An Optimization Algorithm for the Inventory Routing Problem with Continuous Moves

Martin Savelsbergh
Jin-Hwa Song

The Logistics Institute
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205

Abstract

The typical inventory routing problem deals with the repeated distribution of a single product from a single facility with unlimited supply to a set of customers that can all be reached with out-and-back trips. Unfortunately, this is not always the reality. We focus on the inventory routing problem with continuous moves, which incorporates two important real-life complexities: limited product availabilities at facilities and customers that cannot be served using out-and-back tours. We need to design delivery tours spanning several days, covering huge geographic areas, and involving product pickups at different facilities. We develop an integer programming based optimization algorithm capable of solving small to medium size instances. This optimization algorithm is embedded in local search procedure to improve solutions produced by a randomized greedy heuristic. We demonstrate the effectiveness of this approach in an extensive computational study.

1 Introduction

Most of the inventory routing literature deals with the repeated distribution of a single product from a single production facility to a set of customers with a fleet of homogeneous vehicles over a certain planning horizon. Even in situations with multiple production facilities serving the set of customers, the customers are usually aligned with a particular production facility resulting in the need to solve several single facility inventory routing problems. It is implicitly assumed that the production capacity at each facility is sufficient to serve its aligned customers and that all customers can be visited on an out-and-back trip from the production facility with which it is aligned. Consequently, the inventory routing problem seeks to find tours that can be performed by a single driver on a particular day.

In practice, this may not always be the case. In the liquid gas industry, for example, the production and distribution of Argon does not fit this model. Argon is produced in only a few facilities and, as Argon is a by-product of Oxygen production, the rate at which Argon is produced is determined by the demand for Oxygen rather than the demand for Argon. This leads to mismatches between product availability and product demand. In
addition, because only a few facilities can produce Argon (a separate distillation column is required), many customers cannot be “turned” (served using out-and-back tour) as they are too far away. As a result, a substantial number of Argon customers is served using sleeper teams that are on the road for several days in a row covering huge geographic areas, picking up product at different facilities along the way, and moving almost continuously.

In Savelsbergh and Song [27], we introduced the inventory routing problem with continuous moves to capture the complexities eluded to above, i.e., limited product availabilities at facilities, customers that cannot be served using out-and-back tours, and delivery tours that can last for several days. Furthermore, we proposed and extensively tested an innovative randomized greedy heuristic for the solution of inventory routing problems with continuous moves.

In this paper, we continue our investigation of the inventory routing problem with continuous moves, but focus on the development of optimization algorithms. Optimization technology, even if it can only be applied to small instances, will allow us to further improve the quality of the solutions produced by the randomized greedy heuristic by incorporating it as a sophisticated local improvement procedure.

More precisely, in this paper, we present an integer multi-commodity flow formulation on a time-expanded network for the inventory routing problem with continuous moves and develop a customized solution approach for its solution. The solution approach is capable of producing optimal or near optimal solutions for small to medium size instances in a reasonable amount of time. We demonstrate that by integrating the new optimization technology with the randomized greedy heuristic high quality solutions can be obtained for realistic size instances.

The remainder of the paper is organized as follows. In Section 2, we review the inventory routing literature and formally introduce the inventory routing problem with continuous moves. In Section 3, we present a multi-commodity flow formulation of the inventory routing problem with continuous moves. In Section 4, we discuss the techniques we employed to solve the multi-commodity flow formulation in a reasonable amount of time. In Section 5, we present a variety of computational experiments conducted with data sets derived from real-life Argon production and distribution information. Finally, in Section 6 we develop and test an integrated solution approach.

## 2 Problem Description

The inventory routing problem is an extension of the vehicle routing problem in which routing and inventory control decisions are made simultaneously. However, inventory routing problems are quite different from vehicle routing problems. Vehicle routing problems occur when consumers place orders and the producer, on any given day, assigns the orders for that day to delivery routes. In inventory routing problems, the producer, not
the consumer, decides how much to deliver to which consumers each day. There are no consumer orders. Instead, the producer operates under the restriction that the consumers are not allowed to run out of product. Another key difference is the planning horizon. Vehicle routing problems typically deal with a single day, with the only requirement being that all orders have to be fulfilled by the end of the day. Inventory routing problems deal with a longer horizon. Each day the producer makes decisions about which consumers to visit and how much to deliver to each of them, while keeping in mind that decisions made today impact what has to be done in the future. The objective is to minimize the total cost over the planning horizon while making sure no customers run out of product.

Solving inventory routing problems is difficult. Campbell et al. [11] show that there exists a two-customer instance in which deciding how much to deliver to each customer on a delivery route visiting both of them requires solving a nonlinear optimization problem with a non-unimodal objective function. The difficulty of the inventory routing problem has not discouraged researchers from studying inventory routing problems and developing various solution approaches for inventory routing problems.

Several time-discretized integer programs have been proposed to determine the set of customers to be visited in a short-term planning horizon as well as the amount of product to deliver to them. To be able to accurately reflect costs and time related aspects, the integer programs work with a set of potential delivery routes. Fisher et al. [17] and Bell et al. [9] pioneered this approach when they studied the inventory routing problem at Air Products, a producer of industrial gases. Their integer program decides the delivery volumes to customers, the assignment of customers to routes, the assignment of vehicle to routes, and the start times of routes. Other papers using time-discretized integer programs include Campbell and Savelsbergh [12], Bard et al. [7], and Gaur and Fisher [19].

Another approach that has been pursued aggressively is based on the analysis of the optimal timing of deliveries to a single customer. Dror and Ball [16, 15] pioneered this approach. The optimal replenishment day \( t^*_i \), minimizing the expected total cost for customer \( i \), is used to determine the set of customers considered in a short-term planning problem for the next \( T \) days. If \( t^*_i \leq T \), then the customer will be included and will definitely be visited. A value \( c_t \) is computed for each of the days in the planning period that reflects the expected increase in future cost if the delivery is made on day \( t \) instead of on \( t^* \). If \( t^*_i \geq T \), i.e., the optimal replenishment day falls outside the short-term planning period, then a future benefit \( g_t \) can be computed for making an early delivery to the customer on day \( t \) of the short-term planning period. These computed values reflect the long term effects of short term decisions. An integer program is subsequently solved that assigns customers to a vehicle and a day, or just day, that minimizes the sum of these costs plus the transportation costs. Other papers using this approach or variants of this approach include Bard et al. [7, 6].

As it is impossible in practice to solve even medium size instances to optimality, Song and Savelsbergh [28] have developed linear programming based technology to computed
strong lower bounds so as to be able to evaluate the effectiveness of distribution strategies for inventory routing.

The asymptotic analysis of delivery policies has also been popular. Anily and Federgruen [3, 4] have looked at minimizing long run average transportation and inventory costs by determining long term routing patterns. The routing patterns are determined using a modified circular partitioning scheme. After the customers are partitioned, customers within a partition are divided into regions so as to make the demand of each region roughly equal to a truckload. A customer may appear in more than one region, but then a certain percent of his demand is allocated to each region. When one customer in a region gets a visit, all customers in the region are visited. Anily and Federgruen also determine a lower bound for the long run average cost to be able to evaluate how good their routing patterns are. Using ideas similar to those of Anily and Federgruen, Gallego and Simchi-Levi [18] evaluate the long run effectiveness of direct shipping (separate trips to each customer). They concluded that direct shipping is at least 94% effective over all inventory routing strategies whenever minimal economic lot size is at least 71% of truck capacity. Related papers include Bramel and Simchi-Levi [10], and Chan, Federgruen, and Simchi-Levi [13].

