The Self-Driving Pizza Car: Optimizing Control Algorithms for Autonomous Delivery Systems

Nischal Sinha,[1] Bryce Ferguson[2]
[1]*Great Oak High School, 32555 Deer Hollow Way, Temecula, CA 92592*
[2]*Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106*
Email: nish.d.sinha@gmail.com

Recent years have seen an increase in attention to autonomous vehicles, especially in the field of autonomous delivery using self-driving trucks and cars. While the creation of such autonomous vehicles has become fairly straight-forward, a far less explored question is what constitutes an ideal algorithm to determine delivery routes. One unique case of such delivery systems is that of self-driving pizza cars (SDPCs), wherein a vehicle must consider incoming orders, prices, wait times, and other factors, all while baking orders as the car travels. In this paper we pose a computer model to simulate an SDPC on a network of locations. We then develop both greedy and temporal difference (TD) control algorithms, prioritizing various properties related to customer experience and efficiency. Finally, we compare the delivery efficiencies of each algorithm. Ultimately, TD algorithms consistently outperformed their greedy counterparts, while there was no conclusive control which applied optimally over all metrics.

Keywords: Autonomous delivery, Control algorithms, Reinforcement learning, Temporal difference learning

I. INTRODUCTION

Autonomous vehicles have become an increasingly popular and feasible solution to modernizing public transportation networks and delivery systems. One unique and burgeoning application of this technology is in pizza delivery – a $9.8 billion-dollar industry in the US in 2018 [1]. Established companies such as Domino's and Pizza Hut have partnered with automakers (Ford and Toyota, respectively) to automate their delivery infrastructures [2], following the lead of innovative startups such as Mountain View based Zume Pizza [3].

This problem considers a novel idea for a self-driving pizza car (SDPC) which functions much like a self-driving taxi system, but with a specialized twist. The SDPC operates fully autonomously to deliver pizzas from location to location. However, it differentiates itself from other autonomous delivery systems in that it also bakes pizzas fully in-car as it drives. Such a device would be advantageous to pizza delivery companies as it eliminates the need for a human driver while also reducing wait times per order, as the SDPC would not have to return to the brick-and-mortar store to pick up new deliveries.

With this in mind, the SDPC behaves under a number of operating constraints: at any given increment of time, it first receives a series of random orders, then deliberates the optimal path to satisfy them, and finally bakes a pizza as it drives to the order location, one pizza at a time. This final constraint adds an additional time factor which the control algorithm must consider when deciding its optimal path (one which we also see in industries like that of perishable goods and autonomous taxis).

Ultimately, creating an effective decision-making algorithm for such an SDPC has proven difficult. The current state of the literature on routing problems leans heavily towards greedy control algorithms, often leveraging vast repositories of data to calculate trends for optimal car behavior [4, 5]. However, in the case of a fleet of SDPCs (or various other autonomous delivery networks), there is a noted lack of data available due to the fact that the technology has never been implemented before. In light of this, the need for a more dynamic control algorithm becomes apparent [6].

Temporal difference (TD) learning is an alternative approach to control algorithms for systems wherein an object makes decisions in accordance with an approximated cost function generated from the current and all past states of the system [7]. This is a heavily researched field; in the past it has been applied to self-driving cars from the perspective of decision making for driving techniques [8, 9]. Additionally, [4] provides an exploration of how RL can be used to manage off-duty autonomous mobility on demand (AMoD) cars to navigate to optimal pickup locations. Currently, however, there is a marked lack of research into the applications of TD learning in the on-duty elements of an autonomous delivery system, where the algorithm would decide which in which order to satisfy a number of randomly generated orders.

Our solution to the problems presented above is to use RL to optimize the SDPC's behavior for a network with stochastically updating orders. By using temporal difference-lambda (TD(λ)) learning to dynamically adjust the algorithm, the SDPC optimizes its efficiency for the particular network, increasing its performance over time by adjusting its control algorithm with each increment.

