SafeRoute: Learning to Navigate Streets Safely in an Urban Environment

SHARON LEVY, WENHAN XIONG, ELIZABETH BELDING, and
WILLIAM YANG WANG, University of California, Santa Barbara

Recent studies show that 85% of women have changed their traveled routes to avoid harassment and assault. Despite this, current mapping tools do not empower users with information to take charge of their personal safety. We propose SafeRoute, a novel solution to the problem of navigating cities and avoiding street harassment and crime. Unlike other street navigation applications, SafeRoute introduces a new type of path generation via deep reinforcement learning. This enables us to successfully optimize for multi-criteria path-finding and incorporate representation learning within our framework. Our agent learns to pick favorable streets to create a safe and short path with a reward function that incorporates safety and efficiency. Given access to recent crime reports in many urban cities, we train our model for experiments in Boston, New York, and San Francisco. We test our model on areas of these cities, specifically the populated downtown regions with high foot traffic. We evaluate SafeRoute and successfully improve over state-of-the-art methods by up to 17% in local average distance from crimes while decreasing path length by up to 7%.

CCS Concepts: • **Applied computing** → **Transportation**; • **Information systems** → *Spatial-temporal systems*; • **Computing methodologies** → **Planning and scheduling**; *Spatial and physical reasoning*;

Additional Key Words and Phrases: Safe routing, multi-preference routing, deep reinforcement learning

ACM Reference format:
Sharon Levy, Wenhan Xiong, Elizabeth Belding, and William Yang Wang. 2020. SafeRoute: Learning to Navigate Streets Safely in an Urban Environment. *ACM Trans. Intell. Syst. Technol.* 11, 6, Article 66 (September 2020), 17 pages.
https://doi.org/10.1145/3402818

1 INTRODUCTION

Many women take alternative and sometimes longer routes than those that are recommended by navigation applications, such as Google Maps. This is due to fear of harassment or violence when walking alone or with other women on the streets, especially at night. In a street harassment study by Cornell University and Hollaback from 2014, researchers interviewed 4,872 U.S. women. They found that 85% of these women have taken a different route home or to their destination to avoid potential harassment or assault, and 67% changed the time they left an event or location [5]. While those local to an area might know which streets are safe or risky and can take their own

Authors' addresses: S. Levy, W. Xiong, E. Belding, and W. Y. Wang, University of California, Santa Barbara; emails: {sharonlevy, xwhan, ebelding, william}@cs.ucsb.edu.

2157-6904/2020/09-ART66 $15.00
https://doi.org/10.1145/3402818

"safe routes," others visiting a new city will most likely be unaware of the places they should avoid. Tourists have to research criminal activity ahead of their planned trip to stay informed when walking unfamiliar streets. The creation of a safe routing application is more critical than ever. Data collected from 61 metropolitan police agencies showed an 11% increase in homicides in 2016, the second year in a row with such development [10]. With a safe routing application, women, tourists, students, and others will have the resource to increase their safety and peace of mind when walking in an urban environment.

In this article, we focus exclusively on non-vehicular travel, such as walking and biking. Walkable cities like New York City, London, and Boston typically have multiple potential routes between any two points. We want to compute short and risk-free paths and create a balance between the two objectives. Existing state-of-the-art methods also focus on populated cities but have the disadvantage of using crime density maps to create paths that smooth over smaller clusters of crimes and, therefore, ignore smaller crime hotspots.

Recent contributions in deep reinforcement learning (RL) show the possibility of short path navigation [15]. However, our model aims to go beyond this and has the complex goal of navigating safely in addition to finding a short path. To provide safe routes, we choose a solution based on deep RL, which is a natural choice for many data mining problems that require making incremental decisions. Instead of requiring supervised signals at every time step, the policy can be refined based on a single long-term reward signal. This has many advantages over classical algorithms such as Dijkstra [3] and A* [7]. Deep RL allows us to integrate a continuous vector as its agent's state representation and utilize different types of information, such as graph embeddings. We can also directly train the agent to co-optimize several goals at once, according to user preferences. For example, the continuous state vector can include the agent's location on the map and time of day. Classical algorithms cannot explicitly define these multiple optimizations. With deep RL, we can model the user scenarios and perform approximation for these goals, acting as a more practical solution. The flexible architecture also allows us to utilize different learning techniques and learn to adapt to new data. Given that crimes occur daily, safety parameters are continuously changing, and the model would be able to incorporate online learning techniques for these changes in the future [18]. Other existing approaches to safe path-finding do not use RL to generate paths. While there are existing classical RL systems in route planning [1, 25, 31], they cannot provide the many advantages that a deep learning architecture can provide, such as the continuous state space for different types of information, utilization of representation learning, and policy gradient techniques.

We propose SafeRoute, a novel solution to the problem of safe path-finding using deep RL. The goal of SafeRoute is to provide users with a safe and short route to their destination. The RL agent learns to choose a safe street at each step that leads from a starting to destination intersection. We test our results against paths found using an existing—non-reinforcement—algorithm (SafePath) that also utilizes crime information when calculating safe routes.

SafeRoute makes three contributions to the problem of safe path-finding:

- —We are the first to consider deep RL to solve the problem of multi-objective path-finding.
- —We propose a reward function for optimizing safety and length in generated paths. As opposed to using geographical coordinates, we utilize graph embeddings based on local street connectivity to represent maps to create an improved learning environment for our agent.
- —Our model produces results with up to 17% relative improvement in local average distance from crimes over the state-of-the-art model with samples from maps of Boston, New York, and San Francisco.