A growing stream of papers focuses on the stochastic nature of real-life inventory routing problems and formulate and study Markov decision processes of inventory routing problems and thus explicitly incorporate demand uncertainty. Minkoff [23] pioneered this approach. To overcome the computational difficulties caused by large state spaces, he proposes a decomposition heuristic. The heuristic solves a linear program to allocate joint transportation costs to individual customers, and then solves individual customer subproblems. The value functions of the subproblems are added to approximate the value function of the original problem. The main limitation of the proposed approach is that it assumes the availability of a set of delivery routes with fixed delivery quantities for the customers on a route (called itineraries by Minkoff) and that the dispatcher only has to decide which of the delivery routes to use at each decision point. This limitation is removed in the work of Kleywegt, Nori, and Savelsbergh [20, 21] on approximate dynamic programming approaches and that of Adelman [1, 2] on price-directed approaches.

We consider the inventory routing problem with continuous moves (IRP-CM), which differs in several aspects from the standard inventory routing problem. A single product must be distributed from a set $P$ of plants to a set $C$ of customers. A set $V$ of vehicles is available for the distribution. Each customer $i \in C$ has a usage rate $u_i$, a local storage capacity $C_i$, and an initial inventory $I_{0i}$ at time 0. Each plant $j \in P$ has a production rate $p_j$, a local storage capacity $C_j$, and an initial inventory $I_{0j}$ at time 0. Each vehicle $k \in V$ has a tank capacity of $Q$ and becomes available at a location $i \in P \cup C$ (either at a plant or at a customer) at time $t_{0k}$ with an initial volume $v_{0k}$ of product in its tank. Travel time between locations $u \in P \cup C$ and $v \in P \cup C$ is $t_{uv}$, and the cost for traveling between $u$ and $v$ is $c_{uv}$. The objective is to minimize transportation costs over the planning horizon $T$ while trying to ensure that none of the customers experiences a stockout and that none
of the plants have to vent product.

We introduced this variant of the inventory routing problem in Savelsbergh and Song [27] and developed a randomized greedy heuristic for its solution. The only other paper (that we are aware of) considering more than one product pickup location is Bard et al. [7]. Their setting includes satellite facilities where vehicles can be reloaded and customer deliveries continued until the closing time is reached. This setting is still far more restrictive than the setting that inspired the inventory routing problem with continuous moves.

3 An Integer Multi-Commodity Flow Formulation

The integer multi-commodity flow formulation for the IRP-CM models the activities of a vehicle as a commodity in a time-expanded network, in which nodes represent a visit to a site at a particular time. The explicit modeling of time allows us to keep track of the inventory levels of customers, plants, and vehicles.

The use of time-expanded networks or dynamic service networks to model operational decision making in freight transportation is a common practice. For surveys of freight transportation modeling, see Crainic and Laporte [14] and Powell [25].

Assume that the planning horizon has been discretized into $T$ periods, $\{1, 2, \ldots, T\}$ and that $t_{uv}$, the travel time between locations $u \in \mathcal{P} \cup \mathcal{C}$ and $v \in \mathcal{P} \cup \mathcal{C}$, is rounded up to the nearest larger multiple of the time unit. Rounding up guarantees that any solution to the multi-commodity flow formulation is feasible in real time.

For ease of presentation, we first present the formulation for the situation in which there is only a single vehicle.

The set of nodes $N$ consists of regular nodes $N_r = \{(i, t): i \in \mathcal{P} \cup \mathcal{C}, t \in \{1, 2, \ldots, T\}\}$, where node $(i, t)$ represents the possibility for the vehicle to visit site $i$ at time $t$, and one dummy node $(0, T + 1)$, representing the fact that the vehicle has to end up somewhere after time $T$, i.e., $N = N_r \cup \{(0, T + 1)\}$. Since the vehicle becomes available at location $i^0$ at time $t^0$, the vehicle starts its tour at node $(i^0, t^0)$. The set of arcs $A$ represents possible vehicle activities, either traveling between two sites or waiting at a site. A travel arc $((i, t), (j, t + t_{ij}))$, with $(i, t)$ and $(j, t + t_{ij})$ in $N_r$, and $i \neq j$, represents the possibility that the vehicle travels from site $i$ to site $j$, starting at time $t$ and ending at time $t + t_{ij}$. We use $c_a$ to denote the cost of using arc $a \in A$. A waiting arc $((i, t), (i, t + 1))$, with $i \in \mathcal{P} \cup \mathcal{C}$ and $t \in \{1, 2, \ldots, T - 1\}$, represents the possibility that the vehicle waits at site $i$ from time $t$ to time $t + 1$. We assume that there is no cost associated with waiting, i.e., $c_a = 0$ for waiting arcs. The remaining arcs are $((i, T), (0, T + 1))$ for all $i \in \mathcal{P} \cup \mathcal{C}$. The cost of these dummy arcs are set to zero as well. Because travel times between sites are assumed to be positive, the network is directed and acyclic.

Let the parameters $u_{(i,t)}$ and $p_{(j,t)}$ represent the usage of customer $i \in \mathcal{C}$ from time
to time $t$ and the production of plant $j \in \mathcal{P}$ from time $t - 1$ to time $t$, respectively.

A binary decision variable $x_a$ is associated with each arc $a \in A$ indicating whether or not the vehicle traverses arc $a$ in an optimal solution. A continuous decision variable $d_n$ is associated with each node $n \in \mathcal{N}_r$ and represents the delivery or pickup quantity at the corresponding site and time. If $n$ represents a customer, then $d_n$ is a delivery quantity, and if $n$ represents a plant, then $d_n$ is a pickup quantity. These two sets of variables are sufficient to completely specify the vehicle’s schedule, i.e., route, departure times, and delivery/pickup quantities. The formulation is completed by several sets of variables representing implied quantities. A continuous variable $v_t$ represents the amount of product in the tank of the vehicle right after time $t$, with $v_0$ denoting the initial product amount in the tank. A continuous variable $I_n$ is associated with each node $n = (i, t) \in \mathcal{N}_r$ and represents the inventory level at site $i$ right after time $t$, with $I_{(i,0)}$ denoting the initial inventory level at site $i$.

We are now in a position to present a multi-commodity flow formulation for the IRP-CM. The first class of constraints ensures flow balance at the nodes, i.e., if the vehicle arrives at a site, then it will also leave that site.

$$
\sum_{a \in \delta^-(n)} x_a - \sum_{a \in \delta^+(n)} x_a = 0, \quad \forall n \in \mathcal{N}_r \setminus \{(i^0, t^0)\},
$$

$$
\sum_{a \in \delta^+((i^0, t^0))} x_a = 1,
$$

$$
\sum_{a \in \delta^-((0, T+1))} x_a = 1.
$$

The second class of constraints ensures that a delivery or a pickup at a site only occurs when the vehicle is at that site.

$$
d_n \leq Q \sum_{a \in \delta^+(n)} x_a, \quad \forall n \in \mathcal{N}_r.
$$

The third class of constraints ensures quantity balance of the product in the vehicle, i.e., the quantity in the tank of the vehicle at $t$ is equal to the quantity in the tank at $t - 1$ minus the quantity of a delivery between $t - 1$ and $t$ plus the quantity of a pickup between $t - 1$ and $t$

$$
v_{t-1} - \sum_{\{n=(i,t) : n \in \mathcal{N}_r, i \in C\}} d_n + \sum_{\{n=(i,t) : n \in \mathcal{N}_r, i \in \mathcal{P}\}} d_n = v_t, \quad \forall t \in \{1, 2, ..., T\}.
$$