We begin by modelling a single SDPC in a network with stochastic orders and measuring the delivery efficiency of the model while using a series of greedy control algorithms. This serves as a baseline from which we develop TD control algorithms for the same variety of metrics and test their efficiencies before comparing the greedy and TD results. This allows us to investigate approaches to SDPC routing that considerably improve the delivery efficiency of the system over the course of an episode when compared to greedy approaches.

II. PROBLEM MODEL

A. Graph Framework and Stochastic Orders

We begin by developing a model which includes the SDPC and n delivery locations as a framework for our various control algorithms. Here, we represent locations as nodes on an undirected graph H (see FIG. 1). The edges of H represent roads linking two different locations, with weights corresponding to a fixed travel time, or distance, between the locations. The SDPC is denoted on the graph by an additional node on H, with edges that continually update as the system changes states. This process is described in more depth in section *II*, subsection *C*.

At any given point in time, node i receives an order with probability p_i provided there is not an order already pending there. Such an order has three properties: bake time b and price c, where b and c are random variables that do not change until the order has been satisfied, and a continually updating wait time w, which begins at 0 and increases incrementally with time until an order is fulfilled. We can then represent an order as $o_i = (b_i, c_i, w_i)$. For this problem, once an order is placed it remains as an attribute of the node until the SDPC satisfies it or until a set amount of time, after which it expires. Additionally, at any moment, a node can host only one order, meaning the maximum number of orders for a network at t is the network size n.

B. Decision-Making

At $t = 0$ or once the SDPC has delivered an order, it then evaluates its next order to satisfy. This is based on a predetermined control algorithm which chooses a next target $u \in U$, where U represents the set of controls for all possible locations that the SDPC can travel to. The control algorithm considers the state of the system $x = x(t)$, which contains each order currently on the graph as well as the position of the SDPC. A given control algorithm for the system considers the state of the system as an input to identify a control for the SDPC to adopt – that is, which node it should travel to next – based on minimizing the episode's accrued cost. In a general form this is given as:

$$u = argmin_u\, J\big(f(x,u)\big), \tag{1}$$

where $f(x,u)$ denotes the expected state of the system x at the next step if control u is executed and $J\big(f(x,u)\big)$ represents a set cost under state $f(x,u)$. An important constraint placed on the model is that target nodes are evaluated only when the SDPC is no longer satisfying an order. This behavior is based on the simplifying assumption that an SDPC can bake a single order at a time and prevents the SDPC from changing its target location once it has made its decision. For cases featuring complex control algorithms operating on particularly large networks, this has the advantage of reducing the processing power used by the SDPC to more manageable amounts, as it would only be running the algorithm once per completed delivery as opposed to continually.

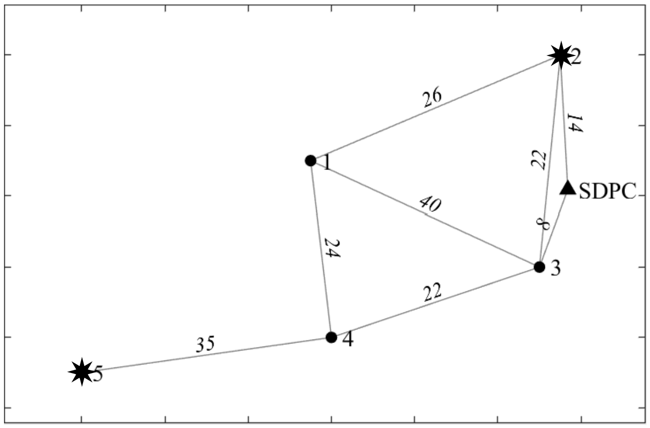

FIG. 1. A sample network with locations $n = 5$. Starred nodes (2,5) represent locations with currently outstanding orders. Here the SDPC is moving from location 3 to satisfy an order at location 2.

