Gamifying the Vehicle Routing Problem with Stochastic Requests

Nicholas D. Kullman
Université de Tours, nick.kullman@gmail.com

Nikita Dudorov
École Polytechnique, nikita.dudorov@polytechnique.edu

Martin Cousineau, Jorge E. Mendoza
Department of Operations Management and Logistics, HEC Montréal, martin.cousineau@hec.ca, jorge.mendoza@hec.ca

Justin C. Goodson
Department of Operations and Information Technology Management, Richard A. Chaifetz School of Business, Saint Louis University, justin.goodson@slu.edu

Abstract. Do you remember your first video game console? We remember ours. Decades ago, they provided hours of entertainment. Now, we have repurposed them to solve dynamic and stochastic optimization problems. With deep reinforcement learning methods posting superhuman performance on a wide range of Atari games, we consider the task of representing a classic logistics problem as a game. Then, we train agents to play it. We consider several game designs for the vehicle routing problem with stochastic requests. We show how various design features impact agents' performance, including perspective, field of view, and minimaps. With the right game design, general purpose Atari agents outperform optimization-based benchmarks, especially as problem size grows. Our work points to the representation of dynamic and stochastic optimization problems via games as a promising research direction.

Funding: This research was enabled in part by support from Calcul Québec, the Digital Research Alliance of Canada, HEC Montréal, and the Institute for Data Valorization (IVADO).

Key words: Gamification, Atari, Vehicle Routing, Dynamic and Stochastic Optimization, Deep Reinforcement Learning

1. Introduction

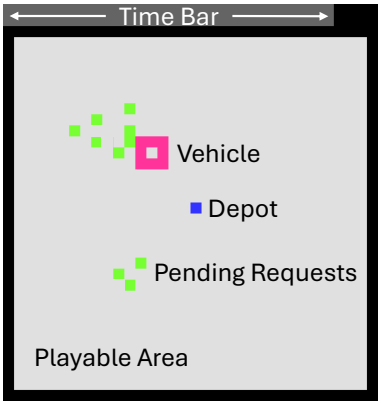
Several of the authors are old enough to remember their very own Atari 2600. A joystick with an orange button, a black game console with a few manual switches, and a cable that connected to a 1980s television. This magical device transported us to pixelated worlds where we spent many blissful hours playing classics like Pong, Centipede, and Frogger. Thirty-five years later, during a time when gamers connect their Xbox consoles to LCDs, our childhood memories are finding new life. With the original Atari games serving as benchmarks for deep reinforcement learning (DRL) methods, we set out to bring our past into our present. As adults, we spend our professional lives wading through our own pixelated world of integer variables, discrete state spaces, and discontinuous functions. Instead of proving theorems and devising algorithms to solve dynamic and stochastic optimization problems, what if we played them like a video game? Can we map one

pixelated world to another and leverage DRL to crack the code? This paper is about our quest to defeat one such problem, with only a vintage joystick and modern AI.

DRL methods have been applied to a variety of tasks involving sequential decisions and uncertainty. These tasks span the domains of healthcare (Liu et al. 2017), image recognition (Choi et al. 2018), and autonomous driving (Sallab et al. 2017), to name a few. In the operational realm, members of our team have applied DRL to dynamic taxi dispatching (Kullman et al. 2022), Amazon has used DRL for online bin packing, newsvendor, and vehicle routing problems (Balaji et al. 2019), and others have applied it to production scheduling (Palombarini and Martínez 2022, Xinquan and Xuefeng 2023). Perhaps the most well-known application is to games, where DRL has achieved superhuman performance in chess (Silver et al. 2018), Go (Silver et al. 2018), Doom (Lample and Chaplot 2017), Texas Hold'em Poker (Heinrich and Silver 2016), and StarCraft II (Vinyals et al. 2019). Notably, the architecture of Mnih et al. (2015) outperforms humans on the majority of 49 Atari games. This is achieved despite differences across the suite of games, such as appearance, goals, rewards, and actions. Indeed, it was the success of Mnih et al. (2015) that led to our reflection on whether similar DRL methods might perform comparably on any game with a related format. Games, like dynamic and stochastic optimization problems, are challenging because they involve long sequences of decisions in the face of uncertain outcomes. Thus, one also wonders if game-based representations of such problems, paired with DRL, can lead to viable solution methodologies. The possibility of bridging these two worlds motivates this paper.

To test our hypothesis, we consider how to gamify the *vehicle routing problem with stochastic requests* (VRPSR). In doing so, we take some poetic license. By gamify, we mean the literal representation of an optimization problem as a video game. The VRPSR is an important problem in modern logistics. It is the problem of dynamically routing a vehicle to service customer requests that occur at random times across an operating horizon and in random places within a service area. The objective is to identify a routing policy that maximizes the expected number of serviced requests. The VRPSR models a range of operational scenarios, including those faced by service technicians, meal delivery services, and couriers. Identifying an optimal VRPSR policy is challenging, especially for instances of any practical size. If the VRPSR can be represented as a game, and if DRL can provide a useful solution methodology, then this would be a notable achievement. Rather than relying on tailored optimization routines to aid online decision making, academics and practitioners alike could instead turn to a more general DRL method.

Figure 1 The VRPSR Game World.



We explore the benefits and drawbacks of various designs for the VRPSR game. Each design presents the player with a different view of the game world shown in Figure 1, and thus provides different input to the DRL algorithm. First, we consider perspective, whether to reposition the vehicle in response to joystick-style movements, or to fix the vehicle in the center of the view and reposition the playable area. Second, we investigate wide and narrow fields of vision, which determine whether the entire playable area is visible to the player at once, or only some portion of it. Third, we examine the impact of including a minimap in the view. When the player's field of vision is narrow, this feature provides the player with a downscaled overview of the entire playable area.

We train agents on each view via the DRL method of Bellemare et al. (2017), which builds on the success of Mnih et al. (2015), then execute them across various VRPSR problem instances. Computational experiments yield five conclusions. First, fixing the vehicle in the center of the view leads to better performance than moving the vehicle within a static view. The improvement follows from an agent's ability to associate each pixel in the view with movement in a particular direction, rather than with a function of the vehicle's location in a static view. Second, a narrow field of vision is helpful for larger playable areas. This feature shifts the problem of learning across a larger playable area from an issue of more complex input to one of additional exploration. Third, including a minimap in the view further improves agent performance. The two features in tandem combine local resolution across a narrow field of vision with a global approximation of the entire playable area. Fourth, in comparison to optimization-based benchmarks, DRL agents perform better as the expected number of requests increases. Because the benchmarks make computations on the fly, the requirement to make timely decisions limits their ability to select good actions. In contrast, following offline training, DRL agents require little time to make online decisions. Fifth, agents

trained on a particular type of problem instance may be adapted to other types of instances via *fine tuning*. A trained agent may adjust its architecture to new settings via a brief period of additional training on the latest instances. This makes agents portable, in the sense that extensive offline training need not be repeated.

Though some of our game designs allow agents to outperform benchmark policies, our aim is not to develop a state-of-the-art procedure. Rather, our contribution is a connection between the seemingly disparate worlds of video games and logistics. More generally, our work points to the representation of dynamic and stochastic optimization problems via games as a promising research direction.

The paper proceeds as follows. In §2, we review related literature. In §3, we detail the game setup and describe four game views. In §4, we describe the DRL training procedure. In §5, we present benchmark methods. In §6, we conduct computational experiments and discuss their implications. We conclude the paper in §7.

2. Related Literature

It seems fitting that the start of VRPSR research coincides with the release of the Atari 2600. The work of Psaraftis (1980) reoptimizes a route through pending requests whenever a new request is made, then uses the route to direct vehicle movement. Just like the Atari influenced video game design for years to come, Psaraftis inspired similar research across several decades. Gendreau et al. (1999), Ichoua et al. (2000), van Hemert and La Poutré (2004), Gendreau et al. (2006), Ichoua et al. (2006), Branchini et al. (2009), and Ferrucci et al. (2013) all present variations on the original idea of Psaraftis. Innovations center on more advanced routing heuristics. While these methods exploit advances in deterministic routing, for the most part they do not explicitly consider future requests.

