Ant Colony Optimization Applied to the Bike Sharing Problem

Cashous W. Bortner[1], Can Gürkan[2], and Brian Kell[3]

[1]Department of Mathematics, University of Nebraska-Lincoln, Lincoln, NE,
cashous.bortner@huskers.unl.edu

[2]Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY,
gurkac@rpi.edu

[3]Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA,
bkell@cmu.edu

Abstract

In this study, we analyze a single vehicle capacitated pickup and delivery problem, namely the bike sharing problem as seen in bike sharing systems around the world. We investigate previous works and formulate our own, novel algorithm for solving the bike sharing problem which is based on an ant colony optimization heuristic. Our algorithm takes into account the total distance traveled, and the distribution of the bikes within the system. We then test our algorithm on random data samples as well as real world data in order to compare our algorithm to other formulations.

1 Introduction

Bike sharing systems have gained wide-spread popularity as cities around the world continue to implement them in an effort to influence citizens to use bikes instead of motorized vehicles to lower green-house gas emissions, induce a more healthy lifestyle, and decrease the traffic flow. These systems allow one to rent a bike from an automated bike-rack, or station, to be ridden to a destination where there is another bike-rack, for returning the bike.

The main factor that determines the success of the system is the demand and how well it is met throughout the daily operation of the system. The demand of the bikes is twofold; there must be enough bikes at every station so that users can take a bike, but there also need to be enough open slots for users that have already rented bikes to return their bikes. To meet this demand, a fleet of vehicles specially designed to efficiently pick up and drop off bikes is sent out from the depot, or the place at which the vehicles are

stationed, to pick up and deliver bikes as they see fit, so that the amount of times a potential customer is turned away from a station, either because there are no bikes to pick up, or because there is no room to return a bike, is reduced to a minimum. The problem of finding an optimal path of the vehicle tasked with repositioning the bikes is a relatively new and interesting question which has recently been investigated in the area of operations research.

This repositioning problem has been studied both in its initial form as a bike sharing problem (BSP), where the goal is to find a route that best meets the demand of the system, and a variation of it in which more constraints are added to focus the problem. What makes this problem so difficult, as well as interesting, is the combination of a routing problem in determining the best path for the repositioning vehicle to minimize distance, as well as an inventory problem in which the vehicle must also attempt to meet the demand as well as possible. Many different approaches have been taken to attain exact solutions using some variation of mixed integer programming; however, these methods proved to be inefficient [3, 5, 6, 8, 12]. Since finding exact solutions is inefficient, many heuristics (rules of thumb which yield good approximation) have been applied to solve the BSP [3, 7, 10, 11, 13, 15]. There has also been an attempt to combine heuristics with exact solution methods to find better solutions more efficiently [3, 7, 13]. For example, Di Gaspero et al. used ant colony optimization and constraint programming to solve the BSP [7]. This was one of the main motivations behind our study.

Ant colony optimization (ACO) was originally introduced in the early 1990's inspired by the actual behavior of ants, particularly, the way the ants gather food and return to their nest with utmost efficiency. The ACO algorithm was first used in an attempt to heuristically solve the traveling salesman problem (TSP) as well as scheduling problems [2]. ACO is an agent based algorithm in which agents (ants) move from node to node based on a probability function. Once they reach the end of the path, the ants travel the same path back home, laying a trail of pheromones on the path, that will increase the probability of using that path in future iterations. The pheromone level depends upon the total distance of the ant's path; for example, a shorter path would yield a higher pheromone level compared to a longer path. This would make the shorter path more appealing to ants in the following iterations.

Our novel ant colony optimization formulation uses the learning capabilities of ACO not only on the path the vehicle travels, but also on the number of bikes the vehicle should transport to solve the BSP. Each ant can be thought of as a redistribution vehicle travelling to each node, picking up or dropping off bikes. We use this method to solve a specific variation of the BSP.