2 RELATED WORK

Safe Routing: There are existing studies that explore the balance between safety and efficiency in routing. One such application, SafePath, creates safe paths for users and optimizes for both safety and distance [4]. Using a crime density map, SafePath assigns risks to streets and outputs paths varying from shortest to safest. A safe path application for Mexico City utilizes tweets and official crime reports to classify and geocode crimes with a naive Bayes classifier [14]. However, it does not consider geographical distance in its algorithm and only focuses on creating paths based on safety. Similarly, SocRoutes creates safe routes using geocoded tweets and routes users through points around unsafe regions [11]. SocRoute performs sentiment analysis on tweets and categorizes regions on a map as unsafe when they have more negative tweets on average. To generate its routes, it first creates the shortest path to the destination and, if the path goes through an unsafe region, it moves its waypoints to be outside of that region. While these approaches focus on crimes on a larger scale, our model concentrates on crimes at the street level. A user would most likely not choose to deviate from the shortest path by too much, so previous works would either ignore smaller crime hotspots or route users in a much longer distance around a larger crime area.

Multi-Preference Routing: Multi-preference or multi-criteria routing algorithms optimize routes for users with more than one objective, such as our model. One such variety considers both distance and either happiness, beauty, or quietness when generating paths by re-ranking a top-k list of shortest paths according to the second objective [17]. However, if the top-k list of paths is not diverse enough from the shortest path, it may be hard to optimize the second objective. One of the more common models of multi-preference routing optimizes distance and traffic conditions. For instance, the PreGo system creates paths based on multiple user preferences such as time, scenery, and road conditions through a single traversal of the graph-based map [8]. Though this system can route based on the risk factor, it does not incorporate distances from risks near edges, which does not give enough information about street safety. Another system, T-Drive, optimizes both distance and time using GPS taxi information and distance to model routes [29]. However, there is no explicit way to include crime information in this process. The Advanced Route Skyline Computing (ARSC) algorithm, introduced in Reference [13], finds non-dominated paths using a best first graph search. Though these models create routes based on multiple criteria, most do not utilize crime information while routing, which can lead to the generation of unsafe paths. Additionally, our model's continuous state space allows us to incorporate various attributes into our state and integrate representation learning of the network with explicit path-finding. Multi-criteria optimization falls into the category of NP-hard problems and is not solvable by theoretical computer science. However, deep RL allows us to perform approximation within the learning-to-search framework. Another reason for the use of deep RL is the ability to integrate human feedback into the model. As risk is not deterministic, other non-deep RL approaches fail as they need explicit functions to define risk. Human feedback can help provide policy updates and continuously learn and shape the model even more toward a human-generated path. To elaborate, users can decide whether they like a provided route, and the model can be rewarded or punished for future generations of paths in the area. With such advantages, it is clear that our task can benefit from the use of deep RL.

RL in Navigation: In Reference [2], a deep RL model learned to localize itself on a 2D map from a 3D perspective and find the shortest paths out of its mazes. The Reinforced Planning Ahead (RPA) model uses a combination of model-based and model-free learning for visual navigation with textual instructions [23]. Recently, Google's DeepMind has created a deep RL model that trains on Google Street View to navigate cities without a map [15]. For tasks of reaching a destination point, the model represents the target in relation to its distances from landmarks nearby.

One of the drawbacks of image-based navigation is the amount of data required for training. Furthermore, SafeRoute attempts to co-optimize the two goals of safety and distance, which would result in additional training to not only find a target with unstructured image data but also classify unsafe streets and avoid them. Reference [31] utilizes Q value-based dynamic programming to find optimal routes based on different objectives. However, its experiments execute on networks of up to 10 nodes, and the algorithm does not scale well to larger networks in real-world settings due to space complexity. Additionally, the model uses discrete states, which utilizes a fixed state-action table and cannot generalize to unseen states. With SafeRoute's simple deep RL architecture, it can easily include other parameters to the model's input for optimization and allow for the learning of a more descriptive network map. When adding new nodes to the network, they will have values/states similar to those of nearby nodes, and the model will continue functioning as is. Graph-based navigation appears in some deep RL frameworks. DeepPath [27] uses deep RL to infer missing links within a knowledge graph. Network packet routing applications also utilize RL [1]. However, this method utilizes only discrete states equalling to the packet's current node in the network and focuses on only packet delivery time. Another method trains on maze navigation recordings to build a topological map and later navigate to a destination within the maze [20].

Despite prior work in safe path-finding and multi-preference routing, SafeRoute differentiates itself by (1) analyzing crimes in the direct path area in order to generate its routes, (2) utilizing continuous state space to allow different types of information as input and incorporate representation learning, and (3) enabling human feedback through policy updates.

3 SAFEROUTE

We can view the route finding process as a Markov Decision Process. At each timestep, the agent decides which compass direction to go next, eventually leading to the final destination. The start and end coordinates of intersections are input into the model, which returns a list of coordinates relating to the agent's incremental decisions. By rewarding the agent for avoiding crime-filled streets, we create a safe path for the user. The following section describes the framework and the training and testing pipeline behind the deep RL architecture of SafeRoute. The environment and policy-based agent are discussed, along with the rewards system, and used to find short and safe paths within a map. Additionally, the two forms of training (pre-training and retraining with rewards), along with the testing algorithm, are described.

3.1 Architecture

The SafeRoute system splits into two parts: an environment with which the RL agent interacts, and the policy network the RL agent represents and uses to make decisions within the environment. We show the SafeRoute architecture in Figure 1. The environment is represented as a Markov Decision Process with tuple $< S, A, P, R >$. S represents the continuous states of the environment and $A = \{a_1, a_2, \ldots a_N\}$ defines all actions available to the agent. $P(S_{t+1} = s_0 | S_t = s, A_t = a)$ determines the probability of moving from one state to another. $R(s, a)$ is the function that rewards the agent for taking action a when in the state s.

