The following publication Y. Yao, Z. Peng and B. Xiao, "Parallel Hyper-Heuristic Algorithm for Multi-Objective Route Planning in a Smart City," in IEEE Transactions on Vehicular Technology, vol. 67, no. 11, pp. 10307-10318, Nov. 2018 is available at https://doi.org/10.1109/TVT.2018.2868942.

Parallel Hyper-Heuristic Algorithm for Multi-objective Route Planning in a Smart City

Yuan Yao, *Member, IEEE,* Zhe Peng, *Student Member, IEEE,* and Bin Xiao, *Senior Member, IEEE*

***Abstract*—Most of commercial navigation products provide route planning service for users. However, they only consider a single metric such as distance, time or other costs, while ignoring a critical criterion: safety. In a smart city, people may prefer to find a safe walking route to avoid the potential crime risk as well as obtain a short distance. This problem can be specified as a Multi-Objective Optimization Problem (MOOP). Many methods were proposed in the past to solve the multi-objective route planning, the Multi-Objective Evolutionary Approach (MOEA) is considered as the most popular one. However, MOEA is non-optimized when used in a large-scale road network and becomes computationally expensive when handling a large population size. In this paper, we propose a Multi-Objective Hyper-Heuristic (MOHH) framework for walking route planning in a smart city. In the search framework, we design a set of low level heuristics to generate new routes. Moreover, we adopt reinforcement learning mechanism to select good low-level heuristics to accelerate searching speed. We further improve the Reinforcement Learning based Multi-Objective Hyper-Heuristic (RL-MOHH) algorithm and implement a parallel version (RL-PMOHH) on General Purpose Graphic Process Unit (GPGPU). Extensive experiments are conducted on the safety-index map constructed from the historical urban data of the New York city. Comprehensive experimental results show that the proposed RL-PMOHH is almost 173, 5.3 and 3.1 times faster than the exact multi-objective optimization (EMLS) algorithm, the RL-MOHH algorithm and the parallel NSGA-II (PNSGA-II) algorithm respectively. Moreover, both RL-MOHH and RL-PMOHH can obtain more than 80% Pareto optimal solutions in a large-scale road network.**

***Index Terms*—Multi-Objective Optimization; Hyper-Heuristics; Route Planning; Safety Index; Parallel Computing.**

I. Introduction

CRIME is a serious issue all over the world which gives rise to widespread fear in cities and brings huge losses to peoples' health and properties. The crime rate has increased over the past decades in many major cities due to the expansion of cities and increasing density of people. A recent report published by Federal Bureau of Investigation (FBI) [1] shows that there are estimated 1,197,704 violent crimes occurred nationwide, an increase of 3.9% from the 2014 estimate.

Manuscript received XX XXX. XXXX; revised XX XXX. XXXX; accepted XX XXX. XXXX. Date of publication XX XXX. XXXX; date of current version XX XXX. XXXX. This work was supported in part by the National Natural Science Foundation of China (No. 61502394, 61772446, 61876151, 61751208), the National Key Research and Development Program of China (No. 2017YFB1001900), the Fundamental Research Funds for the Central Universities (No. 3102017OQD097). (Corresponding author: Yuan Yao.)

Yuan Yao is with the School of Computer Engineering, Northwestern Polytechnical University, Xian, Shaanxi, 710072 China (e-mail:yao yuan@nwpu.edu.cn).

Zhe Peng and Bin Xiao are with The Hong Kong Polytechnic University, Hong Kong (e-mail: cszpeng, csbxiao@.comp.polyu.edu.hk).

Although safety is the first priority factor in guiding our daily lives, people often have very limited information about how safe/dangerous of an area in a city, especially for travellers who come to an unfamiliar city for the first time. Travellers often need a visiting walking route to inform them which region is safe in the city. However, most of commercial products (Google map, Baidu map and AutoNavi Map) only design an optimal path considering a single metric such as distance, time or other costs, while ignoring a critical criterion: safety. In a smart city, citizens and travellers urgently need an application to provide them a safe and short route rather than only considering a single metric. Some cities (e.g. New York City) provide safety heat map by pinning occurred crimes in a city map, but they are very rough safety heat map which only give a safety score of a district. In this paper, we first propose the concept of safety index to give fine-grained description of crime risk in a region. Then, we formulate the problem as a multi-objective route planning in a road network, with the goal of determining routes with low potential crime risk as well as short travel distance. While most of commercial map applications can provide optimal routes in terms of a single objective such as travel efficiency, distance or other costs, we take the crime risk into account and design a safety-aware route for users.

It is well known that multi-objective route planning is an NP-complete problem. Multi-Objective Evolutionary Approaches (MOEA) such as SPEA2 and NSGA-II are widely used in solving this problem. However, they becomes computational expensive when handling a large-scale road network and a large initial population. Therefore, we propose a novel algorithm, Reinforcement Learning based Multi-Objective Hyper-Heuristic (RL-MOHH) for multi-objective route planning in a large-scale smart city. Then, we implement parallel RL-MOHH (RL-PMOHH) on GPGPU based on the Compute Unified Device Architecture (CUDA) framework to further accelerate search speed. Finally, we develop a safe walking route planning application in the smartphone to conduct a real-world case study based on safety index map.

The contributions of the paper is as follows:

- *Low-level Heuristics:* We design a set of low-level heuristics based on hop transition to explore solution space, these low-level heuristics are easy-to-implement and cannot result in circular route or break point in the path;
- *Reinforcement Learning:* The reinforcement learning based heuristic selection can make solutions converging to the optimal Pareto front quickly and improve the solution quality significantly;
- *Parallel Implementation:* The parallel operation of

population initialization, heuristic selection and non-dominated sorting can greatly improve efficiency.

We conduct extensive experiments and show that the proposed RL-PMOHH algorithm is almost 173, 5.3 and 3.1 times faster than the exact multi-objective optimization (EMLS) algorithm, the RL-MOHH algorithm and the parallel NSGA-II (PNSGA-II) algorithm respectively. Moreover, both RL-MOHH and RL-PMOHH can obtain up to 80% Pareto optimal solutions in a large-scale road network. It is worth noting that the proposed reinforcement learning based parallel hyper-heuristic framework is a generic approach which is also suitable for other MOOP problems.

The rest of this paper is organized as follows. Section II briefly presents some relevant work. Section III gives the system overview and the problem definition; Section IV introduces the co-training learning based safety index inference; Section V details the proposed RL-MOHH for route planning; Section VI describes the parallel implementation of RL-MOHH and analyzes the time complexity; Section VII evaluates the proposed algorithm in a large-scale road network; Section VIII draws the conclusion and discusses the future work of this paper.

II. Related work

The multi-objective route planning is widely applied for QoS routing in the communication network, motion design in the robotic control and navigation in the transportation system. It is well known that the route planning with multi-objective is an NP-complete problem [2].

There are two main methods to solve the Multi-Objective Optimization Problem (MOOP): priori method and posterior method. The former one converts a multi-objective problem into a single-objective problem. Some work uses the priori method to solve the multi-objective route planning [3], [4], [5], [6], [7]. By using a weighted-sum function, all objectives are transformed into a single objective. The priori method only obtains a particular trade-off solution according to the weight vector. However, the weights for different objectives are determined by user preference before search. The bias will be imposed during the whole optimization process. Moreover, it is always difficult to determine the weight vector before search due to different magnitudes among multiple objectives. Therefore, the solution of the priori method is often non-optimized.

The posterior method first gets a set of Pareto optimal solutions. Then, the planner selects the most suitable one from the Pareto optimal set according to the user preference. The user preference can be also defined as a weight vector for multiple objectives. In the posterior method, the weight vector is determined after optimization process. The planner is able to know the range of each objective according to Pareto optimal solutions. Thus, it is easy to design the weight vector by normalizing all objectives into the same order of magnitude. The posterior method can be classified into two types: exact method and heuristic method.

