OPTIMIZING THE REGIONS OF PEDAL PEOPLE USING INTEGER PROGRAMMING

MATTHEW HRONES, ALEX MAY, PHILIP MENEGHINI, HANNA MEYER

Abstract. This paper studies the effect of regions on optimal run times for the Pedal People. Using two linear programs, this paper compares results from a system using current routes and regions provided from the Pedal People and results from a system without routes and regions.

Introduction

In this project, we were tasked with considering the regions the Pedal People should use to assign trash days to their customers. Some of the big questions we needed to address were: would different regions, the same regions, or no regions be better? How do we determine if a region is optimal? How are regions affected by adding new customers?

The current business model that Pedal People Coop uses is multilayered. Customer houses are grouped into runs, runs are clustered into routes, and routes into regions.

Each run consists of a certain number of houses dependent on the average amount of trash, recycling, and compost these houses produce. Runs begin and end at one of two trash collection centers. The capacity of the trailer the Pedal People use is a deciding factor in how many houses can be included in a route. In our model, a capacity of 216 gallons for the trailer was used. Based on the way the Pedal People collect waste, only the amount of recycling and compost was taken into account in this measurement; trash bags are simply piled and secured on top of the trailer. When the bike is at approximately 90% capacity to account for any extra waste or new customers, the employee returns to the collection center to empty the trailer and starts on the next run.

Runs are clustered into routes that are traveled by a single employee in a day. Finally, routes are grouped into regions. These regions determine a customer's trash pick up day. The end goal of our project was to determine optimal regions (if possible) or to definitively say that regions are not effective and should not be used.

Similar vehicle routing problems have been researched before. The Capacitated Vehicle Routing Problem studied how best to service customers with specific product demands. In this problem, a fleet of delivery vehicles had a certain capacity and departed from a singular common warehouse. The goal was to deliver all products to customers using these delivery vehicles while minimizing cost [RKPT03]. This is similar to the Traveling Salesman Problem and the Bin Packing Problem. There are significant differences between this problem and our project, however. Unlike in the Capacitated Vehicle Routing Problem, we are optimizing runs and regions with 2 collection points. This added in another dimension to our linear program; we had to determine which trash collection point at which to start and end each route. Additionally, our project deals with collections over a period of time (a week) and we must determine which routes take place on which days as well rather than simply determine the optimal order in which to visit houses.

Date: December 9, 2020.

A problem that has more similarities to our project is the Clustered Vehicle Routing Problem (CluVRP) [BEV14]. The CluVRP divides customers into clusters, similar to the runs in our project for the Pedal People. Clusters are sets or groups of vertices (in this case, customer houses) in number greater than or equal to one. The customers are given as data points on an undirected graph as vertices V with edges to other vertices E. There is a value, c_{ij} that gives the cost of moving along an edge from one vertex to another. In other words, this is the cost of moving along edge x_{ij} from customer i to customer j. In our case, the cost c_{ij} is time, or the time it takes for a biker to travel from customer i to customer j. The variable x_{ij} is a binary variable representing whether or not the path from i to j has been traveled. $x_{ij} = 1$ if the path has been traveled and $x_{ij} = 0$ if it has not. A vehicle visiting a vertex in cluster C must visit all other vertices in C before moving on to another cluster. The aim is to create m number of vehicle routes while minimizing total cost, visiting each customer once exactly, staying within a cluster until all vertices within the cluster are visited, and staying within the vehicle capacity. Rather than clusters, our project deals with runs; each house in a run must be visited before moving on to another run in the route.

Again, a key dissimilarity between our two problems was that we had two different collection points rather than one, and in addition our runs and routes had to be further grouped into optimal regions.

Our approach to this project involved two programs. Our first linear program created optimal runs from the routes currently used by Pedal People. From these optimal runs, we could easily determine the amount of time it would take an employee traveling 10 mph to complete an entire route. Then, we used a linear program to divide routes into regions based on the number of hours employees can work on that day. We decided that it would make sense for there to be 5 regions, one for each weekday. Additionally, if there were more available working hours on a particular day, we assigned more routes that day.

