Utility companies typically send their meter readers out each day of the billing cycle in order to determine each customer’s usage for the period. Customer churn requires the utility company to periodically remove some customer locations from its meter-reading routes. On the other hand, the addition of new customers and locations requires the utility company to add new stops to the existing routes. A utility that does not adjust its meter-reading routes over time can find itself with inefficient routes and, subsequently, higher meter-reading costs. Furthermore, the utility can end up with certain billing days that require substantially larger meter-reading resources than others. However, remedying this problem is not as simple as it may initially seem. Certain regulatory and customer service considerations can prevent the utility from shifting a customer’s billing day by more than a few days in either direction. Thus, the problem of reducing the meter-reading costs and balancing the workload can become quite difficult.

We describe this Balanced Billing Cycle Vehicle Routing Problem in more detail and develop an algorithm for providing solutions to a slightly simplified version of the problem. Our algorithm uses a combination of heuristics and integer programming via a three-stage algorithm. We discuss the performance of our procedure on a real-world data set.

Keywords: vehicle routing problem; heuristics; meter-reading routes

1. INTRODUCTION

A typical utility company issues bills to its customers on a monthly basis. However, because each customer’s usage varies from one month to the next, the utility company must accurately determine the amount of each bill by physically inspecting the customer’s meter. While some utility companies are using technological advances such as RFID to simplify this task [19], many others still send out a fleet of workers each day to individually read each customer’s meter. These meter readers begin their day at a central location, visit a set of specified locations, and then return to this central facility.

Because of the periodic and largely predictable nature of this activity, we expect that most utility companies are very efficient in accomplishing this monthly meter-reading task. However, because the utility’s customer set changes as accounts are closed and new accounts are added, there is the potential for the quality of these meter-reading routes to deteriorate over time if the routes and billing day assignments are not closely monitored.

As an example, consider a fixed customer location and assume that the meter belonging to this customer is read on the 10th day of each month. Since the utility company clearly wishes to minimize the total cost of the meter-reading process, the company will attempt to minimize the total driving time required to read all the necessary meters. Thus, we expect that many meters in the immediate vicinity of this particular location will also be read on the 10th. After a period of time, suppose that the current resident moves away and cancels service with the utility company. A new resident moves in at a later date and begins service, say on the 20th day of the month. We expect that this new resident’s meter will be read on the 10th day of each month instead of the 20th as it is logical to believe that the meter reader servicing this particular area will simply add this new customer to the existing route. However, in practice, this is not always the case as the new resident’s meter is sometimes read on the 20th of the month, rather than on the 10th day.

Over time, as specific meters are transferred among different customers and accounts are canceled and renewed, the utility company’s meter-reading routes become less efficient
as new customers are added to old, inappropriate routes. Thus, utility companies that fail to adjust their routes and billing day assignments in response to this customer turnover end up with inefficient, fractured routes, as well as meter-reading workloads that are not balanced across the billing cycle.

We expect that once a utility realizes that its routes are inefficient and are increasing its operating costs, it would simply adjust certain customers’ monthly billing days in order to quickly cut costs by shifting customers to more appropriate routes. However, certain regulatory constraints forbid a utility company from shifting a given customer’s billing date by more than a few days [4]. In addition, some utilities wish to avoid significant changes in billing dates as many customers resent the potentially large increases in their bills. Thus, even if the utility company knew that it could immediately reduce routing costs by shifting a customer’s meter-reading day from the 20th to the 10th day of the month, there are often external considerations that prevent it from making such a change in a single billing period. After months and even years of this route fracturing phenomenon, a utility can find itself with terribly inefficient and unbalanced routes and faced with the daunting task of creating efficient and balanced routes. Some utilities resort to manual methods and attempt to repair such configurations by hand. Other approaches include a technique called string indexing that attempts to balance routes in terms of the number of customers. A single, giant tour is constructed and then partitioned into routes containing roughly the same number of customers by simply splitting the giant tour in certain places [15]. Clearly, these methods are ad hoc at best, and this article attempts to provide an effective and automated procedure for repairing such fractured configurations.

We now consider the goals of the utility. First and foremost, the utility would like to minimize its meter-reading costs by creating more efficient routes. By improving the routes for each billing day, the utility can reduce its operating costs by decreasing the total mileage traveled by individual meter readers. Additionally, it is possible that the utility can reduce its labor costs since the more efficient routes may also require fewer overall meter readers. The utility company has two secondary goals related to balancing the workload across the different days of the billing cycle. First, the utility company wants to ensure that the number of required meter readers is constant across the billing cycle. In other words, the utility would like to develop meter-reading routes and billing day assignments that do not require extra meter readers or increased (overtime) labor costs on certain days of the billing cycle. Second, the utility wants to ensure that the actual number of customers requiring meter-reading service on any given day of the billing cycle is roughly constant. In certain cases, the billing data and other customer information are stored on older mainframe computer systems. When the meter readers download each day’s billing data onto their personal hand-held devices and then upload the new readings to the main database, a particularly large number of customers on any given day can slow down the system to the point where the next day’s meter reading is delayed [15].

In this article, we consider the following problem. We are given a set of existing meter-reading routes for each day of the billing cycle. We create meter-reading routes for each billing day and assign customers to these routes with the goal of reducing the total length of the meter-reading routes while observing regulatory and customer service considerations when assigning customers to new billing days. We have the secondary goal of balancing the meter readers’ routes in terms of both total route length and the number of meters read per day.

The remainder of this article is organized as follows. In Section 2, we provide a mathematical formulation of this problem. In Section 3, we develop a method for obtaining solutions to this problem. In Section 4, we present the results of our algorithm on a real-world data set. We summarize our progress and present ideas for future research in Section 5.

