- the *demand pattern*, which is characterized by three elements:
  - the set of products P (we chose products composed, on average, of $|K|/2$ components);

  - the proportion ω of all possible destinations with some (positive) demand for at least one product (i.e., $\lceil \omega \times |N_C| \times |T| \rceil = \sum_{t_1 \leq t \leq t_n} N_C(t)$);
  - the interval I_δ over which we randomly generate product demands for each destination according to a uniform law;

- merge center capacities, generated according to the value of a parameter λ:
  - if $\lambda = 0$, the problem is uncapacitated, and we do not generate the capacity constraints (3);
  - otherwise, suppose we let $q_{\max} = \max_{t \in T}\{\sum_{k \in K} \sum_{j \in N_C(t)} l^k d^k_{jt}\}$ denote the maximum volumetric demand that any merge center can satisfy in any time period (recall that l^k is the volume of commodity k). We generate the capacity q_i of merge center i so that $\sum_{i \in N_M} q_i \approx q_{\max}/\lambda$, therefore providing an easy way to control the tightness of the capacities: if $\lambda \leq 1/|N_M|$, the problem is lightly capacitated, while, as λ approaches 1, the problem becomes more tightly constrained. To illustrate how the merge center capacities are generated, suppose we have an instance with five merge centers and $q_{\max} = 50$. If $\lambda = 0.1$, the capacities of the five merge centers could be generated as: $q_1 = 100$, $q_2 = 85$, $q_3 = 90$, $q_4 = 115$, $q_5 = 110$, with $\sum_{i \in N_M} q_i = 500$. If $\lambda = 0.3$, the capacities could be as follows: $q_1 = 30$, $q_2 = 40$, $q_3 = 38$, $q_4 = 34$, $q_5 = 28$, with $\sum_{i \in N_M} q_i = 170$. We give the details of the procedure used to generate the capacities and the initial inventories in Appendix C.

We performed our experiments on a set of 144 instances described in Table 1. We obtained these instances by generating a small-scale and a medium-scale network, in addition to the original large-scale network, by allowing $|T|$ to vary from 1 to 6, and by generating four different demand patterns and two capacity levels for each network. The medium-scale network has the same dimensions as the large-scale network, except that it retains 40 of the original 73 customers. We obtained the small-scale network from the large-scale one by extracting 5 sources, each producing one component, 5 merge centers and 20 customers. The demand patterns, denoted D, are as follows:

- $D = 1$: $|P| = |K|$, $\omega = 0.2$, $I_\delta = [5, 35]$;
- $D = 2$: $|P| = |K|$, $\omega = 0.2$, $I_\delta = [5, 15]$;
- $D = 3$: $|P| = |K|$, $\omega = 0.8$, $I_\delta = [5, 35]$;
- $D = 4$: $|P| = |K|$, $\omega = 0.8$, $I_\delta = [5, 15]$.

We examined two types of instances for the capacities: uncapacitated ($\lambda = 0$) and capacitated, the latter being generated by using $\lambda = 0.1 + 1/|N_M|$ (we analyze results obtained with other values of λ in Section 6.4). We denote every instance using a mnemonic $\text{XY}^{|T|}_{D}$, with X=S, M or L standing for small-scale, medium-scale or large-scale network, Y=U or C representing either an uncapacitated or a capacitated instance, $D \in \{1, 2, 3, 4\}$ denoting the demand pattern, and $|T|$ specifying the number of time periods.

Problem	$\lvert N_S\rvert$	$\lvert N_M\rvert$	$\lvert N_C\rvert$	$\lvert K\rvert$	$\lvert T\rvert$	$\lvert P\rvert$	ω	I_δ	λ
$\mathrm{SU}_1^{\lvert T\rvert}$	5	5	20	5	$\{1,...,6\}$	5	0.2	[5,35]	0.0
$\mathrm{SC}_1^{\lvert T\rvert}$	5	5	20	5	$\{1,...,6\}$	5	0.2	[5,35]	0.3
$\mathrm{SU}_2^{\lvert T\rvert}$	5	5	20	5	$\{1,...,6\}$	5	0.2	[5,15]	0.0
$\mathrm{SC}_2^{\lvert T\rvert}$	5	5	20	5	$\{1,...,6\}$	5	0.2	[5,15]	0.3
$\mathrm{SU}_3^{\lvert T\rvert}$	5	5	20	5	$\{1,...,6\}$	5	0.8	[5,35]	0.0
$\mathrm{SC}_3^{\lvert T\rvert}$	5	5	20	5	$\{1,...,6\}$	5	0.8	[5,35]	0.3
$\mathrm{SU}_4^{\lvert T\rvert}$	5	5	20	5	$\{1,...,6\}$	5	0.8	[5,15]	0.0
$\mathrm{SC}_4^{\lvert T\rvert}$	5	5	20	5	$\{1,...,6\}$	5	0.8	[5,15]	0.3
$\mathrm{MU}_1^{\lvert T\rvert}$	9	10	40	10	$\{1,...,6\}$	10	0.2	[5,35]	0.0
$\mathrm{MC}_1^{\lvert T\rvert}$	9	10	40	10	$\{1,...,6\}$	10	0.2	[5,35]	0.2
$\mathrm{MU}_2^{\lvert T\rvert}$	9	10	40	10	$\{1,...,6\}$	10	0.2	[5,15]	0.0
$\mathrm{MC}_2^{\lvert T\rvert}$	9	10	40	10	$\{1,...,6\}$	10	0.2	[5,15]	0.2
$\mathrm{MU}_3^{\lvert T\rvert}$	9	10	40	10	$\{1,...,6\}$	10	0.8	[5,35]	0.0
$\mathrm{MC}_3^{\lvert T\rvert}$	9	10	40	10	$\{1,...,6\}$	10	0.8	[5,35]	0.2
$\mathrm{MU}_4^{\lvert T\rvert}$	9	10	40	10	$\{1,...,6\}$	10	0.8	[5,15]	0.0
$\mathrm{MC}_4^{\lvert T\rvert}$	9	10	40	10	$\{1,...,6\}$	10	0.8	[5,15]	0.2
$\mathrm{LU}_1^{\lvert T\rvert}$	9	10	73	10	$\{1,...,6\}$	10	0.2	[5,35]	0.0
$\mathrm{LC}_1^{\lvert T\rvert}$	9	10	73	10	$\{1,...,6\}$	10	0.2	[5,35]	0.2
$\mathrm{LU}_2^{\lvert T\rvert}$	9	10	73	10	$\{1,...,6\}$	10	0.2	[5,15]	0.0
$\mathrm{LC}_2^{\lvert T\rvert}$	9	10	73	10	$\{1,...,6\}$	10	0.2	[5,15]	0.2
$\mathrm{LU}_3^{\lvert T\rvert}$	9	10	73	10	$\{1,...,6\}$	10	0.8	[5,35]	0.0
$\mathrm{LC}_3^{\lvert T\rvert}$	9	10	73	10	$\{1,...,6\}$	10	0.8	[5,35]	0.2
$\mathrm{LU}_4^{\lvert T\rvert}$	9	10	73	10	$\{1,...,6\}$	10	0.8	[5,15]	0.0
$\mathrm{LC}_4^{\lvert T\rvert}$	9	10	73	10	$\{1,...,6\}$	10	0.8	[5,15]	0.2