States: The agent's state represents its current status on the map. In our model, the state uses the agent's current and target position. Including the target position allows the agent to relate actions to the end goal, so if it is in the current position at a later time with a destination in the opposite direction from before, the agent will not take the same actions. Furthermore, the agent can generalize better to new unseen states. If the new target is near one the agent has trained on, the agent will take similar actions toward the new goal when starting from the same area. Map information is transformed into a graph with street intersections as nodes and streets connecting

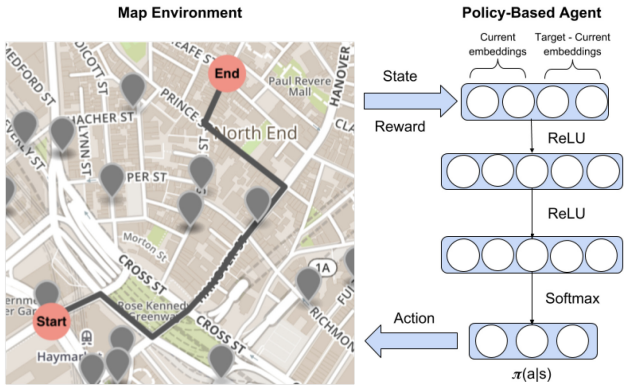


Fig. 1. The SafeRoute model containing the map environment and policy-based agent. The gray pins show recent crimes occurring in the area. We feed the current state into the agent's neural network, which outputs the action to take. After performing the action in the environment, the reward is collected and used to update the agent's policy.

two nodes as edges. We represent the graph as a directed graph with a compass direction for each edge. To represent the continuous states of the RL agent, we use graph embeddings (generated with node2vec [6]) instead of the latitude/longitude coordinates. The reasoning behind using graph embeddings as opposed to coordinates is that coordinates do not give any information as to how intersections connect on the actual map. The embeddings can also learn additional information captured in the road network, such as street popularity, as a user preference by including it as a weight in the road network graph before generating the embeddings. In the initial iterations of the project, we used coordinates to represent states. However, even with pre-training, our model was unable to learn to navigate the map. With the graph embeddings, the model can determine the streets that lead to certain intersections and, eventually, the final destination. The states use embeddings from the agent's current node and target node as follows:

$$\mathbf{s}_t = (\mathbf{e}_t, \mathbf{e}_{target} - \mathbf{e}_t),$$

where $\mathbf{e}_t$ denotes the embeddings of the current node and $\mathbf{e}_{target}$ denotes the embeddings of the target node. The subtraction in the second half of the state helps determine the target's direction and distance from the current position. Due to the model's deep learning architecture, other factors can be included in the state vector, such as time of day and distance flexibility, allowing the model to train for multiple user scenarios. We can include these as vectors concatenated with the existing state input.

Actions: Actions in the environment represent moving from one street intersection to another. The actions themselves are compass directions (North, Northeast, East, Southeast, South, Southwest, West, Northwest). The RL agent learns to pick actions, out of all available actions, that lead in the direction of the target intersection while also moving away from high crime areas.

Policy Network: The policy network representing the RL agent uses a stochastic policy, $\pi_\theta(s, a) = p(a|s; \theta)$, where θ is the list of neural network parameters. We use a stochastic policy instead of a greedy policy to prevent the agent from getting stuck in cycles on the map. Using a stochastic policy, the agent can break free of cycles, such as repeatedly moving toward a dead-end that appears to be leading in the right direction or continually taking a path that will eventually result in a dead-end. The neural network contains two hidden layers, each with a rectified linear unit (ReLU) activation function. The output uses a softmax function and returns a probability

distribution over all actions. We prune the actions for those not available at the current intersection, and the remaining probabilities are normalized and returned. At a high level, the neural network takes as input a state s and outputs a normalized probability distribution over all available actions.

Rewards: The agent optimizes multiple preferences, so the reward function must consider several different factors. Since an integral feature of SafeRoute is to avoid crime areas when creating a route, we add safety into the reward as a function of average distances from previously known crime scenes. A list of recent crimes in the city is traversed for those at a certain radius from the current location. The radius and location vary per edge along the path. The location used is the midpoint along each edge, and the radius is equal to the length of that edge. The average distance from crimes within the radius at each step is then calculated. The average distance is used as opposed to directly calculating the number of crimes because we value more crimes at the edge of the radius as better than fewer crimes directly along the route.

Although the primary goal of SafeRoute is to increase safety by generating routes that lead away from high crime areas, we also want to consider efficiency in our reward represented by the path length. The total average distance from crimes along the path is calculated and then divided by the path length. If there are no crimes near the path, κ is used as a reward. All other paths are assigned a reward proportional to their average distance from crimes divided by the path length. The final reward is defined below:

$$r_{CRIME} = \begin{cases} \frac{\frac{\sum_{i=1}^{n}\sum_{j=1}^{m} distance(x_i, c_{ij})}{\sum_{i=1}^{n} number(c_i)}}{length(p)}, & \text{if } c \neq \emptyset \\ \kappa, & \text{otherwise,} \end{cases}$$

where n is the number of edges along the path, m is the number of crimes within the radius at each node, x is the list of edge midpoints along the path, c is the list of crimes in each radius, p is the path, and κ is a hyperparameter. With this reward in place, shorter paths will be rewarded more than longer paths with similar crime rates.

3.2 Training

Training for SafeRoute comes in two parts: pre-training and retraining with rewards. Without pre-training as the initial step, the agent will have a hard time finding a path to the target node and can end up wandering in random directions. AlphaGo [21] uses imitation learning [9] as the first step in its training process in order to give the agent an initial push when starting to train with rewards. Similarly, we also start the training pipeline of SafeRoute with pre-training as its form of imitation learning.