Some exact posterior methods are based on the Dijkstra's algorithm to get exact Pareto optimal solutions for route planning. Martins proposes a label setting algorithm (MLS) for the shortest path problem [8]. MLS is a direct modification of the classic Dijsktra's algorithm in which the min operator is replaced by a dominance test. Then, an extension of Martins' label setting algorithm (EMLS) is proposed Gandibleux [9]. EMLS modifies the non-dominated test by concerning the weakly non-dominated labels since these labels can improve the determination of efficient paths. Although above algorithms can obtain optimal solutions, the computational complexity is extremely high when they are applied to a large-scale network. The Genetic Algorithm (GA) based on meta-heuristics has emerged as an effective method to get optimal (or near optimal) solutions for the MOOP problem [10]. Many multi-objective genetic algorithm are proposed such as the Niched Pareto Genetic Algorithm (NPGA) [11], the Multi-Objective Genetic Algorithms (MOGA) [12], the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [13] and the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [14]. Among all above algorithms, NSGA-II presents superior performance on not only benchmark problems, but also real-world applications[15]. Therefore, many work adopts NSGA-II in the multi-objective route planning problem from communication networks, to robotic systems and navigation services[16], [17], [18], [19]. However, the heuristic operations such as crossover and mutation in GA may lead to the circular route or break point in a route. Thus, they need extra overhead to repair the newly generated path. Therefore, these NSGA-II based multi-objective route planning methods only implemented and validated in small-scale networks. It is still challenging to obtain solutions as accurate as the exact Pareto optimal set in a large-scale graph network.

The hyper-heuristic approach is a generic method to determine the optimal heuristics by selecting from a set of easy-to-implement low-level heuristics [20]. Compared with the regular heuristic operations in GA, the well-designed low-level heuristics for route planning problem will not result in irrational or infeasible paths. Therefore, some work applies hyper-heuristics to the route planning [21], [22]. However, these work only take a single objective into account.

Due to the large population of initial solutions, GA and hyper-heuristic approaches are extremely time consuming, especially the non-dominated sorting algorithm to evaluate solutions for multi-objective optimization. In order to accelerate the search efficiency, some work adopt the improved NSGA-II with fast non-dominated sorting [23], [24], [25], and the other work implement the parallel GA and hyper-heuristic algorithms based on the naive non-dominated sorting in multi-core platform [26], [27], [28], [29]. Both methods still have the $O(MN^2)$ time complexity (M objectives and N solutions in the initial set).

III. System Overview

This section shows the whole system architecture and gives some notations and definitions that are required as background for other sections.

A. System Architecture

The safety-aware route planning system is composed of a database with historical crime data and safety index map, a GPU-based planning server and a smartphone application as shown in Fig. 1.

Fig. 1: System Architecture

As aforementioned, most of map services and commercial navigation products only provide users a short or time efficient route, while ignore a critical metric safety. Therefore, we propose a novel idea "Safety Index" which allows people know more about the crime risk distribution in a city. Moreover, we can give some safety-aware routes for citizens or travellers based on the safety index map.

Safety Index: Safety Index (SI) represents the potential crime risk of a region in a certain period. There are some very rough safety heat maps, such as NYC crime. These safety heat maps only provide safety index with very low granularity (e.g. the safety index of each district in a city). Thus, they cannot be used for route planning. A nature way to assess the safety index is to associate it with the crime rate (number of crimes per 100,000 residents). The SI is modeled as a function of two types of crime rates: the violent crime (murder, aggravated assault, rape and robbery) rates and the property crime (burglary, larceny, motor vehicle theft, and arson) rates [1]. We calculate the SI as follows:

$$SI = N\left[\alpha R_v^T + (1-\alpha) R_p^T\right], \tag{1}$$

where R_v^T and R_p^T are the violent crime rate and property crime rate in a certain time period T of day respectively, α is a weight which reflects the severity of the crime, $N[\bullet]$ is a function that scales the SI to an index between 0 and 100.

This model provides a static SI value. However, it is unable to compute the real-time SI with dynamic factors. Since an area may be safe in daytime but becomes unsafe at night, the SI can be seen as a spatial and temporal-related variable. Therefore, in a smart city, it is crucial to conduct urban safety analysis to build a periodically updated safety-index map in fine granularity for travellers and citizens.

In our work, we obtain the SI for each road segment according to the spatial and temporal-related features extracted from multiple cross-domain urban location-based data. The bigger the SI, the higher the probability of crime may occur on the street. With the fine-grained SI, we can construct the safety index map for a city. The SI values will be updated every two hours.

GPGPU-based planning server: In our work, we build a GPGPU-based planning server to provide navigation services based on Kepler CUDA. The server conducts route planning according to the updated safety index map and user-specified parameters.

GPGPU enables bidirectional flow exchange between CPU and GPU which has dramatically promoted the parallel computing, which are widely applied in different domains, such as scientific computation, medical treatment, finance, bioinformatics, artificial intelligence etc. [30]. CUDA is one remarkable many-core architecture. Kepler CUDA simplifies the creation of parallel programs and further revolutionizes the high performance computing mechanisms by offering much higher processing power. Kernel is a basic concept in CUDA as an executable task. When a kernel is launched, it always running on on grids of thread blocks. Commonly, threads are logically divided into blocks assigned to a specific multiprocessor which are scheduled in groups of 32 parallel threads, i.e. wraps. Streaming Multiprocessor of Kepler CUDA allows four wraps scheduled concurrently. The Kepler CUDA supports up to 64 wraps (i.e. 2048 threads). When a wrap is scheduled, all threads which perform the same instruction on different data will execute concurrently. Otherwise, treads will deviate resulting in wrap divergence that are executed sequentially.

Smartphone application: Finally, we develop an application on smartphone to provide users a safety-aware navigation service. The application uses heat map to show the current safety index distribution. It also gives two extreme routes (safest route and shortest route) and one optimal route based on user preference (user-specified weight vector).

B. Problem Definition

The route planning is defined as searching an optimal path p between the source and destination nodes in a graph $G(V, E)$ considering two objectives: distance and safety.

Here we give some notations:

- $V = \{v_1, v_2, ..., v_N\} \in V$ is a set of vertexes of graph G denoting road intersections (N is the number of vertexes);
- $E = \{e_1, e_2, ..., e_M\} \in E$ is a set of edges of graph G denoting road segments connecting two vertexes in V (M is the number of edges);
- A path p is a sequence of nodes $\{v_s, v_2, ...v_d\}$ from the source node v_s to the destination node v_d;
- $c_{i,j}$ is the non-negative cost of the j^{th} objective assigned for the i^{th} edge ($i \in [1, M]$ and $j \in [1, K]$, K is the number of objectives). Specifically, K equals to 2 in this paper, $c_{i,1}$ and $c_{i,2}$ are costs of the i^{th} edge for distance and safety index respectively;

- Let $f_j(p)$ be the total cost of all edges in path p for the j^{th} ($j \in [1, K]$) objective. Then, $f(p)$ is the objective vector for path p: $f(p) = \langle f_1(p), f_2(p), ...f_K(p)\rangle$.

A feasible path is composed of a sequence of road segments from the source node to the destination node. Obviously, there are too many feasible paths for a given source and destination pairs in the road network of a city. We need to find the optimal one from them. The "optimality" can be precisely defined in the single-objective route planning. For example, the shortest path is to find an optimal path such that the total length is minimized. However, this concept is not straightforward in the multi-objective problem. Commonly, we make a trade-off among all objectives and try to obtain Pareto optimal solutions.

Definition 1. Pareto Optimal Solution: A path p, when compared with any other path q, its objective vector satisfies the following conditions:

$$\begin{array}{l} f_j(p) \leq f_j(q), \forall j \in [1, K] \\ \exists i \in [1, K], f_i(p) < f_i(q). \end{array} \tag{2}$$

We say that p dominates q or q is dominated by p, which can be represented as $p \prec q$. If there is no solution dominates p, it is called a non-dominated solution, i.e. a Pareto optimal solution. The other dominated solutions are called feasible solutions.

Definition 2. Pareto Optimal Set (Pareto Front): A Pareto optimal set (denoted by $\mathbb{P}$) belonging to a feasible solution set (denoted by $\mathbb{F}$) satisfies following condition:

$$\forall p \in \mathbb{P}, \mathbb{P} \subseteq \mathbb{F}, \nexists x \in \mathbb{F}, x \prec p. \tag{3}$$

The representation of the Pareto optimal set in the objective space is Pareto front as shown in Fig. 2.