Table 1: Set of 144 instances (each row contains six instances)

6.2 Performance Measures

To analyze our computational results, we use three performance measures:

- The lower bound gap $\Delta Z^L = (Z^* - Z^L)/Z^*$ between the lower bound Z^L obtained by a particular relaxation and the best available lower bound Z^*. As specified later, the definition of Z^* varies depending on the experiments performed.

- The upper bound gap $\Delta Z^U = (Z^U - Z^*)/Z^*$ between the upper bound Z^U obtained by a heuristic and the best available lower bound Z^*. Again, the definitions of Z^U and Z^* vary depending on the experiments performed.

- The CPU time on Sun Ultra workstations (we provide the exact specifications of the machines later, because the type of workstations used depends on the experiments). We programmed the code in C and compiled it with the CC compiler using the -O4 option. We solved the linear programs with the dual simplex method implemented in CPLEX 6.0 (ILOG Inc., 1998), using the default options. The branch-and-bound phase also uses the CPLEX 6.0 implementation (we describe the parameter settings in Section 6.4).

6.3 Choice of a Formulation

Our first series of experiments compares (1) the five formulations presented in Section 4.2.2 and (2) the approximation described in Section 4.2.3 to the original cost function. We ran the cutting-plane procedure (without the branch-and-bound phase) on 16 instances extracted from Table 1. We obtained these capacitated instances by using the small-scale and medium-scale networks and two different values of $|T|$, 3 and 5. We did not attempt to solve instances with all the nodes of the original network, since the CPU times would then be too prohibitive for this preliminary phase of our computations. These experiments were performed on a network of Sun Ultra 10/300 workstations (12.10 SPECint95, 12.9 SPECfp95), each with 128 MB of RAM memory.

Table 2 reports the three performance measures ΔZ^L, ΔZ^U and CPU time, averaged over the 16 instances, for the five different formulations and for both the actual cost function ("Origin") and its approximation ("Approx"). In these experiments, $Z^* = Z(DE)$ computed for the original cost function and Z^U is the best upper bound (always with respect to the original cost function) obtained by the rounding heuristics when applied to the formulation.

Three main conclusions emerge from these results:

- The destination-based formulations, DS and DE, are significantly more effective than the component-based ones, the improvements in the lower bounds being also coupled with improvements in the upper bounds. These improvements, however, arise at the expense of significant increases in computation times, the cutting-plane method being then several orders of magnitude slower.

Formulation		ΔZ^L %	ΔZ^U %	CPU (s)
Origin	CB	6.43	26.52	72.01
	CS	6.37	25.29	76.97
	CE	6.36	25.24	159.56
	DS	0.61	14.94	3345.73
	DE	0.00	14.50	50191.48
Approx	CB	6.45	21.65	24.95
	CS	6.39	20.93	31.54
	CE	6.38	20.93	96.16
	DS	0.63	7.82	3710.34
	DE	0.19	7.54	10513.91

Table 2: Choice of a formulation

- The extended formulations, CE and DE, are only slightly more effective than their corresponding strong formulations, CS and DS, but require more computational effort.

- In these experiments, the lower bound gap ΔZ^L, because it is computed with respect to the best lower bound ($Z(DE)$) when the actual costs are used, also allows us to measure the relative difference between the lower bound obtained when using the original costs and the one computed when using the approximate costs. As revealed by these gaps, the model using the approximate costs is very effective since the lower bounds it generates are close to those obtained with the original cost function. Moreover, the upper bounds obtained with the approximation are better, on average, than those obtained with the actual cost function. A detailed instance-by-instance analysis reveals that this behavior is not only true on average, but also for most instances. This analysis also reveals one other fact. The cutting-plane method can become stalled in the sense that the objective function does not change as we introduce new cuts. The model with the approximate cost functions seems less susceptible to stalling, which although leads to a better convergence, also requires more frequent calls to the heuristic (remember that we call the heuristic whenever the lower bound has increased sufficiently from one iteration to the next). This explains why, for formulation DS, the CPU times for the approximation are worse than those for the original cost.

Based on these results, we decided to use the cost approximation in all other experiments. To further compare formulations DS and DE, we have examined how these two formulations behave when the branch-and-bound phase is performed. For the small-scale networks, the branch-and-bound method terminates "normally," as it generally finds the optimal solution to the relaxed problem generated by the cutting-plane procedure. In this case, we observe that, after the branch-and-bound phase, the relative difference between the upper bounds for the two formulations is almost the same as it was at the end of the cutting-plane procedure. Consequently, the relative performance of the two formulations is similar, whether we perform the branch-and-bound phase or not. For the medium-scale networks, we generally terminate the branch-and-bound algorithm once it attains some "reasonable" limits on the CPU time spent, or on the amount of memory needed to store the branch-and-bound tree. In our experiments, we have used limits of 4 hours and 100 MB, respectively. In this case, the branch-and-bound tree explores more nodes when using DS, which usually yields slightly better upper bounds than when using DE. Consequently, we have decided to select formulation DS (which defines commodities for both components and destinations and uses the strong forcing constraints) for the remaining computational experiments.

