Optimizing On-demand Delivery Problems

James Wang
University of Virginia
Charlottesville, VA 22903
jjw6wz@virginia.edu

Abstract

On-demand delivery services are widely used and present several areas for optimization. In this paper, we explore and characterize the deliveryman-request assignment problem in a dynamic setting. We illustrate the problem as a bipartite matching and show an integer programming formulation of the problem. We also introduce a batching algorithm to solve a relaxed linear programming version of the problem so that it can be applied in a dynamic setting. The batching approach is tested against a greedy algorithm in a simple simulation under different supply and demand conditions. Results show that the batching LP approach consistently performs better than a greedy approach, but with large trade-offs in computation.

1 Introduction

On-demand delivery services have become increasingly popular within recent years, ranging from food delivery to furniture delivery. Services like UberEats, Amazon Prime Now, and Instacart promise a variety of items to your doorstep within just a couple of hours. While these types of services vary, they all operate under very similar conditions making it possible to generalize the problem. These services operate as follows: a customer orders for a good from a store to be delivered to them as soon as possible, which results in a deliveryman being dispatched to collect goods from the store and then deliver it to the customer. At scale, this problem can be difficult to optimize but is also very valuable given that delivery services typically operate under thin margins. Potential optimization areas include supply/demand forecasting, pre-positioning, and routing.

In this paper we will explore optimizing delivery dispatch by formally defining the problem as a bipartite matching under different objectives. We then provide an integer programming (IP) formulation and discuss how to solve it under a dynamic setting. Finally we compare a bipartite matching approach to a simple greedy heuristic for dispatching under a dynamic, simulated environment and discuss implications for real-time systems.

1.1 Key Characteristics

In this paper, we list the following as defining characteristics of a on-demand delivery problem:

1. Requests are made with no notice and are expected to be fulfilled as soon as possible
2. A request can only be fulfilled by one specific store
3. A deliveryman can only service one request at a time
4. Matches are automatically

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

(1) should not be confused with the objective of the overall system. While minimizing wait-time can be used as an objective, the purpose of (1) is to avoid the use of a time-window - there is prior knowledge of the request initialization time and no expiration time on the request. For objectives that don't involve wait-time, long wait-times may want to be used as a penalty. (2) provides the most generalized version of this problem. Some delivery services can be fulfilled by several different stores (e.g groceries), but this is uncommon. (3) is interesting as an area for further research - if multiple requests to one store are made in a short amount of time, it could be possible for one deliveryman to fulfill all the requests. (4) states there is no coordination between customer and deliveryman, and all deliverymen are assumed to accept the assignment given.

2 Related Literature

No paper directly discussing this topic could be found, but significant research has gone into optimizing two related problems: the dial-a-ride-problem (DARP) and ride-sharing. DARP features a set of taxis all emanating from one location that fulfill a series of known pickup requests. Cordeau [2006] provides an exact branch-and-bound algorithm to minimize total distance traveled with a time-window constraint. Such an approach, however, is limited computational and does not seem easily scale-able. Xiang et al. [2006] present a large-scale approach using incremental-based heuristics to minimize user and driver costs.

The ridesharing problem is very similar and can take on a number of forms, including multi-passenger, multi-vehicle, dynamic, and static. In ride-sharing, drivers and riders both announce their planned departure time and location, and a system optimizes matches.Najmi et al. [2017] present a rolling horizon solution using a clustering heuristic that batches requests and drivers efficiently. Jia et al. [2016] approach the dynamic ridesharing problem using directed graphs more complicated than a simple bipartite matching graph. Their formulation involves a "task map" that is unique for each driver; it represents the entire set of tasks a driver can take on, and allows a system to chain tasks together. To transfer this into a real-time system, they propose several heuristics including greedy approaches. This approach is particularly interesting to our context since it can be applied also on the on-demand delivery case. A literature review by Agatz et al. [2012] has covered the various objective functions used by other ride-sharing optimization papers. These include total miles traveled by all parties, total time traveling, and total number of participants. They summarize various approaches but conclude by stating there still exists significant optimization for real-time, scale-able systems to ridesharing.

3 Formalizing the On-Demand Delivery Problem

Let D be the set of all deliverymen and R be the set of all requests. A simple and intuitive way to model assignments is using a bipartite graph. A column of nodes is created for each $d \in D$ and another column for each $r \in R$ with a source and sink node attached appropriately.