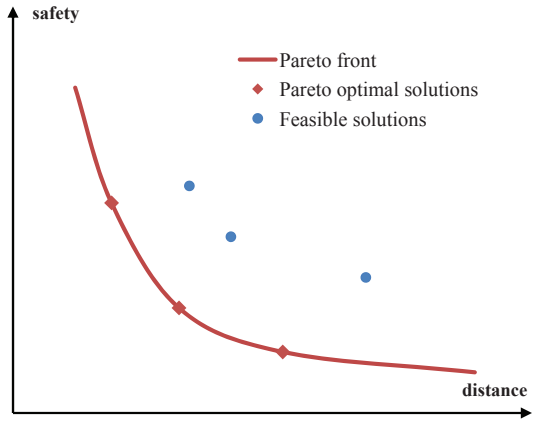


Fig. 2: The Pareto Front

Thus, the problem of multi-objective route planning is to find a safe route from the Pareto optimal set $\mathbb{P}$ such that the total cost C is minimized.

$$C = \min \sum_{j=1, p \in \mathbb{P}}^{K} \varpi_j f_j(p), \sum_{j=1}^{K} \varpi_j = 1, \tag{4}$$

where $\varpi_j \in [0, 1]$ is the weight of the j^{th} objective according to the user's preference which can be set using the smartphone application.

IV. Safety Index Modeling

In this section, we briefly introduce the U-safety system [31], [32] which is used multiple cross-domain urban data to infer safety index throughout a city with high fine granularity.

A. Feature Extraction and Feature Fusion

In this work, we collect a large volumes of cross-domain urban data in New York City. These data include urban map, housing rent and density data, population, positions of police stations, POIs, crime records, and taxi trajectories. In the U-safety system, an urban map is divided into disjointed grids (e.g., the granularity of 200m 200m). We infer the SI of each grid based on the collected urban data.

Feature Extraction: From the collected urban data, we extract various features that can be divided into two categories: spatially-related and temporally-related features.

The spatially-related features include police-station-related features F_s, POI-related features F_p, and housing-related features F_r.

The number of police stations and the distance to them in a region have a strong correlation with crime risk in the area. Therefore, we consider two features: the number of police stations in the grid (f_n) and the distance to the nearest police station for each grid (f_d). The POIs of a region often stands for the function and social environment of the region. It is a good complementary to SI inference. In this work, we select 12 types of POIs such as transportation spot, hospital, factory, shopping mall and supermarket, food and beverage, sport, park, education, entertainment, company, hotel and residential area ($f_p^1, f_p^2, ..., f_p^{12}$) as features for each grid. In some extent, housing illustrates the economic condition and population density. Therefore, we identify following two housing-related features: the number of housing in affecting region f_h and the average rent of housing in affecting region f_r.

The temporally-related features include traffic-related features F_t and human-mobility-related features F_h. The traffic flow has the influence on the urban safety. Thus, we extract the following two features from the spatial trajectories of vehicles traversing the grid in one hour: the number of vehicles in affecting region f_t and the average speed of vehicles f_v. Human mobility is also contributing to urban safety index inference. Therefore, we identify the following two features for each grid: the number of people arriving at the affecting region f_a, and the number of people leaving from the affecting region f_l. These features are extracted from the taxi trajectories which record the pickup and drop off locations in each trip.

Feature Fusion: Feature fusion is to generate more effective feature representation. In our work, we adopt Sparse Auto-Encoder (SAE) [33] to automatically learn the high-level feature representation of the input original features. The structure of SAE is shwon in Fig. 3.

In the encoder module, the input vector $\langle x_1, x_2, ..., x_N\rangle$ of SAE are the original features to be fused. Then, the input vector is mapped into a middle vector $\langle h_1, h_2, ..., h_K\rangle$ through a parameter matrix W_1. The middle vector is seen as the the feature representation of the input original features. Finally, the decode module reconstruct features $\langle y_1, y_2, ..., y_N\rangle$ from the

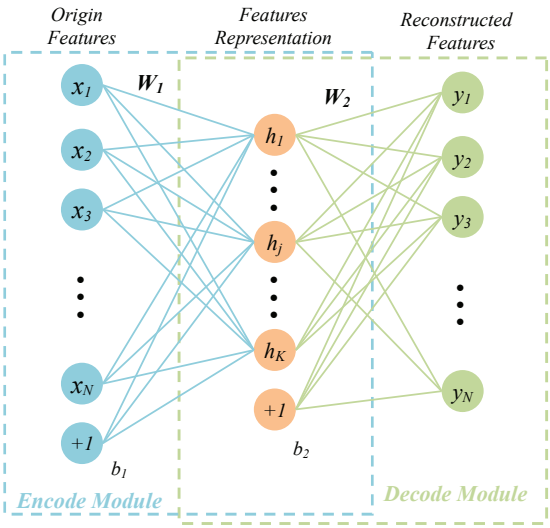


Fig. 3: The structure of SAE

feature representation through a parameter matrix W_2. After training the SAE model, we can derive the parameter matrix W_1, and thus, the feature representation can be obtained from the original features.

For the spatially-related features, the input original features vector X is in a form $\langle f_n, f_d, f_p^1, ..., f_p^{12}, f_h, f_r \rangle$. Different from the spatially-related features, the temporally-related features will change with the time in a day, thus each temporal feature is extended into 24 features extracted in each past hour. Hence, the input vector X of temporally-related features is in a form $\langle f_t^1, ..., f_t^{24}, f_v^1, ..., f_v^{24}, f_a^1, ..., f_a^{24}, f_l^1, ..., f_l^{24} \rangle$.

B. Co-Training Learning based SI Inference

We propose the co-training learning method to infer the safety index of a city. It adopts two separated classifiers to integrate the spatial and temporal features. The proposed co-training framework is shown in Fig. 4.

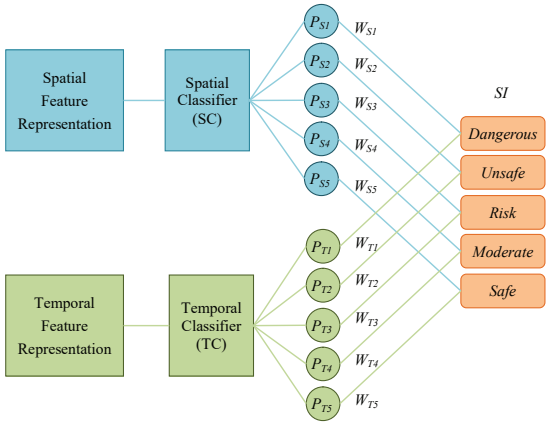


Fig. 4: The co-training learning framework

There are two classifiers in the co-training learning framework. One is the Spatial Classifier (SC) with the input vector of spatially-related features. It models the spatial relationship between crime events in various locations. The other one is the Temporal Classifier (TC), which takes temporally-related features as input to describe the temporal dependency of crime events for a location. Since each classifier can get a safety index, to generate final SI score, we merge these two safety index via a weight matrix $W = [W_S W_T]$. W_{Si} and W_{Ti} represent weights assigned to the i^{th} safety index predicted by SC and TC, which need to satisfy the condition $W_{Si} + W_{Ti} = 1$. The parameters in the weight matrix W are learned from the training data, which can be dynamically adjusted. The precision of safety index inference can reach 90.2% with the co-training learning method in U-safety system.

V. Proposed Route Planning Algorithm

In this section, we introduce the proposed parallel hyper-heuristic approach to multi-objective route planning.

A. Hyper-Heuristic framework

The hyper-heuristic operates on search space of heuristics rather than search space of solutions [20]. It aims to tackle the problem indirectly by selecting which method to adopt in which step of the solving process. The basic search framework is shown in Figure 5.

In this paper, we extend the hyper-heuristic search method to solve the multi-objective route planning problem. The procedure of the search method is as follows.