6.4 Performance Analysis

In this section, we present results obtained when solving the 144 instances of Table 1, using the formulation selected after the preliminary phase. Since the problems to be solved are larger, we use a more powerful network of Sun Ultra 60/2300 workstations (13.0 SPECint95, 23.5 SPECfp), each workstation having a memory 2 GB of RAM (this type of machine is typically 20% faster than the one used in the previous experiments). Following the cutting-plane procedure, we have run the branch-and-bound method implemented in CPLEX 6.0 by imposing a limit of 4 hours (14400 seconds) on the CPU time. In general, most of the algorithm's progress takes place in the first two hours, so we could have reduced this limit by half without significantly affecting the performance of the method. We also imposed a limit of 1 GB on the amount of memory needed to store the tree. The computations almost never attained this limit, except for some large-scale instances having more than 3 periods. We used the default CPLEX 6.0 parameters, except that we enforced the branching rule that explores first the node with the branching variable fixed to 1 (this choice will identify new feasible solutions more rapidly). We also used the so-called "alternate best-estimate" rule to decide which node to explore first when backtracking. Finally, we did not use the preprocessor options and we systematically reset all tolerances to 1e-9.

Instead of presenting results for each instance, we group the instances according to their characteristics, as shown in Table 3, and report the average performance within each group. First, we divide the instances into three classes according to their dimensions, small ("S"), medium ("M") or large ("L"). Within each class (each class has 48 instances), we group instances according to:

- the number of time periods $|T|$, from 1 to 6 (8 instances per group);
- the demand pattern D, from 1 to 4 (12 instances per group);
- whether they are uncapacitated ("U") or capacitated ("C") (24 instances per group).

Each problem appears three times in the table, corresponding to one time period, one demand pattern, and one capacity alternative. We provide two sets of measures, in

Problem			Cutting-plane				Branch-and-bound		
			ΔZ^L %	ΔZ^U %	CPUh (s)	CPU (s)	ΔZ^U %	CPU (s)	Nodes
S	$\|T\|$	1	5.75	11.09	0.61	1.83	4.04	3.28	240
		2	6.40	8.98	2.54	6.67	2.89	7.57	279
		3	1.25	5.95	15.02	26.80	2.83	31.40	535
		4	1.41	7.88	45.84	75.03	3.09	3212.18	36462
		5	1.95	7.20	156.80	213.14	4.48	889.77	5094
		6	2.05	8.52	216.75	329.23	4.34	4446.60	22629
	D	1	4.85	10.70	65.39	91.30	3.67	3329.62	22732
		2	4.36	13.28	33.69	59.39	6.94	1547.71	16378
		3	1.87	4.02	85.72	131.59	1.77	145.17	11888
		4	1.45	5.08	106.91	152.86	2.27	704.71	3553
		U	3.09	7.78	65.60	101.06	3.51	878.35	4769
		C	3.18	8.76	80.25	116.51	3.72	1230.42	16977
M	$\|T\|$	1	8.76	15.10	17.00	50.60	2.57	10613.92	96554
		2	0.46	14.77	158.40	331.54	6.28	14404.50	15950
		3	0.08	9.39	834.25	1778.12	5.47	14405.88	6591
		4	0.02	9.26	2524.90	4911.76	5.59	14408.00	3057
		5	0.00	6.98	6642.19	11093.79	5.40	14410.12	1159
		6	0.00	12.11	12891.54	20501.31	6.77	14416.50	449
	D	1	2.83	15.53	3201.27	5303.88	7.27	14408.17	26321
		2	2.78	18.58	4219.42	6644.03	8.10	13236.00	12697
		3	0.50	5.31	3373.58	6030.05	3.03	13050.94	17372
		4	0.11	5.66	4584.59	7800.12	2.99	14410.83	26117
		U	1.54	11.48	3470.84	6022.91	5.25	13838.56	21134
		C	1.57	11.06	4226.36	6866.13	5.45	13714.41	20119
L	$\|T\|$	1	0.43	12.25	49.77	140.04	6.44	14415.08	58851
		2	0.00	6.86	539.98	1051.01	3.93	14404.38	6202
		3	0.00	4.56	2255.96	4153.70	2.87	14406.86	2363
		4	0.00	4.77	6039.04	15382.98	3.73	12573.90	3766
		5	0.00	5.29	13988.11	26657.78	3.78	11618.76	2148
		6	0.00	5.35	30512.20	56143.87	3.77	13034.41	1608
	D	1	0.04	9.62	6006.98	14040.45	6.14	12362.33	11198
		2	0.15	8.60	8931.22	15943.93	5.86	14420.17	6537
		3	0.04	4.05	11243.48	20122.69	1.65	13761.83	11888
		4	0.06	3.78	9408.33	18912.51	2.11	13113.58	20335
		U	0.07	6.55	8183.13	16544.51	4.16	13203.37	12506
		C	0.08	6.47	9611.88	17961.11	4.01	13614.42	12169

Table 3: Performance analysis

columns "Cutting-plane" and "Branch-and-bound", to better analyze the performance of each major component of the method. We computed all the gaps with respect to the best lower bound identified at the end of the branch-and-bound phase. To analyze the cutting-plane procedure, we show, in addition to the lower and upper bound gaps, the total CPU time, as well as the time spent in performing the rounding heuristics ("CPUh"). For the branch-and-bound phase, we display the upper bound gap, the total CPU time and the total number of nodes explored ("Nodes"). We average the measures over all instances in each group.

We can draw a number of conclusions from these computational experiments:

- As revealed by the upper bound gaps after the branch-and-bound phase, the overall procedure is very effective since we have been able to generate feasible solutions, on average, within 5% of optimality. If we focus on the most realistic large-scale instances with $|T| \geq 3$, the average gap decreases to 3.54%. If we further restrict ourselves to dense instances (corresponding to demand patterns 3-4), the average gap drops to 2.46%.