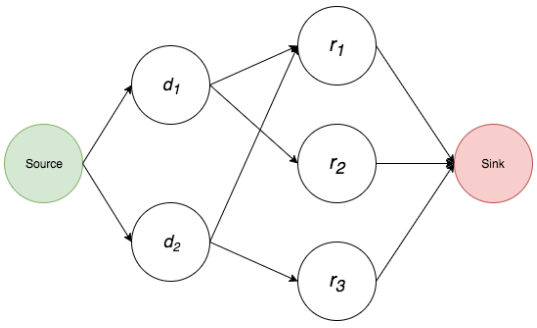

Figure 1: Bipartite Graph of Deliverymen and Requests

We can represent bipartite matching as an integer programming problem. Let $c_{d,r}$ be the weight of the edge between any d, r nodes. Edge weights can be used to represent the objective function - different objectives will be explored later in the paper. Let $x_{i,j}$ be an indicator variable that takes the following states:

$$x_{i,j} = \left\{ \begin{array}{ll} 1, & \text{if an assignment is made between } d, r \\ 0, & \text{else} \end{array} \right\}$$

We can now formulate our integer programming problem

$$\begin{array}{lll} min & \sum_{d \in D} \sum_{r \in R} c_{d,r} x_{d,r} & \\ s.t & \sum_{r \in R} x_{d,r} \leq 1, & \forall d \in D \\ & \sum_{d \in D} x_{d,r} \leq 1, & \forall r \in R \\ & x_{d,r} \in \{0,1\} & \forall d \in D, r \in R \end{array} \tag{1}$$

Our objective is to minimize the total cost of all selected edges. Constraints 2 and 3 ensure at most one matching per request or driver (flow and capacity), and constraint 4 enforces integrality. Since integer programming problems are NP-hard, we need apply some relaxation to it. One common relaxation approach is to apply a Lagrangian relaxation, first proposed by Geoffrion [1974]

$$\begin{array}{lll} min & \sum_{d \in D} \sum_{r \in R} c_{d,r} x_{d,r} - \sum_{r \in R} \lambda_r (\sum_{d \in D} x_{d,r} - 1) & \\ s.t & \sum_{r \in R} x_{d,r} \leq 1, & \forall d \in D \\ & x_{d,r} \in \{0,1\} & \forall d \in D, r \in R \end{array} \tag{2}$$

This reduces the problem into $|D|$ 0-1 knapsnack problems, which can be solved in polynomial time (Fisher [1981]). A second approach is to apply a linear programming relaxation to translate this problem from an IP to an LP. This can be done by simply loosening our integral constraint on $x_{r,d}$

$$\begin{array}{lll} min & \sum_{d \in D} \sum_{r \in R} c_{d,r} x_{d,r} & \\ s.t & \sum_{r \in R} x_{d,r} \leq 1, & \forall d \in D \\ & \sum_{d \in D} x_{d,r} \leq 1, & \forall r \in R \\ & x_{d,r} \geq 0 & \forall d \in D, r \in R \end{array} \tag{3}$$

How can we ensure that we don't create any "fractional" matches now that x is no longer fixed to integer values? For a bipartite matching, it has been shown that optimal or perfect matches will always take on integer values. Let M be the set of all matches for the given graph G. The convex hull of M creates a matching polytope P where

$$P = \{\sum_i \lambda_i m_i |_i \in M, \sum_i \lambda_i = 1\} \tag{4}$$

And let F be the feasible region of the LP, known as the fractional polytope:

$$F = \{\sum_{r,d} x_{r,d} \leq 1, \forall r \in R, d \in D\} \tag{5}$$

Balas and Pulleyblank [1983] show that M = F since we can write any vertex of F as a convex combination two points in F, so there cannot be any non-integer vertices. We can also define a perfect matching polytope and fractional perfect matching polytope for the optimal solution set of our LP and arrive at an equality statement. And because we don't violate the integrality constraint, the relaxed LP and relaxed Lagrangian will arrive at the same optimal result (Geoffrion [1974]).

To choose between the two then, computation time is often the deciding factor. Empirical evidence from Fisher [1981] suggests that for smaller problems the relaxed LP performs better, but as graph

size grows the relaxed Lagrangian begins to outperform. The solver used in this paper, ECOS (Domahidi et al. [2013]), solves for the relaxed Lagrangian using a branch and bound approach.

3.0.1 Dual Problem

From equation 1, we can easily construct the dual problem by introducing dual variables for our inequality constraints.

$$\begin{aligned} max \quad & \sum_{d \in D} \lambda_d + \sum_{r \in R} v_r \\ s.t \quad & \lambda_d + v_r \leq c_{d,r}, \ \forall d \in D, r \in R \end{aligned} \tag{6}$$

This is useful since it allows us to use primal-dual algorithms to solve the LP, though it's not used often in practice.