C. Traveling

The SDPC is represented by an additional node of degree 2 when delivering an order, its neighbors being its previous and next node. As it travels from the previous node to the next, the weights of these edges adjust to represent the SDPC's real-world distance from each respective location. Furthermore, when idle the SDPC's spatial location of being "at" the location is represented by a weight 0 for its edge. If the SDPC arrives at a location before its travel time has reached b, it will rest at the node until sufficient time has elapsed and it can satisfy the order. Finally, if the target order happens to be placed at a location where the SDPC is already resting, it will proceed to wait at that location for the duration of b – as if the SDPC's travel time to the location was 0.

If the shortest path between the current node and the next node includes an intermediate point, the SDPC will consider the intermediate node as one to travel to, but not as one which has an order – this means that the SDPC passing through an intermediate node will not resolve any order there, should one exist.

III. ALGORITHMS

A. Overview and Greedy Algorithms

Given that the goal of our control algorithm is to minimize the total cost J of the system and its controls over the entirety of the episode, two outstanding questions arise from formula (1) in section *II*, subsection *B*: How do we determine what system properties to value for our cost, and how do we evaluate the complete function J so as to find its optimal value?

The first of these questions is open to a number of potential answers, the general goal of each being to maximize the real-world profit an SDPC. For the purposes of our research, we explore three factors in particular: minimizing the distance traveled by the SDPC, minimizing the wait time for an order, and maximizing revenue.

Regarding the question of calculating the optimal value for J, we can first explore a brute-force approach: For a complete iteration of the simulation, or episode, of t timesteps, one may consider the following formula:

$$J_{total} = \sum_{t=1}^{t} \alpha^t \cdot J(x_t) \qquad (2)$$

where α is a discount factor between 0 and 1. This amounts to the SDPC evaluating the cost at each and every state, then adding those together while discounting the value of later states to find the total cost of the episode.

While this approach would in principle provide an optimal policy and thus lead us to the true global minimum of the cost function, at high time complexity it is for too computationally intensive for an SDPC to solve at any realistic network size and number of timesteps. With brute-force ruled out, we must turn to alternative methods to approximate the value of J more efficiently. Among these, two potential approaches are greedy and TD algorithms.

The greedy approach to estimating J only considers the immediate costs associated with transitioning to the next state. It can thus be said that for the cost G of state x and control u that $G_{greedy}(x,u) = g(x,u)$, where $g(x,u)$ represents the up-front cost associated with traveling from x to the next node dictated by u. For both our greedy and TD algorithms we define this $g(x,u)$ to prioritize one of our three target metrics: minimizing distance, minimizing wait, and maximizing revenue.

B. TD(λ)

The alternative approach we pursue is a variant of the TD algorithm called TD(λ) (algorithm 1) which looks beyond the immediate cost of the control, improving its own performance over time to better approximate the true cost function J. To do so, we consider a select number of features which are dependent on state x. For our research, these features are: the maximum wait time among all outstanding orders, the average wait time of all outstanding orders, the maximum price among the orders, the average price of the orders, and the average distance from the SDPC to every node. These are stored in a feature vector $\phi(x)$. If we assign these each of these features a designated weight r_i, we are then able to approximate J at any state as:

Algorithm 1 TD(λ) Algorithm

```
Require: $0 \le \alpha < 1$, $0 \le \lambda < 1$, $T > 0$
Input: State $x$, Step $t$
 1: Approximate $J$ as $\tilde{J}(x,r) = \sum_{i=1}^{i} r_i \cdot \phi_i(x)$
 2: Determine allowable controls $U(x)$
 3: for $i|_{i\in\mathbf{N}} \le \#U(x)$ do
 4:     cost $G(x,u_i) = g(x,u_i) + \alpha \cdot \tilde{J}(x_{u_i}, r_t)$
 5: end for
 6: $u \sim \sigma(x;T) = \frac{1}{Q} \cdot \begin{bmatrix} e^{-G(x,u_1)/T} \\ e^{-G(x,u_2)/T} \\ \vdots \\ e^{-G(x,u_n)/T} \end{bmatrix}$, where Q = $\sum_{i=1}^{n} e^{-G(x,u_i)/T}$
Output: Next state $x^+ := E(x)$ under control $u$
 7: Evaluate temporal difference $\delta_t = G(x,u) - \tilde{J}(x,r_t)$
 8: $z_t = \phi(x^+) + (\alpha \cdot \lambda) z_{t-1}$
 9: $r_{t+1} = r_t + \frac{1}{t} \cdot \delta_t \cdot z_t$
10: repeat at state $x^+$ until episode is complete
```