More anticipatory VRPSR decision making begins with Bent and Van Hentenryck (2004). Rather than direct the vehicle based on reoptimization of a single route, Bent and Van Hentenryck reoptimize a collection of routes, each of which contains a different random sample of future requests. A consensus function then determines vehicle movement. Hvattum et al. (2006) and Ghiani et al. (2009) proceed similarly. Concurrently, Mitrović-Minić and Laporte (2004), Branke et al. (2005), Thomas and White III (2007), and Ghiani et al. (2012) explore waiting strategies. These methods dynamically move and halt the vehicle in anticipation of future requests. Archetti et al. (2020) revisit the reoptimization approach, but explicitly incorporate uncertainty surrounding future requests into decision making.

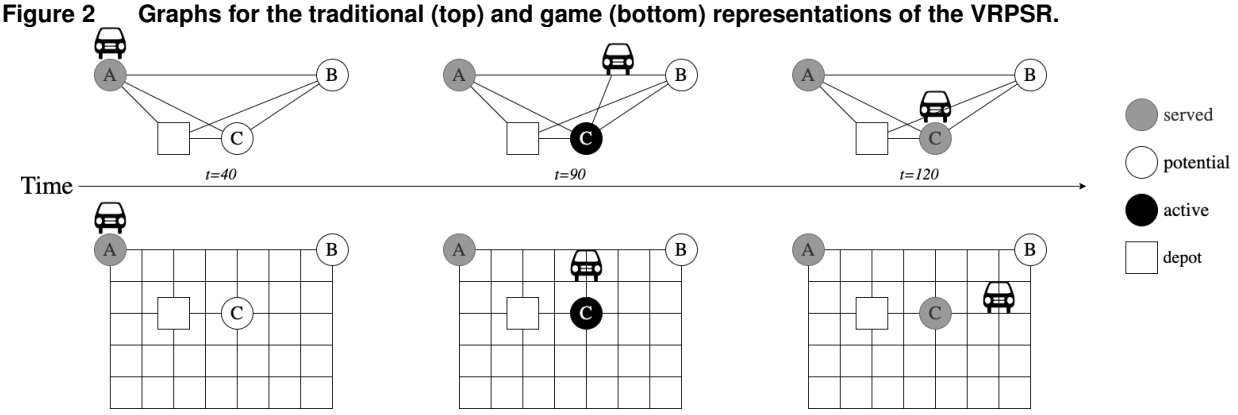
The work of Meisel (2011) begins an era of VRPSR value function approximation, which gives explicit consideration to the timing and locations of future requests. Building on Meisel (2011), Ulmer et al. (2018a) aggregate states around time-based features, then dynamically partition the space as part of an approximate value iteration procedure. Ulmer et al. (2018b) extend these ideas to a multi-period VRPSR and Ulmer et al. (2018c) combine them with rollout algorithms. Even though the method of Ulmer et al. (2018a) adapts to the approximation process, it cannot revert partitioning decisions once made. The adaptive procedure of Soeffker and Ulmer (2019) remedies this issue and yields even better value function approximations. Caspar and Wendt (2024) employ value iteration to exactly solve small problem instances and temporal difference learning to heuristically address larger instances.

More recently, VRPSR value function approximations are made via neural networks. Joe and Lau (2020) show that neural nets can outperform the scenario-based method of Bent and Van Hentenryck (2004) as well as an approximate value iteration procedure. In the context of same-day delivery, Chen et al. (2022) use neural networks to learn the value of fulfilling a request via drone versus via truck. Our approach also relies on neural network approximations of the value function. However, in contrast to the literature, we do not design a value function approximation for the VRPSR. Rather, we seek to design a game-based representation of the VRPSR for a preexisting network architecture. For a more comprehensive examination of dynamic and stochastic vehicle routing literature, see Soeffker et al. (2022).

3. Modeling the VRPSR as a Game

The vehicle routing problem with stochastic requests (VRPSR) dispatches a single vehicle to meet customer requests arriving at random times across a given operating horizon and at random locations across a known service area. The objective is to design a dynamic routing policy, beginning and ending at a depot, that maximizes the expected number of serviced customers. In the taxonomy of Zhang and Woensel (2023), the VRPSR is a dynamic and stochastic vehicle routing problem with random service requests. The *game world*, portrayed in Figure 1, is a visual representation of the problem description. It is composed of several elements. The *playable area* represents the service area, or the portion of the game world within which the vehicle may move. Within the playable area, the depot and requests are represented by single pixels. Customers that have requested service are green and the depot is blue. The pixel depicting each request is invisible before the time of request and after service. The vehicle's location is shown by the open pixel in the center of the

Figure 2 Graphs for the traditional (top) and game (bottom) representations of the VRPSR.



pink square. The vehicle services a request by navigating its open pixel to a customer's position. The playable area is surrounded by a thin border. Above the border is a rectangular *time bar* whose length represents the remaining time before the vehicle must return to the depot.

The VRPSR is conventionally formulated as a Markov decision process: epochs occur when customers are serviced and when customers request service; the state tracks time, vehicle position, and the locations of pending requests; actions direct the vehicle to a pending service request or to wait at the current location; rewards count the number of serviced customers; and transitions account for the likelihood of new requests across space and time. Formal models for related problems may be found in Ulmer et al. (2018c) and Ulmer et al. (2020).

Modeling the VRPSR as a game requires modifications to how policies route the vehicle and to the conventional formulation. The operations research community typically considers policies that move the vehicle among the edges of a complete graph G consisting of nodes for each customer and the depot. To accommodate the pixel-by-pixel movements typical to games, routing policies in the VRPSR game move the vehicle along the edges of a graph G' representing the playable area. Each node in G' is a pixel and edges connect a pixel to adjacent pixels, e.g., the pixels above, below, and to either side. While routing across G allows for a variety of distance metrics (e.g., Euclidian), routing across G' requires Manhattan-style movements. Consequently, the gamified formulation of the VRPSR is a good approximation of vehicle movement in an urban area or a warehouse, but not through a circuitous road network. Figure 2 depicts the VRPSR on both graphs.

A game formulation of the VRPSR modifies actions, epochs, and states. The action space consists of at most five actions: move up, move down, move left, move right, and no movement. If the vehicle is at the boundary of the playable area, the number of feasible actions is fewer. In the conventional formulation, the path between any pair of nodes in G incurs the minimum travel time.

In contrast, moving from one customer to another in G' can be accomplished via many paths. For preemptive routing policies, movement across G' offers more flexibility than movement across G. For example, though detouring the vehicle away from a shortest path may result in more travel time, it may also put the vehicle in closer proximity to potential requests, thus making it possible to service additional requests while en route to a customer. Figure 2 depicts this behavior. In the top part of the figure, the vehicle moves directly from node A to node B through graph G. In the bottom portion, movement toward node B through graph G' anticipates the possibility of a request at node C by moving toward B along a lower and longer path. Epochs reflect the additional routing flexibility afforded by movement across G'. They occur at each moment of a discretized timeline.