2. PROBLEM STATEMENT

The Vehicle Routing Problem (VRP) has been studied by the operations research community for nearly 50 years. The addition of real-world constraints to the problem has led to many variants including the VRP with Time Windows, the Inventory Routing Problem, and the Period Routing Problem [2, 3, 5, 7]. While our current problem shares some characteristics with well-known VRP variants, the Balanced Billing Cycle VRP (BBCVRP) has several aspects that are quite different.

In many VRP variants, the problem is solved from scratch. However, in the BBCVRP, we start with an existing set of unbalanced and inefficient routes and then move towards a more efficient solution that satisfies the routing and billing day constraints.

A second aspect of the BBCVRP that is largely absent from problems in the VRP literature is the notion of balancing the routes. Any company that employs multiple drivers to read meters or deliver goods would like each driver to individually work near capacity on each day, implying some degree of balance across the work days. Despite the simplicity and necessity of this objective in a wide variety of scenarios, it appears that only a handful of papers have considered the issue of balance in any detail.

Renaud, Doctor, and Laporte [18] use the notion of balance when constructing 2-petals as they develop their routes. The notion of balance is also addressed by the heuristic proposed in [1], but the problem they consider involves fewer than 30 nodes. More recently, balancing route length has been addressed in a series of papers by Jozefowiez, Semet, and Talbi [12–14]. They treat the problem as a multi-objective optimization problem and formally propose the Vehicle Routing Problem with Route Balancing (VRPRB). In this problem, the goal is to minimize the total route length as well as the difference between the longest and shortest route. They solve this problem with a genetic algorithm and a local search method they refer to as Target Aiming
Pareto Search. They report the performance of their procedure on some of the classical benchmark problems given in [8] and [9].

We now discuss two different approaches to solving the BBCVRP. An iterative approach takes the existing configuration and tries to create the routes for the following billing period without considering future periods. After successively modifying several billing periods in this fashion, trying to balance the routes in the length and customer dimensions, this procedure ends when no further improvements can be made, thereby reaching a local minimum. A targeted approach has a longer term perspective and begins by constructing an efficient and balanced routing and billing day configuration while ignoring the initial routes altogether. This approach transitions to this final target configuration via a series of intermediate billing periods.

We adopted the targeted approach for two reasons. First, we can be certain that we will eventually reach a desirable configuration after a number of intermediate periods. Second, by ignoring the initial configuration altogether, we can use powerful VRP heuristics when creating the final target configuration from scratch, rather than relying on ad hoc methods to improve an existing set of routes. We note that the targeted approach does not explicitly account for the potential addition and removal of customer locations that may occur during the intermediate transition periods. However, because the number of such locations is likely to be small in any given month, it should be possible to accommodate these changes by slightly modifying the intermediate routes. We envision that our targeted procedure would be applied periodically to correct a severely fractured routing and billing day configuration.

Using the notation presented in Table 1, we now state our objectives more precisely. In the targeted approach, we require a number of intermediate periods to reach the final configuration. The primary objective is to minimize the cost of the routes in the final set of target routes. We wish to minimize the value of $K$ as well as the quantity

$$
\sum_{r=1}^{M} \sum_{j=1}^{D} L_K(r,j).
$$

We now turn to the constraints. For all billing periods $1 \leq t \leq K$, all routes $r$, and days $d$, we must satisfy the individual meter-reading length and capacity constraints

$$
L_t(r, d) \leq \bar{L}
$$

$$
C_t(r, d) \leq \bar{C}.
$$

We also attempt to satisfy the balance constraints imposed by the lower bounds

$$
L_t(r, d) \geq \underline{L}
$$

$$
C_t(r, d) \geq \underline{C}.
$$

To address the constraint that we cannot shift any customer’s billing day by more than $\pm S$ days from one month to the next, we introduce the notion of billing distance. Given two billing days $1 \leq u, v \leq D$, we define $\|u, v\|_D$ as

$$
\|u, v\|_D = \|v, u\|_D = \min(u - v \mod D, v - u \mod D).
$$

In other words, $\|u, v\|_D$ is simply the minimum number of billing days separating $u$ and $v$ in a $D$-day cycle if we allow for wraparounds. Thus, when we assign some customer $i$ to a new billing day in period $t + 1$, we must ensure that

$$
\|d_t(i), d_{t+1}(i)\|_D \leq S.
$$

### 3. SOLUTION ALGORITHM

As mentioned earlier, our approach to solving the BBCVRP is targeted. We construct an idealized set of target routes and then transition to this configuration through a series of intermediate routes. Before describing our three-phase procedure, we discuss our assumptions.

#### 3.1. Assumptions

First, we assume that only a single meter reader operates on each day of the billing cycle. Although this clearly makes the problem simpler as we are now required to create only a single meter-reading route for each day of the billing cycle, we argue that this assumption is not very restrictive. Given existing routes for $M$ meter readers, we can simply run our procedure $M$ separate times, thereby producing a set of balanced target routes for each meter reader. As the original customer sets for each meter reader are assumed to be disjoint, we produce a set of $M$ routes per day that are balanced and efficient. If the initial routes are highly fractured and imbalanced so that this approach is not easy to execute, we could still proceed under the assumption of having a single

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>The number of meters that must be read in each billing cycle</td>
</tr>
<tr>
<td>$(\underline{C}, \bar{C})$</td>
<td>The minimum and maximum number of customers that an individual meter reader can visit in a given day</td>
</tr>
<tr>
<td>$(\underline{L}, \bar{L})$</td>
<td>The minimum and maximum allowed route length for an individual meter reader</td>
</tr>
<tr>
<td>$D$</td>
<td>The number of days in the billing cycle</td>
</tr>
<tr>
<td>$S$</td>
<td>The maximum number of days that we are allowed to shift a customer’s billing day in either direction</td>
</tr>
<tr>
<td>$M$</td>
<td>The number of meter readers used daily by the utility in its existing routes</td>
</tr>
<tr>
<td>$L_t(r, d)$</td>
<td>The length of route $r$ on day $d$ in billing period $t$</td>
</tr>
<tr>
<td>$C_t(r, d)$</td>
<td>The number of customers on route $r$ on day $d$ in billing period $t$</td>
</tr>
<tr>
<td>$d_t(i)$</td>
<td>The billing day of customer $i$ during billing period $t$</td>
</tr>
<tr>
<td>$f(i)$</td>
<td>The final billing day of customer $i$</td>
</tr>
</tbody>
</table>
(very fast) meter reader and then partition each route in the resulting solution into $M$ separate routes.