$$\tilde{J}(x,r) = \sum_{i=1}^{i} r_i \cdot \phi_i(x) \qquad (3)$$

From here, the algorithm begins the process of deciding its next control and updating weights r for the next timestep. For every element u of the set of allowable controls $U(x)$ at state x, the SDPC evaluates a cost $G(x,u_i)$. This cost is expressed as the sum of the immediate cost associated with implementing control u_i – much like we see in the greedy algorithm – but with the additional addend of the expected cost of the next state times a constant discount factor $\alpha \in [0,1)$. A control u is then drawn from the normalized Gibbs distribution of G, with tuning factor $T > 0$, and the next state x^+ is defined as the expected value of x after control u is implemented. Following this, the temporal difference δ_t is calculated as the difference between G and $\tilde{J}$. The vector z is then calculated as $z_t = \phi(x^+) + \alpha \cdot \lambda \cdot z_{t-1}$, λ being another value between 0 and 1. This finally results in the updated weights being evaluated as $r_{t+1} = r_t + \frac{1}{t} \cdot \delta_t \cdot z_t$. This algorithm is then repeated for every timestep until the episode is completed. Thus the TD algorithm is able to improve its performance for a designated metric over the duration of the episode by reactively recalculating the weights it assigns to each feature.

IV. IMPLEMENTATION AND DISCUSSION

Our model operates on a graph of size 5, not including the SDPC itself. At any given node, orders appear with probabilities ranging from 0.5% to 1.0% per timestep, dependent on the node. Order prices are drawn from a normal distribution with means ranging from 16 to 20, again dependent on the node (see table 1 for more detailed information). For the TD learning algorithms, we set $\alpha = 0.5$, $\lambda = 0.9$, and $T = 10$. Initial weights r all begin at 1, and each episode lasted 35,000 time increments.

For our implementation, we tested both a greedy algorithm and a TD algorithm that prioritized each of three system properties: minimizing selected distance, maximizing selected prices, and maximizing selected wait times. We then gathered efficiency data for these based on three primary metrics: total accrued revenue, distance traveled per order, and average wait time per order. Complete results are displayed in table 1.

On the metric of total revenue, the SDPC operating with the goal of maximizing price per order on a TD framework outperformed the rest. This is expected as that program is designed to accrue as much value as possible per order, while also being able to make a move with lower price so as to maximize future returns. Performing second-best is the greedy algorithm minimizing distance: as the SDPC here is spending less time travelling, it is able to gain more revenue through the sheer satisfying of more orders. Finally, the mild performance of the greedy max price algorithm can be explained by recognizing that always targeting the highest price order neglects the opportunity to satisfy more orders in the same time for a higher total income.

For the metric of average distance travelled, the SDPC performed best under the TD algorithm designed to minimize distance – again consistent with predictions that the TD algorithm would be more effective. This outperformed the other algorithms by a margin of 7 points.

The metric of average wait time presents a less intuitive result. While TD algorithms outperformed their greedy counterparts for all priorities, the worst performing program was that which seeks to satisfy the maximum wait using greedy techniques. The TD algorithm of the same kind performed marginally better, yet it still failed to outperform the other methods in a meaningful way. Given that the wait algorithms did not perform optimally for any other metric either, we are drawn to the conclusion that wait time is not an ideal metric for which to optimize an SDPC. The low wait associated with the TD min distance algorithm indicates that it satisfied orders effectively, in a brief amount of time.