Policy Pre-Training. In SafeRoute, one of the criteria for a good path is a short distance. Therefore, we initially train the policy network by rewarding the shortest paths for each training episode. In addition to helping optimize one of our goals of a short path, the pre-training helps the agent balance exploration vs. exploitation during the second portion of training and not rely entirely on exploration. This effect of pre-training is seen in other deep RL models, such as DeepPath [27]. The training samples used include a randomly sampled starting intersection on the map and several endpoints at a 5-hop distance on the respective graph. We shuffle these samples, and each episode uses Dijkstra's algorithm for shortest paths using the edge length as the weight. At the end of each episode, the neural network parameters θ update to reward the actions taken at each state. Each state-action pair, corresponding to a step t along the path, is rewarded equally using Monte-Carlo Policy Gradient (REINFORCE) [26]. The reward given for each episode's steps during

ALGORITHM 1: Algorithm for retraining with rewards

```
for episode ← 1 to N do
    Initialize max_rwd ← 0.0
    Initialize avg_rwd ← 0.0
    Initialize num_success ← 0.0
    Initialize max_path ← ∅
    for i ← 0 to T do
        Initialize state vector s_t ← s_0
        Initialize episode length num_steps ← 0
        while num_steps < max_len do
            Randomly sample action a ~ π(a|s_t)
            Add < s_t, a > to path
            if success or num_steps = max_len then
                if success and R_CRIME > max_rwd then
                    max_rwd = R_CRIME
                    max_path = path
                break
            Increment num_steps
            s_t ← s_{t+1}
    if max_rwd ≠ 0.0 then
        b ← avg_rwd / num_success or 1
        for < s_t, a > in path do
            Update θ with g ∝
            ∇_θ Σ_t logπ(a = r_t|s_t; θ)(R_{p−(t−1)} − b)
        Increment num_success
        avg_rwd ← avg_rwd + max_rwd
```

policy pre-training is +1, so our final gradient when updating the policy is equal to:

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_t \sum_{a \epsilon A} \pi(a|s_t;\theta) \nabla_\theta log\pi(a|s_t;\theta) \\ &\approx \nabla_\theta \sum_t log\pi(a = a_t|s_t;\theta),\end{aligned}$$

where a_t is the corresponding action taken at time t along the path.

Retraining with Rewards. After pre-training, the agent retrains to avoid crime-filled areas. We run the model T times for every episode. Due to the stochastic nature of the policy, all T paths found by the agent will likely exhibit some variation. While the pre-training limits the agent's exploration, the stochastic policy reduces this by allowing the agent to generate a variety of paths to choose from to reward, thus finding a balance and not relying entirely on exploitation. For each of the successful paths generated, we look at the rewards calculated along the path. When updating the policy, we only consider the path with the highest rewards, since a higher reward indicates a more optimal path, and we want future queries to lead to similar paths. The model does not update for the other successful and non-successful paths, as updates with negative rewards

generated worse results and we do not want to encourage the model to recreate less optimal paths. Algorithm 1 illustrates the training procedure.

Instead of immediately updating the policy with the rewards for the best path, we use a baseline value b, such as the one used in Reference [30]. The baseline we use is a running average of the current rewards in the current epoch, using the rewards from the most successful path in each episode. With the baseline in place, the most successful path from each episode will be rewarded only if its rewards are greater than the baseline.

In addition to using a baseline value, we also do not reward each state-action pair equally. Instead, each is rewarded for its actual value in creating the path. At each timestep t, the current action is rewarded with only the remaining path in mind. Any information coming from previous edges along the path is not included in the calculation of the reward. The final gradient is shown below:

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_t log\pi(a = r_t|s_t;\theta)(R_{p-(t-1)} - b)$$

In terms of our model's convergence properties, our usage of REINFORCE allows us to determine that after training, the model should converge to a local optimum [26]. If it is simple Q-learning, then we can use the Bellman equation and dynamic programming to obtain the optimal results [24]. However, here we are considering deep RL where the state is continuous and hidden, and there are no theoretical results on optimality to date. For our model, we cease training once the rewards obtained on the validation set stop increasing for multiple epochs so that our model does not overfit to the training data. We discuss the number of training samples and epochs for our model in Section 4.1.

Testing. For all the evaluations, we use beam search to create our paths with a beam size of 5. The policy runs several times for every current path in the beam, and these paths extend into several new paths. Paths that remain in the beam are those that are highly favored by the policy. Once five paths have successfully reached the target, or the step limit is reached, the path with the highest local average distance from crimes, as described in Section 4, is chosen as the final path. The path created will occasionally contain loops leading back to an intermediate node along the path due to the agent's stochastic nature. Post-processing removes these external loops and returns the resulting path. Algorithm 2 illustrates the testing procedure.

4 EXPERIMENTS

4.1 Dataset

Though our baseline utilized public crime information for its safe routing, we did not have access to SafePath's dataset and thus created our own for our experiments.[1] Map information was collected from OpenStreetMap, a free collaborative world map [16]. We chose to export map information for the downtown areas of Boston, New York, and San Francisco. This is due to the fact that many tourists visit these areas, and as they are urban centers, there are typically multiple viable paths between any two points. The resulting graphs for the three cities were unequal in size, and as a result, San Francisco was trained with 2,000 samples (episodes), Boston with 3,000 samples, and New York (the largest) was trained with 4,000 samples. All three models trained for 60 epochs. Hyperparameter tuning was used to determine the value of κ in the reward function. The neural network parameters were updated using Adam Optimizer [12]. We achieved the best results with a learning rate of 0.0005. The model layer dimensions were 256 for the input, 512 and 1,024 for the hidden layers, and eight action choices in the output layer. We chose five as our value for T

[1]Source code and dataset will be available: https://github.com/sharonlevy/SafeRoute.

ALGORITHM 2: SafeRoute algorithm for testing

```
for episode ← 1 to N do
    Initialize state vector s_t ← s_0
    Initialize environment e_t ← e_0
    Initialize probability p_t ← 1.0
    Initialize current paths curr_paths ← (s_t, e_t, p_t)
    Initialize successful paths success_paths ← ∅
    Initialize episode length num_steps ← 0
    Initialize new paths new_paths ← ∅
    while num_steps < max_len and |success_paths| ≠ 5 do
        new_paths ← ∅
        for s_t, e_t, p_t in curr_paths do
            for i ← 1 to 5 do
                Randomly sample action a ~ π(a|s_t) with probability p
                Copy s_t, e_t to s_c, e_c and take action a in new environment
                p_c ← p_t * p
                if success then
                    Add s_c, e_c, p_c to success_paths
                else
                    Add s_c, e_c, p_c to new_paths
                    s_t ← s_{t+1}
        while |new_paths| > 5 − |successful_paths| do
            Remove < s_t, e_t, p_c > from new_paths with lowest p_c
        Increment num_steps
        curr_paths ← new_paths
    if |successful_paths| > 0 then
        Choose path from successful_paths with greatest local average distance from crimes
        Post-process and return path
```