Second, we treat the meter-reading problem as a node routing problem rather than an arc routing problem where a street network is traversed. While this assumption allows us to use many of the standard solution techniques developed for the VRP, many parts of our procedure would apply to an arc routing setting as well.

Third, we assume that the utility is willing to employ additional meter readers during the transition periods, if needed. Because we are able to present a final, balanced set of routes that will be reached after a bounded number of intermediate periods, this assumption is reasonable in that these additional resources are required for only a few months.

### 3.2. Phase 1

In Phase 1 of the algorithm, we completely ignore the existing routing configuration and create a set of target routes. As there is only a single meter reader, we create a single meter-reading route for each day. We generate the routes by applying a modified version of the VRP record-to-record travel algorithm (see [11] and [16]). We chose this meta-heuristic for its simplicity, its ability to handle different types of constraints, and its ability to quickly generate high-quality solutions for large VRPs as demonstrated in [16]. We modified the general VRP algorithm given in [16] by forbidding certain moves and including penalties and rewards for moves that hurt or improve the balance of the routes (the details of the record-to-record travel algorithm for the VRP are described in the Appendix).

We begin by generating a set of exactly $D$ routes that obey the limits imposed by (2) and (3). The record-to-record travel algorithm does not explicitly attempt to reduce the number of routes. However, the algorithm’s performance on benchmark problems indicates that it typically finds solutions with a minimum number of routes [16]. We rely on this property of the record-to-record travel procedure in Algorithm 1 where we attempt to find a solution with exactly $D$ routes. Additionally, we note that since we are typically provided with an initial (unbalanced and potentially inefficient) solution with $D$ routes, we expect to encounter little difficulty in finding a solution with the desired number of routes.

At the conclusion of this procedure, we have a set of exactly $D$ routes, one for each day of the billing cycle. Each of these routes is feasible in terms of the maximum route length and the maximum number of customers, but typically is not feasible in terms of the minimums imposed by constraints (4) and (5). The next step is to try to modify the existing routes in order to meet these balancing constraints.

We again utilize the general record-to-record travel framework, except that we now guide the search by forbidding certain moves and perturbing the evaluation of other types of moves. We focus solely on those moves that involve the exchange of nodes between two separate routes.

First, when we consider the acceptance of any inter-route move, we check to see if constraints (4) and (5) are satisfied by the current solution. If not, then we reject any move that decreases either the current minimum route length or the current minimum number of customers on a route. Second, as long as constraints (4) and (5) are violated by some route, we perturb the evaluation of certain moves as follows. Suppose we have some inter-route move $m$ that involves route $r$. Let $\delta(m)$ represent the change in the objective function value if the move were made. If route $r$ is currently the shortest route in the solution and the length of route $r$ increases after making the move $m$, then we would like to reward $m$. Similarly, if route $r$ contains the fewest number of customers and the number of customers in route $r$ increases if we make the move $m$, then we would also like to reward this move. In either case, we set $\delta(m) = \delta(m) - \beta \times |\delta(m)|$ where $\beta \in (0, 1)$ is a fixed balance parameter. This modification has the effect of making $m$ appear more attractive when compared to other potential moves.

On the other hand, if our guided record-to-record travel search discovers a solution that does not violate constraint (4) or (5), then we refrain from rewarding any more moves but we forbid any moves that lead to violations of these constraints. Computational experiments (discussed in Section 4) have shown that this procedure is typically able to find solutions that satisfy constraints (4) and (5) with little sacrifice in quality.

### 3.3. Phase 2

At the end of Phase 1, we have $D$ different meter-reading routes, one for each day of the billing cycle. Each of the customers on a route had an original billing day. The goal of Phase 2 is to assign a single billing day to each of our target routes so that the transition from our initial routes and billing day configuration will be as quick and efficient as possible.

A simple way of assigning billing days to our target routes would be to use the modal value. For each route, we find the most common original billing day among all customers in this route, and then assign this billing day to each customer in this route. However, there are at least two problems with this approach. First, we could encounter two different routes.

---

**Algorithm 1: Generate Initial Routes**

```plaintext```
input : A set of $N$ customer locations
output: A set of $D$ routes that visit all $N$ locations and obey the length and capacity upper bounds $L$ and $C$.
begin
  Construct an initial solution $I$ using concurrent Clarke-Wright [10], stopping if we reach a solution with $D$ routes.
  while The number of routes in $I$ is greater than $D$
    do
      Apply the record-to-record travel algorithm
    end
end
```

---

246 NETWORKS—2009—DOI 10.1002/net

with the same modal billing day. Second, if we choose this modal billing day for all the customers in this target route, there could be some customers that can only reach this billing day following many billing day shifts due to the shifting constraint (7). In other words, this strategy does not account for those customers which are far away from the modal value in terms of the billing day distance.