The fourth class of constraints ensures inventory balance at the sites, i.e., the inventory at $t$ is equal to the inventory at $t - 1$ minus product usage plus quantity delivered between
\( t - 1 \) and \( t \) for customers, and the inventory at \( t - 1 \) plus production minus quantity picked up between \( t - 1 \) and \( t \) for plants

\[
I_{(i,t-1)} + d_n - u_{(i,t)} = I_{(i,t)}, \quad \forall i \in C, \forall n = (i, t) \in N_r,
\]

\[
I_{(i,t-1)} - d_n + p_{(i,t)} = I_{(i,t)}, \quad \forall i \in P, \forall n = (i, t) \in N_r.
\]

Finally, the bounds on the variables ensure that none of the customers experience a stockout and that none of the plants have to vent product

\[
x_a \in \{0, 1\} \quad \forall a \in A,
\]

\[
d_n \geq 0 \quad \forall n \in N_r,
\]

\[
0 \leq v^t \leq Q \quad \forall t \in \{1, 2, ..., T\},
\]

\[
I_{(i,t-1)} \geq u_{(i,t)}, \quad \forall i \in C, \forall (i, t) \in N_r,
\]

\[
I_{(i,t)} \leq C_i, \quad \forall i \in C, \forall (i, t) \in N_r,
\]

\[
I_{(i,t-1)} \leq C_i - p_{(i,t)}, \quad \forall i \in P, \forall (i, t) \in N_r,
\]

\[
I_{(i,t)} \geq 0, \quad \forall i \in P, \forall (i, t) \in N_r.
\]

The objective is to minimize total transportation costs

\[
\min \sum_{a \in A} c_ax_a.
\]

This single-vehicle formulation can be easily extended to a multi-vehicle formulation. Each vehicle has its own arc set, but shares the node set. The complete multi-vehicle formulation can be found in the appendix.

Before discussing solution techniques, we point out some variations of the problems that can easily be handled.

In the formulation presented above, we minimize transportation costs over the planning horizon, while ensuring that none of the customers experience a stockout and none of the plants vent product. In case stockouts at customers or venting at plants is unavoidable in an instance, there will be no feasible solution. The formulation can easily be expanded with variables that capture potential stockouts at customers and potential venting at plants. It is then possible to apply a hierarchical optimization approach in which in the first phase we minimize stockouts and venting, and in the second phase we minimize transportation costs. It is easy to handle more elaborate usage or production patterns by appropriately modifying the \( u_{(i,t)} \) and \( p_{(j,t)} \) parameters. Various other practical issues can be handled simply by forcing delivery/pickup variables to zero, e.g., delivery windows at customers and plant shut downs, or forcing certain waiting variables to one, e.g., vehicle maintenance.
4 Solution Technology

In practice, a minimum delivery quantity at a customer is often imposed. As volume delivered per mile is one of the key performance indicators tracked in this environment, enforcing a minimum delivery quantity seems reasonable. For customers far away from a plant, it is obvious that delivering only a small quantity is undesirable as it implies that that customer will have to be visited again soon. Of course, there is more to it than that, because two far-away customers that are close together may be best visited together on a single trip potentially delivering a relatively small quantity to one of them. Thus, any minimum delivery quantity enforced at a customer should be determined carefully, based on the customer’s usage rate, its storage capacity, its distance to a plant, and the presence of other customers in its vicinity. We have adopted the use of minimum delivery quantities as part of our optimization approach. Optimal solutions to inventory routing problems over a given planning horizon tend to let the inventory at customers get close to zero towards the end of the planning horizon and tend to make only small deliveries towards the end of the planning horizon. Enforcing a minimum delivery quantity will avoid this behavior to some extent.

A well-known disadvantage of formulations involving time-expanded networks is their size. The size of the instance, in terms of the number of variables, increases rapidly with the length of the planning horizon and the chosen time discretization. To keep the number of variables under control, we apply a variety of reduction techniques. Furthermore, to better solve the resulting integer programs, we develop a set of valid inequalities to strengthen the LP relaxations and we develop a specialized branching scheme to guide the search.

4.1 Problem Size Reduction

Inventory routing problems are typically solved using a rolling horizon framework. A distribution plan is constructed for the current planning period, but only the decisions for the first few days are implemented. The planning period is moved forward and the process repeats. As a consequence, the produced distribution plan needs to be precise for the first few days, but an outline of the schedule for the remaining days suffices because the role of the decisions for these days is to capture the effects of the decisions for the portion of the schedule that will be implemented. The decisions for the later days in the planning period will be revisited when we move to the next planning period. We exploit this flexibility by adopting a different time discretization for the first few days and the remaining days (a finer grain for the near future, and a coarser grain for the not so near future), i.e., we employ a telescoping planning horizon.

To reduce the number of nodes in the network, we make use of the notions of economic fill level and economic window. The economic fill level of a customer is defined as his storage capacity minus his minimum delivery quantity. Only when inventory drops below
the economic fill level a delivery is possible. The economic window is the period between the time when inventory drops below the economic fill level and the time when the inventory reaches zero. We can make deliveries only within the economic window. Therefore, customer nodes for which the inventory level lies outside the economic window can be eliminated from the network. Figures 1 and 2 demonstrate the use of economic windows to reduce the number of nodes.

Figure 1: Customer storage capacity is less than vehicle tank size

Figure 1 illustrates the situation in which a customer’s storage capacity is less than the vehicle’s tank size and Figure 2 illustrates the situation in which a customer’s storage capacity is larger than the vehicle’s tank size. The solid line represents the inventory level over time when deliveries are always made at the earliest possible times, i.e., always receiving the minimum delivery quantity. The dotted line represents the inventory level over time when deliveries are always made at the latest possible times, i.e., always receiving a delivery right when a zero inventory is reached. A simple analysis of the two inventory graphs allows the identification of periods of time in which it is feasible to deliver product to the customer.
Figure 2: Customer storage capacity is greater than or equal to vehicle tank size

More specifically, the delivery periods can be computed as follows. Let $d_i$ denote the minimum delivery quantity for customer $i$. First, suppose that customer $i$ has an initial inventory level $I^0_i$ that is higher than his economic fill level $e_i = C_i - d_i$. Let $\overline{d}_i = \min\{C_i, Q\}$ denote the maximum quantity that can be delivered at customer $i$. The first delivery period for customer $i$ starts at time $t^{s}_{D^1_i} = \frac{I^0_i - e_i}{u_i}$ and ends at time $t^{e}_{D^1_i} = \frac{I^0_i}{u_i}$.

The second delivery period starts at time $t^{s}_{D^2_i} = t^{s}_{D^1_i} + \frac{d_i}{u_i}$ and ends at time $t^{e}_{D^2_i} = t^{e}_{D^1_i} + \overline{d}_i$. The last delivery period satisfies either $t^{s}_{D^k_i} < T$, $t^{e}_{D^k_i} \geq T$ or $t^{e}_{D^k_i} < T$, $t^{s}_{D^{k+1}_i} \geq T$. For the time being, we will assume that the delivery periods do not overlap, i.e., $t^{e}_{D^k_i} < t^{s}_{D^{k+1}_i}, \forall k = 1, 2, ..., l - 1$. When the initial inventory is less than or equal to the economic fill level, the first delivery period changes to $t^{s}_{D^1_i} = 0$ and $t^{e}_{D^2_i} = \frac{I^0_i + d_i - e_i}{u_i}$; the others remain the same.