Algorithm	Mean Distance	Total Revenue	Mean Wait
Greedy, Min Distance	**22.166**	10268.53	68.734
TD, Min Distance	**14.780**	8872.62	55.395
Greedy, Max Price	22.982	**9487.66**	68.236
TD, Max Price	20.540	**10635.93**	61.180
Greedy, Max Wait	22.046	8371.60	**73.255**
TD, Max Wait	21.281	8937.04	**67.463**

TABLE 1. Delivery efficiency performances for each of the control algorithms tested. For revenue, higher values denote better performance, while for distance and wait times lower values are considered better. Revenues are truncated to two decimal places to emulate real-world price values, and distances and wait times are truncated to three decimals for display convenience.

Ultimately, the consistently better performance of the TD algorithms compared to their greedy counterpart – outperforming them in 7 of 9 cases – indicates that they lend themselves well to optimizing delivery performances on a given network. With regards to prioritization of variables, no algorithm optimizes performance on all metrics, but the low wait time and low distance of the TD min distance algorithm indicate that it is the strongest for delivery efficiency, while the high revenue of the TD max price algorithm indicates its strength in maximizing profit.

V. CONCLUSION AND FUTURE WORK

In this paper, we presented a method with which to model a single-car autonomous delivery system, as well as numerous greedy and machine learning algorithms to optimize its performance. In running the simulation and measuring efficiencies for each algorithm, we showed that TD algorithms consistently outperformed greedy algorithms when prioritizing the same system property. In future work, we seek to expose this model to a more robust set of factors that would be seen in a real-world environment such as baking multiple orders at a time and guaranteed delivery times. Furthermore, we seek to expand the model to a fleet of vehicles and eventually determine a single control algorithm of optimal performance across all tested metrics.

ACKNOWLEDGEMENTS

This research was conducted under the UC Santa Barbara Department of Electrical and Computer Engineering. The authors would like to thank them and a number of individuals for their contributions to the research process: Dr. Lina Kim, our program director, provided a number of invaluable resources to guide our research, and both Dr. Michael Hughes and A S M Iftekhar, our TA, gave us specialized feedback to fine tune our research and its presentation. Finally, our lab mates Adit Shah and Siddharth Ganesan were valuable partners in navigating the research process. Without these people's support, this report would not have been possible.

REFERENCES

[1] “Pizza delivery: consumer spending US 2018,” Statista. [Online]. Available: https://www.statista.com/statistics/259168/pizza-delivery-consumer-spending-in-the-us/. [Accessed: 14-Jul-2019].

[2] P. Holley, “Domino’s will start delivering pizzas via an autonomous robot this fall,” Washington Post, 17-Jun-2019.

[3] “Order Delicious Pizza and Have it Delivered Zume Fast,” Zume Pizza, Order Online. [Online]. Available: https://zumepizza.com/. [Accessed: 14-Jul-2019].

[4] M. Han, P. Senellart, S. Bressan, and H. Wu, “Routing an Autonomous Taxi with Reinforcement Learning,” in Proceedings of the 25th ACM International on Conference on Information and Knowledge Management - CIKM ’16, Indianapolis, Indiana, USA, 2016, pp. 2421–2424.

[5] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie, “T-Finder: A Recommender System for Finding Passengers and Vacant Taxis,” IEEE Trans. Knowl. Data Eng., vol. 25, no. 10, pp. 2390–2403, Oct. 2013.

[6] M. Pavone, “19 Autonomous Mobility-on-Demand Systems for Future Urban Mobility,” p. 16.

[7] D. Bertsekas, Dynamic Programming and Optimal Control, 3rd ed., vol. 1. Athena Scientific, 2005.

[8] S. Shalev-Shwartz, S. Shammah, and A. Shashua, “Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving,” arXiv:1610.03295 [cs, stat], Oct. 2016.

[9] A. Sallab, M. Abdou, E. Perot, and S. Yogamani, “Deep Reinforcement Learning framework for Autonomous Driving,” Electronic Imaging, vol. 2017, no. 19, pp. 70–76, Jan. 2017.

[10] K. Treleaven, M. Pavone, and E. Frazzoli, “Asymptotically Optimal Algorithms for Pickup and Delivery Problems with Application to Large-Scale Transportation Systems,” p. 36, 2012.