in the training algorithm in order to balance the speed of retraining versus the diversity of the generated paths. A beam size of five was used in the testing algorithm. We collect crime data from Spotcrime, which shows recent crime incident information with details such as geographic coordinates and type of crime [22]. This source only allows us to retrieve information from the past year. However, utilizing recent crimes from this time range allows us to obtain a sizable amount of data that allows us to train the model for long-lasting crime trends. Using only very recent crimes introduces sparsity during training and does not allow the model to learn general trends. Types of crimes are treated equally. We previously experimented with different severity levels and concluded that it is not a feasible addition to our model since the many different types of crimes introduced too much sparsity during training. For SafeRoute, we chose to use the crimes of shooting, assault, and robbery, as they are crimes related to street harassment and assault.

A study done by Reference [28] found that walking trips in the U.S. are 0.7 miles on average and with a median of 0.5 miles. Therefore, we test our model on paths near these lengths. The model trains on start and end points that are a 5-hop distance away from each other in the graph. These points represent initial and final points that can be reached in five decisions, though the actual length may vary depending on the city's grid and chosen path. When doing experiments,

we randomly sample nodes in the graph and test on 5-hop and 10-hop paths. We chose to test on these numbers of hops because they resulted in paths ranging from 0.20–1.0 miles on average when tested using Dijkstra's shortest path algorithm. Therefore, we cover the spectrum of most walking distances. However, because we are prioritizing both distance and safety, our generated paths are usually longer than their shortest path distance.

4.2 Evaluation Settings

We evaluate our model in several experiments. The quality of the paths is measured in three different tests: average distance from crimes (local), average distance from crimes (global), and path length. These are all measured in terms of miles.

Local Average. The local average distance from crimes only considers crimes near the agent as it traverses the path and is calculated similarly to the agent's rewards:

$$AvgCrime(local) = \begin{cases} \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} distance(x_i, c_{ij})}{\sum_{i=1}^{n} number(c_i)}, & \text{if } c \neq \emptyset \\ AvgMinCrime, & \text{otherwise,} \end{cases}$$

where n is the number of edges along the path, m is the number of crimes within the radius at each edge, x is the list of edge midpoints along the path, and c is the list of crimes in each radius. If a path happens to have no crimes within its radius, then it uses the average minimum distance from crimes and is consistent with our safety-first approach—highly valuing edges with no crimes around them.

Global Average. The global average distance from crimes experiment is equivalent to the local test but considers all crimes at each edge along the path. This is seen below:

$$AvgCrime(global) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} distance(x_i, c_j)}{n},$$

where m is the list of all crimes in the city, and c is the list of crimes in the city.

Path Length. We sum up the edge lengths along a path in order to determine the final path length:

$$Length = \sum_{i=1}^{n} length(n)$$

We evaluate SafeRoute against the baselines on the different metrics. We selected these evaluations based on what we believe to be easily explainable criteria that are likely to be valued by humans trying to navigate a new city safely. Though we propose the metrics of both local average and global average distance from crimes, we value the local average distance metric the most. This metric calculates the average distance to a set of crimes, constrained to the crimes within a certain distance from the edges traversed by the path. It is very similar to the reward function used by the RL agent and, therefore, provides a measure of how well the agent learned the given task. Furthermore, we believe this metric to be a good approximation of the overall intuitive safety of a path, which is why we developed our model to train on it. By keeping a high average distance from local crimes, we can measure how well the agent learns to avoid certain unsafe streets and even navigate within the more crime-filled areas of the city. As an additional metric, we also examine the global average distance along our paths as it considers the entire crime list in its score as opposed to the local average distance. However, this metric is less significant when traversing a path. The global average distance captures information from the whole city, and users would likely not care about crime hotspots miles away from their paths.

We compare our model against two baselines: SafePath and Dijkstra's algorithm. SafePath creates safe paths for users by producing non-dominated paths when considering both safety and

Table 1. Results for the Average Distance from Crimes (Local), Average Distance from Crimes (Global), and Length Experiments on 5-Hop and 10-Hop Test Datasets

		5-hops			10-hops		
City	Model	Local	Global	Length	Local	Global	Length
Boston	Dijkstra	0.0554	0.8620	0.2023	0.0484	0.8479	0.4200
	SafePath(Median)	0.0566	0.8648	0.2044	0.0498	0.8639	0.4372
	SafePath(Safest)	0.0568	0.8651	0.2050	0.04817	0.8768	0.4619
	SafeRoute	**0.0630**	0.8627	0.2361	**0.0571**	0.8510	0.4903
San Francisco	Dijkstra	0.0882	0.9268	0.4704	0.0964	0.8966	1.0121
	SafePath(Median)	0.0934	0.9423	0.4821	0.1046	0.9344	1.1063
	SafePath(Safest)	0.0937	0.9489	0.5001	**0.1084**	0.9650	1.2402
	SafeRoute	**0.0990**	0.9880	0.5468	0.1036	0.9611	1.1659
New York	Dijkstra	0.1344	1.1978	0.4222	0.1016	1.1842	0.7907
	SafePath(Median)	0.1341	1.2125	0.4515	0.0987	1.2267	0.8748
	SafePath(Safest)	0.1344	1.2242	0.4935	0.0978	1.2537	0.9731
	SafeRoute	**0.1454**	1.2004	0.47267	**0.1179**	1.1992	0.8915

Distance is calculated in terms of miles. In our evaluation, larger values for the local and global metrics are better while smaller values for the length are favorable.

distance. When generating safe paths, it assigns streets a risk factor determined by a city's crime density map. This approach outputs multiple paths on a varying scale of safety and distance using Dijkstra's algorithm. SafePath represents paths on a graph by visualizing risk versus length. In order to generate the paths in between, the algorithm re-weights streets using the gradient of the line between two paths. The final paths appear as the lower convex hull on the risk vs. length graph. For our evaluation, we compare against SafePath's safest paths and the median paths that it outputs, meaning that these paths balance safety and distance. We use Dijkstra's algorithm with distance as the edge weights for comparison against shortest paths, as is done in SafePath.