Under the assumption that delivery periods do not overlap, it is easy to see that except for the final delivery period exactly one delivery has to take place in each delivery period. In the final delivery period, a delivery may or may not be required. (As an aside, in the real-world data sets used in our computational experiments, there are no customers with
overlapping delivery periods when we set the planning horizon to five days, but there are overlapping delivery periods for larger planning horizons.) There are many computational advantages to having no overlapping delivery periods. Therefore, we force \( t^{s}_{D^k_{i+1}} = t^{e}_{D^k_i} + \epsilon \) if \( t^{s}_{D^k_{i+1}} \leq t^{e}_{D^k_i} \), where \( \epsilon \) is a small positive value. Of course doing so may impact the solution quality, but we believe that the impact will be negligible in practice because overlapping delivery periods tend to happen only towards the end of the planning period, and only decisions relating to beginning of the planning period will be implemented within a rolling horizon framework.

Another opportunity for problem size reduction is transportation. Transportation over long distances, resulting in long travel times, is costly and therefore not likely to occur in an optimal distribution plan. Therefore, we may remove some of the transportation options from the problem, i.e., we may eliminate some arcs from the network. The following two rules guide the elimination of arcs from the network. The \((\alpha, \beta_c)\)-rule is concerned with connectivity between customers, and the \(\beta_p\)-rule is concerned with connectivity between customers and plants.

The \((\alpha, \beta_c)\)-rule specifies that if the distance from customer \( j \) to customer \( i \) is greater than or equal to \( \alpha \), then customers \( j \) and \( i \) will not be connected. However, if this results in fewer than \( \beta_c \) customers being connected to customer \( i \), then we create connections to the next closest customers until the number of connections is equal to \( \beta_c \). If customer \( j \) is connected to customer \( i \), then customer \( i \) is also connected to customer \( j \). When examining a possible connection between customers \( i \) and \( j \), we also take into account their minimum delivery quantities. If \( d_i + d_j > Q \), then there is no need for a direct connection, because a vehicle cannot visit these two customers without an intermediate pickup.

The \(\beta_p\)-rule specifies that customer \( i \) is connected only to the \(\beta_p\) nearest plants. If customer \( i \) is connected to a plant, then the plant is automatically connected to \( i \) as well.

We do not allow any arcs between plants, i.e., there is no accessibility between plants. Sometimes this does happen in practice. A vehicle visits a supply facility to deliver a certain amount of product. Even though we do not consider this in order to maintain the simplicity of the model, it is not difficult to allow deliveries to plants in the model. By allowing \( d_n \) to take negative values where \( n \) is a plant node, deliveries to plants will be possible. If no vehicle visits node \( n \), then \( d_n \) has to remain 0.

The choice of parameters impacts both the solution quality and time. Reasonable values for \( \alpha \), \( \beta_c \), and \( \beta_p \) depend on instances.

Once the connectivity lists for customers and plants are built, we define the arc set \( A \) based on the lists and nodes in the delivery periods. The arc set \( A \) consists of the following elements. A waiting arc \(((j, t), (j, t + 1))\) indicates waiting at plant \( j \), where \( j \in \mathcal{P} \) and \( t \in \{1, 2, \ldots, T - 1\} \). \(((j, T), (0, T + 1))\) for all \( j \in \mathcal{P} \) are also in \( A \) as before. For each delivery period of customer \( i \), \(((i, t), (i, t + 1))\) indicates an arc waiting at customer \( i \), where
t \in \{t^i_{D_1}, ..., t^i_{D_r} - 1\}. The last node in a delivery period has to be accessible to the dummy node. So \(((i, [t^i_{D_r}])), (0, T + 1)) is also in A for each delivery period of customer i. Note that customer node \((i, t) \in N\) indicates that \(t\) is in one of delivery periods of customer i. Suppose that i and j are accessible. \(((i, t), (j, t + t_{ij}))\) is an arc between two sites, where \((i, t) \in N, (j, t + t_{ij}) \in N,\) and \(i \neq j\). Suppose that customer i and customer j are accessible and \([t^i_{D_q}] + t_{ij} < [t^j_{D_r}]\). Then \(((i, [t^i_{D_q}]), (j, [t^j_{D_r}])))\) is an arc from the last node in the \(q^{th}\) delivery period of customer i to the first node in the \(r^{th}\) delivery period of customer j.

If needed, we can reduce the arc set even further by limiting the set of plants and customers a particular vehicle can visit.

### 4.2 Customized Integer Programming Techniques

Due to the minimum delivery quantities, a set of customers for which the sum of their minimum quantities exceeds the vehicle capacity \(Q\) cannot be visited together on a single trip between two product pickups. Let \(F\) be a subset of \(C\). A trivial lower bound on the number of vehicle trips needed to make at least one delivery to the customers in \(F\) is \(\lceil \sum_{i \in F} d_i / Q \rceil\). This simple observation forms the basis for a set of valid inequalities, which we call delivery cover inequalities. Before presenting a more detailed discussion of delivery cover inequalities, we have to introduce a few additional concepts.

A supernode \(s^k_u\) represents the set of nodes in the \(k^{th}\) delivery period (as defined above) for customer u. A supernode is called closed if the end time of the corresponding delivery period occurs before the end of the planning horizon; otherwise it is called open. A superarc \((s^k_u, s^k_v)\) represents the set of arcs going between any node in supernode \(s^k_u\) and any node in supernode \(s^k_v\). A superarc \((s^k_u, j)\) represents the set of arcs going between any node in supernode \(s^k_u\) and a node of plant j. A superarc \((j, s^k_u)\) represents the set of arcs going from a node of plant j to any node in supernode \(s^k_u\). A superarc \((s^k_u, (0, T + 1))\) represents the set of arcs going from any node in supernode \(s^k_u\) to the dummy node \((0, T + 1)\). Figure 3 shows these four types of superarcs.

Let \(S\) denote the set of supernodes, \(S^o\) denote the set of open supernodes, and \(S^c\) denote the set of closed supernodes. Furthermore, if we denote by \(\delta^+(s^k_u)\) the set of arcs that belong to superarcs that have supernode \(s^k_u\) as their tail, then we can add the following constraints to the formulation

\[
\sum_{n \in s^k_u} d_n \geq d^i_j, \quad \forall s^k_i \in S^c, \quad (1)
\]

\[
\sum_{a \in \delta^+(s^k_u)} x_a = 1, \quad \forall s^k_i \in S^c. \quad (2)
\]
Constraints (1) and (2) ensure that a closed supernode is visited exactly once and that the delivery size is greater than or equal to the minimum delivery quantity associated with that supernode. Constraints (3) and (4) ensure that when an open supernode is visited the delivery size is greater than or equal to the minimum delivery quantity associated with that supernode.

Next, we present a more detailed description of delivery cover inequalities.

Define $d(H) = \sum_{s^k \in H} d_n$ for any $H \subset S$. Denote the set of arcs belonging to superarcs with both head and tail supernodes in $H$ by $\gamma(H)$. Furthermore, for a set of arcs $A$ let $x(A) = \sum_{a \in A} x_a$. For any subset $H \subset S$, $H \neq \emptyset$, we have that $x(\gamma(H)) \leq |H| - \left\lceil \frac{d(H)}{Q} \right\rceil$.
is a valid inequality, because \( \left\lceil \frac{d(H)}{Q} \right\rceil \) is a lower bound on the number of vehicle trips that visit all supernodes in \( H \).

Figure 4: Simple example of a violated delivery cover inequality

Figure 4 shows a simple example of a delivery cover inequality that is violated by \( \hat{x} \), the optimal solution to the LP relaxation. If we take \( H = \{1, 2, 3\} \), then \( d(H) = 12 \) and \( \hat{x}(\gamma(H)) = 1.8 \), which implies that \( x(\gamma(H)) > |H| - \left\lceil \frac{d(H)}{Q} \right\rceil \), i.e., the delivery cover inequality is violated. The interpretation is as follows. Since the vehicle tank capacity is 10, which is less than the sum of the minimum delivery quantities of the supernodes in \( H \), the vehicle cannot visit the supernodes in \( H \) in a single trip without an intermediate product pickup. That is, at least two vehicle trips are necessary to serve the supernodes in \( H \).