We then used our first linear program to create optimal runs from the set of all customer houses the Pedal People service. If our linear program were to find that the total time it would take to completely service all houses with optimal runs is far less than the time it would take employees to stop at every house using their current system, we could conclude that the use of these regions is not optimal. If runs are geographically close enough in a single route and trailers operate within 90% capacity, regions would be unnecessary. Any new customers would be comfortably fit into the closest route rather than the closest region. If, however, our model showed that the current system and optimal runs that we found yield a similar run time, then we could conclude that the current model of using regions is in fact more or less optimal.

Model: First Linear Program

The first task in our project was to determine optimal runs from houses in a given route. Each run started at a trash collection center and would collect trash from houses in a route until the bike trailer was filled to 90% capacity; capacity was one of our constraints. The objective function for this linear program minimized time and used volume of recycling and compost as a another constraint.

Initially, we reasoned that an optimal run would be the least "costly" for the employees. We considered a work function based on distance and elevation/incline. In other words, a run would minimize the amount of distance covered by a cyclist as well as minimize the uphill cycling done. However, it would be difficult to decide a "maximum amount of work" that an employee can exert in a single day. Thus, we came to the conclusion that we should minimize the total amount of time it takes to collect the trash from every house in a route.

Our linear programs used both values and variables, described below.

Values: First Linear Program

- The volume capacity of the bike's trailer, *cap*, is 216 gallons. This approximately 90% of the bike's total capacity for compost and paper and plastic recycling. Any trash bags thrown on top are assumed to fit no matter how much waste they contain.
- The distance from house i to house j is denoted d_{ij}. We created a large matrix of the distances between all customer houses in miles.
- Elevation values are similar to distance values. The elevation between house i and house j is given as e_{ij}. We also created a large matrix of elevations between each house that Pedal People services.
- r_h is a value needed specifically for the route optimization program. This gives all houses in a route h.
- The time it takes for an employee traveling an average of 10 miles per hour between house i and house j is denoted as t_{ij}. We generated a matrix of time values based on our distance matrix.
- The two collection centers are represented as values c. This set is of size 2 because there are only two transit stations. c represents which indices d_{ij}, t_{ij} and e_{ij} correspond to the collection centers.
- The set of values denoting the average volume of waste (recycling and compost) of each house is $trash_i$.
- Because we want to be aware of incline, we have a set of values $incline_k$. This is a set of integer values taken from houses i and j which determines which paths between pairs of houses have inclines greater than 3 degrees. We took these values of e_{ij} from the elevation matrix.

Variables: First Linear Program

- Variable x_{ij} is a binary variable representing whether or not the edge from vertex (house) i to vertex j has been traveled. If $x_{ij} = 0$, the path between houses i and j has not been traveled. Conversely, if $x_{ij} = 1$, the path between houses i and j has been traveled by an employee.
- The array of integer variables equaling the volume of a bike's trailer at any given house w in its theoretical optimal run is denoted as u_w.
- r is an integer variable that equals the number of optimal runs.

Objective Function: First Linear Program

Now that all of our variables and values are thoroughly described, we can move on to defining the objective function of the first program.

The objective function we agreed upon was a minimization function based on time and the variable x_{ij}, whether or not a path between houses i and j has been traveled:

$$\text{minimize} \sum_{\forall i} \sum_{\forall j} t_{ij} x_{ij}$$

In other words, we minimized the amount of time it would take an employee to pick up all of the waste from a route.

Constraints: First Linear Program

In order to get reasonable solutions, our first linear program required a distinct set of constraints.

Our first constraint enforced that every house has an incoming path as well as an outgoing path (in other words, each house has a degree of 2). The first constraint checks that there is an edge leaving each house and the second constraint insures that there is an edge entering each house.

$$\sum_{\forall i} \left(\sum_{\forall j \backslash c} x_{ij} = 1 \right)$$
$$\sum_{\forall i \backslash c} \left(\sum_{\forall j} x_{ji} = 1 \right)$$

Our second constraint enforced a similar concept as the first, except this time with collection centers. Both collection centers should collectively have r incoming edges and r outgoing edges where r is the variable defining the total number of runs. The first equation establishes this for entering edges and the second is for leaving edges.

$$\sum_{\forall i \in c} \sum_{\forall j \backslash c} x_{ij} = r$$
$$\sum_{\forall i \backslash c} \sum_{\forall j \in c} x_{ij} = r$$

The third constraint imposed the bike trailer volume capacity with the u_w variable, the cap value of 216, and the $trash_i$ value. This constraint guaranteed that each house in a run would not overburden the bike with too much waste before going back to a collection center.