2 Problem Description

The problem considered in this study can be formalized as follows. Let $G = (V, E)$ denote a complete directed graph where $V = \{0, 1, \ldots, n\}$ is the set of vertices representing the stations with vertex 0 representing the depot. Each vertex $i \in V$ has an initial state $s_i \in \mathbb{Z}^+$ and a target state $r_i \in \mathbb{Z}^+$ representing the initial and target number of bikes respectively. The final number of bikes in a station, that is, the number of bikes after an iteration is completed, is denoted by q_i. Additionally each vertex has a capacity $C_i \in \mathbb{Z}^+$ which represents the size of the station. Note that the number of bikes at a station i cannot exceed C_i. Let $d_i = r_i - s_i$ be the demand of each station. If $d_i < 0$ then d_i bikes should be removed from the station i, whereas if $d_i > 0$ then d_i bikes should be supplied to station i. If $d_i = 0$ no further action is required for that station. A traveling cost c_{ij} is associated with every arc $(i, j) \in E$. There is a single vehicle with capacity Q that traverses the graph and transports the bikes.

The goal of the problem is to minimize a function

$$f : \mathcal{A} \to \mathbb{R}$$

where $\mathcal{A}$ is a set of bijections of the form

$$g : \{1, \ldots, n'\} \to \{i \in V : d_i \neq 0\}$$

with n' denoting the number of stations visited. In other words $\mathcal{A}$ is a set that consists of sequences of vertices such that it starts and ends with the vertex 0 and every other vertex has a non-zero demand.

For a given sequence $A \in \mathcal{A}$,

$$R = \sum_{v \in A} |q_v - r_v|$$

also referred to as the residual, is the total difference between the current number of bikes and the target number of bikes at each station, and

$$L = \sum_{u,v \in A} c_{u,v}$$

is the total traveling cost, where u and v are consecutive elements in A.

The objective function considered in this paper is

$$f = aR^{\bar{a}} + bL^{\bar{b}}$$

with the parameters a, $\bar{a}$, and b, $\bar{b}$ that determine the relative importance of R and L respectively.

3 Previous Work

There are similar problems discussed in the literature, many of which are analyzed in the context of bike sharing problems. Most of these studies have used exact solution methods, for example, integer programming [3,5,6, 8,12] as well as several heuristic approaches, namely variable neighborhood search (VNS) [11], local searches including tabu search [3], and genetic algorithms [15].

Most of the BSP variants can be classified as static or dynamic. Solving the static case simply means that the system is at rest when repositioning occurs, for example during the night. The dynamic case is about repositioning the bikes while they are being used, for example during the rush hours [1].

Some of the previous work on the static case of the BSP are as follows.

One of the initial studies on the bike sharing problem was conducted by Benchimol et al. The authors considered a variation of the C-delivery traveling salesman problem, where one truck with capacity C visited every node exactly once and balanced the number of bikes while minimizing the distance traveled. They showed that the BSP is NP-hard even in simplified cases [1]. This means that there does not exist an efficient algorithm for solving the problem exactly,unless P=NP, which leads to the use of heuristics— methods that can only find an approximate answer but which are also very efficient. This was one of the reasons that a novel heuristic was considered to solve the BSP in this paper.

In 2004, Hernández-Pérez and Salazar-González attempted to use 0-1 integer programming coupled with a branch and cut algorithm to solve the pickup and delivery traveling salesman problem. They also formulated an accepted convention of producing random data to test the pickup and delivery traveling salesman problem [8]. We implemented these conventions in this paper as well.

Chemla et al. worked on a more realistic variation of the problem in which a vertex can be visited multiple times and vertices with the target number of bikes are not necessarily visited but they can be used to temporarily store bikes. The authors presented an exact model of the problem but they also proposed a relaxation of it as the exact formulation is intractable. They used branch and cut algorithms to solve the relaxation integer program and utilized a tabu search heuristic to solve the problem more efficiently by finding the upper bounds [3].

Raviv et al. formulated multiple mixed integer linear programs (MILP) in an attempt to solve the static case of the BSP. They were successful in creating two reasonable MILP's for the BSP, although they struggled to solve larger instances efficiently [12].

In a recent paper, Dell'Amico et al. used a mixed integer linear program aided by the branch and cut algorithm to solve the BSP. The authors also used a set of benchmark instances of real world data to test their algorithm [5]. These instances were used to test the algorithms introduced in this paper as well.