In addition to the general model we describe in Section 3, we also conduct two additional experiments. The first compares the Boston SafeRoute model with our chosen crime distance radius (length of the current edge) against models using half the radius and double the radius. The second experiment trains a SafeRoute model that considers time. The model is trained in Boston with 5-hops. This model trains for daytime and nighttime paths. The model has the same state input as that in the general model. However, we concatenate a time vector, which relates to either daytime or nighttime. We then train the model for both scenarios with the same training set. When rewarding the paths, we only consider crimes that occur within the same time frame: daytime or nighttime.

4.3 Results

Table 1 shows a numerical comparison between the experimental results of SafeRoute and the baselines. It is evident by how different the distances are that each city has a different structure and distribution of crimes. However, even with these variations, SafeRoute can perform well in each city with the same rewards function. Comparing the local average crime distances between the three cities for 5-hop paths shows the numbers are much lower for Boston, implying that it is harder to navigate away from crime in this city. When analyzing the crime density per city, this shows to be accurate as Boston had the highest crime density with San Francisco following closely behind. Meanwhile, New York had a density of about half of Boston. The crime distance metrics in Table 1 follow the same pattern, with Boston having the shortest distances from crimes and

Table 2. Percent Improvement in Results Over the State-of-the-Art Model (SafePath) in Safest Mode

	5-hops			10-hops		
City	Local(%)	Global(%)	Length(%)	Local(%)	Global(%)	Length(%)
Boston	10.4	− 0.3	− 20.9	16.2	− 0.8	− 12.4
San Francisco	4.5	4.1	− 7.6	− 6.6	0.3	5.3
New York	6.9	− 2.1	2.2	17.1	− 4.7	6.5
Average	7.3	0.6	− 8.8	8.8	− 2.3	− 0.4
Standard Deviation	2.7	2.6	9.9	11.3	2.2	9.4

Models are evaluated on local average, global average, and path length metrics. Three models were trained for each city and results in the table correspond to the average results for each.

Table 3. Comparison of Boston SafeRoute Model with Crime Radius Equal to Edge Length vs. Other SafeRoute Models with Half and Double Radius Lengths

	5-hops			10-hops		
Radius Length	Local	Global	Length	Local	Global	Length
Normal	**0.0630**	0.8627	**0.2361**	**0.0571**	0.8510	**0.4903**
Half	0.0567	**0.8641**	0.2397	0.0555	**0.8723**	0.5960
Double	0.0583	0.8612	0.2422	0.0570	0.8683	0.5026

Distance is calculated in terms of miles.

New York with the highest. However, even with this entailment, SafeRoute can navigate further from the crime spots than the baselines. When traversing longer distances at a 10-hop radius, it is noticeable that the local average crime distance decreases by about the same factor for Boston and San Francisco, and by a larger one for New York. This implies that traveling long distances in New York City requires one to go closer to crime spots so that length is not compromised significantly. Nonetheless, SafeRoute finds paths that are further away from local crimes on average for this city.

In order to decrease variance in our results, we create three models for each city and average the results for each. We also compute the average and standard deviation of the nine results for each metric. Table 2 shows SafeRoute's percent increase/decrease when compared to SafePath's safest routes. Because our agent receives rewards based on its local average distance from crimes, it is no surprise that our model surpasses SafePath by a large percentage. It is also worth noting that while our model increases path lengths for most experiments, as is expected, there were some test sets in which our average length was shorter than the SafePath while still excelling in some experiments. When comparing results for 5-hop and 10-hop paths, it is evident that SafeRoute has better results for the shorter paths due to the smaller search space and the lower likelihood that the agent will take a wrong action before it reaches the goal. This is also clear through the standard deviation results, where there is much less variation in the local average results for 5-hops than 10-hops.

It is expected that we do not have shorter paths than Dijkstra since we are co-optimizing two criteria. Nevertheless, we demonstrate that it is necessary to compromise length to ensure safety along a path, as confirmed in Table 2. The overall results of the experiments imply that our SafeRoute model has learned to successfully avoid dense crime areas on a map by creating paths that are further from local crimes on average and can find a balance between distance and safety.

We show the results of our crime radius length experiments in Table 3. Given these results, we can validate our chosen radius length as the most optimal. Using half the chosen radius may lead to very few crimes being evaluated throughout the path while training. A large radius focuses less on local crime hotspots and moves further away from the objective of our model.

(a) Daytime (b) Nighttime

Fig. 2. An example of paths generated by a time-based SafeRoute model. The left image shows a path learned for the daytime and the right image's path is for nighttime.

Table 4. Time Complexity Comparisons of SafeRoute, SafePath, and Dijkstra at Test Time

Models	Time Complexity
Dijkstra	$O(V^2)$
SafePath	$O(S(E + VlogV)$
SafeRoute	$O(V)$

V corresponds to the number of nodes in the graph, S is the number of paths generated, and E corresponds to the edges in the graph.

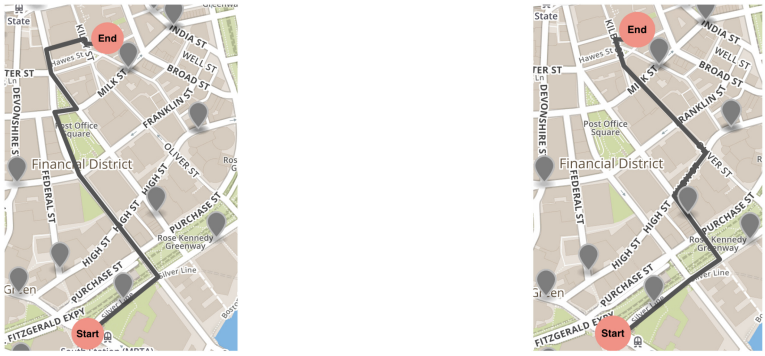
For our time-based SafeRoute model, we provide a sample of our results in Figure 2. It is clear that the left image, which shows a path constructed for the daytime, avoids the small cluster of crimes that appear on Washington St. However, a distance-only based path would lead the user on this street as it takes the user on a shorter path. The right image, which represents a nighttime-based path, takes this shorter path as there are only scattered crimes and no local clusters of crimes to avoid.