$$\sum_{\forall i \backslash c} \sum_{\forall j \backslash c} u_j - u_i + cap(1 - x_{ij}) \geq trash_j$$

The fourth constraint was meant to ensure that a bike would not experience too great of a change in elevation. It enforced the idea that a bike would not go up a hill with greater than a 3 degree incline with a full trailer. In the following constraint, j is the second integer value in the two values represented by k.

$$\sum_{\forall k} u_j x_k \leq trash[j]$$

Unfortunately, this constraint ended up yielding too many infeasible solutions. In the end, our results were unaffected since we were strictly comparing two sets of outputs. One output was the amount of time it would take to collect trash from all houses using given routes and regions and the other was the total amount of time it would take to service all houses without these structures.

The fifth constraint was undiscovered until we began coding the program. We needed to enforce that every x_{ii} value is zero. Without this constraint, the program would just use $x_i i = 1$ to satisfy all of the constraints and create a false optimal solution.

$$\sum_{\forall i} x_{ii} = 0$$

Our last constraints are used to make sure that all of the variables stay within the correct range. We defined x_{ij} as binary and our r variable needed to be above 0 (there cannot be zero runs). Finally, the u_w variable needed to stay in a range between the volume of trash at that house and the bike volume capacity.

$$r \geq 1$$
$$x_{ij} \in [0, 1]$$
$$\forall i \; trash_i \leq u_i \leq cap$$

This linear program provided us a set of minimum times of each run in a route and we thus found the minimum amount of time that a route would take. In addition, it found optimal runs for all houses that pedal people serviced (not just a set of houses in a route). This program was the bulk of our project since it served as a very easy way to compare times with regions and routes as opposed to times without.

Model: Second Linear Program

Our second linear program aimed to split up routes as proportionally as possible into a certain number of regions (pick up days). Every weekday has a different number of employees and working hours so some days will have bigger regions or contain routes that take more time because there are more people working on that day. The input to this linear program was a set of route times (which we computed optimally with the integer program above) and the output was a set of routes in given regions.

Values: Second Linear Program

We used all values from the first integer program as well as these three new values.

- R_k is an array of values that shows the optimal amount of time that each route k will take to finish.
- $region_i$ is an array of values that has the times we are aiming to obtain for each region i. This This is calculated by taking the R_k values, summing them and then multiplying by the proportion of workers we have available on each weekday.
- Lastly, *routes* is a value of how many routes we have.

Variables: Second Linear Program

- reg_{ij} is a binary matrix of variables. If $reg_{ij} = 1$, then route j is in region i.
- x_i is an array of abstract variables we used to eliminate the absolute value from our objective function. This variable will become more clear when we lay out the entire program.
- y_i is an array of abstract variables similar to x_i. When paired together, x_i and y_i eliminate the nonlinear aspects of the program. We denote the subscript as i since i represents a region and there is an x and y variable for each region.

Objective Function: Second Linear Program

$$\text{minimize} \sum_{\forall i} x_i + y_i$$

This is essentially minimizing $|reg_i - region_i|$. In other words, this linear program minimizes the difference between the most optimal division of employee hours on a given day and the actual amount of time it takes employees to travel routes on a certain day.

Constraints: Second Linear Program

Our first constraint enforces the idea that every route belongs to one or more regions. No route should be without a region.

$$\sum_{\forall i} \sum_{\forall j} reg_{ij} = routes$$

The second constraint is simply used to make sure that we do not include a route in multiple regions so we sum all entries in reg_{ij} for a given route in all regions and it needs to be equal to exactly one.

$$\sum_{\forall j} \left(\sum_{\forall i} reg_{ij} = 1 \right)$$

The third constraint ensures that the x_i and y_i variables have the correct assignments. $x_i + y_i$ should equal $|reg_i - region_i|$ and $x_i - y_i$ should equal $reg_i - region_i$ to make this work.

$$\sum_{\forall i} x_i - y_i + \left(\sum_{\forall j} reg_{ij} R_j \right) - region_i = 0$$

The last constraints for this linear program are simply in place to ensure that abstract variables x_i and y_i are non-negative and that reg_{ij} is binary.

$$x_i \geq 0,\ y_i \geq 0$$
$$reg_{ij} \in [0, 1]$$

Computational Results and Analysis

The computations of our models yielded interesting results. First, we computed optimal run times for each route that the Pedal People currently uses. These routes were projected to take no longer than 4.027 hours, and the total time to finish all routes was approximately 79.58 hours. Below is a table of all optimal route times sorted by Route ID.