Delivery cover inequalities are similar (but not identical) to the capacity constraints of the *Capacitated Vehicle Routing Problem* (CVRP). A large body of literature on branch-and-cut approaches for the CVRP exists; Naddef and Rinaldi [24], Ralphs et al. [26] and Lysgaard et al. [22] contain overviews of the recent major research activities in this area. The separation of the capacity constraints is known to be NP-complete (see [5]).
Therefore, we use heuristics for the separation of delivery cover inequalities.

### 4.2.1 Separation Heuristics

Two heuristics are used to identify violated delivery cover inequalities.

The first heuristic is the integer connected components separation heuristic, and the second is called the connected components separation heuristic. The connected components heuristic is used only when the integer connected components heuristic fails to return a violated delivery cover inequality.

Let $G_{\hat{x} > 0}$ and $G_{\hat{x} = 1}$ denote the support graphs defined by arcs with strictly positive values and values equal to 1, respectively. $G_{\hat{x} = 1}$ is a subgraph of $G_{\hat{x} > 0}$. If a node in supernode $s$ is connected to a node in supernode $s'$, then we say that supernode $s$ and $s'$ are connected.

The integer connected components separation heuristic is inspired by the simple example of Figure 4. The integer connected components separation heuristic starts with the construction of maximally connected supernode components in $G_{\hat{x} = 1}$. For each supernode component $H$, we find a supernode $s \notin H$ connected to $H$ in $G_{\hat{x} > 0}$. If $\hat{x}(\gamma(H \cup \{s\})) > |H| + 1 - \left\lceil \frac{d(H \cup \{s\})}{q} \right\rceil$, then a violated delivery cover inequality has been found.

It is easy to see that the integer connected components heuristic detects the delivery cover inequality that is violated by $\hat{x}$ in Figure 4. In Figure 4, $H = \{1, 2\}$ is a maximal connected supernode component in $G_{\hat{x} = 1}$ and supernode 3 is connected to $H$ in $G_{\hat{x} > 0}$. When the supernode set $H \cup \{3\}$ is checked, the violated inequality $\hat{x}(\gamma(\{1, 2, 3\})) \leq 1$ is detected.

If the integer connected components separation heuristic fails to determine a violated inequality, we apply the connected components separation heuristic. The connected components separation heuristic is an appropriate modification of the connected components separation heuristic for the CVRP presented in [26]. The connected components separation heuristic begins with the construction of maximally connected supernode components in $G_{\hat{x} > 0}$. For each supernode component $H$, the heuristic checks if $\hat{x}$ violates the delivery cover inequality associated with $H$. If a violation is detected, we add $H$ to $\mathcal{H}$. Otherwise, we identify a supernode $s \in H$ such that the chance of a violation of $H \setminus \{s\}$ is greater than that of $H$. If such a node exists, we set $H \leftarrow H \setminus \{s\}$ and check the violation again. Unless a violated delivery cover inequality is found, we continue this process until no such supernode exists. A precise description of the connected components separation heuristic can be found in Algorithm 1. The input for the connected components heuristic is $G_{\hat{x}}$ and its output is a set of delivery cover inequalities violated by $\hat{x}$. The output delivery cover inequalities are determined by $\mathcal{H}$.

The connected components separation heuristic is only used when the integer connected components separation heuristic fails to determine a violated inequality, as it is
Algorithm 1 Connected components separation heuristic

Find supernode sets $H_1, \ldots, H_r$ of maximally connected supernode components of $G_{\hat{x}>0}$

for $i = 1, 2, \ldots, r$ do
  if $\hat{x}(\gamma(H_i)) > |H_i| - \left\lceil \frac{d(H_i)}{Q} \right\rceil$ then
    Add $H_i$ to $\mathcal{H}$.
  else
    while $\exists s \in H_i$ such that $\left\lceil \frac{d(H_i)}{Q} \right\rceil = \left\lceil \frac{d(H_i \setminus \{s\})}{Q} \right\rceil$ and $|H_i| - 1 - \left\lceil \frac{d(H_i \setminus \{s\})}{Q} \right\rceil - \hat{x}(\gamma(H_i)) < |H_i| - \left\lceil \frac{d(H_i \setminus \{s\})}{Q} \right\rceil$ do
      if $\hat{x}(\gamma(H_i \setminus \{s\})) > |H_i| - 1 - \left\lceil \frac{d(H_i \setminus \{s\})}{Q} \right\rceil$ then
        Add $H_i \setminus \{s\}$ to $\mathcal{H}$
        Break while
      else
        $H_i \leftarrow H_i \setminus \{s\}$
      end if
    end while
  end if
end for

more time consuming. Figure 5 shows an example in which the integer connected components separation heuristic fails to find a violated delivery cover inequality, but the connected components separation heuristic does find one. The integer connected components separation heuristic constructs maximally connected supernode components $\{1, 2\}$ and $\{3, 4\}$. Since the delivery cover inequalities determined by $\{1, 2, 3\}$ and $\{2, 3, 4\}$ are not violated by $\hat{x}$, the integer connected components separation heuristic fails to find a cut. On the other hand, the connected components separation heuristic constructs maximally connected supernode component $H = \{1, 2, 3, 4\}$, which immediately leads to a violated delivery cover inequality.

4.2.2 Search Strategies

The effectiveness of a branch-and-cut algorithm may also be enhanced by incorporating customized branching schemes. We have experimented with two customized branching schemes.

The first scheme favors branching on fractional variables corresponding to transportation early in the planning horizon. The motivation behind this scheme is that determining the earlier part of the schedule will allow the rest of the schedule to fall into place naturally.

The second scheme involves branching on superarcs. It is easy to see that branching on transportation variables $x_a$ leads to unbalanced search trees. Fixing $x_a = 1$ significantly
Figure 5: Simple example for the connected component heuristic

 reduces the solution space (and thus potentially the LP bound), whereas fixing \( x_a = 0 \) has little impact on the solution space. Branching on superarcs will lead to more balanced search trees. When we branch on a superarc, say \((s, s')\), we divide the solution space into schedules in which \( \sum_{a \in (s, s')} x_a = 1 \) and schedules in which \( \sum_{a \in (s, s')} x_a = 0 \). Branching on superarcs is similar to branching on Special Ordered Sets of type I [8].

5 Computational Experiments

We conducted various computational experiments to analyze the performance of the branch-and-cut algorithm. Our base instance involves 7 production facilities, about 200 customers, and a homogeneous fleet of 7 vehicles. We have historical information about production rates at plants and usage rates at customers as well as vehicle capacities and storage capacities at customers. (The data was provided by Praxair Inc., a producer and distributor of industrial gases and long-time member of the Leaders in Logistics program at Georgia Tech.) There is considerable diversity in the set of customers in terms of
their storage capacities and usage rates. As a result, some customers require one or two deliveries every week whereas others require only a single delivery every other month.