In addition to the above evaluations, we also compare the time complexities of SafeRoute, SafePath, and Dijkstra when creating paths. These can be seen in Table 4. It is clear that at test time, SafeRoute generates paths faster than the two baseline models.

We also show a sample result from SafeRoute and our baseline on a map of Boston in Figure 3. While, initially, the two go along the same path, SafeRoute stays in a mostly crime-free area for the rest of the path and, therefore, maps the user away from crimes. Meanwhile, SafePath routes in another direction and ends up closer to crime points instead.

4.4 Discussion

As can be seen in Table 1, SafePath evaluation results do not significantly change when attempting to create a safe route vs. a short route. SafePath creates a smooth crime density map in order to utilize Dijkstra's algorithm. The crime density map is created using a Gaussian kernel density estimation (KDE) [19]. The density map used by SafePath for our San Francisco dataset is shown in Figure 4. It can be seen that the density map has one major peak of crime intensity and does not capture the local variations in crime on the map. Due to the smoothing of the density map, small



(a) SafeRoute (b) SafePath

Fig. 3. An example of paths generated by SafeRoute and the state-of-the-art (SafePath) in its safest mode.

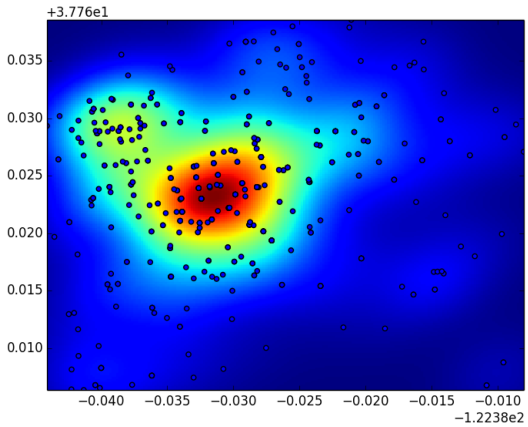


Fig. 4. Crime density map of San Francisco with the black dots representing crimes. The *x*-axis and *y*-axis represent the longitude and latitude coordinates, respectively.

clusters of crimes go seemingly unnoticed and are visualized as similar to single points of crime, as seen in Figure 4 at (−0.035, 0.014). In addition, most edges in the same area will appear to have very similar risk weights. These weights only have significant changes when traversing long distances. As a result, when using Dijkstra's to create risk-free paths for a short distance, as to SafePath, the output will most likely be the one with the fewest hops and longer paths will be overlooked by this algorithm. It is also apparent that the differences between SafePath's safest and median modes are minimal. When comparing the results for the local average experiment in Table 1, the resulting values are very close. Meanwhile, SafeRoute is shown to have a significant increase over the two in its local average metric. This reveals that SafePath's algorithm does not do very well in compromising its length for a safer path. In contrast, SafeRoute is inherently local and puts a high value on avoiding local crime spots. This is done using a non-linear reward system. By rewarding our agent based on local crimes, we train it to move toward safer streets that are nearby. With this reward system in place, it is evident that SafeRoute will generate paths co-optimized for safety and distance. Because SafeRoute co-optimizes for both parameters, some of our experiments revealed paths that were not longer than those chosen by SafePath in its safest mode.

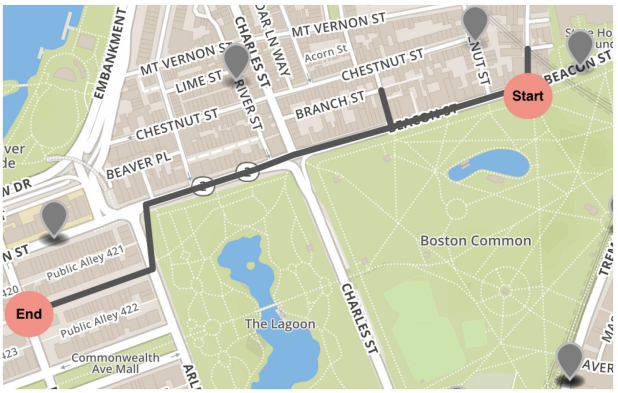


Fig. 5. Example of looping in a path generated by our model. The agent makes two wrong turns along the path in its initial steps.

As mentioned in Section 3, some paths generated by SafeRoute will contain cycles. These decisions will take the agent on a wrong turn, but eventually, lead back to an intermediate node along the path and continue to the destination. To handle this, we post-process our paths to remove external loops and return the resulting path. We do not allow the agent to take the same action in a specific state twice to prevent future cycles. Without this, the agent would end up in an endless cycle between two nodes. An example of a looping path before post-processing can be seen in Figure 5. When analyzing the path, we see the agent makes a wrong turn on its first step and goes North instead of West toward the destination. This occurs again shortly after. However, after the initial steps, the agent is able to reach the goal in an optimal path directly. To better understand this, we examined each step along the path. We found that the agent had high probabilities of going North initially, which decreased as the steps progressed toward the goal and switched to maximizing probabilities for the correct actions. One explanation for this is that not enough training samples crossed the initial area of the path. As a result, the agent maximized probabilities for the few directions it trained. A solution to this is increasing the size of our training sets and diversifying the start and end locations. Another reason for these loops is that initial steps for the agent are harder to decide since it is further away from the goal. This seems to be a plausible explanation because the percentage of loops in paths increased for our 10-hop datasets. However, since the 10-hop datasets also have longer paths, there is more room for error and making wrong turns.