- First, we generate an initial set of feasible solutions (denoted by P_0) with population size of N_s. A feasible solution is a sequence of nodes $\{v_s, v_2, , v_k, , v_d\}$ from the source node v_s to the destination node v_d. In a given road network, each node v_k has an integer ID. So a solution is encoded as a sequence of integers. In other to quickly obtain the feasible solutions, we use A* algorithm [34] with different weights for each objective. Since we deal with the problem as a single-objective route planning for each individual solution and solve it using the priori method, most of solutions in the initial set are not Pareto optimal.
- Second, we choose a low-level heuristic for each path $p \in P_t$ based on the heuristic utility. This operation generates an intermediate solution set P'_t.
- Third, the planner will make comparison of all individuals in the union set composed of $P_t \bigcup P'_t$ and evaluate the quality of each solution. The non-dominated sort algorithm is adopted to conduct the solution comparison. The top N_s solutions are chosen to generate the next solution set P_{t+1}. The reinforcement learning algorithm is used to update the utility of selected low-level heuristic according to the optimality of the generated solution.
- Finally, all operations are executed iteratively until stopping conditions are met. In our work, stopping conditions are the maximum iteration number and the average crowding distance which is detailed in Section VII-A.
- To further improve the efficiency of the hyper-heuristic search approach, we parallelize the above framework on the GPGPU with the specific design of parallel non-dominated sort algorithm. The extended parallel hyper-heuristic approach is detailed in Section VI.

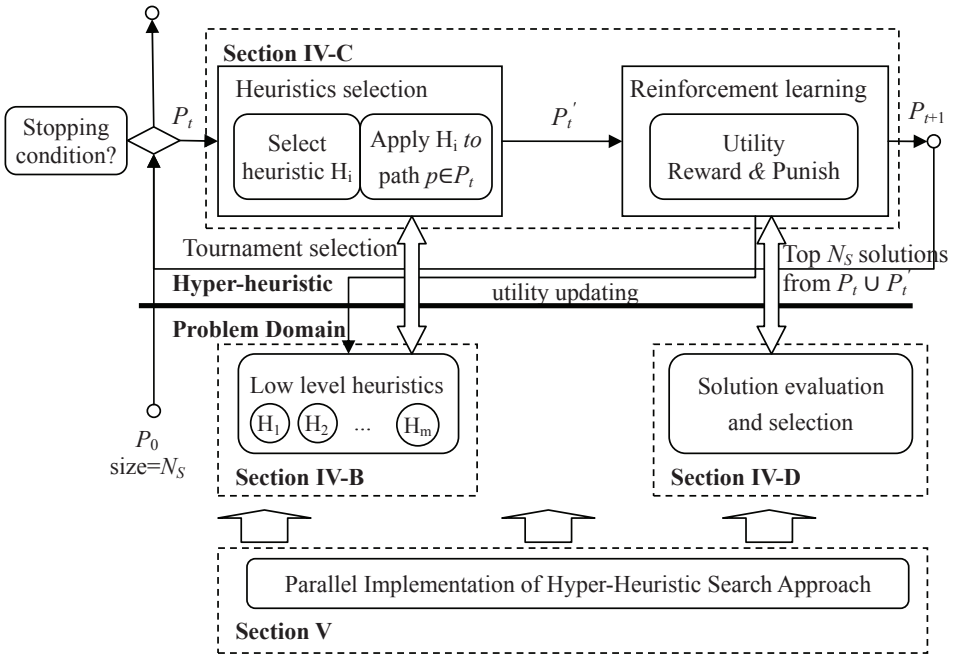


Fig. 5: The hyper-heuristic framework

B. Low-level Heuristics

Unlike GA designing three regular heuristic operators: reproduction, crossover and mutation, the low-level heuristics in hyper-heuristic search approach are implemented based on specific applications. We design 6 types of low-level heuristics as shown in Fig. 6. To accelerate the online search speed, a preliminary process of finding the neighboring hops for each node is done in the map construction. We only store at most 3-hop paths for one intersection. Thus, the low-level heuristic is to simply change a certain m-hop path to other feasible n-hop path (e.g. 1-hop path to 2-hop path). For each individual solution, the start intersection for hop transition is randomly chosen and we only take the forward hops of the feasible path into account. Applying the low-level heuristics on the feasible solution can gradually extend the search space based on the initial solution set P_0. Then, the reinforcement learning mechanism will guide the search towards a region of Pareto optimal solutions.

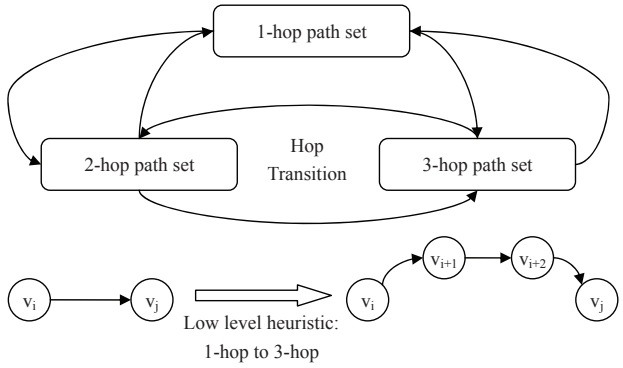


Fig. 6: Low-level heuristics

C. Reinforcement Learning based Heuristic Selection

We consider two heuristic selection methods as follows:

- **Random Selection**, choose a low-level heuristic function randomly in each iteration.
- **Reinforcement Learning**, assign a utility value to each low-level heuristic and choose the heuristic according to these values. The utility of each low-level heuristic is updated at each step based on the quality of the generated solution. The reinforcement learning mechanism is used to reward or punish the utility values.

We adopt a Q-learning algorithm for reinforcement learning based mechanism. The algorithm is composed of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$ per state and the $Q(s, a)$ as the state-action pairs and their corresponding values. These elements are detailed below:

State (s): is a four-tuple $\langle p, node_s, hops, direction\rangle$. p is a feasible path of the current feasible set P_t. $node_s$ is a randomly selected start intersection in path p for hop transition. $hops$ represents the number of edges from the start intersection to the end intersection $node_e$ in the path p for hop transition. $direction$ is the angle between the start intersection and the next intersection. We consider 8 directions including north, east, south, west, northeastern, southeastern, northwestern and southwestern, each covering a sector of 45 degrees.

Action (a): is the hop transition defined in Subsection IV-B, i.e. the low-level heuristic.

Q-Value ($Q(s, a)$): is the value of a certain state-action pairs, i.e. the utility of the low-level heuristic. The updating of Q-Value is defined as:

$$Q(s_t, a_t) = r + \gamma \max_{a \in \mathcal{A}} \{Q(s_{t+1}, a)\}, \tag{5}$$

where s_t is the current state and s_{t+1} is the new state after taking action a_t on the current state. r is the immediate reward

function. $\gamma \in [0, 1]$ (set to be 0.5 in experiments) is the discount factor which determines the importance of future rewards. γ is close to 0, the system will tend to consider the current rewards. Otherwise, the system will consider the future rewards.

Reward function is important in a reinforcement learning algorithm. The purpose of this function is to reward an improving state while punish the degraded one. In this paper, the reward value r is calculated according to the quality of the feasible path p in a state.

$$r = (\mathcal{F}(p_{t+1}) - \mathcal{F}(p_t)), \tag{6}$$

where $\mathcal{F}$ is the non-dominated front rank for path p which is detailed in Section V-D.

Initially, all low-level heuristics of each node are assigned with equal utility (Q-value). If we get a better solution, the reward function will return a positive value to enhance the utility of the selected low-level heuristic. Otherwise, the utility will be reduced by a negative value. Tournament selection is used to choose the low-level heuristics according to the utilities. The probability of being selected for a low-level heuristic is the proportion of its utility to the cumulative utilities of all heuristics. In order to build a better estimation of the optimal Q Matrix, we adopt the ϵ-greed policy for exploration in the training process. It means that we choose a random low level heuristic with probability ϵ, and choose a low-level heuristic with the highest utility with probability 1-ϵ.

D. Solution Evaluation and Selection

We use the non-dominated sorting algorithm to evaluate the quality of a given solution. As shown in Fig. 7, the Pareto optimal front is composed of a set of non-dominated solutions $\mathbb{P}_1$. We assign a front rank $\mathcal{F}(p) = 1$ for all path $p \in \mathbb{P}_1$. Then, we identify non-dominated paths in the rest solutions excluding paths $p \in \mathbb{P}_1$. This identified new non-dominated solutions are assigned the front rank $\mathcal{F}(p) = 2$. The sorting process is continued until all solutions are classified into a certain front.