- The cutting-plane procedure is very efficient at reducing the size of the LP without sacrificing the quality of the relaxation. For example, for problem LC_4^6, formulation DS has 478,078 integer variables and, potentially, 562,336 constraints. At the end of the cutting-plane procedure, we have generated only 36,618, or 6.5%, of these constraints. In particular, we note that 259,179 constraints would be necessary to represent destination-based flow balance equations, but the procedure generates only 20,427, or 7.9%, of these constraints. Once we have added these equations, the formulation contains 287,136 strong forcing constraints. The cutting plane method generates only 3,522, or 1.2%, of these constraints.

- For medium and large-scale instances, the branch-and-bound phase always terminates after reaching the time or memory limit (except for a few 1-period medium-scale instances). This explains why the lower bound gaps are so small for these instances: the branch-and-bound algorithm did not have sufficient resources to improve the cutting-plane lower bound.

- The branch-and-bound phase terminates "normally" for almost all small-scale instances. These results allow us to evaluate the effectiveness of the cutting-plane lower bound. In particular, when $|T| \geq 3$, the cutting-plane lower bound is, on average, within 2% of the lower bound obtained after the branch-and-bound, which is an indication of the strength of relaxation DS. If $|T| < 3$, the gap significantly increases (on average, more than 6%), which indicates that the LP relaxation is weaker. This outcome is explained by the fact that, with such a small time horizon, a solution uses air and TL more often, but these modes incur significant fixed costs, and the LP relaxations notoriously provide poor approximations of such costs. This phenomenon also explains the relatively poor upper bounds obtained by the cutting-plane heuristic over all instances when $|T| < 3$ (on average, 11.51% compared to 7.27%, when $|T| \geq 3$).

- The rounding heuristics consume approximately half of the total CPU time of the cutting-plane procedure, but this investment pays off. Indeed, the upper bound generated by the heuristics is near-optimal, especially for the most realistic instances. For example, on the large-scale dense instances with $|T| \geq 3$ (these are a subset of the instances with demand patterns 3-4), the average gap after the cutting-plane procedure is 3.62%, already very close to the gap of 2.46% obtained after the branch-and-bound phase.

- Problems with dense demand patterns and large demands provide significantly better bounds than other, sparser, instances. The most convincing example of this result arises when we compare the gaps (after the branch-and-bound phase) of 2.30% (on average over all problem sizes) for problems with demand patterns 3-4 (80% of all possible destinations are demand points) and gaps of 6.33% for problems with demand patterns 1-2 (20% of all destinations are demand points). Similarly, if we compare problems with large demands (with demands in the interval [5,35]) to instances with small demands (with demands in the interval [5,15]), we obtain, on average, a gap of 3.92% compared to a gap of 4.71%. The same phenomenon explains why the results obtained on the large-scale instances are better than those for the medium-scale instances, since the latter are characterized by a sparser demand pattern (they share a common network with the large-scale instances, except they have significantly fewer customers).

- As expected, capacitated instances are harder to solve than uncapacitated ones. The cutting-plane procedure requires, on average, 10% more CPU time on capacitated problems, while the branch-and-bound method runs 40% slower on average for the small-scale instances. For medium and large-scale capacitated instances, the branch-and-bound phase generates fewer nodes on average, which is another indication of the difficulty of solving capacitated problems (recall that we terminate the branch-and-bound after a fixed time limit). Surprisingly, however, the gaps for capacitated problems are comparable to those obtained when solving uncapacitated instances (for large-scale problems, the average gap is even slightly better).

We have performed additional experiments to evaluate the impact of tighter capacity constraints on the performance of the solution method. In Table 4, we report the results obtained by varying λ on instance SC_4^3, which is representative of the behavior of other problems. We selected this problem because the branch-and-bound method completely explores the enumeration tree within the imposed limits. The results indicate that, as λ increases, it becomes increasingly difficult, during the cutting-plane phase, to identify good feasible solutions via the rounding heuristics: the upper bound gaps then increase significantly. Nonetheless, the upper bound gaps at the end of the branch-and-bound phase remain almost constant, since they rely only on the effectiveness of the (w, u)-heuristic, which is fairly independent of the capacities. From these results, it appears that we can attribute the significant increases in the size of the tree as λ increases to the large corresponding upper bound gaps. We also note that the branch-and-bound algorithm performs a best-first-like exploration of the tree, for which one of the most

λ	Cutting-plane				Branch-and-bound		
	ΔZ^L %	ΔZ^U %	CPUh (s)	CPU (s)	ΔZ^U %	CPU (s)	Nodes
0.00	0.82	6.72	20.13	33.42	1.84	13.83	98
0.30	0.80	3.54	30.93	46.53	1.79	19.73	166
0.32	0.90	3.77	30.51	44.86	2.66	22.69	205
0.34	0.94	6.13	21.27	35.24	2.18	17.33	266
0.36	0.30	9.12	15.03	29.61	2.04	123.01	2044
0.38	2.83	11.24	20.40	37.05	1.93	8615.78	78741
0.40	1.65	18.85	21.06	37.21	2.05	3793.14	48221

Table 4: Effect of capacity variations – SC_4^3

influential factors of performance is the quality of the lower bound. Thus, even though the lower bound gap is less than 3% when $\lambda \geq 0.38$, its increase, when compared to the gaps when $\lambda < 0.38$, might also explain the lack of efficiency of the branch-and-bound method for more tightly capacitated problems.

7 Conclusion

We have described an integer programming model that captures the essential features of MIT distribution systems, accounting for various complex problem elements including the integration of inventory and transportation decisions, the dynamic and multimodal components of the application, the single shipment constraints at customers' sites, and the non-convex piecewise linear structure of the cost functions. To accurately model these cost functions, we have used disaggregation techniques that allowed us to derive a hierarchy of LP relaxations. To solve these LP relaxations, we have proposed a cutting-plane procedure combining constraint and variable generation in a way that exploits the simple structure of the forcing constraints. We have embedded rounding heuristics within this cutting-plane procedure to obtain feasible solutions that we subsequently improve by using a branch-and-bound algorithm.