Erdoğan et al. used a combination of a separation algorithm as well as a construction heuristic and integer programming to solve a variant of the BSP in which they require the system to be at equilibrium, meaning every demand is filled, at the end of the iteration [6]. We were able to compare our results to theirs, since they tested their algorithm on real world data collected by Dell'Amico.

Contardo et al. considered the dynamic variant of the BSP in which the system must be balanced while considering the constant change within the system [4].

Schuijbroek et al. worked on a dynamic variant of the BSP as well. They unified the inventory balancing problem and the routing problem for the first time and modeled the stochastic demand by viewing the inventory at each station as a queuing system. In other words, they presented a novel formulation of the BSP. They used clustered routing heuristics and constraint programming (CP) to solve the problem by clustering the vertices so that each cluster is balanced by a single vehicle. In doing so they introduced one of the first constraint programming formulations of the bike sharing problem. They found that the clustered routing heuristic performs better than CP for instances with a large vehicle fleet and a low number of stations per vehicle, where CP is better for lower number of vehicles and longer routes [13].

In 2014 Hernández-Pérez and Salazar-González consider a more generalized version of the capacitated pickup and delivery traveling salesman problem (CPDTSP) where there are m different types of commodities instead of just one. In other words the multi-commodity pickup-and-delivery traveling salesman problem (m-PDTSP) is the problem of finding a route for a single vehicle such that it picks up and delivers all the quantities of the m different types of products satisfying the vehicle-capacity limitation and minimizing the total travel cost. They explicitly suggest that this problem can be used to model bike sharing systems with two different kinds of bikes [9]. We believe that our ACO formulation is very well suited to accommodate this variation.

Stützle and Hoos created a variant of the ACO in an attempt to create an algorithm that converges faster than the normal ACO, and also tends to have a lower probability to converge prematurely on a route that is less than optimal for the traveling salesman problem. This algorithm only allows the "ant" with the best path in a certain iteration to lay pheromones, rather than every ant laying pheromones [14].

In a recent paper, Di Gaspero et al. used a hybridization of the ant colony optimization method and constraint programming (ACO+CP) with the strategy of using the learning ability designed within the ant colony optimization algorithm in conjunction with the solving power of constraint programming. This strategy proved to be effective in solving the BSP as they were far more efficient than solving with a traditional constraint programming method, and also even outperformed other widely used methods including mixed integer linear programming in solving more complicated instances [7].

4 Ant Colony Optimization

Our ant colony optimization algorithm is designed to use a number of agents m, which we will refer to as ants, to explore possible paths in search of an optimal path, based on a probabilistic choice which we will define in more detail later in this section. These ants are specifically designed to pick up any bikes at a station that has an excess number of bikes, and drop off bikes at a station that has too few, without exceeding the capacity of bikes they can carry. We will make use of many variables in the description of our algorithm. An overview of their meanings is given below; they will be defined more precisely later in this section.

$\tau_{ij}(t)$	represents the pheromone on the arc from station i to station j at iteration t.
$p_{ij}^{(k)}$	represents the probability that an ant k will travel from station i to station j for j that has not been visited.
η_{ij}	is a function of the distance from station i to station j as part of the probability function of traveling from i to j.
$\lambda_{ij}^{(k)}$	is a measure of the potential effectiveness of the ant k traveling from station i to station j in distributing bikes efficiently as part of the probability function of traveling from station i to station j.

q_i	represents the number of bikes at station i after an iteration has been completed.
r_i	represents the target number of bikes at station i.
$B_i^{(k)}$	represents the best possible number of bikes we would have at station i, if an ant k traveled to station i next.
$H_i^{(k)}$	represents the number of bikes held by ant k at station i (after dropping off or picking up bikes at station i).
Q	represents the maximum number of bikes that an ant can carry.