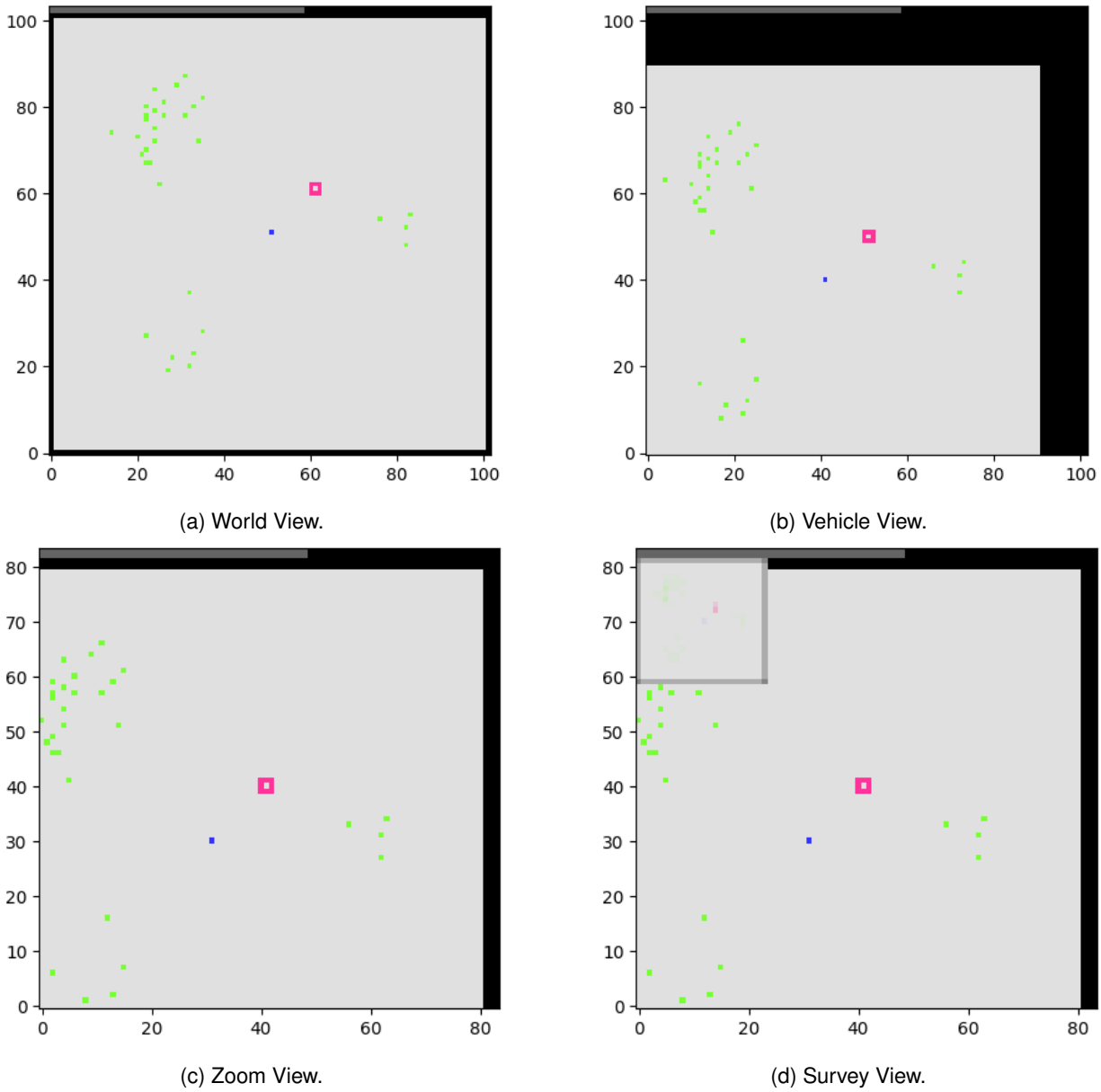
The state of a VRPSR game is the player's *view* of the game world at an epoch. A view consists of the time bar plus the player's *field of vision*. The player's field of vision is the visible portion of the playable area. We consider four views, each of which is depicted in Figure 3. In the *World View*, the field of vision includes the entire playable area and the player simply moves the vehicle within the area. In the *Vehicle View*, the vehicle is fixed in the center of the field of vision and movement adjusts the position of the field of vision across the playable area. When the field of vision is positioned away from the center of the playable area, portions of the playable area move out of the view. The *Zoom View* is the Vehicle View with a narrower field of vision, meaning portions of the playable area are obscured even when the vehicle is positioned in the center of the playable area. To observe portions of the playable area outside the field of vision, the vehicle must move to those regions. The *Survey View* is the Zoom View plus a minimap. In this view, players see a small overview of the playable area in the top left corner. The overview is an aggregation, where the color of each aggregated pixel is the average color of the pixels it represents.

Because each view is a different state representation, the four views constitute four different VRPSR games. While the World View provides a complete observation of the state variable, the latter three views result in partial observations.

4. Agent Architecture

We use the *categorical Deep Q-Network* (DQN) approach of Bellemare et al. (2017) to train agents for all four views of the VRPSR game. This method has demonstrated particular success across a suite of Atari 2600 games. In contrast to classical reinforcement learning techniques, which estimate expected Q-values, categorical DQN approximates Q-value distributions. The method builds on the celebrated DQN architecture of Mnih et al. (2015), which employs a collection of neural network

Figure 3 Game views for a 100-by-100 pixel playable area.

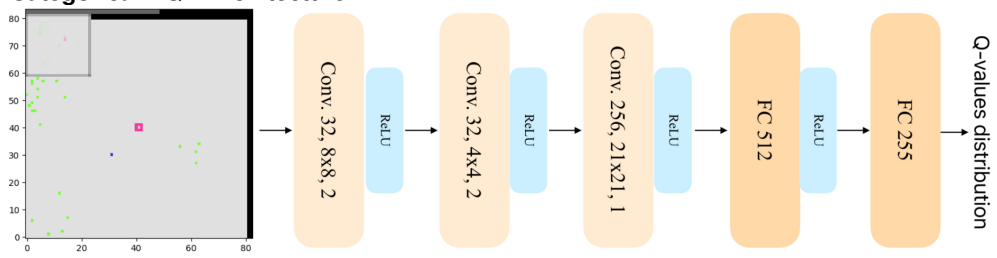


Note. Zoom and Survey Views are depicted with an 84-by-84 pixel field of vision.

techniques. As in Mnih et al. (2015), categorical DQN takes the classical ϵ-greedy approach to learning. However, instead of minimizing squared loss between Q-values, it minimizes sample loss between distributions of Q-values.

Our network architecture is shown in Figure 4. It takes as input a game view, represented as an array of pixels. It consists of three convolution layers and two fully connected layers. The architecture mirrors that of Bellemare et al. (2017), with one modification. Because each customer occupies only a single pixel, a customer's exact location may become lost in the convolution layers.

Figure 4 **Categorical DQN Architecture.**



To prevent this, in the first convolution layer we double the number of filters and cut the stride in half. After an agent is trained, Q-value distributions are used to make decisions by selecting an action that maximizes the expected Q-value in a given state.

5. Benchmarks

We benchmark agents' performance against two reoptimization policies, two nearest neighbor policies, and the expected value with perfect information (EVPI), which serves as a dual bound on the value of an optimal policy (Brown et al. 2010). At a given state, the reoptimization policies select an action by solving a mixed-integer linear program (MILP). The MILP seeks a route that serves a maximum number of pending service requests. The action is movement of the vehicle toward the first customer visited by the route. We refer to these policies as reoptimization with preemption (Reopt-WP) and reoptimization without preemption (Reopt-WOP). The Reopt-WP policy solves the MILP when customers are serviced and whenever service requests are realized. Thus, a vehicle en route to one customer can be diverted to a different customer. The Reopt-WOP policy only solves the MILP when customers are serviced, as is customary in the VRPSR literature. This policy commits the vehicle to the selected request. Reopt-WP and Reopt-WOP may be viewed as players of the VRPSR game with a World View. As discussed in §2, solving deterministic MILPs in rolling horizon fashion to make dynamic decisions is a common means of deriving policies from static methods. The policies provide a sense of how classical optimization-based methods perform relative to a state-of-the-art reinforcement learning method. As we discuss below, the same MILP can be used to obtain the EVPI.