Several extensions and studies are possible for SafeRoute. In the future, we plan to optimize SafeRoute further to enable longer paths and larger maps. In addition, we will study the portability of the model across urban environments. As discussed in Section 2, there has been much research in visual navigation. Therefore, another extension we plan for is to augment our agent with additional information to inform its decision, namely Google Street View images of the neighborhood. By combining multiple sources of information, we hope to boost the real-life relevance of our model further. Another exciting direction would be to harness human feedback to the generated paths in our evaluation of the quality of the paths. We hope to validate the results of our work and provide the generated feedback to an Inverse-Reinforcement learning framework to determine differences between our reward structure and the inferred one.

5 CONCLUSION

In this project, we developed and tested a deep RL model that aims to prevent street harassment by routing users away from local crime areas. We showed that an agent could learn to route users

around high-density crime areas without strict supervision and can do so based on external data not contained within the graph. Our model can produce high-quality paths between two nodes in the graph that co-optimize to avoid crime areas without unreasonably sacrificing short travel distances and surpass our baselines in evaluation. With personal safety as a continual issue in people's lives, we hope SafeRoute can be a useful step in crime avoidance and helping users reach their destinations safely.

REFERENCES

[1] Justin A. Boyan and Michael L. Littman. 1994. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems*. 671–678.

[2] Gino Brunner, Oliver Richter, Yuyi Wang, and Roger Wattenhofer. 2017. Teaching a machine to read maps with deep reinforcement learning. *AAAI 2018*.

[3] Edsger W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1, 1 (1959), 269–271.

[4] Esther Galbrun, Konstantinos Pelechrinis, and Evimaria Terzi. 2016. Urban navigation beyond shortest route. *Information Systems* 57, C (April 2016), 160–171.

[5] Maria Grillo, Rebecca Paluch, and Beth Livingston. 2014. Cornell International Survey on Street Harassment.

[6] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 855–864.

[7] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

[8] Abdeltawab M. Hendawi, Aqeel Rustum, Dev Oliver, David Hazel, Ankur Teredesai, and Mohamed Ali. 2015. Multi-preference time dependent routing. *Technical Report UWT-CDS-TR-2015–03-01, Center for Data Science, Institute of Technology, University of Washington, Tacoma, Washington* (2015).

[9] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation learning: A survey of learning methods. *ACM Computing Surveys* 50, 2, Article 21 (April 2017), 35 pages.

[10] David Johnson and Josh Sanburn. 2017. Violent crime is on the rise in U.S. cities. *Time* (2017).

[11] Jaewoo Kim, Meeyoung Cha, and Thomas Sandholm. 2014. SocRoutes: Safe routes based on tweet sentiments. In *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 179–182.

[12] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *ICLR 2015* (2014).

[13] Hans-Peter Kriegel, Matthias Renz, and Matthias Schubert. 2010. Route skyline queries: A multi-preference path planning approach. In *Proceedings of the IEEE 26th International Conference on Data Engineering (ICDE'10)*. IEEE, 261–272.

[14] Felix Mata, Miguel Torres-Ruiz, and Giovanni Guzman. 2016. A mobile information system based on crowd-sensed and official crime data for finding safe routes: A case study of Mexico City. *Mobile Information Systems* 2016, Article 8068209 (2016).

[15] Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. 2018. Learning to navigate in cities without a map. In *Advances in Neural Information Processing Systems*. 2419–2430.

[16] OpenStreetMap contributors. 2017. Planet dump. Retrieved from https://planet.osm.org. https://www.openstreetmap.org.

[17] Daniele Quercia, Rossano Schifanella, and Luca Maria Aiello. 2014. The shortest path to happiness: Recommending beautiful, quiet, and happy routes in the city. In *Proceedings of the 25th ACM Conference on Hypertext and Social Media*. ACM, 116–125.

[18] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*. 693–701.

[19] Murray Rosenblatt. 1956. Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics* 27, 3 (09 1956), 832–837.

[20] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. 2018. Semi-parametric topological memory for navigation. In *Proceedings of the International Conference on Learning Representations*. https://openreview.net/forum?id=SygwwGbRW.

[21] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (Jan. 2016), 484–489.

[22] SpotCrime 2007. Retrieved from https://spotcrime.com/.
[23] Xin Wang, Wenhan Xiong, Hongmin Wang, and William Yang Wang. 2018. Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 37–53.
[24] Christopher John Cornish Hellaby Watkins. 1989. Learning from delayed rewards. Ph.D. Dissertation. King's College, Cambridge, UK. http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.
[25] Fangfang Wei, Shoufeng Ma, and Ning Jia. 2014. A day-to-day route choice model based on reinforcement learning. *Mathematical Problems in Engineering* 2014 (Sep. 2014), 14–32. DOI: https://doi.org/10.1155/2014/646548
[26] Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 3 (May 1992), 229–256. DOI: https://doi.org/10.1007/BF00992696
[27] Wenhan Xiong, Thien Hoang, and William Yang Wang. 2017. DeepPath: A reinforcement learning method for knowledge graph reasoning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP 2017)*. ACL, Copenhagen, Denmark.
[28] Yong Yang and Ana V. Diez-Roux. 2012. Walking distance by trip purpose and population subgroups. *American Journal of Preventive Medicine* 43, 1 (2012), 11–19.
[29] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive: Driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'10)*. ACM, New York, NY, 99–108.
[30] Wojciech Zaremba and Ilya Sutskever. 2015. Reinforcement learning neural turing machines. *CoRR* abs/1505.00521 (2015). arxiv:1505.00521 http://arxiv.org/abs/1505.00521.
[31] Mortaza Zolfpour-Arokhlo, Ali Selamat, Siti Zaiton Mohd Hashim, and Hossein Afkhami. 2014. Modeling of route planning system based on Q value-based dynamic programming with multi-agent reinforcement learning algorithms. *Engineering Applications of Artificial Intelligence* 29 (2014), 163–177.

Received January 2019; revised March 2020; accepted May 2020