Other than the fact that the ants will always begin at the designated depot, the first iteration of the algorithm is completely random, in that each ant has the same probability to visit any unvisited station. Upon arrival at the final unvisited station, the ant then returns directly to the depot. Now, the ant will essentially retrace its steps along the path that it had just traveled, laying "pheromones" along the path. The pheromone level for each ant is determined by the following formula:

$$\Delta\tau_{ij}^{(k)}(t) = \begin{cases} \left(\frac{1}{L^{(k)}(t)}\right)^{\sigma} + \left(\frac{1}{R^{(k)}(t)+1}\right)^{\delta} & \text{if arc}(i,j) \text{ is used by ant } k \\ & \text{in iteration } t, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $L^{(k)}(t)$ is the length of the path taken by the k^{th} ant in iteration t, and $R^{(k)}(t)$ represents the "residual" number of bikes, or the total difference between the final number of bikes and the target number of bikes for each station by the k^{th} ant in iteration t. Additionally, $\sigma > 0$ is the parameter that stresses the relative importance placed on path length for the pheromone level, and similarly $\delta > 0$ is the parameter stressing the relative importance of the residual for the pheromone level. This $\Delta\tau_{ij}^{(k)}$ rewards routes that are both short, and have a small residual, meaning that those routes would have a higher pheromone level and would be more likely to be chosen by an ant in the following iterations, as seen in Equation (2).

Once the first iteration is complete, the ants will now start the second iteration. During this iteration, the ants will each make a probabilistic choice as to which station they will visit next. The probability function that a given ant will travel from the station it is currently in, say station i, to another station that is yet to be visited, say station j, is determined by a mixture of both learned long term knowledge of good paths to take, and greedy short term decisions based on what the ant can do at very next step:

$$p_{ij}^{(k)} = \frac{\left[\tau_{ij}(t)\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}\left[\lambda_{ij}^{(k)}\right]^{\gamma}}{\sum_{l\in\mathcal{N}_i^{(k)}}\left[\tau_{il}(t)\right]^{\alpha}\left[\eta_{il}\right]^{\beta}\left[\lambda_{il}^{(k)}\right]^{\gamma}} \quad \text{if } j \in \mathcal{N}_i^{(k)} \tag{2}$$

where α, β, and γ are parameters that stress the relative importance of the pheromone level, the distance, and the potential effectiveness of the ant traveling from station i to station j respectively. The set of unvisited stations is $\mathcal{N}_i^{(k)}$. Also, as previously seen, τ_{ij} is the level of pheromone on the path from i to j. The function for distance is $\eta_{ij} = 1/c_{ij}$ where c_{ij} represents the distance from station i to j (which rewards smaller values of c_{ij}). The function for the potential effectiveness of the ant traveling from station i to station j is as follows:

$$\lambda_{ij}^{(k)} = \frac{|s_j - B_j^{(k)}|}{|r_j - B_j^{(k)}| + 1} \tag{3}$$

where s_j is the starting number of bikes at station j, r_j is the target number of bikes at station j, and $B_j^{(k)}$ is the function that determines the best possible number of bikes at station j, if the ant travels to station j next. That is:

$$B_j^{(k)} = \begin{cases} s_j - \left(Q - H_i^{(k)}\right) & \text{if } Q - H_i^{(k)} < s_j - r_j \\ s_j + H_i^{(k)} & \text{if } H_i^{(k)} < r_j - s_j \\ r_j & \text{otherwise} \end{cases} \tag{4}$$

where $H_i^{(k)}$ represents the number of bikes that ant k has after performing necessary actions at station i (i.e., the station that the ant currently sits at). The maximum number of bicycles that an ant can carry is denoted by Q. Notice that, the first output of the piecewise function is used if we cannot pick up enough bikes to meet the demand at station j, the second part of the function is evaluated if we cannot drop off enough bikes to meet the demand at station j, and the last part is for if we are able to meet the demand at station j.

We require that if $p_{ij}^{(k)}(t) < \Pi$ then $p_{ij}^{(k)}(t) = \Pi$, and similarly, if $p_{ij}^{(k)}(t) > \Theta$ then $p_{ij}^{(k)}(t) = \Theta$, for $0 < \Pi < \Theta < 1$, so that the minimum and maximum probabilities of $p_{ij}^{(k)}(t)$ are restricted by Π and Θ. After adjusting $P_{ij}^{(k)}$ we renormalize the probabilities. This restriction is enforced so that the distribution of probabilities of going to any given node that has not yet been visited is never zero which aids in assuring that premature convergence occurs less frequently.

Once the second iteration is completed, the pheromone levels are adjusted as follows:

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{(k)}(t) \tag{5}$$

where $\rho \in (0,1)$ is a fixed parameter that represents the evaporation rate, or the amount at which the pheromone level is disintegrated from each arc per iteration. From this point, the algorithm repeats exactly what it did in the second iteration for a designated amount of time and returns a solution only if it was better than the previous best solution.