3.1 Objectives

The flexibility of the bipartite matching formulation allows us to consider different objectives simply through changing edge weights. We will look at a few examples.

Maximize Matches If our goal was to fulfill every request, we could change our objective into a maximization.

$$max \quad \sum_{d \in D, r \in R} x_{d,r} \tag{7}$$

Minimize Distance Traveled Let $l(d)$ be the deliveryman's current location, $l(r_s)$ be the location of the requested store, and $l(r_c)$ be the location of the customer. Let $h(x, y)$ be the Haversine distance between x, y - we can set our edge weights to the Haversine sum of traveling this path and minimize the distance. We use this objective function for our simulations.

$$min \quad \sum_{d \in D, r \in R} h(l(d), l(r_s)) + h(l(r_s), l(r_c)) x_{d,r} \tag{8}$$

Minimize Wait Time Let r_t be the time a request is created and r_f be the time a request is fulfilled. Suppose we want to penalize higher wait times i.e waiting 5-10 minutes has less penalty than 50-55 minutes. We can use a simple quadratic.

$$min \quad \sum_{d \in D, r \in R} (\mathrm{r}_f - r_t)^2 \, \mathrm{x}_{d,r} \tag{9}$$

Maximize Profit Let p be the fixed payment amount per delivery, and c be the cost per mile traveled by the deliveryman.

$$max \quad \sum_{d \in D, r \in R} (p - c * h(l(d), l(r_s)) + h(l(r_s), l(r_c))) x_{d,r} \tag{10}$$

4 Implementation

4.1 Simulation

To benchmark our assignment algorithms in a dynamic environment, we developed a simple on-demand delivery simulation.[1] The simulation contains three types of objects: stores, deliverymen, and requests, and runs in discrete time-steps, where each time-step represents one second. For each step, we update the status of the world as follows.

[1] Full source code and visualization: https://github.com/jameswang14/delivery-optimizer/

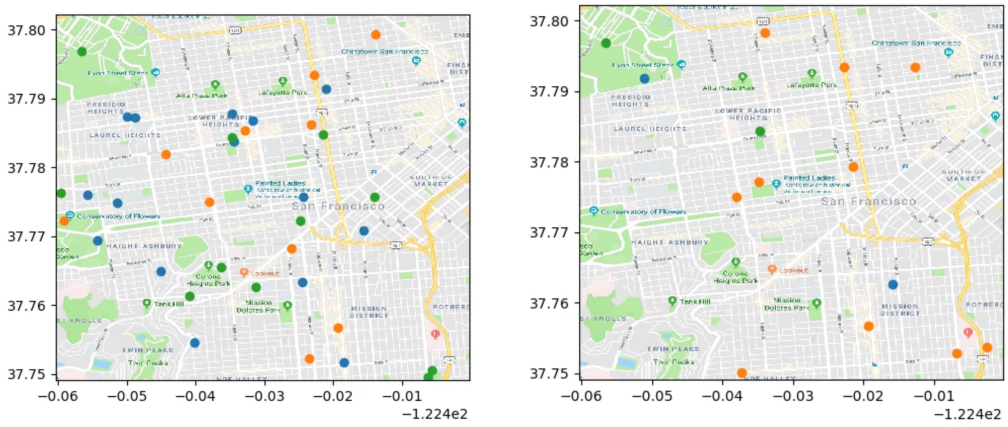

Figure 2: Screenshots of the Simulation. Blue dots represent drivers, orange dots represent stores, and green dots represent requests.

Requests have a probability of a_r of being generated every step with a random location and requested store - if generated they enter a queue. If the requests queue is not empty and there are free deliverymen, we make an assignment according to the algorithm being used. Deliverymen then have their location updated, moving in a direct line to a store or customer at roughly 10mph. For each mile a deliveryman travels from a store to a customer's location, they accumulate some cost c. Upon delivery, they receive a payment fixed payout p.

4.1.1 Batch Matching Assignment

The batch assignment algorithm accumulates new requests and newly freed drivers over ten second intervals, at the end of which it constructs the bipartite graph and solves the IP according to the formulation in equations 2 and 8. The algorithm is implemented using CVXPY from Diamond and Boyd [2016] and the ECOS_BB (branch and bound) solver from Domahidi et al. [2013].

4.1.2 Greedy Assignment

The greedy assignment works on a per-request basis, where new requests are added to a queue and popped off so long there is an available deliveryman. For each request, we consider the set of available deliverymen and assign the deliveryman with the smallest Haversine distance to the store being requested.