From this base instance, we derived three data sets. The first data set of 10 small instances \((s_1, s_2, ..., s_{10})\) involve two plants (randomly selected), about 50 customers (around the two plants), and two vehicles. We randomly generated the initial inventory levels for the customers at the start of the planning period. This set of instances is used to analyze the performance of the various settings of the branch-and-cut algorithm. The second data set of 10 medium size instances \((m_1, m_2, ..., m_{10})\) involves two plants (randomly selected), about 100 customers, and three vehicles. Again, we randomly generated the initial inventory levels for the customers at the start of the planning period. This set of instances is used to compare the optimization approach with the randomized greedy heuristic (RGH). The third data set of 5 large instances \((l_1, l_2, ..., l_{5})\) involves all plants, all customers, and all vehicles. We randomly generated the initial inventory levels and for instances \(l_3, l_4, \text{ and } l_5\) we randomly relocated some of the customer sites to create larger diversity among the instances. This set of instances is used in the next section to evaluate the potential benefits of embedding our optimization approach in a local search scheme to improve the performance of the randomized greedy heuristic. The historical data also contains for each customer the minimum delivery quantity guideline provided to the planners at Praxair. The planners are encouraged to plan only delivery quantities larger than the guideline. The guideline typically sets the minimum delivery quantity at 90% of the maximum delivery quantity. We have used these guidelines as the minimum delivery quantities in all experiments.

CPLEX 9.0 was used as the integer programming solver. The experiments were run on a SUN 280R machine with a 900MHz UltraSparc-III-Cu processor.

The planning horizon used in the experiments is five days. We use different time discretizations for the first two days and the remaining three days. A time discretization \((a,b)\) indicates that the time unit for the first two days is \(a\) hours and the time unit for the next three days is \(b\) hours. We use the following settings for the branch-and-cut algorithm, unless specifically stated otherwise. The time discretization is \((2,4)\), the \((\alpha, \beta_c)\)-rule is \((300,5)\), no \(\beta_p\)-rule is in effect, and no plant-vehicle alignments are specified. That is, each customer can be served from all plants by all vehicles. The chosen parameters for the \((\alpha, \beta_c)\)-rule were such that there was almost no impact on the small and medium size instances, but that for large instances about 20% of the arcs could be eliminated.

By default the superarc branching scheme is used and only delivery cover inequalities are added as cuts, i.e., CPLEX cuts are turned off. Cuts are generated at the first 20 nodes and every 20th node after that.

In the following section, we provide some experimental results to compare these base settings with many other options.
5.1 The Branch-and-Cut Algorithm

In this section, we solve the set of eight small instances to evaluate the effects of various settings and to determine appropriate settings for the branch-and-cut algorithm.

5.1.1 Time Discretization

We experimented with various time discretizations to study the impact of the chosen time discretization on the solution quality and the solution time. The results are presented in Table 1. The last two columns show the percentage increase in solution quality and the percentage increase in solution time when changing from time discretization (2,4) to time discretization (1,1). We see that most of the time, the exception being instance s9, using a finer time discretization leads to only a small improvement in solution quality at the expense of a huge increase solution time. Consequently, we have chosen to use the coarser time discretization (2,4) in our further experiments.

More detailed analysis of the results shows that the chosen time unit has to be small enough to ensure that the delivery periods of customers contain at least few nodes.

<table>
<thead>
<tr>
<th></th>
<th>(2,4)</th>
<th>(2,2)</th>
<th>(1,2)</th>
<th>(1,1)</th>
<th>quality increase from (2,4) to (1,1)</th>
<th>time increase from (2,4) to (1,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>1,474.1</td>
<td>1,474.1</td>
<td>1,474.1</td>
<td>1,474.1</td>
<td>189</td>
<td>4755.1%</td>
</tr>
<tr>
<td>s2</td>
<td>1,515.7</td>
<td>1,515.7</td>
<td>1,515.7</td>
<td>1,515.7</td>
<td>1923</td>
<td>1459.7%</td>
</tr>
<tr>
<td>s3</td>
<td>1,730.4</td>
<td>1,730.4</td>
<td>1,716.9</td>
<td>1,716.9</td>
<td>805</td>
<td>2155.4%</td>
</tr>
<tr>
<td>s4</td>
<td>1,148.9</td>
<td>1,148.9</td>
<td>1,105.4</td>
<td>1,105.4</td>
<td>2,104</td>
<td>771.6%</td>
</tr>
<tr>
<td>s5</td>
<td>1,519.8</td>
<td>1,508.4</td>
<td>1,488.3</td>
<td>1,488.3</td>
<td>375</td>
<td>816.4%</td>
</tr>
<tr>
<td>s6</td>
<td>1,619.3</td>
<td>1,619.3</td>
<td>1,619.3</td>
<td>1,619.3</td>
<td>465</td>
<td>2517.5%</td>
</tr>
<tr>
<td>s7</td>
<td>1,852.2</td>
<td>1,792.5</td>
<td>1,778.3</td>
<td>1,778.3</td>
<td>743</td>
<td>1116.0%</td>
</tr>
<tr>
<td>s8</td>
<td>1,161.3</td>
<td>1,161.3</td>
<td>1,161.3</td>
<td>1,161.3</td>
<td>29</td>
<td>308.8%</td>
</tr>
<tr>
<td>s9</td>
<td>2,073.2</td>
<td>2,073.2</td>
<td>1,628.5</td>
<td>1,628.5</td>
<td>9,924</td>
<td>62907.6%</td>
</tr>
<tr>
<td>s10</td>
<td>2,237.4</td>
<td>2,237.4</td>
<td>2,178.5</td>
<td>2,178.5</td>
<td>10,856</td>
<td>4236.4%</td>
</tr>
<tr>
<td>avg</td>
<td>1,633.2</td>
<td>1,626.1</td>
<td>1,568.6</td>
<td>1,566.6</td>
<td>2,741.3</td>
<td>8104.5%</td>
</tr>
</tbody>
</table>

Table 1: Comparison of various time units

5.1.2 Delivery Cover Inequalities

To study the value of delivery cover inequalities and the impact of the algorithmic choices related to delivery cover inequalities we conducted a variety of experiments. First, we
focused on the value of the delivery cover inequalities. Table 2 compares using only the cuts automatically generated by CPLEX, with using only delivery cover inequalities, and with using both the cuts automatically generated by CPLEX and the delivery cover inequalities. The headings of the sub-columns are self-explanatory. The results in Table 2 clearly demonstrate the value of delivery cover inequalities. In fact, the differences are even more pronounced when the instances get larger. When the instances get larger, the differences between using just delivery cover inequalities and using both the cuts automatically generated by CPLEX and the delivery cover inequalities also become larger, see Table 3. Consequently, we have chosen to use only delivery cover inequalities in our further experiments.

<table>
<thead>
<tr>
<th>Instance</th>
<th>CPLEX cuts only</th>
<th>DC cuts only</th>
<th>CPLEX cuts and DC cuts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time (sec.)</td>
<td>B&amp;C nodes</td>
<td>cuts</td>
</tr>
<tr>
<td>s1</td>
<td>24</td>
<td>264</td>
<td>73</td>
</tr>
<tr>
<td>s2</td>
<td>704</td>
<td>4,542</td>
<td>328</td>
</tr>
<tr>
<td>s3</td>
<td>66</td>
<td>216</td>
<td>71</td>
</tr>
<tr>
<td>s4</td>
<td>17,432</td>
<td>83,924</td>
<td>390</td>
</tr>
<tr>
<td>s5</td>
<td>19,618</td>
<td>89,494</td>
<td>184</td>
</tr>
<tr>
<td>s6</td>
<td>38</td>
<td>510</td>
<td>208</td>
</tr>
<tr>
<td>s7</td>
<td>465</td>
<td>5,572</td>
<td>313</td>
</tr>
<tr>
<td>s8</td>
<td>38</td>
<td>316</td>
<td>57</td>
</tr>
<tr>
<td>s9</td>
<td>410</td>
<td>1,663</td>
<td>19</td>
</tr>
<tr>
<td>s10</td>
<td>9,440</td>
<td>20,992</td>
<td>385</td>
</tr>
<tr>
<td>average</td>
<td>4,823</td>
<td>20,749</td>
<td>203</td>
</tr>
</tbody>
</table>

Table 2: The effect of delivery cover inequalities

As discussed in the previous section, we implemented two separation heuristics to identify violated delivery cover inequalities. With default settings, the connected components separation heuristic is used only when the integer connected components separation heuristic fails to identify a violated delivery cover inequality. Of course, other choices are possible. Table 4 shows the results when only the integer connected components separation heuristic is used (Setting 1), when only the connected components separation heuristic is used (Setting 2), and when the default settings are in effect (Setting 3). The default setting seems to have a slight advantage over the other choices. The instances are solved faster on average and the search trees are smaller on average.