5 Computational Results

The algorithm was coded in Python 2.7, and tested on an Intel Core i7-3540 CPU, 3.00 GHz, 4.00 GB single processor. We tested our ACO algorithm on random data produced in accordance with the conventions of Hernández-Pérez and Salazar-González [8], as well as on real data collected by Dell'Amico et al. [5].

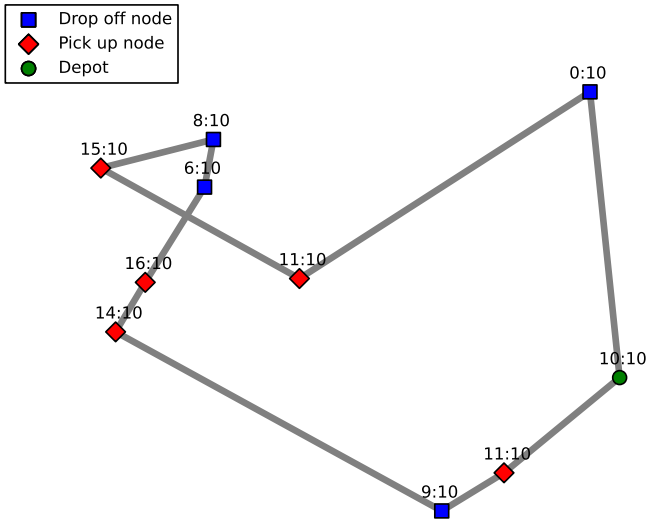

Figure 1: Best route found by the ACO method for a random graph, where the first label is the initial number of bikes and the second label is the final number of bikes for each station.

Figure 1 depicts an example of a random set of vertices with random demands at each vertex. The grey arcs represent the best solution found by our algorithm. As the legend depicts, the green circle represents the depot node, which is where the ants begin their routes. The blue squares represent the stations that have a positive demand, meaning that they require bikes to be dropped off in order to reach their target number of bikes. The red diamonds represent stations that have a negative demand meaning that they are overfilled and require bikes to be picked up to reach their target number of bikes. The target number of bikes for this instance is 10 for each node. The two numbers at each node that are separated by a colon represent the initial number of bikes at that node (on the left), and the final number of bikes at that node (on the right), i.e, the number of bikes at a node after the algorithm is run. This graph depicts an instance of the bike sharing problem and its solution. As it can be seen, every node has the target number of bikes, so the demand is completely met. The parameters set for all tests used in this paper are as follows: $a = 1$, $\bar{a} = 2$, $b = 0.2$, $\bar{b} = 1$, $\sigma = 1$, $\delta = 1$, $\alpha = 1$, $\beta = 1$, $\gamma = .05$, $\Omega = .95$, $\Pi = \frac{1}{(n+1)^2}$, and $\rho = .91$.

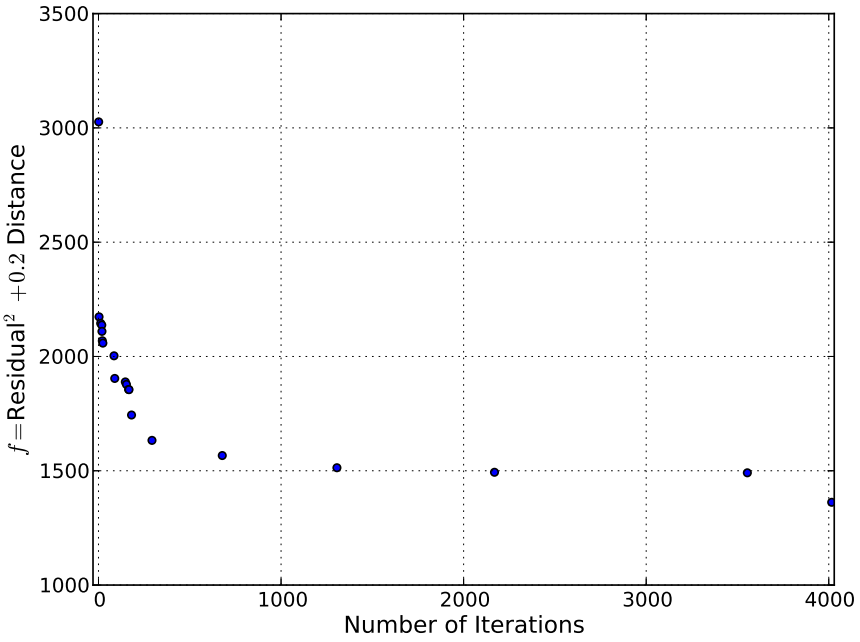

Figure 2: The most recent objective value as function of the number of iterations.