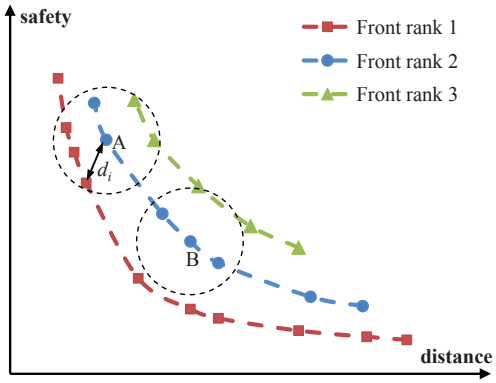


Fig. 7: Solution evaluation

We also adopt the concept of crowding distance to compare two solutions in the same front. The NSGA-II only computes the crowding distance considering solutions in the same non-dominated front, the less crowded points are selected to the next feasible solution set since it can preserve uniform spread of solutions and avoid many solutions from converging to a region. This cannot truly reveal the density of a point. For example, from Fig. 7, we notice that point A is more crowded than point B. But the NSGA-II will select point A because it is less crowded than B in the non-dominated front rank 2. To overcome this counter-intuitive solution selection, we redefine this distance as the average distance within k-nearest solutions.

$$\mathcal{D} = \frac{1}{k}\sum_{i=1}^{k} d_i. \tag{7}$$

Then, we can compare solutions according to these two attributes: non-dominated front rank $\mathcal{F}$ and crowding distance $\mathcal{D}$. We define a path p is superior to a path q if

$$\begin{aligned} &\mathcal{F}(p) < \mathcal{F}(q) \ OR \\ &\mathcal{D}(p) > \mathcal{D}(q) \ when \ \mathcal{F}(p) = \mathcal{F}(q). \end{aligned} \tag{8}$$

VI. Parallel Implementation on GPGPU

The framework of Hyper-Heuristic algorithm is very similar to Evolutionary Algorithm (EA). The same parts include initialization, metaheuristic-based search and evaluation. Many works implement the parallel EA on GPGPU according to concurrent operations on multiple individuals in the population. These steps can be easily parallelized due to the independence of different individual solutions. However, the Hyper-Heuristic framework has a non-dominated sorting which needs to compare every pair of solutions to identify the front rank for each individual. Some previous works try to design the parallel Hyper-Heuristic algorithm on multi-core platform. However, they implement the algorithm based on the naive non-dominated sorting which still has a high computational complexity. In this section, we design the parallel Hyper-Heuristic algorithm according to the fast non-dominated sorting. The fast non-dominated sorting in NSGA-II requires to calculate two variables for each solution i: (1) the domination count n_i (the number of solutions dominate the solution i); (2) the dominated set S_i (a set of solutions dominated by solution i). The detailed fast non-dominated sorting is as below:

Steps 1 to 14 find the first non-dominated front with complexity $O(KN_s^2)$ (K is the number of objectives and N_s is the population size of solutions). Steps 15 to 26 calculate the rest fronts iteratively with complexity $O(N_s^2)$. Therefore, the overall computational complexity of the fast non-dominated sorting is $O(KN_s^2)$. The fast non-dominated sorting requires to maintain S_i for each solution $P[i]$ to find higher-level front rank. Due to the different size of S_i in each iteration, it is hard to implement concurrent execution on GPGPU. So, we record the dominance relationship (stored in a $N_s \times N_s$ matrix Dom_h on host CPU/Dom_d on GPU device) among different solutions rather than obtain the set of S_i in each iteration. This approach identifies the first non-dominated front by comparing each pair of solutions and get the dominance relationship matrix. Then, it finds the next non-dominated front according to the matrix. Finally, the procedure continues until all fronts are

Algorithm 1 Fast Non-dominated Sorting Algorithm

Input: $P[N_s][K]$: Solution Matrix, K objectives
Output: $FR[N_s]$: Front Rank Array

1: $n_i \leftarrow 0$, $S_i \leftarrow \emptyset$ $(i \in [0, N_s])$;
2: **for** $j = 1$ to N_s **do**
3: **if** $j \neq i$ **and** $P[j] \prec P[i]$ **then**
4: $n_i \leftarrow n_i + 1$;
5: **else if** $j \neq i$ **and** $P[j] \succeq P[i]$ **then**
6: $S_i \leftarrow S_i \cup \{j\}$;
7: **end if**
8: **end for**
9: **for** $i = 1$ to N_s **do**
10: **if** $n_i = 0$ **then**
11: $FR[i] \leftarrow 0$;
12: **end if**
13: **end for**
14: $P_{fr} \leftarrow \{P[i] | FR[i] = 0\}$, $fr \leftarrow 1$;
15: **while** $P_{fr} \neq \emptyset$ **do**
16: $Q \leftarrow \emptyset$;
17: **for all** i $(P[i] \in P_{fr})$ **do**
18: **for all** j $(P[j] \in S_i)$ **do**
19: $n_j \leftarrow n_j - 1$;
20: **if** $n_j = 0$ **then**
21: $Q \leftarrow Q \cup \{j\}$, $FR[j] \leftarrow fr + 1$;
22: **end if**
23: **end for**
24: **end for**
25: $fr \leftarrow fr + 1$, $P_{fr} \leftarrow Q$;
26: **end while**
27: **return** $FR[N_s]$;

Algorithm 2 Parallel Non-dominated Sorting Algorithm

Input: $P_h[N_s][K]$: Solution Matrix on Host CPU
Output: $FR_h[N_s]$: Front Rank Array on Host CPU

1: $fr \leftarrow 1$; ▷ current front rank to identify
2: $FR_h[i] \leftarrow -1$; ▷ initialize FR_d on Host CPU
3: $Dom_h[N_s][N_s] \leftarrow 0$; ▷ initialize Dom_h on Host CPU
4: $P_d[N_s][K] \leftarrow P_h[N_s][K]$; ▷ initialize P_d on GPU device
5: $Dom_d[N_s][N_s] \leftarrow 0$; ▷ initialize Dom_h on GPU device
6: $\mathcal{K}_{CUDAQuickSort}(P_d[N_s][0])$; ▷ launch kernel CUDAQuickSort on GPU device to sort P_d according to objective 1
7: $\_synchthreads()$; ▷ synchronize GPU Device
8: $P_h[N_s][K] \leftarrow P_d[N_s][K]$;▷ copy P_d to P_h on Host CPU
9: $\mathcal{K}_{FFR}(P_d[N_s][K], FR_d[N_s], fr)$; ▷ launch kernel PNDS on GPU device to perform parallel non-dominated sort
10: $\_synchthreads()$; ▷ synchronize GPU Device
11: $FR_h[N_s] \leftarrow FR_d[N_s]$; ▷ copy FR_d to FR_h on Host CPU
12: $Dom_h[N_s][N_s] \leftarrow Dom_d[N_s][N_s]$; ▷ copy Dom_d to Dom_h on Host CPU
13: $n_{assign} \leftarrow CountAssigned(Dom_h[N_s][N_s])$; ▷ the number of solutions assigned by front rank
14: **while** $n_{assign} < N_s$ **do** ▷ not all solutions are assigned a front rank
15: $\mathcal{K}_{AFR}(FR_d[N_s], Dom_d[N_s][N_s], fr, n_{assign})$; ▷ launch kernel PNDS on GPU device to perform parallel non-dominated sort
16: $\_synchthreads()$; ▷ synchronize GPU Device
17: $fr \leftarrow fr + 1$;
18: **end while**

identified. Algorithm 2 gives the pseudo-code of the parallel non-dominated sorting.

Algorithm 2 is the main thread on Host CPU to assign the front rank for each solution. In Step 6, we adopt CUDAQuickSort [35] to sort P_d using the objective 1 as key. The complexity of CUDAQuickSort is $O(\frac{N_s}{N_T} log(N_s))$. N_T is the maximum number of threads executed concurrently for a task. In Step 9, we launch the Kernel FFR (as shown in Algorithm 3) to identify the first front rank of solutions by checking dominance among objectives $2 \sim K$. This requires $(K-1)N_s^2$ comparisons concurrently executed by N_T threads. Therefore, the complexity of Kernel FFR is $O(\frac{1}{N_T}(K-1)N_s^2)$. In Step 15, we launch the Kernel AFR (as shown in Algorithm 4) to identify the next front rank. It needs N_s computations with complexity $O(\frac{1}{N_T}N_s)$. In the worst case, suppose that each solution belongs to a front rank, the while loop will be executed N_s times. Thus, the overall complexity of finding the rest front ranks is $O(\frac{1}{N_T}N_s^2)$. Since in our platform, the number of concurrently executed threads N_T supported by GPGPU is much greater than the solution population size N_s, the complexity of parallel non-dominated sorting is $O(log(N_s) + KN_s)$.