Finally, we evaluated whether generating cuts throughout the search is necessary or whether a cut-and-branch strategy, i.e., generating violated inequalities only at the root node, would be sufficient. Table 5 shows the results when cuts are generated only at the root node, when cuts are generated at the root node plus at every 20th node evaluated
Table 3: The effect of delivery cover inequalities with larger instances

<table>
<thead>
<tr>
<th>Instance</th>
<th>DC cuts only</th>
<th>CPLEX cuts and DC cuts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time (sec.)</td>
<td>B&amp;C nodes</td>
</tr>
<tr>
<td>m1</td>
<td>177,927</td>
<td>12,735</td>
</tr>
<tr>
<td>m2</td>
<td>71,488</td>
<td>7,267</td>
</tr>
<tr>
<td>m3</td>
<td>61,055</td>
<td>6,807</td>
</tr>
<tr>
<td>m4</td>
<td>179,422</td>
<td>26,343</td>
</tr>
<tr>
<td>m5</td>
<td>451,106</td>
<td>35,232</td>
</tr>
<tr>
<td>m6</td>
<td>25,135</td>
<td>7,625</td>
</tr>
<tr>
<td>m7</td>
<td>88,681</td>
<td>10,175</td>
</tr>
<tr>
<td>m8</td>
<td>17,978</td>
<td>1,519</td>
</tr>
<tr>
<td>m9</td>
<td>16,910</td>
<td>3,069</td>
</tr>
<tr>
<td>m10</td>
<td>35,524</td>
<td>3,649</td>
</tr>
<tr>
<td>average</td>
<td>112,523</td>
<td>11,442</td>
</tr>
</tbody>
</table>

Table 4: Comparison of separation heuristics

(a skip factor of 20), and when cuts are generated at every node in the search tree. The results clearly demonstrate that generating cuts throughout the search process significantly reduces solution times.

5.1.3 Branching

We experimented with three branching schemes: CPLEX default branching, a branching scheme that favors branching on fractional variables corresponding to transportation early
Table 5: Frequencies of delivery cover generation

in the planning horizon, and branching on superarcs. The results are presented in Table 6. The results for the branching scheme that favors branching on fractional variables corresponding to transportation early in the planning horizon were disappointing and counter to our expectations. On the other hand, superarc branching appears to be very effective.

Table 6: Comparison of branching strategies
5.2 Comparing Heuristic and Optimization Approaches

One of the motivations for developing an optimization approach for the IRP-CM was to strengthen our confidence in the performance of the randomized greedy heuristic we developed earlier [27]. Therefore, in this section, we compare the performance of the randomized greedy heuristic RGH with that of the optimization approach based on the multi-commodity flow formulation. The comparison is based on the performance on ten medium size instances. The results are presented in Table 7. The integer programming gap is computed as \( \frac{v(RGH) - v(IP)}{v(RGH)} \times 100 \) and the linear programming gap is computed as \( \frac{v(RGH) - v(LP)}{v(RGH)} \times 100 \), where \( v(LP) \) is the linear programming value after adding cuts.

<table>
<thead>
<tr>
<th>Instance</th>
<th>( v(RGH) )</th>
<th>( v(LP) )</th>
<th>( v(IP) )</th>
<th>time (sec.)</th>
<th>LP GAP (%)</th>
<th>IP GAP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>2,470.3</td>
<td>2,009.1</td>
<td>2,245.7</td>
<td>177,927</td>
<td>18.67%</td>
<td>9.09%</td>
</tr>
<tr>
<td>m2</td>
<td>1,852.8</td>
<td>1,590.1</td>
<td>1,840.0</td>
<td>71,488</td>
<td>14.18%</td>
<td>0.70%</td>
</tr>
<tr>
<td>m3</td>
<td>2,722.6</td>
<td>2,465.0</td>
<td>2,784.5</td>
<td>61,055</td>
<td>9.46%</td>
<td>-2.27%</td>
</tr>
<tr>
<td>m4</td>
<td>3,017.9</td>
<td>2,353.5</td>
<td>2,920.4</td>
<td>179,422</td>
<td>22.01%</td>
<td>3.23%</td>
</tr>
<tr>
<td>m5</td>
<td>3,107.1</td>
<td>2,681.6</td>
<td>2,970.5</td>
<td>451,106</td>
<td>13.69%</td>
<td>4.40%</td>
</tr>
<tr>
<td>m6</td>
<td>3,051.0</td>
<td>2,412.2</td>
<td>2,844.0</td>
<td>25,135</td>
<td>20.94%</td>
<td>6.78%</td>
</tr>
<tr>
<td>m7</td>
<td>2,833.0</td>
<td>2,171.1</td>
<td>2,669.7</td>
<td>88,681</td>
<td>23.37%</td>
<td>5.76%</td>
</tr>
<tr>
<td>m8</td>
<td>2,371.5</td>
<td>2,024.5</td>
<td>2,345.3</td>
<td>17,978</td>
<td>14.63%</td>
<td>1.10%</td>
</tr>
<tr>
<td>m9</td>
<td>2,594.3</td>
<td>2,263.7</td>
<td>2,544.1</td>
<td>16,910</td>
<td>12.74%</td>
<td>1.94%</td>
</tr>
<tr>
<td>m10</td>
<td>2,741.7</td>
<td>2,003.1</td>
<td>2,471.1</td>
<td>35,524</td>
<td>26.94%</td>
<td>9.87%</td>
</tr>
<tr>
<td>average</td>
<td>2,676.2</td>
<td>2,197.4</td>
<td>2,563.5</td>
<td>112,523</td>
<td>17.66%</td>
<td>4.06%</td>
</tr>
</tbody>
</table>

Table 7: Comparison of RGH with the optimization approach

A first examination of the results shows that optimization approach does not always produce a better solution than the randomized greedy heuristic. The reason for this is the time discretization. By restricting events to occur at a limited set of points in time, we may eliminate certain feasible schedules. Furthermore, it is also clear that the computational requirements of the optimization approach are too large to make it of practical interest. The randomized greedy heuristic produced solutions for all these instances in only a few minutes.

Overall, we conclude that the solutions produced by the randomized greedy heuristics are good, but that there is room for improvement. This is the topic of the next section, where we use the optimization approach in a local search setting to improve the solution produced by the randomized greedy heuristic.
6 Local Search Using Integer Programming

The optimization approach presented above cannot be used to solve real-life instances directly because of prohibitive computational times, but it can effectively solve small instances. This suggests that the optimization approach may be used successfully to improve portions of the schedule produced by the randomized greedy heuristic. We investigate the potential of such an approach in this section. We use the randomized greedy heuristic [27] to produce a complete schedule and then use the optimization approach to improve portions of that schedule. To ensure that the optimization approach is always able to return a schedule that is at least as good as the current schedule, we modify the underlying network structure in such a way that the current schedule constitutes a feasible solution. That is, we expand the underlying network by introducing, if needed, additional nodes at the times corresponding to deliveries and pickups in the current schedule. This has the additional advantage that we can provide the integer programming algorithm with a bound on the optimal objective function value, which will often result in reduced solution times.