Figure 2 is a plot of the most recent "best" objective value found by our algorithm as a function of iterations. The solutions plotted converge very quickly to a "good approximation" and then level off but keep improving

over time. Notice that a lower objective value is found even after 4016 iterations. This suggests that our algorithm can find good solutions in a short amount of time, but it takes a while for it to find better solutions after a certain number of iterations. It can also be seen that the effects of the driving mechanism behind the ant colony optimization algorithm, that is, its learning capability, allows it to make better decisions about which paths to take in the future based on its past experiences.

n	**Q**	**Residual**	**Length**	**O.V.**
20	10	4	3600.79	736.158
20	15	0	3448.77	689.754
20	20	0	3409.17	681.834
20	25	0	3067.87	613.574
20	30	0	3199.92	639.984
30	10	2	6341.72	1272.344
30	15	0	5330.59	1066.118
30	20	0	5192.02	1038,404
30	25	0	5094.58	1018.916
30	30	0	4993.09	998.618
40	10	4	8081.86	1632.372
40	15	0	7949.14	1589.828
40	20	2	7303.26	1464.652
40	25	2	7144.89	1432.978
40	30	2	7485.77	1501.154
50	10	6	7855.84	1607.168
50	15	2	8818.31	1767.662
50	20	0	6779.23	1355.846
50	25	0	7525.27	1505.054
50	30	0	7345.95	1469.19

Table 1: Performance of the ACO method on randomly generated instances.

For our randomly generated instances, we used the conventions of Hernández-Pérez and Salazar-González [8] to formulate the random set of nodes and demands. For each randomly generated instance, we tested our algorithm for vehicle capacities $Q \in \{10, 15, 20, 25, 30\}$, and number of nodes $n \in \{20, 30, 40, 50\}$. The last column of the table represents the actual value of the objective function, which is $f = aR^{\bar{a}} + bL^{\bar{b}}$. We allowed our algorithm to run for fifteen minutes for testing the randomly generated instances as well as for any other instance considered in this study. Our solutions are similar to those of Hernández-Pérez and Salazar-González, especially for smaller instances [8].

Instance	**n**	**Q**	**Best Known**	**ACO**	**% Gap**	**Path**
16LaSpezia30	20	30	20,746	21,518	3.65	[0, 1, 15, 11, 7, 16, 9, 14, 17, 19, 12, 4, 2, 5, 3, 10, 8, 6, 13, 18, 0]
17LaSpezia20	20	20	20,746	23,488	12.40	[0, 1, 15, 11, 10, 7, 16, 9, 14, 17, 19, 4, 12, 2, 5, 3, 6, 8, 18, 13, 0]
18LaSpezia10	20	10	22,811	23,908	4.70	[0, 1, 15, 13, 11, 10, 7, 16, 9, 17, 14, 4, 2, 5, 3, 12, 19, 8, 6, 18, 0]
21Ottawa30	21	30	16,202	17,178	5.85	[0, 1, 4, 13, 14, 15, 16, 11, 12, 17, 18, 19, 20, 9, 10, 8, 7, 6, 5, 3, 2, 0]
22Ottawa20	21	20	16,202	17,178	5.85	[0, 1, 4, 13, 14, 15, 16, 12, 11, 17, 18, 19, 20, 9, 10, 8, 7, 6, 5, 3, 2, 0]
23Ottawa10	21	10	17,370	17,604	1.34	[0, 1, 4, 13, 16, 14, 15, 12, 11, 10, 9, 17, 18, 19, 20, 8, 7, 6, 5, 3, 2, 0]
33Madison30	28	30	29,246	34,029	15.12	[0, 21, 16, 25, 5, 6, 3, 1, 4, 2, 24, 13, 14, 12, 10, 27, 26, 7, 22, 23, 8, 19, 18, 11, 15, 9, 20, 17, 0]
34Madison20	28	20	29,839	34,706	15.08	[0, 21, 16, 13, 24, 1, 5, 6, 3, 4, 2, 14, 12, 27, 10, 7, 26, 8, 19, 18, 23, 22, 11, 15, 9, 25, 20, 17, 0]
35Madison10	28	10	33,627	38,677	13.97	[0, 21, 16, 13, 1, 4, 3, 5, 6, 2, 24, 14, 11, 15, 12, 10, 18, 22, 23, 8, 19, 27, 7, 26, 9, 25, 20, 17, 0]