VII. Evaluation

In this section, we carry out experiments to evaluate the performance of the proposed approach. We compare the Random Selection based MOHH (RS-MOHH), the Reinforcement Learning based MOHH (RL-MOHH) as well as the parallel version of both algorithms (RS-PMOHH and RL-PMOHH) with the NSGA-II [14] and Parallel NSGA-II (PNSGA-II) [29] in terms of both computation time and solution quality. Moreover, the Extended Martins' Label Setting (EMLS) [9] algorithm is used to get the exact Pareto optimal solutions.

A. Experiment setup

We carry out experiments on an Intel SandyBridge E5-2609 (4-core, 2.4GHz) CPU with 32GB main memory (run the serial program) and a GPU NVIDIA Tesla Kepler k20c (run the parallel program). The GPU has total 2496 CUDA cores and 5GB GDDR5 global memory. The compute capability is 3.5. The safety index is established by analyzing the relationship between the crime risk and various urban data. The urban data includes historical crime data, taxi trip data, housing rent price, population, position of police stations and other POIs (Point Of Interests) [31]. Then, a safety index map is constructed based on a large amount of data collected form New York city as shown in Fig. 8a. For each road segment of the city, it has a value ranging from 0 to 1 that indicates the safe level of the road (0 is very safe and 1 is very dangerous).

We apply all approaches to 3 network instances from small to large size including a partial region of Manhattan (RN_1: 5382 intersections and 6483 road segments), Manhattan

Algorithm 3 Kernel to identify the first front rank

Kernel $FFR(P_d[N_s][K], FR_d[N_s], Dom_d[N_s][N_s], fr)$
$i \leftarrow blockIdx.x \times blockDim.x + threadIdx.x$;
if $FR_d[i] = -1$ **then**
  $n[i] \leftarrow 0$; ▷ $n[i]$ is the number of solutions that dominate $P_d[index]$
  for $j \leftarrow 0$ to i **do**
    $Dom_d[j][i] = Check(P_d[j][1:K-1], P_d[i][1:K-1])$; ▷ check dominance between $P_d[j]$ and $P_d[i]$ according to objectives $2 \sim K$
    if $Dom_d[j][i] = 1$ **then** ▷ $P_d[j] \succeq P_d[i]$
      $n[i] \leftarrow n[i] + 1$;
    end if
  end for
  if $n[i] = 0$ **then**
    $FR_d[i] \leftarrow fr$;
  end if
end if
EndKernel

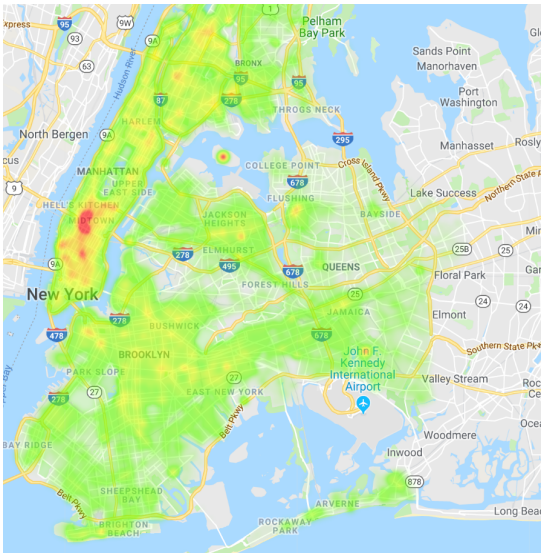
Algorithm 4 Kernel to assign all front ranks

Kernel $AFR(FR_d[N_s], Dom_d[N_s][N_s], fr, n_{assign})$
$i \leftarrow blockIdx.x \times blockDim.x + threadIdx.x$;
if $FR_d[i] = fr$ **then** ▷ $P_d[i]$ is assigned front rank
  for j=0 to N_s **do**
    if $Dom_d[j][i] > 0$ **then**
      $Dom_d[j][i] \leftarrow Dom_d[j][i] - 1$;
    end if
  end for
  for j=0 to N_s **do**
    $n[j] = sum(Dom_d[j][0:N_s-1])$
    if $n[j] = 0$ **then**
      $FR_d[j] \leftarrow fr + 1$;
      $n_{assign} \leftarrow n_{assign} + 1$;
    end if
  end for
end if
EndKernel

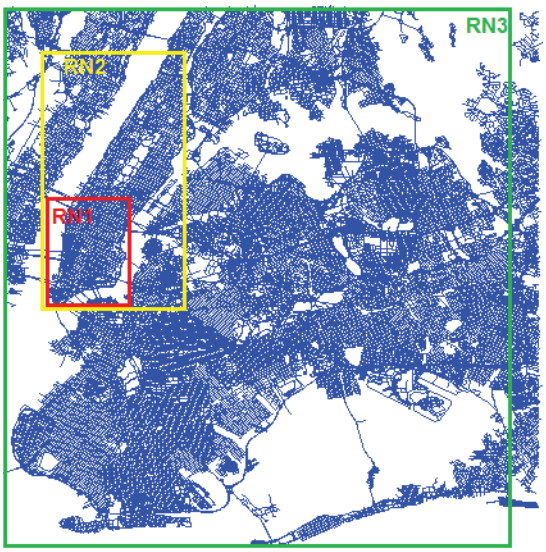
district (RN_2: 27051 intersections and 33358 road segments) and New York city (RN_3: 125383 intersections and 160124 road segments) as shown in Fig. 8b.

The utilities are trained based on 200000 source and destination pairs in the map before experiments. Thus, we do not activate the learning process in the on-line planning.

To make the comparison as fair as possible, all algorithms adopt the same initial population size (N_s=200) and exactly the same initial routes. They will be stopped until either the loop reaches the maximum iteration number (I_{MAX}=200) or only one non-dominated front rank exists in the population and the $\Delta\mathcal{D}$ is smaller than a given predefined threshold (set to be 10 in experiments). To eliminate the impact of different magnitudes on metric measurements, all solutions of each objective are normalized to the range of $[0, 1]$ according to the min-max normalization.



(a) Safety index map



(b) Abstraction of road network

Fig. 8: Map construction

$$\Delta\mathcal{D} = \overline{\mathcal{D}}\left(P_{t+1}\right) - \overline{\mathcal{D}}\left(P_t\right), \tag{9}$$

where $\overline{\mathcal{D}}(P)$ is the average crowding distance of all individuals in the population P.

The detailed parameter settings are summarized in Table I.

TABLE I: Parameter settings

NSGA-II		MOHH	
Parameters	**Values**	**Parameters**	**Values**
population size	200	population size	200
maximum iteration	200	maximum iteration	200
crossover rate	0.9	exploration rate ϵ	0.1
mutation rate	0.2	discount factor γ	0.5

B. Metrics

We consider two metrics: efficiency and solution quality.

Efficiency: We use the computation time to evaluate the efficiency of the proposed method. The computation time of

EMLS has taken as the baseline. We compute the speed-up ratio (denoted by ϕ) of all other methods to the EMLS. The ratio is defined as:

$$\phi = T_{EMLS}/T_x. \tag{10}$$

Solution Quality: The optimal solution ratio (denoted by δ) is adopted to evaluate the accuracy of the proposed method. It defined as the proportion of exact optimal solutions to the total number of obtained solutions.

$$\delta = N_{exact}/N_{total}. \tag{11}$$

To evaluate the solution quality in a fine-grained manner, we define the average error (denoted by ρ) to measure the extent of convergence to exact Pareto optimal solutions. It is the average distance of σ_i, which is the minimum Euclidean distance from each obtained solution to the real Pareto optimal solution as shown in Fig. 9.

$$\rho = \frac{1}{N_s}\sum_{i=1}^{N_s}\sigma_i, \tag{12}$$

where N_s is the size of the initial population.