To address these problems, we formulate an assignment problem with an appropriate cost function to determine how to allocate our resources to the modal value in terms of the billing day distance. We let \( b_{ij} \) to be the cost of assigning billing day \( j \) to customer \( i \). This cost is calculated in terms of the billing distance:

\[
b_{ij} = \begin{cases} 
0, & \text{if } \|d_0(i), j\|_D \leq S \\
\|d_0(i), j\|_D, & \text{otherwise.}
\end{cases}
\]

If customer \( i \) can be shifted from the original billing day to day \( j \) in a single shift of size less than or equal to \( S \), then we incur no cost. If the shift is greater than \( S \), then we simply use the billing distance. We let \( C_{Rj} \) denote the cost of assigning billing day \( j \) to all the customers in target route \( R \) and set this quantity to be the sum of these billing shift costs for each customer in the route, that is, \( C_{Rj} = \sum_{i \in R} b_{ij} \). Letting \( x_{Rj} = 1 \) if day \( j \) is assigned to route \( R \) and \( x_{Rj} = 0 \) otherwise, we have the following assignment problem.

\[
\text{Minimize } \sum_{R=1}^{D} \sum_{j=1}^{D} C_{Rj} x_{Rj} \quad (8a)
\]

s.t.

\[
\sum_{R=1}^{D} x_{Rj} = 1 \text{ for } j = 1, 2, \ldots, D \quad (8b)
\]

\[
\sum_{j=1}^{D} x_{Rj} = 1 \text{ for } R = 1, 2, \ldots, D \quad (8c)
\]

\[
x_{Rj} \in \{0, 1\} \text{ for } R = 1, 2, \ldots, D \text{ and } j = 1, 2, \ldots, D. \quad (8d)
\]

In Table 2, we show how the solutions to the assignment problem change as we increase \( S \) in an illustrative example with a 10-day billing cycle. For each of the 10 target routes, we give the original billing day mixtures, modal value, and assigned final billing days.

In Table 2, the modal value is not unique as day 7 is assigned to the two target routes corresponding to columns 7 and 11, and day 8 is assigned to the two target routes corresponding to columns 5 and 8. As an example of the effect of the shift size on the solution to the assignment problem, consider the fourth column that represents a route where 31 customers have original billing day 3, 13 customers have original billing day 9, and the remaining 51 customers have original billing day 10. For \( S = 2, 3, \) and 4, all customers are able to transition to the assigned final billing day in a single move. However, the actual final billing day itself varies as larger shifts are allowed. For \( S = 3 \), we see that these customers are assigned the final day of 10 and receive the billing day 6 for \( S = 4 \). Billing day 10 is given to the customers in the third column when \( S = 4 \), as this assignment allows all of these customers to transfer to their final day in a single shift.

After solving the assignment problem for a specific value of \( S \), we have a set of \( D \) different target routes that have now been assigned a unique billing day. In other words, for each customer \( i \), we now have a final billing day \( f(i) \). Next, we construct a sequence of intermediate transition routes that allow us to move customers from their original routes and billing days to their final routes and billing days.

### 3.4. Phase 3

In Phase 3, we start with the initial routes at time \( t = 0 \) and iteratively construct the routes for billing period \( t + 1 \) in terms of the period \( t \) routes until all customers are transitioned to their final billing days. We first find all customers that can be transitioned from their billing day at time \( t \) to their final billing day in a single shift that does not violate (7). In terms of billing distance, we find all customers \( i \) such that \( \|d_l(i), f(i)\|_D \leq S \). Letting \( \mathcal{A} \) denote this set of assigned customers and \( \mathcal{U} \) represent the complementary set of unassigned customers, we create routes for the next period by first constructing a set of skeleton routes. We do this by taking each target route and restricting it to only those customers that are in \( \mathcal{A} \). Having constructed these skeleton routes, we take the unassigned
customers in $\mathcal{U}$ and place them in an appropriate intermediate route while trying to ensure that these routes are as efficient and balanced as possible.

For the remaining unassigned customers in $\mathcal{U}$, we assign them to the existing skeleton routes by solving a sequence of mixed integer programs (MIPs). Given customer $i \in \mathcal{U}$ and skeleton route $j$, we let $x_{ij} = 1$ when customer $i$ is inserted into route $j$ and $x_{ij} = 0$ otherwise. We let $c_{ij}$ represent the cheapest cost of feasibly inserting customer $i$ into route $j$. One approach is to try to minimize the sum of all these insertion costs while attempting to maintain feasible and balanced routes. Letting $L_j$ and $C_j$ represent the length and number of customers on skeleton route $j$, we have the following MIP denoted by INT-ROUTES:

**Minimize**

$$\sum_{i \in \mathcal{U}} \sum_{j=1}^{R} c_{ij} x_{ij}$$  \hspace{1cm} (9a)

**s.t.**

$$\sum_{j=1}^{R} x_{ij} = 1 \text{ for } i \in \mathcal{U}$$  \hspace{1cm} (9b)

$$C_j + \sum_{i \in \mathcal{U}} x_{ij} \leq \tilde{C} \text{ for all routes } j$$  \hspace{1cm} (9c)

$$C_j + \sum_{i \in \mathcal{U}} x_{ij} \geq \bar{C} \text{ for all routes } j$$  \hspace{1cm} (9d)

$$L_j + \sum_{i \in \mathcal{U}} c_{ij} x_{ij} \leq \tilde{L} \text{ for all routes } j$$  \hspace{1cm} (9e)

$$L_j + \sum_{i \in \mathcal{U}} c_{ij} x_{ij} \geq \bar{L} \text{ for all routes } j$$  \hspace{1cm} (9f)

$$x_{ij} = 0, \text{ if } \|d_i(i), f(i)\|_D > S$$  \hspace{1cm} (9g)

$$x_{ij} = 0, \text{ if } \|d_i(i), f(i)\|_D \leq \|j, f(i)\|_D$$  \hspace{1cm} (9h)

$$x_{ij} \in \{0, 1\}.$$  \hspace{1cm} (9i)

In (9a), we minimize the sum of the individual insertion costs. Constraint (9b) ensures that each customer in $\mathcal{U}$ is assigned to a route, (9c) and (9d) require that we maintain balance across the routes in terms of the number of customers, and (9e) and (9f) attempt to ensure that each route remains feasible and balanced in terms of total length. Finally, constraint (9g) restricts the choices of customer $i$’s new billing day to only those days that can be reached by a feasible shift. Constraint (9h) ensures that whenever we shift an unassigned customer’s billing day, we are always moving this customer’s billing day closer to the final billing day. We incorporate this constraint into the formulation in an attempt to minimize the number of required transition periods.