To formalize the MILP, let $\bar{t}$ be the current time and let node n in G' be the location of the vehicle at time $\bar{t}$. Denote by $C \subset G'$ the subset of nodes in G' corresponding to customers who have requested service but who have not yet been visited. Let 0 in G' represent the depot and denote by $\bar{C} = C \cup \{n\} \cup \{0\}$ the set of nodes comprising customer locations, vehicle location, and the depot. For each pair of nodes (i, j) in $\bar{C} \times \bar{C}$, let d_{ij} be the duration of the shortest travel time from i to j,

where distances are Manhattan. Denote by r_i the time of the request at node i in C. Let $l_i = T - d_{i0}$ be the latest time the request at node i in C may be serviced such that the vehicle can return to the depot by the end of the operating horizon at time T.

Decision variables include which customers to visit and when. Let h_i be 1 if the request at node i is serviced by the route and 0 otherwise. Let y_i be 1 if the request at node i is the first request to be serviced and 0 otherwise. Let x_{ij} be 1 if the request at node j is serviced immediately after the request at node i and 0 otherwise. Finally, let t_i be the time at which the request at node i is serviced, if it is serviced at all. Constraints require the vehicle to begin routing from its current location, to visit customers at or after the time at which service is requested, to visit each customer at most once, and to conclude service with enough time to return to the depot before the end of the operating horizon.

The MILP is formulated as follows:

$$\text{maximize} \quad \sum_{i \in C} h_i \tag{1}$$

$$\text{subject to} \quad \sum_{i \in C} y_i \leq 1 \tag{2}$$

$$y_j + \sum_{i \in C} x_{ij} = h_j, \quad j \in C \tag{3}$$

$$\sum_{j \in C} x_{ij} \leq h_i, \quad i \in C \tag{4}$$

$$t_i \geq r_i + (\bar{t} + d_{ni} - r_i)\, y_i, \quad i \in C \tag{5}$$

$$t_j - t_i \geq r_j - l_i + \left(d_{ij} - r_j + l_i\right) x_{ij}, \quad (i, j) \in C \times C \tag{6}$$

$$r_i \leq t_i \leq l_i, \quad i \in C \tag{7}$$

$$h_i \in \{0, 1\}, \quad i \in C \tag{8}$$

$$y_i \in \{0, 1\}, \quad i \in C \tag{9}$$

$$x_{ij} \in \{0, 1\}, \quad (i, j) \in C \times C \tag{10}$$

The objective in Equation (1) seeks to maximize the number of serviced requests. Constraint (2) allows at most one request to be routed first. Constraints (3) allow each node j in C to be routed only if j is selected for service. Constraints (4) allow departure from node i in C only if node i is marked for service. If the request at node i in C is routed first, then Constraints (5) require the

time of service at node i to be at or after the current time plus the travel time from the vehicle's current location to node i. If the request at node j in C is routed after the request at node i in C, then Constraints (6) require the time of service at node j to be at or after the time of service at node i plus the travel time from node i to node j. Constraints (7) ensure that service at each node i in C occurs between r_i and l_i. Constraints (8) and (9) mark decision variables h_i and y_i as binary for each i in C. Similarly, Constraints (10) require x_{ij} to be binary for each pair of nodes in $C \times C$.

The Reopt-WP and Reopt-WOP policies extract actions from feasible MILP solutions as follows. Let $i^\star$ be the node in C such that $y_{i^\star} = 1$. Node $i^\star$ is the first request serviced by the solution. Both Reopt-WP and Reopt-WOP advance the vehicle toward $i^\star$ on a path that first moves horizontally and then moves vertically. If $i^\star$ does not exist and $\bar{t} + d_{n0} < T$, then the action is to wait at n, the vehicle's current location. Otherwise, the action moves the vehicle along a shortest path to the depot.

The nearest neighbor policies move the vehicle to the closest pending service request. As with the reoptimization policies, we consider actions that do (NN-WP) and do not (NN-WOP) permit preemption.

The EVPI is the expected value of an optimal policy with perfect foresight of the times and locations of service requests. If we represent a realization of requests by C and their times by $r = (r_i)_{i \in C}$, then the MILP models the problem of identifying an optimal routing in response to C and r. Denote by $f(C, r)$ the value of this routing. The EVPI is the expected value $\mathbb{E}[f(C, r)]$ across all possible realizations.

6. Computational Experience

In this section, we conduct four sets of experiments designed to demonstrate the strengths and weaknesses of agents trained on each of the four game views. Problem instances and algorithmic parameters are described in § 6.1. Results are presented in § 6.2.

6.1. Problem Instances

Across all experiments, instances are structured similar to those of Ulmer et al. (2018c). Instances for the first set of experiments are characterized by a 20 km square service region, a 6-hour operating horizon, and 30 expected requests. A 100-by-100 pixel playable area represents the service region, where each pixel is 0.2 km square. Time is discretized into 750 steps, each with a duration of 0.008 hours. This leads to 749 epochs plus a final time step that concludes service. The vehicle moves at a constant speed of 25 km/h, or one pixel per time step. The depot is located in the center of the playable area. The number and location of requests are independent random variables.

The number of customers that request service in a given time step is Poisson distributed with rate $30/749$, resulting in 30 expected requests across the operating horizon. Each request is located in one of three areas with likelihoods $1/4$, $1/2$, and $1/4$, respectively. Request locations within each area follow a bivariate normal distribution. Referencing the bottom-left corner of the playable area as the origin, mean locations for the areas are $(5\text{ km}, 5\text{ km})$, $(5\text{ km}, 15\text{ km})$, and $(15\text{ km}, 10\text{ km})$, respectively. Standard deviations for both horizontal and vertical dimensions are 1 km and the correlation between dimensions is 0.

The second set of experiments modifies instances for the first set by increasing the resolution of the playable area from 100-by-100 pixels to 200-by-200 pixels. With the area of each pixel at 0.1 km square, time is discretized into 1500 steps, each with a duration of 0.004 hours. The Poisson rate parameter is adjusted accordingly to $30/1499$. The third set of experiments alters instances from the first set by increasing the expected number of customers from 30 to 100. All other parameters remain the same.

The fourth set of experiments consists of four parts, labeled 4a, 4b, 4c, and 4d. Instances in each part modify the instances in the third set of experiments. In part 4a, the number of expected requests is 150. In part 4b, the likelihoods of request location are uniform, or $1/3$ across each of the three areas. In part 4c, requests are not received from the first area, and the likelihoods of request location are $2/3$ and $1/3$ for the second and third areas, respectively. In part 4d, requests are not received from the third area. Instead, they are received from the first area, the second area, and two additional areas. The additional areas are located at $(15\text{km}, 5\text{km})$ and $(15\text{km}, 15\text{km})$. Likelihoods of request location are again uniform, or $1/4$ across each of the four areas.

We use categorical DQN to train agents on VRPSR games via the World, Vehicle, Zoom, and Survey Views. We refer to the agents by the views on which they are trained. The field of vision for the Zoom and Survey Views is 84-by-84 pixels, the same resolution as the images used by Mnih et al. (2015). To accelerate exploration, each training episode begins by moving the vehicle to a random location. In the first and third sets of experiments, each agent is trained for 50 million epochs, with episodes terminating whenever the remaining time is such that the agent's only feasible moves are those bringing it back to the depot. In the second set of experiments, the number of training epochs increases to 180 million. In the fourth set of experiments, agents are trained twice for each part. An initial training of 50 million epochs is conducted on problem instances associated with the third set of experiments. A second training, which we call *fine tuning*, exposes the agent to part-specific problem instances for an additional 10 million epochs. We use RLlib (2024) to implement

Table 1 Ray RLlib Hyperparameter Values.

Parameter	Value
num_atoms	51
replay_buffer_config: capacity	1,000,000
target_network_update_freq	8,000
lr	0.0000625
adam_epsilon	0.00015
hiddens	[512]
train_batch_size	32
exploration_config: epsilon_timesteps	200,000
exploration_config: final_epsilon	0.01

categorical DQN. It provides the pre-tuned hyperparameters for Atari displayed in Table 1. Training is conducted in serial on Nvidia V100 GPUs.