Route ID	Time with Optimal Runs (Hrs)		Route ID	Time with Optimal Runs (Hrs)
4	1.827857091		42	3.293531145
7	2.783813575		45	0.038995137
8	3.131002491		53	2.397299879
11	4.027384127		75	2.183256538
12	1.702180915		178	1.600558705
13	2.083592462		204	2.026314097
14	1.431317145		207	2.067451288
15	1.171654785		217	1.503471511
18	2.32725488		239	1.833294072
20	2.14164036		314	1.625084962
23	1.173317397		333	1.406686729
24	1.926553491		336	1.805888863
25	2.206544031		377	0.447701102
26	1.657217694		421	2.470390501
27	2.692725826		428	1.642679144
28	2.834142889		539	1.54545588
29	1.653812532		553	2.340758227
33	2.060853432		560	1.030947664
34	2.888893606		561	1.268744278
35	1.733607773		575	0.956471013
37	2.64168076			

Figure 1. Routes, sorted by ID and their respective optimal run times.

While most of these route times make sense, we did encounter some issues in our distance/time matrix that yielded some strange results. One of these odd results is prominent in the optimal time for route 45. This route supposedly takes 0.038995 hours which is not reasonable. With that being said, our results for this first linear program were relatively stable. If we could find the bug in our distance matrix that yielded this small result, our results would of course be more accurate.

The second linear program parsed up routes into regions based on projected employee hours on that particular day, shown in the table below. Unfortunately, the error between these regions is 28.63 hours when we tried to create most proportional regions. There is a large gap between times of the regions we constructed and those we ideally wanted. The table showing how our program parsed routes into routes is shown below.

Region	Monday	Tuesday	Wednesday	Thursday	Friday
Routes in Region	11	7	4	27	12
	13	8	18	28	15
	24	14	20	53	25
	26	23	34	314	178
	29	33	37	575	207
	333	35	42		539
	377	45	75		560
		204	217		
		239	336		
		421	428		
		553			
		561			
Total hours in Region	13.203	22.292	22.256	10.506	11.325

FIGURE 2. The results of our code that split routes into regions.

It is fairly obvious that this second linear program could be more robust in order to yield more reliable results. The error between regions is large and the program would need to be altered to provide consistently accurate outputs.

The most interesting result was found when we ran our first linear program over the entire set of Pedal People customer houses. After optimizing runs for all of the houses Pedal People service, we found that the total time it would take to collect trash from all houses would be 41.648 hours. We acknowledge that this result was found by manually interrupting the solving process after our code ran for an hour on Gurobi. Thus, we know the integer program results were not fully optimal. Still, the resulting total run time is significantly lower (by 47.6%) than the total run time found by using routes given to us by the Pedal People. We can confidently conclude that the current system of using these given routes and regions is not optimal.

While our linear programs did produce significant results, there are certainly ways that our process could be improved. For example, we could produce optimal routes from the optimal runs we found when using data collected from the entire set of houses serviced by the Pedal People. These optimal routes would be fed into our second linear program in order to discover the best way to split up routes by days of the week based on the distribution of employee working hours. All told, the questions we ended up answering were fairly ambitious. We were able to successfully compare the total run times for the current system and a new system without the restriction of given routes and regions. Though the program processing the optimal runs for the entire set of customer houses had to be terminated, we still obtained a value for our objective function that showed there is indeed a better way to go about parsing up houses into regions.

We can draw interesting comparisons between the way the new regions look geographically

vs. the old regions by using k-means. Below is a graphic showing each house as a single dot in the set of all customer houses. There are 33 regions based on geographic location but due to some coding restrictions, we could not get 33 separate colors for each region.