However, given skeleton route $j$ with length $L_j$, the value $L_j + \sum_{i} c_{ij} x_{ij}$ in (9e) and (9f) can sometimes overestimate this route’s length following the insertions that we made. For example, if two customers that are very near one another are inserted into a particular route, then the sum of these minimum insertion costs can be nearly double the actual increase in the route’s length. We address this issue in two ways. First, rather than solving INT-ROUTES for all $|\mathcal{U}|$ customers at once, we solve a related formulation several times, each time using only a portion or batch of the customers in $\mathcal{U}$. Second, we initially solve a relaxation of the problem and then gradually tighten the constraint until no feasible solution can be found.

To determine which subset of customers from $\mathcal{U}$ to insert, we define a batch size $B$. We then select $B$ customers at a time from $\mathcal{U}$ by considering their insertion costs. In particular, for each customer in $\mathcal{U}$, there is a particular subset of skeleton routes into which we can insert this customer without violating constraints (9g) and (9h). We compute the cheapest insertion costs for all these possibilities, find the largest insertion cost for each customer, and then sort the list of customers in $\mathcal{U}$ in terms of this largest insertion cost. We then solve a related integer program for the first $B$ customers in this list. Our reasoning is that the customers with the largest insertion costs are potentially the most troublesome for our procedure and we would like to insert them early on while there is still some slack in the skeleton routes. After removing $B$ customers from the set $\mathcal{U}$, we apply intra-route improvements to the individual routes that were affected by the insertion of these particular customers. Then we recalculate the insertion costs for the remaining customers in $\mathcal{U}$ and sort the customers as before in order to select the next set of $B$ customers.

The second part of our strategy in solving the MIP involves the right-hand side of constraint (9e). Because of the potential error involved in the left-hand side of this constraint, we replace the route length maximum $L$ with a larger value $\tilde{L}$. This change leads to the following alternative formulation of INT-ROUTES, denoted by BATCH-ROUTES:

**Minimize**

$$\sum_{i=1}^{B} \sum_{j=1}^{R} c_{ij} x_{ij}$$  \hspace{1cm} (10a)

**s.t.**

$$\sum_{j=1}^{R} x_{ij} = 1 \text{ for } i = 1, 2, \ldots, B$$  \hspace{1cm} (10b)

$$C_j + \sum_{i=1}^{B} x_{ij} \leq \tilde{C}$$  \hspace{1cm} (10c)

$$L_j + \sum_{i=1}^{B} c_{ij} x_{ij} \leq \tilde{L} \text{ for all routes } j$$  \hspace{1cm} (10d)

$$x_{ij} = 0, \text{ if } \|d_i(i), f(i)\|_D > S$$  \hspace{1cm} (10e)

$$x_{ij} = 0, \text{ if } \|d_i(i), f(i)\|_D \leq \|j, f(i)\|_D$$  \hspace{1cm} (10f)

$$x_{ij} \in \{0, 1\}.$$  \hspace{1cm} (10g)

In BATCH-ROUTES, we reduce the number of customers under consideration, remove constraints (9d) and (9f), and relax (9e). We embed this alternative formulation into Algorithm 2. Notice that we begin with some $\bar{L} > \tilde{L}$ for which a feasible solution can be found, and then gradually decrease this value until either no solution can be found, or $L \leq \tilde{L}$.  

Algorithm 2: Generate Intermediate Routes

\textbf{input}: A set of skeleton routes and a set $\mathcal{U}$ of unassigned customers at time $t$

\textbf{output}: A set of intermediate routes for time $t + 1$

\begin{algorithmic}
\State Set $B = 20$, $\alpha = .05$
\While {$|\mathcal{U}| > 0$}
\ForAll {$i \in \mathcal{U}$}
\For {$j = 1..D$}
\State Calculate $c_{ij}$, the minimum cost of inserting customer $i$ into route $j$, for all routes $j$ for which (10e) and (10f) are feasible
\EndFor
\EndFor
\State Sort the list of customers in $\mathcal{U}$ in terms of the largest $c_{ij}$
\State Select the first $\min(B, |\mathcal{U}|)$ customers from the list
\State Set $\tilde{L} = 2 \times \tilde{L}$
\While {\textsc{BATCH-ROUTES} is infeasible}
\State Set $\tilde{L} = \tilde{L} + \alpha \times \tilde{L}$
\EndWhile
\EndWhile
\State Make the insertions determined by the $x_{ij}$ variables from the last feasible solution to \textsc{BATCH-ROUTES}
\State Apply intra-route improvement operations to the resulting routes
\State Update $\mathcal{U}$ by removing the just inserted $\min(B, |\mathcal{U}|)$ customers from the list
\EndFor
\EndWhile
\EndFor
\EndWhile
\EndFor
\EndWhile
\EndAlgorithm

Given the routes and billing period assignments at time $t$, we are able to use Algorithm 2 to generate new routes and billing period assignments for time $t + 1$. Since constraint (10f) requires that a customer is always moving closer to the final billing day, this procedure will terminate after a finite number of iterations. We begin with the original routing and billing day configuration of time $t = 0$ and repeatedly apply Phase 3 of our algorithm in order to generate a sequence of intermediate billing periods and routes before finally arriving at the target configuration.

3.5. Additional Considerations

For a problem that is tightly constrained in terms of maximum length $\tilde{L}$ and maximum number of customers $\tilde{C}$, the intermediate routes of Phase 3 may be infeasible. This is simply due to the fact that these intermediate routes must be stretched in order to include customers which are temporarily visited on the corresponding billing day. Since our procedure guarantees that each customer’s billing day moves closer by at least one day during successive billing periods, the maximum number of intermediate periods is bounded by $\lfloor D/2 \rfloor - 1$. Furthermore, because of the temporary nature of these intermediate routes as well as the promise of efficient balanced routes in the near future, it is reasonable for a utility to use additional meter-reading resources during the intermediate periods, especially since the exact number of required intermediate periods is completely determined upon the completion of Phase 3.