Table 2: Performance of the ACO method on real world instances.

The real world data we used to test our algorithm with, is introduced by Dell'Amico et al. The data we used was collected from La Spezia, Italy; Madison, United States; and Ottawa, Canada [5]. We chose these instances because the total demand in the system (i.e, the sum of all demands in the system) was close to zero. This meant that the system had enough bikes to satisfy the total demand, and that the stations could be balanced by visiting each node once for the instances considered. Table 2 is a comparison of our results with those of Dell'Amico's [5]. In this table, n represents the number of nodes and Q represents the capacity of the delivery vehicle. Dell'Amico's results are named 'Best Known' since they are the best known solutions. Our results are labeled 'ACO', with '% Gap' denoting the percentage difference between the two solutions. We changed our original algorithm slightly to include the stipulation that every station's demand must be met exactly in an attempt to better compare the results to those of Dell'Amico. As it can be seen, we were very successful in finding good solutions for the real world instances. Our results seem to outperform some of the recently developed methods [6].

6 Conclusion

In this paper, we formulate a novel algorithm for solving the bike sharing problem that is both efficient, and effective for solving small to moderate size instances. This study is a first attempt at using solely Ant Colony Optimization for solving the single vehicle capacitated pickup and delivery problem, namely the BSP. We found that our algorithm was very effective compared to other methods that have recently been used to solve the BSP. We believe that our approach also shows promise for solving other variations of the BSP, as described in the next section.

7 Future Work

We believe that one of the true advantages of using the ACO algorithm for solving the BSP is its flexibility. This algorithm could possibly be changed to accommodate many different variations of the BSP as well as any other type of single vehicle capacitated pick up and delivery problems. One interesting avenue of study in a relatively unexplored area is that of the multi-commodity capacitated pick up and delivery problem [9], which we believe the ACO algorithm is very well suited for. In this problem, rather than attempting to reposition one commodity in a graph, one needs to reposition m different types of commodities. This problem is motivated by the idea of a BSP with two kinds of bikes, or perhaps a bike and a scooter that the vehicle must reposition simultaneously. The ACO algorithm would handle

this by adding another short-term greedy function to the probability function and adding some type of factor of the new commodity to the pheromone function. An example of the implementation of another commodity into the probability and pheromone functions is as follows:

$$p_{ij}^{(k)} = \frac{\left[\tau_{ij}(t)\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}\left[\lambda_{ij}^{(k)}\right]^{\gamma}\left[\mu_{ij}^{(k)}\right]^{\phi}}{\sum_{l\in\mathcal{N}_i^{(k)}}\left[\tau_{il}(t)\right]^{\alpha}\left[\eta_{il}\right]^{\beta}\left[\lambda_{il}^{(k)}\right]^{\gamma}\left[\mu_{il}^{(k)}\right]^{\phi}} \quad \text{if } j \in \mathcal{N}_i^{(k)}$$

$$\Delta\tau_{ij}^{(k)}(t) = \begin{cases} \left(\frac{1}{L^{(k)}(t)}\right)^{\sigma} + \left(\frac{1}{R_1^{(k)}(t)+1}\right)^{\delta} + \left(\frac{1}{R_2^{(k)}(t)+1}\right)^{\omega} & \text{if arc}(i,j) \text{ is used by ant } k \text{ in iteration } t, \\ 0 & \text{otherwise.} \end{cases}$$

In the first equation, $\mu_{ij}^{(k)}$ represents the function for the potential effectiveness of ant k traveling from station i to station j in distributing the second commodity, with ω as a parameter that stresses the relative importance of the second commodity. In the second equation, $R_2^{(k)}$ represents the residual of the second commodity that is being distributed. Otherwise, all other functions would be essentially the exact same.