It is important to realize that the integer programs do not have to be solved to proven optimality. We can stop either at the first feasible solution (which already corresponds to an improved schedule), or after a fixed amount of time.

We have implemented the idea outlined above by optimizing the schedules of a subset of the vehicles while keeping the schedules of the remaining vehicles fixed. Consider two vehicles, say $k_1$ and $k_2$. Fixing all variables associated with vehicles other than $k_1$ and $k_2$ at values corresponding to the current schedule defines an integer program that optimizes the schedules of vehicles $k_1$ and $k_2$. Of course, this approach can be applied iteratively, each time optimizing the schedules for two vehicles. Therefore, the suggested approach can be viewed as a neighborhood search scheme that relies on integer programming to explore neighborhoods. The neighborhood consists of all feasible schedules for the specified set of vehicles, customers, and plants.

There are still several decisions that have to be made concerning an actual implementation. How do we select the two vehicles that define the neighborhood? When do we stop the neighborhood search? Do we only fix the travel itinerary for the other vehicles (through the binary variables) or do we also fix the pickup and delivery quantities for the other vehicles (through the continuous variables)?

As our goal is to establish the potential value of this type of neighborhood search, we chose a relatively simple and straightforward scheme. Let $C_k$ denote the set of customers visited by vehicle $k$ in the current schedule. Among the subsets of two vehicles that have not been selected before, we select the subset of vehicles $\{k_1, k_2\}$ with the largest number of customers being visited by both $k_1$ and $k_2$, i.e., for which $|C_{k_1} \cap C_{k_2}|$ is maximum. In case of ties, we choose the subset of vehicles visiting the largest number of customers, i.e., for which $|C_{k_1} \cup C_{k_2}|$ is maximum. We fix the travel itinerary for all the other vehicles and solve the resulting optimization problem using our branch-and-cut algorithm.
(to optimality). We stop after all subsets of two vehicles have been evaluated.

This scheme has been evaluated on five large instances, each with 7 plants, 200 customers, and 7 vehicles, and covering a planning period of five days. The initial schedule was produced with the randomized greedy heuristic. Since there are 7 vehicles, a total of 21 integer programs is solved for each instance. The results can be found in Table 8. We present the value of the initial schedule ($v_{RGH}$), the value of the improved schedule ($v_{LS}$), the percentage improvement, the average time per neighborhood exploration in seconds, and the number of neighborhood explorations that were successful, i.e., produced an improved solution. Given that the randomized greedy heuristic already produces good quality solutions, it is promising to see that this type of neighborhood search is able to improve the overall solution quality by more than 3% on average.

<table>
<thead>
<tr>
<th>Instance</th>
<th>$v_{RGH}$</th>
<th>$v_{LS}$</th>
<th>perc. improvement</th>
<th>average time (sec.)</th>
<th>#successes</th>
</tr>
</thead>
<tbody>
<tr>
<td>l1</td>
<td>5541.0</td>
<td>5233.4</td>
<td>5.55%</td>
<td>413.8</td>
<td>2</td>
</tr>
<tr>
<td>l2</td>
<td>8909.0</td>
<td>8614.8</td>
<td>3.30%</td>
<td>194.5</td>
<td>5</td>
</tr>
<tr>
<td>l3</td>
<td>8546.3</td>
<td>8243.8</td>
<td>3.54%</td>
<td>145.8</td>
<td>2</td>
</tr>
<tr>
<td>l4</td>
<td>7515.4</td>
<td>7374.0</td>
<td>1.88%</td>
<td>103.5</td>
<td>3</td>
</tr>
<tr>
<td>l5</td>
<td>6290.8</td>
<td>6234.5</td>
<td>0.90%</td>
<td>16.0</td>
<td>3</td>
</tr>
<tr>
<td>average</td>
<td>7360.5</td>
<td>7140.1</td>
<td>3.03%</td>
<td>174.7</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 8: Improvements from neighborhood search

In a more sophisticated implementation of this neighborhood search scheme, a careful analysis of the current schedule should guide the selection of subsets of vehicles, which may lead to a different sequence of optimization problems, and the procedure may be terminated without solving an optimization problem for every possible subset of two vehicles.

References


26


Appendix

Below we present the complete multi-vehicle multi-commodity flow formulation for the inventory routing problem with continuous moves. The index \(k\) is used here to represent each vehicle.

\[
z = \min \sum_{k \in V} \sum_{a \in A^k} c_a x_a \tag{5}
\]

s.t.

\[
\sum_{\{a \in A^k : a \in \delta^- (n)\}} x_a - \sum_{\{a \in A^k : a \in \delta^+ (n)\}} x_a = 0, \forall k \in V, \forall n \in N_r \setminus (i_0^k, t_0^k), \tag{6}
\]

\[
\sum_{\{a \in A^k : a \in \delta^+ (i_0^k, t_0^k)\}} x_a = 1, \forall k \in V, \tag{7}
\]

\[
\sum_{\{a \in A^k : a \in \delta^- (0,T+1)\}} x_a = 1, \forall k \in V, \tag{8}
\]

\[
v_k^{t-1} - \sum_{\{n=(i,t) : i \in C\}} d_n^k + \sum_{\{n=(i,t) : i \in P\}} d_n^k = v_k^t, \forall t \in \{1, 2, \ldots, T\}, \forall k \in V, \tag{9}
\]

\[
I_{(i,t-1)} + \sum_{k \in V} d_n^k - u_{(i,t)} = I_{(i,t)}, \forall i \in C, \forall n = (i, t) \in N_r, \tag{10}
\]

\[
I_{(i,t-1)} - \sum_{k \in V} d_n^k + p_{(i,t)} = I_{(i,t)}, \forall i \in P, \forall n = (i, t) \in N_r, \tag{11}
\]

\[
I_{(i,t)} \geq u_{(i,t)}, \forall i \in C, \forall (i, t) \in N_r, \tag{12}
\]

\[
I_{(i,t)} \leq C_i - p_{(i,t)}, \forall i \in P, \forall (i, t) \in N_r, \tag{13}
\]

\[
d_n^k \leq Q \sum_{\{a \in A^k : a \in \delta^+ (n)\}} x_a, \forall k \in V, \forall n \in N_r, \tag{14}
\]

\[
x_a \in \{0, 1\} \forall a \in A^k, \forall k \in V, \tag{15}
\]

\[
d_n^k \geq 0 \forall k \in V, \forall n \in N_r, \tag{16}
\]

\[
0 \leq v_k^t \leq Q \forall t \in \{1, 2, \ldots, T\}, \forall k \in V, \tag{17}
\]

\[
I_{(i,t)} \leq C_i, \forall i \in C, \forall (i, t) \in N_r, \tag{18}
\]

\[
I_{(i,t)} \geq 0, \forall i \in P, \forall (i, t) \in N_r. \tag{19}
\]

The flow conservation constraints (6)-(8) ensure that the solution contains a path for vehicle \(k\) from node \((i_0^k, t_0^k)\) to node \((0, T+1)\). Constraints (9) keep track of the product quantity in the vehicle. Constraints (10) and (11) keep track of the product inventory at customers and plants, respectively. Constraints (12) ensure that customers do not experience a stock out. Constraints (13) ensure that plants do not have to vent product.
Constraints (14) ensure that we only make deliveries/pickups at a time when there is a vehicle at the site. Constraints (15)-(19) represent boundary conditions. The objective (5) is to minimize total transportation costs over the planning horizon $T$. 