The results in this paper show that integer programming modeling and solution methods can be effective in solving merge-in-transit applications. Indeed, this approach has been able to solve an extensive set of test problems derived from actual data from the computer industry. Although these models are very large, some containing up to 500,000 integer variables, the modeling and solution procedure has been able to generate feasible solutions that are provably within 5% of optimality on average.

This study suggests several potentially fruitful research issues. To adequately describe the particular application that prompted this research, we have imposed several assumptions in our problem definition. Other applications might invoke different assumptions, with possible impact on the modeling and the solution methods. Most notably, the following variants might be worth investigating:

- Make merging optional, i.e., allow shipments directly between the source and the customer if direct shipments are more cost effective.
- Include limited supplies at sources, or production time for make-to-order systems.
- Account for uncertainty of the demands.
- Allow early or late arrivals with penalties.
- Allow time at the merge centers for additional processing, e.g., delayed production, customization, and other problem features.
- Allow orders to arrive in multiple shipments at customers' sites.

The formulations we propose for the MIT problem are based on disaggregation techniques that exploit the piecewise linear structure of the cost functions. In a companion paper (Croxton, Gendron and Magnanti, 2000a), we establish the effectiveness of similar formulations applied to other network flow problems with the same type of cost structure. These results suggest the potential of adapting our cutting-plane method to solve large-scale instances of these applications. We plan to explore this prospect in future investigations. Finally, although our computational results demonstrate the effectiveness of the destination-based LP relaxation for uncapacitated or lightly capacitated instances, they also suggest room for improvement in the LP bound for tightly capacitated problems. We can imagine of two ways to improve the LP relaxations: add strong valid inequalities that account for the capacity constraints, or improve the bounds used in the forcing constraints, using so-called capacity improvement techniques (Bell, Lamar and Wallace, 1998). Results along these lines could improve our ability to solve complex distribution problems when capacity plays a prominent role.

Acknowledgments

Special thanks are due to Serge Bisaillon for his help in implementing and testing our solution method. This project started from a collaboration with the Hudson staff at Caliber Logistics, Inc. (now, FDX Supply Chain Services). We are grateful to Tom Escott and Adam Ruff who introduced us to this problem and provided us the data. Bernard Gendron was a Visiting Scientist at the Operations Research Center, Massachusetts Institute of Technology, in 1996 when this project began. His visit was made possible due to a grant from the Département d'informatique et de recherche opérationnelle, Université de Montréal. The Fonds FCAR (Québec), NSERC (Canada), NSF and the U.S. Department of Transportation provided additional financial support for this project.

References

[1] Ahuja R.K. (1998), “Flows and Paths”, Chapter 17 in *Annotated Bibliographies in Combinatorial Optimization*, Dell’Amico M., Maffioli F. and Martello S. (eds.), John Wiley and Sons.

[2] Ahuja R.K., Magnanti T.L., and Orlin J.B. (1993), *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall.

[3] Balakrishnan A., and Graves S.C. (1989), “A Composite Algorithm for a Concave-Cost Network Flow Problem”, *Networks* 19, 175-202.

[4] Balakrishnan A., Magnanti T.L., and Mirchandani P. (1998), “Network Design”, Chapter 18 in *Annotated Bibliographies in Combinatorial Optimization*, Dell’Amico M., Maffioli F. and Martello S. (eds.), John Wiley and Sons.

[5] Balakrishnan A., Magnanti T.L., and Wong R.T. (1989), “A Dual-Ascent Procedure for Large-Scale Uncapacitated Network Design Problems”, *Operations Research* 37, 716-740.

[6] Bell G.J., Lamar B.W., and Wallace C.A. (1998), *Capacity Improvement, Penalties, and the Fixed Charge Transportation Problem*, working paper, The MITRE Corporation.

[7] Bramel J., and Simchi-Levi D. (1997), *The Logic of Logictics*, Springer-Verlag.

[8] Chan L., Muriel A., and Simchi-Levi D. (1997), “Supply-Chain Management: Integrated Inventory and Transportation”, working paper, Department of Industrial Engineering and Management Sciences, Northwestern University.

[9] Cole M.H., and Parthasarathy M. (1998), “Optimal Design of Merge-in-Transit Distribution Networks”, Mack-Blackwell National Rural Transportation Study Center, University of Arkansas.

[10] Cominetti R., and Ortega F. (1997), “A Branch-and-Bound Method for Minimum Concave Cost Network Flows Based on Sensitivity Analysis”, working paper, Departamento de Ingenieria Matematica, Universidad de Chile.

[11] Crainic T.G. (1999), “Long-Haul Freight Transportation”, *Handbook of Transportation Science*, R.W. Hall (ed.), 433-491, Kluwer, Norwell, MA.

[12] Crainic T.G., Frangioni A., and Gendron B. (1998), “Bundle-Based Relaxation Methods for Multicommodity Capacitated Fixed-Charge Network Design Problems”, forthcoming *Discrete Applied Mathematics.*

[13] Crainic T.G., and Laporte G. (1997), “Planning Models for Freight Transportation”, *European Journal of Operational Research* 97, 409-438.

[14] Croxton K.L. (1999), "Modeling and Solving Network Flow Problems with Piecewise Linear Costs, with Applications in Supply Chain Management", Ph.D. Thesis, Operations Research Center, Massachusetts Institute of Technology.

[15] Croxton K.L., Gendron B., and Magnanti T.L. (2000a), "Variable Disaggregation in Network Flow Problems with Piecewise Linear Costs", working paper, Operations Research Center, Massachusetts Institute of Technology.

[16] Croxton K.L., Gendron B., and Magnanti T.L. (2000b), "A Comparison of Mixed-Integer Programming Models for Non-Convex Piecewise Linear Cost Minimization Problems", working paper, Operations Research Center, Massachusetts Institute of Technology.

[17] Gendron B., Crainic T.G., and Frangioni A. (1998), "Multicommodity Capacitated Network Design", Chapter 1 in *Telecommunications Network Planning*, Sansò B. and Soriano P. (eds.), Kluwer Academics Publisher.