The expected values of agents' decisions and of each benchmark policy are estimated via simulation. We do this by randomly generating request realizations across the operating horizon and service area, executing the policy, recording the number of serviced requests, then repeating a total of 250 times. The average number of serviced requests is an unbiased and consistent estimator of the expected number of serviced requests. For each set of request realizations, the Reopt-WOP policy is allocated a computing budget equal to the 6-hour operating horizon. In the first and second sets of experiments, this provides up to 12 minutes to solve the MILP for each of 30 expected requests. In the third and fourth sets of experiments, Reopt-WOP receives up to 3.6 minutes to solve the MILP for each of 100 expected requests. The Reopt-WP policy receives the same allocations for each MILP. If a MILP is not solved to optimality within the allocated time, then the best feasible solution is used to direct action selection. The EVPI is estimated via simulation across the same 250 instances. For each realization of requests, each MILP instance is allocated 6 hours of computing time. If the solver does not identify an optimal solution to the MILP within that time, then we use the solver's smallest upper bound instead. Though this may overestimate the EVPI, it is also a dual bound on the value of an optimal policy. The EVPI is only estimated for experiments with 30 expected requests. Tractability issues prevent estimation of meaningful bounds for larger instances. All MILPs are solved with Gurobi Optimizer on Intel Xeon Gold 6148 (2.4 GHZ) CPUs. The experimental setup is summarized in Table 2.

6.2. Results and Discussion

The first set of experiments highlights the benefit of a Vehicle View. Figures 5 and 6 depict the results. Figure 5 displays the performance of each agent and of the benchmarks while Figure 6 shows the performance of each agent as a function of the number of training epochs. The expected

Table 2 Experimental Setup.

Experiment Set	**1**	**2**	**3**	**4a**	**4b**	**4c**	**4d**
Expected Requests	30	30	100	150	100	100	100
Probability (5 km, 5 km)	1/4	1/4	1/4	1/4	1/3	–	1/4
Probability (5 km, 15 km)	1/2	1/2	1/2	1/2	1/3	2/3	1/4
Probability (15 km, 10 km)	1/4	1/4	1/4	1/4	1/3	1/3	–
Probability (15 km, 5 km)	–	–	–	–	–	–	1/4
Probability (15 km, 15 km)	–	–	–	–	–	–	1/4
Playable Area (px^2)	100	200	100	100	100	100	100
World/Vehicle Views Field of Vision (px^2)	100	200	100	100	100	100	100
Zoom/Survey Views Field of Vision (px^2)	84	84	84	84	84	84	84
Reopt-WOP/Reopt-WP CPU minutes per MILP	12	12	3.6	3.6	3.6	3.6	3.6
Training Iterations per Agent (millions)	50	180	50	50	50	50	50
Fine Tuning Iterations per Agent (millions)	–	–	–	10	10	10	10

number of serviced requests by the World Agent (3.4) is particularly low. The Vehicle (21.9), Zoom (21.8), and Survey Agents (22.3) each demonstrate substantial improvements over the World Agent and also outperform the Reopt-WOP policy (21.0). While these three agents perform similarly, the Survey Agent's zoom and minimap features together provide an edge over the Vehicle View in isolation. Figure 6 underscores these results. While all four agents improve as the number of training epochs approaches 50 million, the World Agent lags behind the other three, and the two agents trained on the zoom feature surpass the performance of the Vehicle Agent. Although better performance is posted by the Reopt-WP (23.7), NN-WOP (23.1), and NN-WP (23.5) policies, the Survey Agent is not far behind. The EVPI (27.7) sits between the expected number of requests serviced by Reopt-WP and 30, the expected number of total requests. Of the 250 instances used to calculate the EVPI, nearly 25 percent solve to optimality within the allotted time. Across the vast majority of the remaining instances, the solver reports optimality gaps below 20 percent.

The performance disparity between the World Agent and the Vehicle Agent points to a Vehicle View as an important factor in modeling the VRPSR as a game. Vehicle Views allow agents to associate each pixel in the view with specific actions, e.g., pixels in the top half of the view require upward movement and pixels on the right side of the view require movement to the right. In contrast, the action required to move the vehicle to the same pixel in the World View is a function of the vehicle's location in the view. The difference is one of gauging value relative to the view versus value relative to the vehicle's position in the view. It seems that dependency on vehicle location introduces non-trivial complexity. While it is straightforward for humans to recognize the connection between the World View and the Vehicle Views, to the deep Q-networks that drive the categorical DQN method, Vehicle Views are more amenable to learning.

Figure 5 **Performance in the First Set of Experiments.**

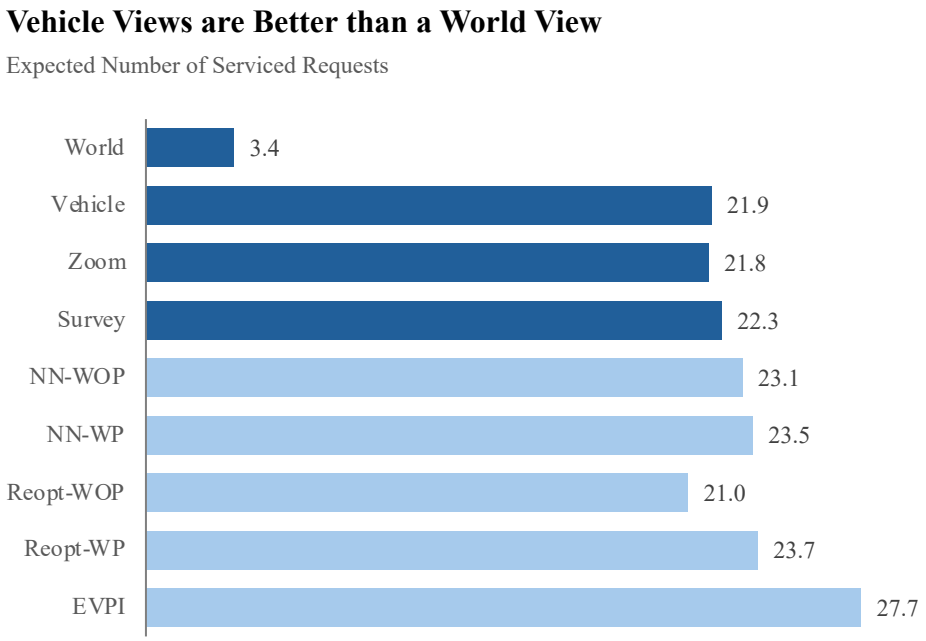
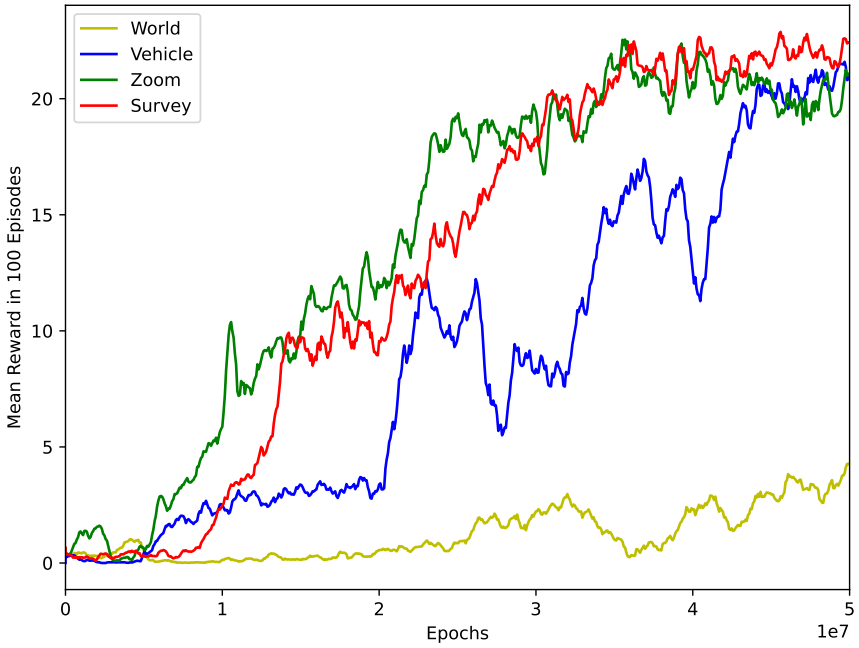


Figure 6 **Training Curves for the First Set of Experiments.**



The second set of experiments showcases the benefit of Zoom Views and confirms the utility of the Survey View. Figures 7 and 8 depict the results. The format of these figures mirrors that of Figures 5 and 6. The World Agent is excluded from these experiments. Because the increased resolution does not affect the benchmarks, their performance values are not displayed. Relative to the first set of experiments, the expected number of serviced requests decreases for each agent.