5 Results

Three simulations were run under different setting to observe different scales of the problem. The settings for each scale can be seen in table 1. Results are in table 2. All times are in seconds. Simulations were carried out on macOS with a 2.9ghz processor.

Table 1: Simulation Settings For Different Scales

		Small	Medium	Large
n_d	number of drivers	50	75	100
n_r	initial number of requests	15	30	50
n_s	number of stores	10	15	20
a_r	seconds per new request	20	10	1
c	cost per mile	0.3	0.3	0.3
p	payment	7.5	7.5	7.5

Table 2: Simulation Results For Different Scales

	Small		Medium		Large	
	Greedy	LP	Greedy	LP	Greedy	LP
Total Revenue	875	910	1092	1624	1596	5320
Total Cost	629.63	614.77	949.43	1238.08	1331.21	1874.44
Total Number of Requests	177	177	337	337	3600	3600
Total Number of Unfulfilled Requests	53	47	181	105	3372	2840
Total Wait Time	135430	128507	224121	226909	484131	486274
Longest Wait Time	2019	1976	2464	2467	3381	3527
Computation Time	2.60	6.77	3.639	93.05	8.76	31765.70

6 Discussion

Our batch optimization approach outperforms a greedy approach in all cases across nearly every metric. The batch approach is able to fulfill more requests and generate more profit, but wait times are near equal between the two. This could be a consequence of the ten second batch time we chose. The batch optimization also gains huge improvements over greedy as scale increases. With a larger set of deliverymen and requests, the greedy algorithm will more have more chances to make a non-optimal chance. The drawback is paid in computation time however - solving the LP repeatedly makes our batch run-time grow significantly more compared to the greedy approach, which grows at a linear rate. This and long wait-times suggest a lower batch time may be beneficial.

Long computation times also have strong implications for real-time production systems; solving graphs efficiently requires a combination of choosing the right solver, partitioning data correctly (e.g by geography), and using heuristics when possible. At smaller scales, the difference between the two approaches is nearly negligible. An adaptive approach that chooses between the two depending on the state of the environment may better optimize for run-time while still achieving near-optimal results.

7 Conclusion and Future Directions

We have introduced the on-demand delivery problem as a bipartite matching and integer programming problem. We've show relaxation techniques that can be applied to transform the problem into a solvable LP and introduced various objective functions. We've created a simple but scale-able simulation to benchmark a greedy and IP batching approach to the assignment problem and discussed the results for real-world applications.

There is still a large swath of optimization to be explored for this problem and related problems. One consequence of using bipartite matching is that the model can be too simple and prevents further optimization. Using a more complex graph such as the one used by Jia et al. [2016] that features arcs between each location would allow request chaining (assign multiple requests to one deliveryman) and multiple deliveries from one store. Routing optimization can also affect assignment - the matching graph can be expanded to deliverymen en route to a store but have not picked up anything yet, and also deliverymen about to finish their current request.

References

N. Agatz, A. Erera, M. Savelsbergh, and X. Wang. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295 – 303, 2012. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2012.05.028. URL `https://www.sciencedirect.com/science/article/pii/S0377221712003864`.

E. Balas and W. Pulleyblank. The perfectly matchable subgraph polytope of a bipartite graph. *Networks*, 13(4):495–516, 1983. ISSN 1097-0037. doi: 10.1002/net.3230130405. URL `http://dx.doi.org/10.1002/net.3230130405`.

J.-F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3): 573–586, 2006. doi: 10.1287/opre.1060.0283. URL `https://doi.org/10.1287/opre.1060.0283`.

S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, pages 3071–3076, 2013.

M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management science*, 27(1):1–18, 1981.

A. M. Geoffrion. *Lagrangean relaxation for integer programming*, pages 82–114. Springer Berlin Heidelberg, Berlin, Heidelberg, 1974. ISBN 978-3-642-00740-8. doi: 10.1007/BFb0120690. URL https://doi.org/10.1007/BFb0120690.

Y. Jia, W. Xu, and X. Liu. An optimization framework for online ride-sharing markets. *CoRR*, abs/1612.03797, 2016. URL http://arxiv.org/abs/1612.03797.

A. Najmi, D. Rey, and T. H. Rashidi. Novel dynamic formulations for real-time ride-sharing systems. *Transportation Research Part E: Logistics and Transportation Review*, 108:122 – 140, 2017. ISSN 1366-5545. doi: https://doi.org/10.1016/j.tre.2017.10.009. URL https://www.sciencedirect.com/science/article/pii/S1366554517300017.

Z. Xiang, C. Chu, and H. Chen. A fast heuristic for solving a large-scale static dial-a-ride problem under complex constraints. 174:1117–1139, 02 2006.