[18] Holmberg K. (1994), "Solving the Staircase Cost Facility Location Problem with Decomposition and Piecewise Linearization", *European Journal of Operational Research* 75, 41-61.

[19] Holmberg K., and Ling J. (1997), "A Lagrangean Heuristic for the Facility Location Problem with Staircase Costs", *European Journal of Operational Research* 97, 63-74.

[20] Magnanti T.L., and Wong R.T. (1984), "Network Design and Transportation Planning: Models and Algorithms", *Transportation Science* 18, 1-55.

[21] Mirchandani P.B., and Francis R.L. (1990), *Discrete Location Theory*, Wiley.

[22] Nemhauser G.L., and Wolsey L.A. (1988), *Integer and Combinatorial Optimization*, Wiley.

[23] Popken D. (1994), "An Algorithm for the Multiattribute, Multicommodity Flow Problems with Freight Consolidation and Inventory Costs", *Operations Research* 42, 274-286.

[24] Shapiro J.F. (1993), "Large-Scale Assembly Systems", Chapter 8 in *Handbooks in Operations Research and Management Science, volume 4: Logistics of Production and Inventory*, Graves S.C., Rinnooy Kan A.H.G. and Zipkin P. (eds.), Elsevier Science Publishers.

[25] Vidal C.J., and Goetschalckx M. (1997), "Strategic Production-Distribution Models: A Critical Review with Emphasis on Global Supply Chain Models", *European Journal of Operational Research* 98, 1-18.

Appendix A: Problem Formulation

Sets
N: nodes, subdivided into three sets:
N_S: sources;
N_M: merge centers;
N_C: customers.
A: arcs, subdivided into two sets:
A_{SM}: arcs from sources to merge centers;
A_{MC}: arcs from merge centers to customers.
$N^+(i)$: outward neighbors of node $i \in N$.
$N^-(i)$: inward neighbors of node $i \in N$.
T: time periods ($T = \{t_1, ..., t_n\}$).
M: modes.
K: commodities (components).
$N_C(t)$: customers with some (positive) demand at time $t \in T$.
S_{jitm}: segments of the cost function for mode $m \in M$ on arc $(j, i) \in A_{SM}$ at time $t \in T$.

Data
g_{ijtm}: cost of the total weight transported from merge center $i \in N_M$ by mode $m \in M$ to satisfy demand of customer $j \in N_C(t)$ at time $t \in T$.
e_i^k: unit inventory cost of commodity $k \in K$ at merge center $i \in N_M$.
c_{jim}^s: cost per weight unit, on segment $s \in S_{jitm}$, of using mode $m \in M$ on arc $(j, i) \in A_{SM}$.
f_{jim}^s: fixed cost, on segment $s \in S_{jitm}$, of using mode $m \in M$ on arc $(j, i) \in A_{SM}$.
b_{jitm}^s: upper limit on the weight of flow, on segment $s \in S_{jitm}$, transported using mode $m \in M$ on arc $(j, i) \in A_{SM}$ at time $t \in T$ ($b_{jitm}^0 = 0$).
r_{ijm}: transport time on arc $(i, j) \in A$ using mode $m \in M$.
h^k: weight per unit of commodity $k \in K$.
l^k: volume per unit of commodity $k \in K$.
q_i: volume capacity at merge center $i \in N_M$.
d_{jt}^k: demand of customer $j \in N_C(t)$ for commodity $k \in K$ at time $t \in T$.
$a_{jitm}^k = \min\left\{\sum_{t \leq \tau \leq t_n} \sum_{\iota \in N_C(\tau)} d_{\iota\tau}^k, \left\lfloor \frac{q_i}{l^k} \right\rfloor\right\}$: upper bound on the flow of commodity $k \in K$ on arc $(j, i) \in A_{SM}$ at time $t \in T$ using mode $m \in M$.
$a_{jitm}^{ks} = \min\left\{a_{jitm}^k, \left\lfloor \frac{b_{jitm}^s}{h^k} \right\rfloor\right\}$.
$a_{jitm}^{k\iota\tau} = \min\left\{d_{\iota\tau}^k, \left\lfloor \frac{q_i}{l^k} \right\rfloor\right\}$: upper bound on the flow of commodity $k \in K$ on arc $(j, i) \in A_{SM}$ at time $t \in T$ using mode $m \in M$ destined to customer $\iota \in N_C(\tau)$ at time $\tau \in T$ ($t \leq \tau \leq t_n$).
$a_{jitm}^{k\iota\tau s} = \min\left\{a_{jitm}^{k\iota\tau}, \left\lfloor \frac{b_{jitm}^s}{h^k} \right\rfloor\right\}$.

Variables
v_{ijtm} (*single shipment variable*): 1, if the demand of customer $j \in N_C(t)$ at time $t \in T$ is

satisfied by merge center $i \in N_M$ using mode $m \in M$; 0, otherwise.
w^k_{jitm} (*flow variable*): unit flow of commodity $k \in K$ transported on arc $(j,i) \in A_{SM}$ arriving at time $t \in T$ using mode $m \in M$.
u^k_{it} (*inventory variable*): inventory of commodity $k \in K$ at merge center $i \in N_M$ at the beginning of time period $t \in T$.
x^s_{jitm} (*auxiliary variable*): the total weight transported on arc $(j,i) \in A_{SM}$ at time $t \in T$ using mode $m \in M$, if it lies on segment $s \in S_{jitm}$; 0, otherwise.
y^s_{jitm} (*indicator variable*): 1, if the total weight transported on arc $(j,i) \in A_{SM}$ at time $t \in T$ using mode $m \in M$ lies on segment $s \in S_{jitm}$; 0, otherwise.
w^{ks}_{jitm} (*extended flow variable*): unit flow of commodity $k \in K$, if the total weight transported on arc $(j,i) \in A_{SM}$ at time $t \in T$ using mode $m \in M$ lies on segment $s \in S_{jitm}$; 0, otherwise.
$u^{k\iota\tau}_{it}$ (*destination-based inventory variable*): inventory of commodity $k \in K$ at merge center $i \in N_M$ at the beginning of time period $t \in T$ destined to customer $\iota \in N_C(\tau)$ at time $\tau \in T \, (t \le \tau \le t_n)$.
$w^{k\iota\tau}_{jitm}$ (*destination-based flow variable*): unit flow of commodity $k \in K$ transported on arc $(j,i) \in A_{SM}$ arriving at time $t \in T$ using mode $m \in M$ destined to customer $\iota \in N_C(\tau)$ at time $\tau \in T \, (t \le \tau \le t_n)$.
$w^{k\iota\tau s}_{jitm}$ (*destination-based extended flow variable*): unit flow of commodity $k \in K$ destined to customer $\iota \in N_C(\tau)$ at time $\tau \in T$, if the total weight transported on arc $(j,i) \in A_{SM}$ at time $t \in T$ using mode $m \in M$ lies on segment $s \in S_{jitm}$; 0, otherwise.