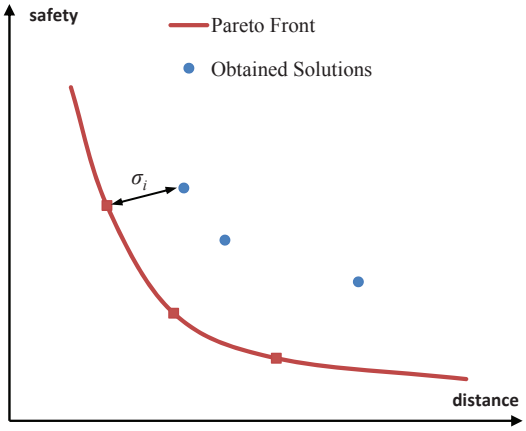


Fig. 9: The minimum distance to the Pareto optimal front

C. Results

We carry out experiments on each network instance with 50 independent runs and calculate an average of obtained results. For an individual run, the source and destination nodes are randomly selected from the network.

To illustrate the detailed solving processes of these algorithm, we plot the growth of solutions in the first non-dominated front rank versus the iterations in an individual run. From Fig. 10, we observe that the proposed RL-PMOHH can quickly generate better solutions and the number of solutions in the first non-dominated front rank increases up to the whole population size rapidly. However, the curve of RS-PMOHH presents linear growth trend which is much slower than RL-PMOHH. This proves that the pre-trained utilities by reinforcement learning can effectively guide the planner toward to the optimal solutions.

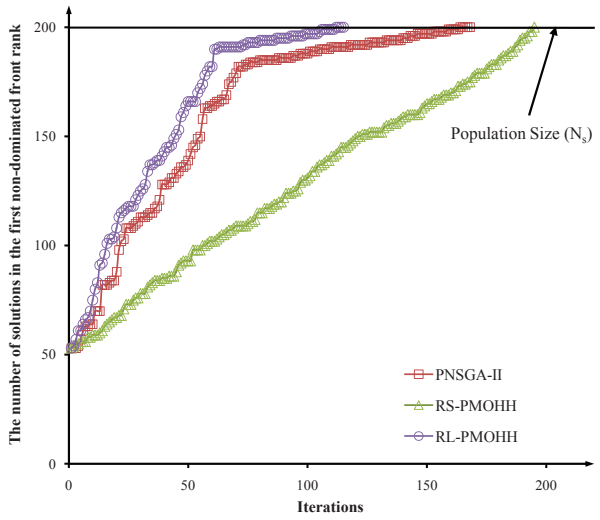


Fig. 10: The number of solutions in the first non-dominated front rank

Fig. 11 gives all solutions obtained when those algorithms meet the stopping condition. The figure shows that RL-PMOHH gets better accuracy in converging to the exact Pareto optimal front.

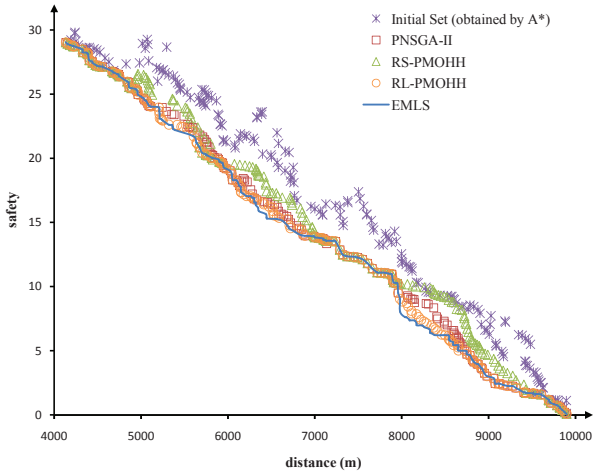


Fig. 11: Comparison of solutions

Table II and III show the comparison of the efficiency and the accuracy. The experiment results show that the proposed RL-MOHH performs the best among serial programs in all network instances. It is almost 33 and 1.3 times faster than EMLS and NSGA-II respectively. RL-PMOHH also outperforms other parallel programs, which achieves about 173 and 3.1 times speedup than EMLS and PNSGA-II respectively. Comparing with the RL-MOHH, the RL-PMOHH with parallel implementation on GPGPU yields an average 5.3 times performance improvement. Moreover, both RL-MOHH and RL-PMOHH can obtain up to 80% Pareto optimal solutions even in a large-scale road network.

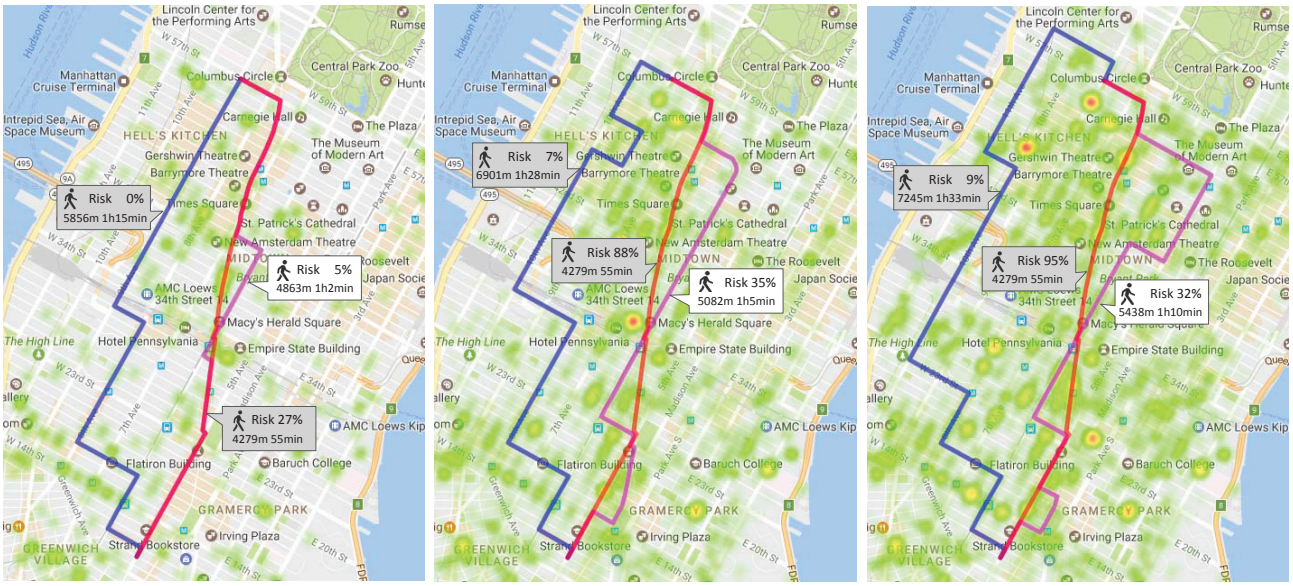
We develop a safe walking route planning application in the smartphone to conduct a real-world case study. As shown in Fig. 12, the application will give 3 options: two extreme routes (the shortest one and the safest one) and one trade-off route

TABLE II: Comparison of efficiency

Network	Computation Time (msec) / Speed-up Ratio (ϕ)						
	EMLS	**NSGA-II**	**RS-MOHH**	**RL-MOHH**	**PNSGA-II**	**RS-PMOHH**	**RL-PMOHH**
RN_1	14376 / 1	580($\pm$20.8) / 24.79	766($\pm$34.1) / 18.77	445($\pm$25.3) / 32.31	261($\pm$15.2) / 55.08	296($\pm$16.8) / 48.57	85($\pm$8.9) / 169.13
RN_2	48383 / 1	1928($\pm$72.5) / 25.09	2428($\pm$97.6) / 19.93	1580($\pm$61.4) / 30.62	865($\pm$44.5) / 55.93	987($\pm$48.6) / 49.02	281($\pm$17.1) / 172.18
RN_3	151110 / 1	6089($\pm$215.2) / 24.82	8155($\pm$235.8) / 18.53	4202($\pm$165.3) / 35.96	2602($\pm$96.2) / 58.07	2915($\pm$108.8) / 51.84	847($\pm$42.7) / 178.41