Figure 7 **Performance in the Second Set of Experiments.**

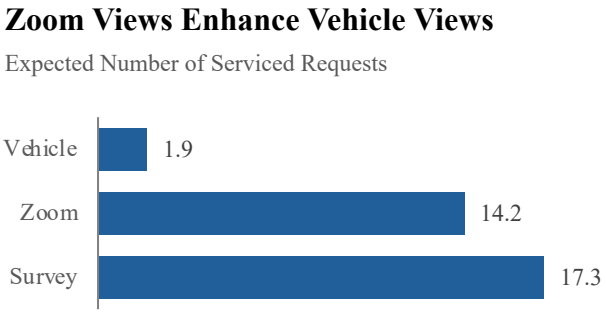
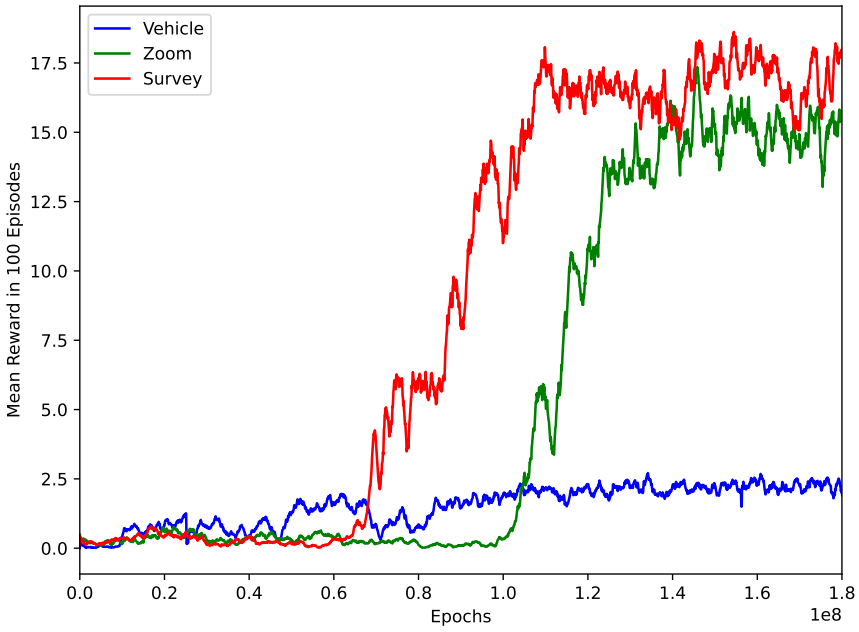


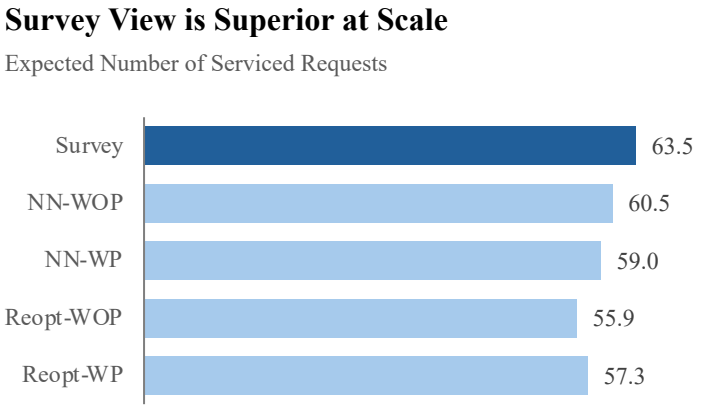
Figure 8 **Training Curves for the Second Set of Experiments.**



However, the Zoom (14.2) and Survey Agents (17.3) both improve substantially over the Vehicle Agent (1.9). The training curves in Figure 8 emphasize this difference. Across 180 million training epochs, the Vehicle Agent demonstrates only minor performance gains whereas the other two agents exhibit marked improvement.

Theses results indicate that as view resolution increases, performance of the agents decreases. For the Vehicle Agent, quadrupling the number of pixels from 100-by-100 to 200-by-200 quadruples the size of the input array for the deep Q-networks that underlie the categorical DQN method. As demonstrated by the training curves in Figure 8, this is a more difficult learning task. The zoom feature included in the Zoom and Survey Agents partially offsets the challenges of a larger playable area by maintaining the size of the view at 84-by-84 pixels. This shifts the problem of learning across a larger playable area from an issue of more complex input to one of additional spatial

Figure 9 Performance in the Third Set of Experiments.



exploration. In the first set of experiments, this difference is not evident because the Zoom View captures more than 70 percent of the playable area. In the second set of experiments, the Zoom View captures four times less, not even 18 percent, and leads to substantially better performance.

In both the first and second sets of experiments, the performance of the Survey Agent exceeds that of the other agents. This points to the minimap feature as an important part of VRPSR game design. In contrast to the zoom feature, which provides high resolution locally, the minimap feature offers a global approximation of the entire playable area. Combining these features into one view provides the Survey Agent with detail to facilitate exploitation and with a wide field of vision to promote exploration.

The contrast to conventional optimization-based approaches is also notable. While the performance of our agents depends heavily on the granularity of the playable area, methods like NN-WOP, NN-WP, Reopt-WOP, and Reopt-WP do not. Although agent performance is sensitive to view resolution, as we show next, it is robust to increases in problem size.

The third set of experiments spotlights advantages of DRL when the problem size grows. These experiments examine the performance of the Survey Agent alongside the benchmarks when the expected number of requests more than triples from 30 to 100. Figure 9 depicts the results. In contrast to the first and second sets of experiments, the expected number of requests serviced by the Survey Agent (63.5) exceeds those of NN-WOP (60.5), NN-WP (59.0), Reopt-WOP (55.9), and Reopt-WP (57.3).

Increasing the problem size highlights an important difference between the Survey Agent and the reoptimization policies. Because a larger number of expected requests leads to appreciably larger MILPs, the Reopt-WOP and Reopt-WP policies struggle to select good actions within the allotted computing time of 3.6 minutes per MILP. Significant increases in computing time may

lead to better policy performance, but this is not practical. The Reopt-WOP and Reopt-WP policies operate online. Because operational limitations typically require decisions to be made in a matter of minutes, this precludes the possibility of allocating more computing time to solving MILPs on the fly. In contrast, the Survey Agent may be trained offline for large periods of time without negative service consequences. Then, when it is deployed, actions are selected in a matter of seconds. Thus, as problem size grows, DRL yields better performance in typical operating conditions.

Despite the fast execution typically enjoyed by DRL methods, a common concern is whether an agent can perform well on instances that differ from the instances on which it is trained. If not, then a new agent must be trained, and the lengthy offline training period may offset the benefit of fast online execution. The fourth set of experiments explores this. In each part, the Survey Agent is trained on the instances designed for the third set of experiments. Then, it is executed on instances with more requests (4a), different request location likelihoods (4b), fewer request areas (4c), and more request areas (4d). We also explore whether fine tuning can improve the Survey Agent's performance. At only one-fifth the number of training iterations, fine tuning requires much less time than training a new agent.