Model

Formulation CB

- *objective*

$$\min \sum_{(j,i)\in A_{SM}} \sum_{t\in T} \sum_{m\in M} \sum_{s\in S_{jitm}} (c^s_{jim} x^s_{jitm} + f^s_{jim} y^s_{jitm}) +$$
$$\sum_{i\in N_M} \sum_{t\in T} \sum_{k\in K} e^k_i u^k_{it} + \sum_{(i,j)\in A_{MC}} \sum_{t\in T} \sum_{m\in M} g_{ijtm} v_{ijtm}.$$

- *single shipment constraints*

$$\sum_{i\in N_M} \sum_{m\in M} v_{ijtm} = 1, \quad j \in N_C(t), t \in T.$$

- *flow balance constraints*

$$\sum_{j\in N^-(i)} \sum_{m\in M} w^k_{jitm} + u^k_{it} = \sum_{j\in N^+(i)} \sum_{m\in M} d^k_{jt'} v_{ijt'm} + u^k_{i(t+1)},$$
$$i \in N_M, t \in T, k \in K \ (t' = t + r_{ijm} \le t_n).$$

- *capacity constraints*

$$\sum_{k\in K}\sum_{j\in N^+(i)}\sum_{m\in M} l^k d^k_{jt'} v_{ijt'm} + \sum_{k\in K} l^k u^k_{i(t+1)} \le q_i, \quad i\in N_M, t\in T \ (t' = t + r_{ijm} \le t_n).$$

- *basic variable definition constraints*

$$\sum_{k\in K} h^k w^k_{jitm} = \sum_{s\in S_{jitm}} x^s_{jitm}, \quad (j,i)\in A_{SM}, t\in T, m\in M.$$

$$\sum_{s\in S_{jitm}} y^s_{jitm} \le 1, \quad (j,i)\in A_{SM}, t\in T, m\in M.$$

- *basic forcing constraints*

$$b^{(s-1)}_{jitm} y^s_{jitm} \le x^s_{jitm} \le b^s_{jitm} y^s_{jitm}, \quad (j,i)\in A_{SM}, t\in T, m\in M, s\in S_{jitm}.$$

Formulation CS

- *strong forcing constraints*

$$w^k_{jitm} \le a^k_{jitm} \sum_{s\in S_{jitm}} y^s_{jitm}, \quad (j,i)\in A_{SM}, t\in T, m\in M, k\in K.$$

Formulation CE

- *extended variable definition constraints*

$$w^k_{jitm} = \sum_{s\in S_{jitm}} w^{ks}_{jitm}, \quad (j,i)\in A_{SM}, t\in T, m\in M, k\in K.$$

$$x^s_{jitm} = \sum_{k\in K} h^k w^{ks}_{jitm}, \quad (j,i)\in A_{SM}, t\in T, m\in M, s\in S_{jitm}.$$

- *extended forcing constraints*

$$w^{ks}_{jitm} \le a^{ks}_{jitm} y^s_{jitm}, \quad (j,i)\in A_{SM}, t\in T, m\in M, k\in K, s\in S_{jitm}.$$

Formulation DS

- *destination-based flow balance constraints*

$$\sum_{j\in N^-(i)}\sum_{m\in M} w^{k\iota\tau}_{jitm} + u^{k\iota\tau}_{it} = \sum_{m\in M} d^k_{\iota\tau} v_{i\iota\tau m} + u^{k\iota\tau}_{i(t+1)}$$

$$i\in N_M, t\in T, k\in K, \iota\in N_C(\tau), t\le\tau\le t_n.$$

- *destination-based strong forcing constraints*

$$w^{k\iota\tau}_{jitm} \leq a^{k\iota\tau}_{jitm} \sum_{s\in S_{jitm}} y^s_{jitm}, \quad (j,i)\in A_{SM}, t\in T, m\in M, k\in K, \iota\in N_C(\tau), t\leq\tau\leq t_n.$$

Formulation DE

- *destination-based extended variable definition constraints*

$$w^{k\iota\tau}_{jitm} = \sum_{s\in S_{jitm}} w^{k\iota\tau s}_{jitm}, \quad (j,i)\in A_{SM}, t\in T, m\in M, k\in K, \iota\in N_C(\tau), t\leq\tau\leq t_n.$$

$$x^s_{jitm} = \sum_{k\in K}\sum_{\iota\in N_C(\tau)}\sum_{t\leq\tau\leq t_n} h^k w^{k\iota\tau s}_{jitm}, \quad (j,i)\in A_{SM}, t\in T, m\in M, s\in S_{jitm}.$$

- *destination-based extended forcing constraints*

$$w^{k\iota\tau s}_{jitm} \leq a^{k\iota\tau s}_{jitm} y^s_{jitm}, \quad (j,i)\in A_{SM}, t\in T, m\in M, k\in K, \iota\in N_C(\tau), t\leq\tau\leq t_n, s\in S_{jitm}.$$

Appendix B: Calibration of Parameters

In this Appendix, we specify how we calibrated the parameters of the cutting-plane procedure using instances and workstations described in Section 6.3.