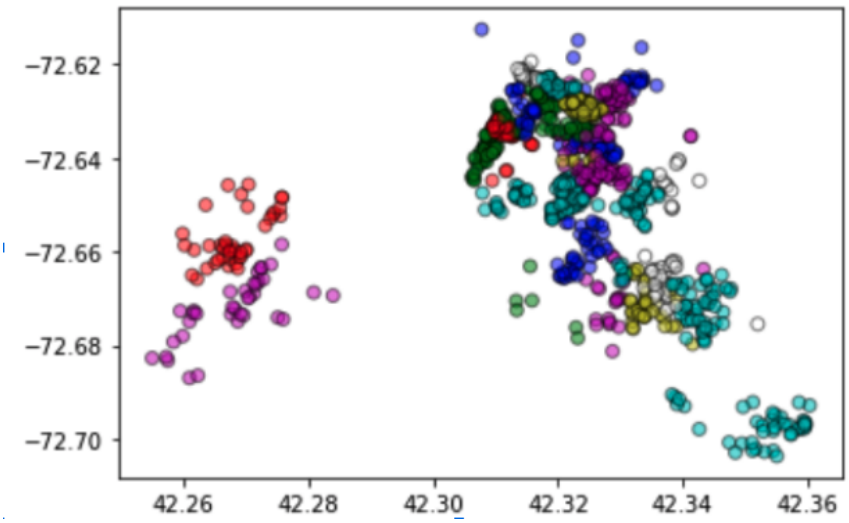

FIGURE 3. Old regions.

The new regions are shown below. Each dot represents the average location of a run. As depicted, there are 5 distinct regions, one for each day of the week. Unlike the first graphic, regions were not determined by geographic location but by the way our code divided up runs in regards to employee hours on a particular day.

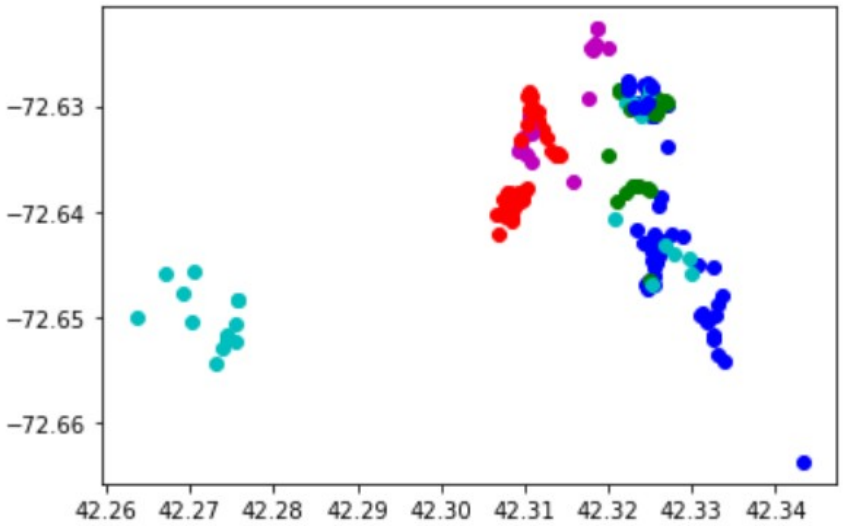

FIGURE 4. Runs in new regions.

Conclusion

Given the results yielded by our linear programs, our recommendations for the Pedal People are relatively straightforward. Clustering houses into 33 small regions is incredibly sub-optimal. While we understand the rationale for houses to be grouped by regions (such as "State Street" or "Fort Hill") from a human standpoint, if we abolish the idea of clustering houses into regions we can more optimally create runs and routes. It may be easier to understand groups of houses based on location, but creating regions undoubtedly restricts the optimality of the total run time. Instead, we suggest redoing runs altogether and grouping the new, more streamlined routes based on how many employees, and how long they work, on any given day. Any new customers can be added to the nearest route. Because we recommend keeping the trailer at or below 90% capacity, any extra trash or new customers should comfortably fit in existing routes. Any set of routes can formed into regions be used in the cluster routes program. This includes potentially optimal routes that were found by another group, routes found by our route-optimization program, or the old routes defined by the Pedal People. Regardless, our proposal is to cluster the routes as our second linear program to determine trash days and again discard the old regions.

References

[BEV14] Maria Battarra, Günes Erdogan, and Daniele Vigo. Exact algorithms for the capacitated vehicle routing problem. *Operations Research*, 62(1), 2014.

[RKPT03] T.K. Ralphs, L. Kopman, W.R. Pulleyblank, and L.E. Trotter. On the capacitated vehicle routing problem. *Mathematical Programming*, 94, 2003.

Department of Mathematics and Statistics, University of Massachusetts, Amherst