Figures 10–13 show the results of each part of experiment four. They display the performance of the Survey Agent, with and without fine tuning, alongside the benchmark policies. In parts 4a, 4b, and 4c, the Survey Agent without fine tuning performs well relative to the benchmark policies. Fine tuning nudges performance higher by 1 percent in part 4a, 3.1 percent in part 4b, and 4 percent in part 4c. This performance boost is enough to put the Survey Agent at the top of the pack in each part. The instances comprising part 4d are a drastic departure from the instances that make up the third set of experiments. Not only is one request area removed, but two new request areas are added. Thus, it is not surprising that the Survey Agent without fine tuning performs poorly relative to the benchmark policies. Its initial training is conducted on instances that are very different. In this case, fine tuning provides more than a bump in performance improvement. It raises the expected requests served by 51.5 percent, which is enough to best the benchmark policies. Across the board, fine tuning counteracts differences between training instances and instances the Survey Agent sees in operation. Thus, complete retraining is unnecessary. With only a fraction of the training epochs, fine tuning yields superior performance.

7. Conclusion

A pixelated "Game Over" was the message that flashed across our CRT televisions when we ran out of lives. We would start again, play until the next "Game Over," and repeat until our mothers

Figure 10 **Performance in Experiment Set 4a.**

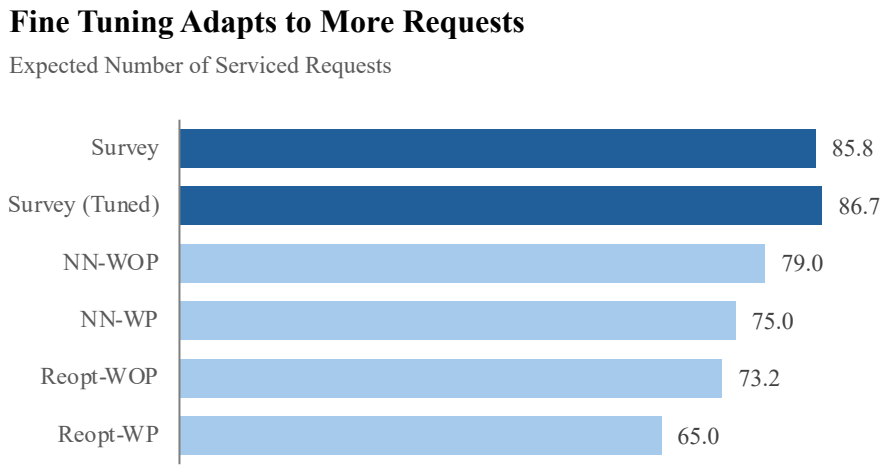


Figure 11 **Performance in Experiment Set 4b.**

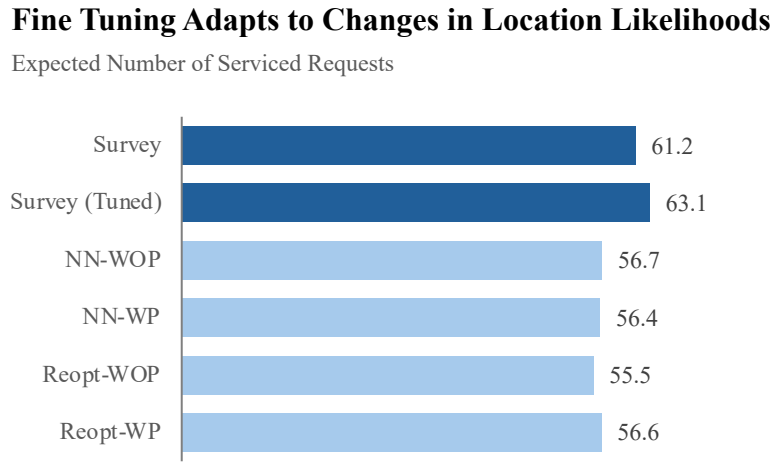


Figure 12 **Performance in Experiment Set 4c.**

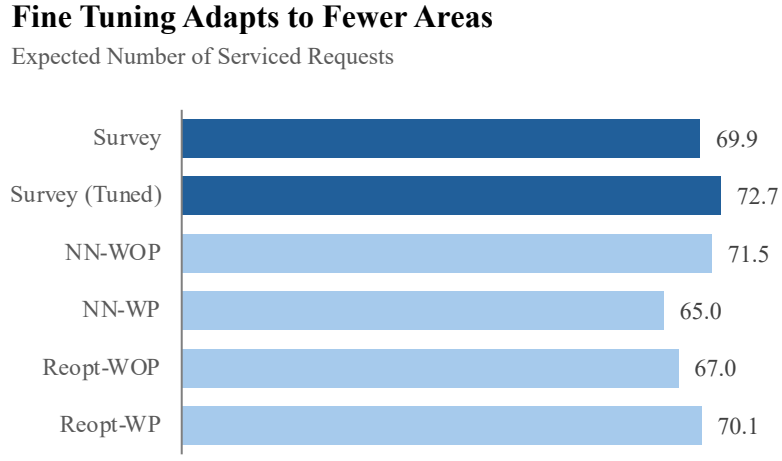
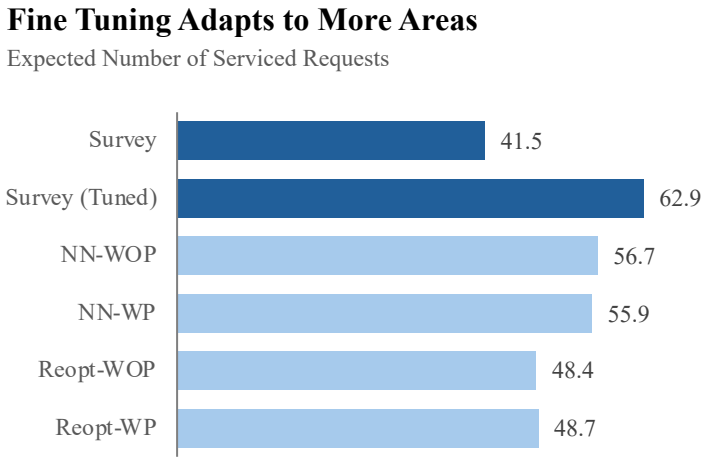


Figure 13 Performance in Experiment Set 4d.



told us our eyeballs would turn into squares if we played any longer. The same "Game Over" also signaled the conclusion of a game. It marked completion of every level. Sometimes this unlocked new content, and sometimes we loaded a new cartridge into the console. In any case, as long as we kept playing, the game was never really over.

Game worlds, playable areas, zoom views, minimaps, deep-Q networks. These things are not the end. This is only level one. New quests await us. Games, like optimization problems, have always been about discovery of a winning policy. Whether that happens through theorems and proofs, or by sitting an agent in front of a screen through millions of "Game Overs," the goal is the same. This paper shows that it is possible to represent the VRPSR as a game and to train a neural network to play it better than some optimization algorithms. It teases out aspects of the VRPSR game design that lead to strong performance. Just like certain optimization models pair well with certain methods—e.g., a set partitioning model and column generation—we see that some game views are more amenable to DRL than others.

To keep going down this road, we must find new connections between games and optimization problems. Sequentiality is likely to be an essential element. Even in the absence of stochasticity, the ability to iteratively construct a solution seems like a critical connection between a conventional optimization model and a gamified model. For example, the classical knapsack problem may be formulated as an integer program that selects items all at once or as a dynamic program that packs items one by one. Because the ability to choose items in sequence bears a stronger resemblance to gameplay, the dynamic program formulation may be a better starting point for gamification.

A visual depiction of the problem, one that changes in response to decisions and uncertainty, also seems essential. Such a representation is often natural for optimization problems rooted in real

applications. For example, scheduling, production, assortment, and vehicle routing problems naturally lend themselves to visualizations. Abstract problems, on the other hand, may pose challenges. For instance, it is not immediately clear how to best visualize vectors of decision variables and objective function coefficients that do not represent something recognizable to the eye.