Preliminary experiments allowed us to identify the most "critical" parameters, i.e, those that have the most significant impact on the performance of the method. These parameters are related to the interaction between the cutting-plane procedure and the rounding heuristics: $\overline{v}$, which determines how much effort should be dedicated every time we call the heuristic procedure, and ϵ, which specifies how often the heuristic is to be called. We first experiment with several combinations of values of these parameters, and let the other parameters fixed. More specifically, we statically generate the variables x^s and y^s when we solve the initial LP, and we use $C = 0.01 \times M$ to decide when the cleanup procedure should be called. Table 5 reports the results (averaged over the 16 instances) obtained by the cutting-plane procedure with six combinations of values of $\overline{v}$ and ϵ. We measure the gaps with respect to $Z^* = Z(DE)$ the best lower bound when the approximation to the cost function is used.

The first combination of the parameters in the table corresponds to frequent calls to the heuristic, with relatively little work per call: when $\overline{v} = 1.0$, the heuristic fixes all single shipment variables at once and, consequently, calls the cutting-plane procedure only once to solve the restricted problem. In contrast, when $\overline{v}$ approaches 0.5, the heuristic can make several calls to the cutting-plane procedure , each call following a successful attempt to fix v variables greater than $\overline{v}$. The results in the table support the general conclusion that it is preferable to spend more effort each time the heuristic is performed, than to call it more frequently. In this respect, $\overline{v} = 0.5$ performs uniformly better than the two other tested values of $\overline{v}$. However, it is still necessary to call the heuristic "reasonably" frequently to obtain effective solutions. The upper bounds obtained when ϵ = 1.0e-3,

$\overline{v}$	ϵ	Formulation	ΔZ^U %	CPU (s)
1.0	1.0e-4	DS	17.10	2501.84
		DE	16.45	9214.69
1.0	1.0e-3	DS	16.56	2508.50
		DE	15.89	7513.27
0.7	1.0e-4	DS	11.74	3953.95
		DE	11.00	12524.64
0.7	1.0e-3	DS	12.52	2976.76
		DE	12.19	10161.58
0.5	1.0e-4	DS	7.54	4548.54
		DE	6.82	17084.62
0.5	1.0e-3	DS	8.03	3710.34
		DE	7.75	10513.91

Table 5: Calibration phase – I

which are comparable to the ones computed when $\epsilon = 1.0\text{e-}4$, support this observation. Note that we have performed some preliminary tests with $\epsilon = 1.0\text{e-}2$ and $\epsilon = 1.0\text{e-}5$: either the upper bounds seldom improve or the computation times become prohibitive. Following these tests, we have decided to use the values $\overline{v} = 0.5$ and $\epsilon = 1.0\text{-}3$ for all other experiments, since they represent a fair balance between solution quality and computational efficiency.

The next series of experiments concern the calibration of the remaining parameters of the cutting-plane procedure: C, the threshold used to decide when to apply the cleanup procedure, and the decision whether to generate the auxiliary and indicator variables statically when solving the initial LP, or dynamically, during the course of the algorithm. We have tested three values of C: $0.001 \times M$, which means that the cleanup procedure is frequently applied, $0.01 \times M$ and $1.0 \times M$, which, in practice, means that the cleanup procedure is never applied. When combined with the two strategies for generating the basic variables x^s and y^s ("Static" and "Dynamic"), these values yield six different combinations, the results of which are reported in Table 6.

These results demonstrate that these parameters do not have a strong influence on the performance of the method, since both the upper bound gaps and the CPU times are comparable for all parameter settings. We observe that the dynamic strategy is, on average, slightly more effective than the static one, but consumes more CPU time. With respect to the parameter C, the value $0.01 \times M$ is slightly more effective than the two others. Based on these results, we have selected a static strategy with $C = 0.01 \times M$ for all other experiments, as it represents a fair compromise between effectiveness and efficiency.

Strategy	C	Formulation	ΔZ^U %	CPU (s)
Static	$0.001 \times M$	DS	8.73	3495.37
		DE	7.76	10883.95
Static	$0.01 \times M$	DS	8.03	3710.34
		DE	7.75	10513.91
Static	$1.0 \times M$	DS	8.59	3783.88
		DE	8.13	10666.71
Dynamic	$0.001 \times M$	DS	8.61	3864.89
		DE	7.60	13535.13
Dynamic	$0.01 \times M$	DS	7.78	4829.09
		DE	7.51	13905.36
Dynamic	$1.0 \times M$	DS	8.61	4284.41
		DE	7.61	12956.57

Table 6: Calibration phase – II

Appendix C: Generation of Capacities and Inventories

We generated the merge centers capacities as follows:

1. Compute $q_{\max} = \max_{t \in T}\{\sum_{k \in K} \sum_{j \in N_C(t)} l^k d^k_{jt}\}$ and $q_\lambda = \lceil q_{\max}/(\lambda \times |N_M|) \rceil$.

2. Select randomly $\lfloor |N_M|/2 \rfloor$ pairs of merge centers, so that all selected pairs cover all merge centers, except one if $|N_M|$ is odd, in which case we set the capacity of the remaining merge center to q_λ.

3. For each pair (i_1, i_2), compute $q_{i_1} = (1+\gamma) \times q_\lambda$ and $q_{i_2} = (1-\gamma) \times q_\lambda$, with γ randomly chosen in an interval $I_\gamma \subseteq [0, 1)$ (in all instances, we used $I_\gamma = [0, 0.2]$).

We generated the initial inventories of the merge centers as follows:

1. $U^k = \min\left\{\sum_{i \in N_M} \left\lfloor \frac{q_i}{l^k} \right\rfloor, \sum_{j \in N_C(t_1)} d^k_{jt_1} + \left\lfloor \beta \times \sum_{t_1 < t \le t_n} \sum_{j \in N_C(t)} d^k_{jt} \right\rfloor \right\}$, is the total initial inventory of each component k, where β is a parameter (in all instances, we chose $\beta = 0.2$).

2. Divide U^k randomly among all merge centers, so that the initial inventory assigned to every merge center never exceeds the merge center capacity.