Another interesting avenue of study that the ACO algorithm could potentially map to is the time constricted BSP in which the user designates a period of time that the vehicle has to perform its activities; however, each activity has a cost of time. An example of this would be an unloading and loading time for bikes, as well as a travel time between bike stations. The motivation behind this variation of the BSP is to create a more realistic routing algorithm that takes into account the actual amount of time allotted to repositioning process. This problem could also potentially implement the revisitation of stations in an effort to create a more realistic model. Within this time constricted variation of the BSP, we believe that it would be also be possible to implement a dynamic element. Here, the idea would be to have the vehicle work towards optimizing the current system, and every period have the system update to the new information about station demand, and adjust the ACO's goals accordingly.

The last branch from BSP that we considered was using a system with more than one vehicle present. We hypothesize that one could potentially implement ACO into a BSP like this by pairing certain ants together to work as teams in repositioning the bikes. This would create a more realistic model for systems that implement a method that uses multiple vehicles for the redistribution of their bikes.

References

[1] Mike Benchimol, Pascal Benchimol, Benot Chappert, Arnaud de la Taille, Fabien Laroche, Frederic Meunier, and Ludovic Robinet. Balancing the stations of a self service bike hire system. *RAIRO - Operations Research*, 45:37–61, 1 2011.

[2] Christian Blum. Ant colony optimization: Introduction and recent trends, 2005.

[3] Daniel Chemla, Frederic Meunier, and Roberto Wolfler Calvo. Bike sharing systems: Solving the static rebalancing problem. *Discrete Optimization*, 10(2):120 – 146, 2013.

[4] Claudio Contardo, Catherine Morency, and L.-M. Rousseau. Balancing a dynamic public bike-sharing system. Technical Report CIRRELT-2012-09, Universitè de Montrèal, Montrèal, Canada, 2012.

[5] M. Dell'Amico, E. Hadjicostantinou, M. Iori, and S. Novellani. The bike sharing rebalancing problem: Mathematical formulations and benchmark instances. *Omega*, 45:7–19, 2014.

[6] Güneş Erdoğan, Maria Battarra, and Roberto Wolfler Calvo. An exact algorithm for the static rebalancing problem arising in bicycle sharing systems. *European Journal of Operational Research*, 245(3):667–679, 2015.

[7] Luca Di Gaspero, Andrea Rendl, and Tommaso Urli. A Hybrid ACO+CP for Balancing Bicycle Sharing Systems, 2013.

[8] Hipólito Hernández-Pérez and Juan-José Salazar-González. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145(1):126 – 139, 2004. Graph Optimization {IV}.

[9] Hipólito Hernández-Pérez and Juan-José Salazar-González. The multi-commodity pickup-and-delivery traveling salesman problem. *Networks*, 63(1):46–59, 2014.

[10] Christian Kloimüllner, Petrina Papazek, Bin Hu, and Günther R. Raidl. Balancing Bicycle Sharing Systems: An Approach for the Dynamic Case. In Christian Blum and Gabriela Ochoa, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 8600 of *LNCS*, pages 73–84. Springer, 2014.

[11] Nenad Mladenovi, Dragan Uroevi, Sad Hanafi, and Aleksandar Ili. A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research*, 220(1):270 – 285, 2012.

[12] Tal Raviv, Michal Tzur, and Iris A. Forma. Static repositioning in a bike-sharing system: models and solution approaches. *EURO Journal on Transportation and Logistics*, 2(3):187–229, 2013.

[13] Jasper Schuijbroek, Robert Hampshire, and Willem-Jan van Hoeve. Inventory rebalancing and vehicle routing in bike sharing systems. 2013.

[14] Thomas Stützle and Holger H. Hoos. MAX-MIN Ant System, 1999.

[15] Fanggeng Zhao, Sujian Li, Jiangsheng Sun, and Dong Mei. Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Industrial Engineering*, 56(4):1642 – 1648, 2009.