When we started this research, it felt a bit unusual. Our university coursework covered methods to solve equations and construct algorithms. Game design wasn't part of the curriculum. In hindsight, however, joining our pixelated past with our pixelated present seems natural. A joystick and a game console are just new ways to think about objectives, constraints, and decision variables. Where we go from here depends on how you want to play.

Acknowledgments

This research was enabled in part by support from Calcul Québec (`calculquebec.ca`), the Digital Research Alliance of Canada (`alliancecan.ca`), HEC Montréal, and the Institute for Data Valorization (IVADO). The authors especially thank Clément Grodecoeur for his efforts in the development and prototyping of an early VRPSR game.

References

Archetti C, Feillet D, Mor A, Speranza M (2020) Dynamic traveling salesman problem with stochastic release dates. *European Journal of Operational Research* 280(3):832–844.

Balaji B, Bell-Masterson J, Bilgin E, Damianou A, Moreno Garcia P, Jain A, Luo R, Maggiar A, Narayanaswamy B, Ye C (2019) Orl: Reinforcement learning benchmarks for online stochastic optimization problems. *arXiv preprint arXiv:1911.10641* .

Bellemare M, Dabney W, Munos R (2017) A distributional perspective on reinforcement learning. *Proceedings of the 34th International Conference on Machine Learning* 70:449–458.

Bent R, Van Hentenryck P (2004) Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research* 52(6):977–987.

Branchini R, Armentano A, Løkketangen A (2009) Adaptive granular local search heuristic for a dynamic vehicle routing problem. *Computers and Operations Research* 36:2955–2968.

Branke J, Middendorf M, Noeth G, Dessouky M (2005) Waiting strategies for dynamic vehicle routing. *Transportation Science* 39(3):298–312.

Brown D, Smith J, Sun P (2010) Information relaxations and duality in stochastic dynamic programs. *Operations Research* 58(4):785–801.

Caspar M, Wendt O (2024) Reinforcement learning applied to the dynamic capacitated profitable tour problem with stochastic requests. Gervasi O, Murgante B, Garau C, Taniar D, C Rocha AMA, Faginas Lago MN, eds., *Computational Science and Its Applications – ICCSA 2024*, 346–363 (Cham: Springer Nature Switzerland).

Chen X, Ulmer M, Thomas B (2022) Deep q-learning for same-day delivery with vehicles and drones. *European Journal of Operational Research* 298:939–952.

Choi J, Kwon J, Lee KM (2018) Real-time visual tracking by deep reinforced decision making. *Computer Vision and Image Understanding* 171:10–19.

Ferrucci F, Bock S, Gendreau M (2013) A pro-active real-time control approach for dynamic vehicle routing problems dealing with the delivery of urgent goods. *European Journal of Operational Research* 225(1):130–141.

Gendreau M, Guertin F, Potvin JY, Séguin R (2006) Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies* 14(3):157–174.

Gendreau M, Guertin F, Potvin JY, Taillard E (1999) Parallel tabu search for real-time vehicle routing and dispatching. *Transportation science* 33(4):381–390.

Ghiani G, Manni E, Quaranta A, Triki C (2009) Anticipatory algorithms for same-day courier dispatching. *Transportation Research Part E: Logistics and Transportation Review* 45(1):96–106.

Ghiani G, Manni E, Thomas BW (2012) A comparison of anticipatory algorithms for the dynamic and stochastic traveling salesman problem. *Transportation Science* 46(3):374–387.

Heinrich J, Silver D (2016) Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121* .

Hvattum L, Løkketangen A, Laporte G (2006) Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. *Transportation Science* 40(4):421–438.

Ichoua S, Gendreau M, Potvin J (2006) Exploiting knowledge about future demands for real-time vehicle dispatching. *Transportation Science* 40(2):211–225.

Ichoua S, Gendreau M, Potvin JY (2000) Diversion issues in real-time vehicle dispatching. *Transportation Science* 34(4):426–438.

Joe W, Lau H (2020) Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers. Beck J, Buffet O, Hoffmann J, Karpas E, Sohrabj S, eds., *Proceedings of the international conference on automated planning and scheduling*, volume 30, 394–402.

Kullman N, Cousineau M, , Goodson J, Mendoza J (2022) Dynamic ride-hailing with electric vehicles. *Transportation Science* 56(3):775–794.

Lample G, Chaplot D (2017) Playing fps games with deep reinforcement learning. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2140–2146, AAAI'17 (AAAI Press), URL `http://dl.acm.org/citation.cfm?id=3298483.3298548`.

Liu Y, Logan B, Liu N, Xu Z, Tang J, Wang Y (2017) Deep reinforcement learning for dynamic treatment regimes on medical registry data. *2017 IEEE International Conference on Healthcare Informatics (ICHI)*, 380–385 (IEEE).

Meisel S (2011) *Anticipatory Optimization for Dynamic Decision Making*, volume 51 of *Operations Research/Computer Science Interfaces Series* (Springer).

Mitrović-Minić S, Laporte G (2004) Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological* 38(7):635–655.

Mnih V, Kavukcuoglu K, Silver D, Rusu A, Veness J, Bellemare M, Graves A, Riedmiller M, Fidjeland A, Ostrovski G, et al. (2015) Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Palombarini J, Martínez C (2022) End-to-end on-line rescheduling from gantt chart images using deep reinforcement learning. *International Journal of Production Research* 60(14):4434–4463.

Psaraftis HN (1980) A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science* 14(2):130–154.

RLlib (2024) Industry-grade reinforcement learning. URL `https://docs.ray.io/en/latest/rllib/rllib-algorithms.html#dqn`, accessed on June 16, 2024.

Sallab A, Abdou M, Perot E, Yogamani S (2017) Deep reinforcement learning framework for autonomous driving. *Electronic Imaging* 2017(19):70–76.

Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, Lanctot M, Sifre L, Kumaran D, Graepel T, et al. (2018) A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* 362(6419):1140–1144.

Soeffker N, Ulmer D Mand Mattfeld (2019) Adaptive state space partitioning for dynamic decision processes. *Business and Information Systems Engineering* 61(3):261–275.

Soeffker N, Ulmer M, Mattfeld D (2022) Stochastic dynamic vehicle routing in the light of prescriptive analytics: A review. *European Journal of Operational Research* 298:801–820.

Thomas BW, White III CC (2007) The dynamic shortest path problem with anticipation. *European Journal of Operational Research* 176(2):836–854.

Ulmer M, Mattfeld D, Köster F (2018a) Budgeting time for dynamic vehicle routing with stochastic customer requests. *Transportation Science* 52(1):20–37.

Ulmer M, Soeffker N, Mattfeld D (2018b) Value function approximation for dynamic multi-period vehicle routing. *European Journal of Operational Research* 269:883–899.

Ulmer MW, Goodson JC, Mattfeld DC, Hennig M (2018c) Offline–online approximate dynamic programming for dynamic vehicle routing with stochastic requests. *Transportation Science* 53(1):185–202.

Ulmer MW, Goodson JC, Mattfeld DC, Thomas BW (2020) On modeling stochastic dynamic vehicle routing problems. *EURO Journal on Transportation and Logistics* 9(2):100008.

van Hemert JI, La Poutré JA (2004) Dynamic routing problems with fruitful regions: Models and evolutionary computation. *Parallel Problem Solving from Nature-PPSN VIII*, 692–701 (Springer).

Vinyals O, Babuschkin I, Czarnecki W, Mathieu M, Dudzik A, Chung J, Choi D, Powell R, Ewalds T, Georgiev P, et al. (2019) Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* 575(7782):350–354.

Xinquan W, Xuefeng Y (2023) A spatial pyramid pooling-based deep reinforcement learning model for dynamic job-shop scheduling problem. *Computers and Operations Research* 160:106401.

Zhang J, Woensel TV (2023) Dynamic vehicle routing with random requests: A literature review. *International Journal of Production Economics* 256:108751.