Finally, we note that after the intermediate routes are generated via Phase 3, the utility company can transition to the target routes at its own pace. If the utility is concerned about shifting a customer’s billing day for several successive months, the utility can choose to slow down this process by using the same set of intermediate routes for two or more periods in order to prevent these successive shifts. In practice, utility companies are typically hesitant to shift customers’ billing days several times in succession. Stretching out the transition period is a response to this hesitancy [15].

4. Computational Results

In this section, we conduct computational experiments with our procedure. We coded our algorithm in C/C++ and used the GMPL modeling language in conjunction with CPLEX 10.2 to handle the MIPs of Phase 3. When run on an existing set of routes containing between 3,000 and 4,000 customers, the entire procedure required roughly one hour of computing time on a 2GHz Intel processor. The MIPs themselves presented little difficulty and the construction of the balanced target routes generally required about half of the total computing time.

We had the opportunity to test the performance of our procedure on a real-world data set provided by Routesmart Technologies, Inc. Confidentiality agreements prevent us from describing all of the specifics of the data set, but we are able to offer the following details. The 17,775 customer locations are shown in Figure 1 where the depot is represented by the large circle in the bottom center. These meter locations are serviced by a total of 13 meter readers over the standard 20 day (monthly) billing period. Four of these meter readers service the region on all 20 days of the billing cycle and nine readers visit customers anywhere from only one day per cycle to 10 days per cycle.

We focused on the four full-time meter readers, and ran our procedure on each of the four sets of 20 routes. The four data sets contain 2,894, 3,011, 3,942, and 3,972 customers, accounting for 13,819 of the 17,775 total customer locations. For each meter reader, we were provided with the customer latitude and longitude coordinates along with the existing billing day assignments. Because we treat the problem from a node-routing perspective rather than an arc-routing perspective, we did not use the existing routes that traversed a...
FIG. 1. Original customer set.

street network. Instead, we used the TSPLIB GEO [17] norm to calculate a symmetric distance matrix (in km) by using the customer coordinates. We then used these distances to create daily routes for each individual meter reader by computing a TSP tour traversing all customers on each day of the billing cycle. We used our record-to-record travel algorithm to generate these tours. In Table 3, we summarize important properties of the existing routes. It is clear that the existing routes are very imbalanced in terms of both the number of customers serviced each day as well as the total route length. For example, on one billing day, meter reader 1 visits 22 customers and travels 35 km. On another day of the cycle, this same meter reader visits 359 customers and travels 382 km (in this case, many of these locations are very near one another, possibly in an apartment complex where the meters are in a single location).

Next, we determined appropriate upper and lower bounds on the route length and number of customers per route. The maximum number of customers per route \( \bar{C} \) is based on the existing routes from the data set. Determining the route length maximum \( \bar{L} \) was more difficult. In order to generate solutions containing exactly 20 routes, we set \((\bar{L}, \bar{C}) = (170, 160)\) for meter readers 1 and 2, and \((\bar{L}, \bar{C}) = (225, 205)\) for readers 3 and 4. It is clear from Table 3 that these limits are substantially lower than the current maximum values. The balancing lower bounds \( C \) and \( L \) were set to 0.8 times the upper bounds.

We ran our procedure and generated a set of 20 target routes for each of the four meter readers using a balance parameter \( \beta = 0.99 \). Setting \( \beta = 1 \) causes unusual behavior in the record-to-record travel algorithm due to the distinction between deteriorating and improving moves, so we set \( \beta \) very close to 1 in order to reward balancing moves as much as possible. Our Phase 1 algorithm was able to construct routes that were well-balanced and efficient. We also ran a general record-to-record VRP algorithm on each of the four data sets, disregarding the lower bounds \( L \) and \( C \), in order to gain some measure of how the balancing constraints affect the final solution. The results of these computational experiments are presented in Table 4. Using the guided record-to-record procedure of Phase 1, we were able to find solutions that satisfy the upper and lower bound constraints with only one exception where a single route for meter reader 1 visited 126 customers compared to the lower bound of 128. Interestingly, we found that the balanced routes were very competitive with routes generated without using the lower bounds, and were even more efficient in certain cases (due to the nature of heuristics).

Phase 2 of the procedure provided billing day assignments for each of the target routes by solving the assignment problem (8). When running Phase 3 of the algorithm, we found that we were not always able to meet the balancing constraint (10c), especially for small shift sizes when there are relatively few possibilities for each billing day shift. When we encountered such a situation where BATCH-ROUTES was infeasible, we increased the value of \( \bar{C} \) by the minimum amount in order to produce a feasible solution and continued the procedure. The required increase was relatively modest. Due to the temporary nature of these intermediate routes, a small increase in \( \bar{C} \) would seem tolerable, even if the utility company had to pay for additional overtime costs during some of the intermediate periods.

We ran our algorithm four times on each of the four data sets, using shift sizes \( S = 2, 3, 4, \) and \( 5 \). Each data set contains the customer locations, billing day assignments, and existing routes for one of the four full-time meter readers. The results are summarized in Table 5. The first column gives the instance and the maximum allowed shift size (2, 3, 4, or 5), the second column lists the total length of the original routes and the total length of the final target routes, and the third column

<table>
<thead>
<tr>
<th>Meter reader</th>
<th>Number of total stops</th>
<th>Min,Max</th>
<th>Total length of all routes</th>
<th>Min,Max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Number of customers</td>
<td></td>
<td>Route lengths</td>
</tr>
<tr>
<td>1</td>
<td>2,894</td>
<td>22,359</td>
<td>3,214</td>
<td>35,382</td>
</tr>
<tr>
<td>2</td>
<td>3,011</td>
<td>78,306</td>
<td>3,496</td>
<td>94,324</td>
</tr>
<tr>
<td>3</td>
<td>3,942</td>
<td>69,355</td>
<td>4,522</td>
<td>89,381</td>
</tr>
<tr>
<td>4</td>
<td>3,792</td>
<td>12,342</td>
<td>4,157</td>
<td>31,366</td>
</tr>
</tbody>
</table>
gives the number of required intermediate periods. In column four, we count the number of customers who are not yet in their final route during each intermediate period. Column five lists the total length of all 20 routes during each required intermediate period, and the rightmost two columns list the minimum and maximum route lengths and the minimum and maximum number of customers across all intermediate periods.

As expected, we found that larger shift sizes led to fewer intermediate periods as well as more efficient and balanced intermediate routes. For meter readers 1 and 3, a shift size of $S = 2$ requires seven intermediate periods while meter readers 2 and 4 require five intermediate periods for this shift size. The routes for $S = 2$ are generally more costly and less balanced than the routes for larger shift sizes, and we were unable to meet the constraints regarding the route length and the number of customers, particularly for meter readers 3 and 4. If we allow shifts of size $S = 5$, we require at most three intermediate periods with meter readers 2 and 3 requiring no intermediate periods at all as we are able to transfer all customers from their original billing days to their final billing days in a single shift of fewer than five days.

In terms of total route length, the general trend is an initial increase over the target route length in the first intermediate period with the total route length of the subsequent intermediate periods then gradually decreasing before reaching the final configuration. In general, we were somewhat surprised at the high quality of the intermediate routes in terms of total length as we were typically able to avoid expensive insertions. In terms of percentage increase over the length of the target routes, the worst set of routes is the first set of intermediate routes for meter reader 4 and $S = 2$ with a

<table>
<thead>
<tr>
<th>Meter reader</th>
<th>Total length</th>
<th>Min,Max Route length</th>
<th>Min,Max Number of customers</th>
<th>Total length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Balanced</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3,200</td>
<td>137,170</td>
<td>126,150</td>
<td>3,205</td>
</tr>
<tr>
<td>2</td>
<td>3,342</td>
<td>146,170</td>
<td>135,157</td>
<td>3,342</td>
</tr>
<tr>
<td>3</td>
<td>4,325</td>
<td>187,224</td>
<td>177,200</td>
<td>4,322</td>
</tr>
<tr>
<td>4</td>
<td>4,176</td>
<td>183,225</td>
<td>166,205</td>
<td>4,174</td>
</tr>
</tbody>
</table>

Table 5: The performance of our procedure on four actual data sets.

<table>
<thead>
<tr>
<th>Instance, shift size</th>
<th>Original length, target length</th>
<th># Intermediate periods</th>
<th># Customers not in final route</th>
<th>Total Route length</th>
<th>Min,Max Route length</th>
<th>Min,Max Number of customers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2</td>
<td>384,213,105, 55,37,13,2</td>
<td>7</td>
<td>3231,322,3214, 3214,3206,3203, 3201</td>
<td>108,208</td>
<td>95,194</td>
<td></td>
</tr>
<tr>
<td>1,3</td>
<td>3214,3200</td>
<td>5</td>
<td>3219,3219,3215, 3204,3201</td>
<td>129,179</td>
<td>109,165</td>
<td></td>
</tr>
<tr>
<td>1,4</td>
<td>99,43,10</td>
<td>3</td>
<td>3208,3205,3200</td>
<td>128,174</td>
<td>114,157</td>
<td></td>
</tr>
<tr>
<td>1,5</td>
<td>25,11,3</td>
<td>3</td>
<td>3201,3201,3201</td>
<td>137,174</td>
<td>129,156</td>
<td></td>
</tr>
<tr>
<td>2,2</td>
<td>196,59,48, 31,6</td>
<td>5</td>
<td>3358,3343,3343, 3344,3343</td>
<td>124,198</td>
<td>107,176</td>
<td></td>
</tr>
<tr>
<td>2,3</td>
<td>59,34,5</td>
<td>3</td>
<td>3347,3343,3343</td>
<td>124,185</td>
<td>107,168</td>
<td></td>
</tr>
<tr>
<td>2,4</td>
<td>48</td>
<td>1</td>
<td>3343</td>
<td>124,187</td>
<td>107,168</td>
<td></td>
</tr>
<tr>
<td>2,5</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>3,2</td>
<td>468,326,194, 53,50,18,2</td>
<td>7</td>
<td>4408,4372,4397, 4369,4363,4347, 4325</td>
<td>114,299</td>
<td>92,263</td>
<td></td>
</tr>
<tr>
<td>3,3</td>
<td>280,95,43, 23,6</td>
<td>5</td>
<td>4367,4363,4382, 4337,4326</td>
<td>140,242</td>
<td>122,215</td>
<td></td>
</tr>
<tr>
<td>3,4</td>
<td>63,19</td>
<td>2</td>
<td>4340,4342</td>
<td>187,231</td>
<td>171,205</td>
<td></td>
</tr>
<tr>
<td>3,5</td>
<td>—</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>4,2</td>
<td>641,412,317, 150,37</td>
<td>5</td>
<td>4264,4218,4258, 4233,4198</td>
<td>137,294</td>
<td>118,261</td>
<td></td>
</tr>
<tr>
<td>4,3</td>
<td>403,160, 86,28,4</td>
<td>5</td>
<td>4239,4210,4204, 4196,4181</td>
<td>138,254</td>
<td>118,226</td>
<td></td>
</tr>
<tr>
<td>4,4</td>
<td>192,65,34,3</td>
<td>4</td>
<td>4208,4198, 4217,4189</td>
<td>183,241</td>
<td>168,215</td>
<td></td>
</tr>
<tr>
<td>4,5</td>
<td>101,48</td>
<td>2</td>
<td>4185,4195</td>
<td>183,245</td>
<td>166,215</td>
<td></td>
</tr>
</tbody>
</table>
FIG. 2. Original, intermediate, and target routes for a single meter reader. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]
total length of 4264 km compared to the target routes with a length of 4176 km. While this increase is very modest at just over 2%, it is important to note that these efficient intermediate routes do come at the expense of occasionally violating the maximum route length \( L \) and maximum number of customers per route \( C \), as seen in the rightmost two columns of Table 5. However, when compared to the existing extremes found in the original routes given in Table 3, these temporary violations would seem to be quite tolerable.

In Figure 2, we present a sequence of plots (produced by http://www.gpsvisualizer.com) derived from the data set for meter reader 4 with shift size \( S = 4 \). In these plots, the route for each billing day is represented by a different color and we have removed the edges radiating from the depot for clarity. In this example, the TSP tours that we computed for each of the existing billing days provide an initial solution with a total length of 4157 km. The intermediate routes are quite efficient, increasing the total route length by less than 0.3% over both the initial and target configurations. A visual inspection of the routes indicates that our procedure creates these efficient intermediate routes by moving clusters of customers from one route to the next, incurring minimal insertion costs. In other words, groups of neighboring customers are generally assigned the same final billing day and our procedure creates intermediate routes that preserve these clusters of customers.

5. CONCLUSION

The Balanced Billing Cycle Vehicle Routing Problem is a new problem in the VRP literature. In the BBCVRP, we start with an initial billing day configuration that can be extremely poor. We are not allowed to start from scratch as is assumed in the typical vehicle routing problem. We developed a relatively simple heuristic method that produces solutions to the standard VRP with routes that are balanced in both the number of customers and length. Our procedure combines heuristic and exact methods to generate solutions to a new variant of the standard VRP that has unusual complicating constraints.

We tested the performance of our algorithm on a utility company’s data set. Our algorithm produced efficient and balanced target routes along with a set of intermediate routes. In general, our procedure performed very well, allowing the utility to move from its existing, imbalanced initial configuration to a more efficient and more balanced configuration in a small number of steps. Furthermore, we found that the required intermediate routes remained quite balanced with a relatively small increase in cost.

In future work, we hope to investigate a modification of our approach that relaxes our assumption of having only one meter reader per day as this would possibly allow for a more efficient configuration when all the meter readers are taken into account. This modified approach may allow us to achieve a better balance of the workload across the different meter readers as well as across the different days of the billing cycle.

Finally, we note that some of our techniques may be useful for solving variants of the standard VRP. Our procedure for constructing balanced routes could be applied to any VRP where balancing is an important issue. In practice, balancing is rarely unimportant. Furthermore, there are real-world problems such as commercial sanitation collection which exhibit the features of the well-known Period Vehicle Routing Problem (PVRP) [6]. In such PVRP instances, some customers are visited once a week while others require more frequent service. In the literature, the PVRP is always solved from scratch. In practice, however, a current, inefficient set of daily routes may exist and the challenge is to improve these routes while minimally disrupting the current solution. The techniques presented in this article can be applied to such a scenario.

APPENDIX

In this appendix, we describe the record-to-record travel algorithm that we use to improve VRP solutions [16]. Before running the algorithm, we construct a sorted list of the 50 nearest neighbors for each node in the problem. When searching for potential solution modifications involving some node \( k \), we begin with \( k \)’s nearest neighbor and move through this list searching for an acceptable move. We accept only feasible moves and apply three well-known improvement operators: One-point move, Two-point move, and Two-opt (see Fig. 3).

- **One-point move**: Given a node \( j \) that is in \( k \)’s neighbor list, remove node \( k \) from its current position in the solution and try to insert it either before or after \( j \).
- **Two-point move**: For a node \( j \) in \( k \)’s neighbor list, try to exchange the positions of \( j \) and \( k \) in the current solution.
- **Two-opt move**: Select a pair of edges from the current solution, one connected to node \( k \), and the other connected to a node \( j \) that is in the neighbor list. Remove this pair of edges and reconnect the solution so that a feasible solution is preserved.

These three moves are used in Algorithm 3 that takes a feasible solution and tries to improve it using the record-to-record travel approach.

![Figure 3](image-url)

**FIG. 3.** The three improvement operators used in the record-to-record algorithm. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]
Algorithm 3: Record-to-Record Travel

input: An existing feasible solution to an N-node VRP
output: An improved feasible solution to the VRP

Set $S = 30$, $T = 5$, $t = 0$, set $Record$ equal to the current objective function value, and $Deviation = 0.01 \times Record$

Set $Operators = \{One-Point Move, Two-Point Move, Two-Opt Move\}$

Start:
for $i = 1$ to $S$ do
  for $j = 1$ to $3$ do
    for $k = 1$ to $N$ do
      Apply $Operators [j]$ and search node number $k$'s neighbor list for moves that obey the length and capacity constraints, accepting the first improving move if one is found. If no improving moves are found, then accept the best deteriorating move if the resulting objective function is less than $Record + Deviation$.
    end
  end
  if The current solution is a new record then
    Update $Record$ and $Deviation$
  end
end
while Improving moves can be found do
  for $j = 1$ to $3$ do
    for $k = 1$ to $N$ do
      Apply $Operators [j]$ and search node number $k$'s neighbor list for improving moves that obey the length and capacity constraints, accepting the first improving move that is found.
    end
  end
  if The current solution is a new record then
    Update $Record$ and $Deviation$
  Set $t = 0$
end
$t = t + 1$
if $t = T$ then
  Return the best solution found
else
  Return to Start
end

Acknowledgments

The authors thank Dr. Larry Levy of Routesmart Technologies, Inc. for describing this problem to us, for valuable conversations regarding our solution approach, and for supplying actual data sets for our computational experiments. They also thank the anonymous reviewers for their helpful suggestions.

REFERENCES