TABLE III: Comparison of accuracy

Network	Optimal Solution Ratio (δ) / Average Error (ρ) $\times 10^{-2}$						
	EMLS	**NSGA-II**	**RS-MOHH**	**RL-MOHH**	**PNSGA-II**	**RS-PMOHH**	**RL-PMOHH**
RN_1	100% / 0	68.5%($\pm$2.5%) / 3.83	46.3%($\pm$2.1%) / 5.51	86.2%($\pm$1.8%) / 1.37	67.3%($\pm$2.3%) / 3.61	45.7%($\pm$2.1%) / 5.31	87.5%($\pm$1.4%) / 1.34
RN_2	100% / 0	65.2%($\pm$2.3%) / 3.92	45.4%($\pm$2.1%) / 6.73	82.4%($\pm$2.2%) / 1.82	64.5%($\pm$2.5%) / 4.03	43.6%($\pm$1.7%) / 6.84	83.4%($\pm$1.9%) / 1.62
RN_3	100% / 0	62.8%($\pm$2.5%) / 4.66	42.1%($\pm$2.0%) / 7.69	80.9%($\pm$2.0%) / 2.76	62.5%($\pm$1.9%) / 4.82	40.6%($\pm$2.2%) / 7.49	81.6%($\pm$1.5%) / 2.11



(a) Routes for travelling at 10:00 (b) Routes for travelling at 20:00 (c) Routes for travelling at 23:00

Fig. 12: Walking routes planned by the smartphone application for travelling at different time periods

according to user's preference (the default weight vector is $\langle 0.4, 0.6 \rangle$, and the vector can be set by users before planning). Fig. 12a - 12c show planning results at different time of Day. The crime risk increases greatly when the time gets later in the day. The application can change the route according to updated safety index of the map and well support users to avoid roads with high crime risk.

VIII. Conclusion

In this paper, we consider the safety index in navigation services, and propose a Reinforcement Learning based Multi-Objective Hyper-Heuristic (RL-MOHH) scheme for route planning in a smart city. A set of easy-to-implement low-level heuristics is designed based on the hop transition to explore the solution space. A Q-learning algorithm is operating on the heuristic space to guide the search toward to the Pareto optimal front. Moreover, we implement a parallel RL-MOHH (RL-PMOHH) algorithm on GPGPU to accelerate search speed. Furthermore, we develop the safe walking route planning application in smartphone and carry out a real-world case study on the safety index map constructed by a large amount historical urban data from New York city. Extensive experimental results show that the proposed algorithm is superior than NSGA-II/PNSGA-II on both the efficiency and optimality. However, the proposed MOHH/PMOHH frameworks still need a lot of time to train utilities of the low level heuristics. In our future work, we will implement a more efficient online learning algorithm to improve continuous learning ability of MOHH/PMOHH.

References

[1] FBI, "Uniform crime report crime in the united states, 2015," Federal Bureau of Investigation, Tech. Rep., 2015.

[2] R. G. Garroppo, S. Giordano, and L. Tavanti, "A Survey on Multi-Constrained Optimal Path Computation: Exact and Approximate Algorithms," *Computer Networks*, vol. 54, pp. 3081–3107, 2010.

[3] Z. Li, I. Kolmanovsky, E. Atkins, J. Lu, D. Filev, and J. Michelini, "Cloud Aided Safety-based Route Planning," in *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics*, 2014, pp. 2495–2500.

[4] Z. Li, I. V. Kolmanovsky, E. M. Atkins, J. Lu, D. P. Filev, and Y. Bai, "Road Disturbance Estimation and Cloud-Aided Comfort-based Route Planning," *IEEE Transactions on Cybernetics*, vol. 47, no. 11, pp. 3879–3891, 2017.

[5] Y. Zeng, P. Zhang, and Y. Luo, "A Multi-Constrained Routing Optimization Algorithm in the IP Networks," in *Proc. of IEEE ICNC*, 2015, pp. 314–318.

[6] B. Zhang, J. Hao, and H. T. Mouftah, "Bidirectional Multi-Constrained Routing Algorithms," *IEEE Transactions on Computers*, vol. 63, no. 9, pp. 2174–2186, 2014.

[7] S. V. Mallapur, S. R. Patil, and L. V. Agarkhed, "Multi-constrained Reliable Multicast Routing Protocol for MANETs," in *Proc. of COMSNETS*, 2016.

[8] E. Martins, "On a Multicriteria Shortest Path Problem," *European Journal of Operational Research*, pp. 236–245, 1984.

[9] X. Gandibleux, F. Beugnies, and S. Randriamasy, "Martins Algorithm Revisited for Multi-objective Shortest Path Problems with a Maxmin Cost Function," *4OR - A Quarterly Journal of Operations Research*, vol. 4, no. 1, pp. 47–59, 2006.

[10] A. Konak, D. W. Coit, and A. E. Smit, "Multi-objective Optimization using Genetic Algorithms: A Tutorial," *Reliability Engineering and System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[11] J. Horn, N. Nafpliotis, and D. E. Goldberg, "A Niched Pareto Genetic Algorithm for Multiobjective Optimization," in *Proc. of IEEE ICEC*, 1994, pp. 82–87.

[12] T. Murata and H. Ishibuchi, "MOGA: Multi-Objective Genetic Algorithms," in *Proc. of the IEEE International Conference on Evolutionary Computation*, 1995, pp. 289–294.

[13] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," Swiss Federal Institute of Technology, Tech. Rep., 2001.

[14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Compution*, vol. 6, no. 2, pp. 182–197, 2002.

[15] S. Huband, P. Hingston, L. Barone, and L. While, "A Review of Multiobjective Test Problems and a Scalable Test Problem Toolkit," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 5, pp. 477–506, 2006.

[16] H. Chenji and R. Stoleru, "Pareto Optimal Cross Layer Lifetime Optimization for Disaster Response Networks," in *Proc. of IEEE ICCSN*, 2014.

[17] M. Hadded, R. Zagrouba, A. Laouiti, P. Muhlethaler, and L. A. Saidane, "A Multi-Objective Genetic Algorithm-based Adaptive Weighted Clustering Protocol in VANET," in *Proc. of ICEC*, 2015, pp. 994–1002.

[18] J. Zeng, X. Zhang, and X. Guan, "Path Planning for General Aircrafts Under Complex Scenarios Using an Improved NSGA-II Algorithm," *Journal of Computational Information Systems*, vol. 9, no. 16, pp. 6545–6553, 2013.

[19] P. Mulik, "Optimal Trajectory Planning of Industrial Robot with Evolutionary Algorithm," in *Proc. of ICCPEIC*, 2015, pp. 256–263.

[20] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. O. zcan, and R. Qu, "Hyper-Heuristics: A Survey of the State of the Art," *Journal of the Operational Research Society*, vol. 64, pp. 1695–1724, 2013.

[21] E. Akar, H. R. Topcuoglu, and M. Ermis, "Hyper-Heuristics for Online Uav Path Planning Under Imperfect Information," *EvoApplications*, pp. 741–752, 2014.

[22] N. R. Sabar, X. J. Zhang, and A. Song, "A Math-Hyper-Heuristic Approach for Large-Scale Vehicle Routing Problems with Time Windows," in *Proc. of ICEC*, 2015, pp. 830–837.

[23] G. Liu, J. Zhang, R. Gao, and Y. Shang, "An Improved Parallel Adaptive Genetic Algorithm Based on Pareto Front for Multi-objective Problems," in *Proc. of KAM*, 2009, pp. 212–215.

[24] M. K. Marichelvam, T. Prabaharan, and X. S. Yang, "A Discrete Firefly Algorithm for the Multi-Objective Hybrid Flowshop Scheduling Problems," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 301–305, 2014.

[25] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary Multi-Objective Workflow Scheduling in Cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1344–1357, 2016.

[26] K. Seridi, L. Jourdan, and E.-G. Talbi, "Parallel Hybrid Metaheuristic for Multi-objective Parallel Hybrid Metaheuristic for Multi-objective," in *Proc. of IPDPS Workshop*, 2012, pp. 625–633.

[27] Z. Shen, K. Wang, and F.-Y. Wang, "GPU based Non-dominated Sorting Genetic Algorithm-II for Multi-objective Traffic Light Signaling Optimization with Agent Based Modeling," in *Proc. of ITSC*, 2013, pp. 1840–1845.

[28] C.-J. Ye and M.-X. Huang, "Multi-Objective Optimal Power Flow Considering Transient Stability Based on Parallel NSGA-II," *IEEE Transactions on Power Systems*, vol. 30, no. 2, pp. 857–866, 2015.

[29] S. Gupta and G. Tan, "A Scalable Parallel Implementation of Evolutionary Algorithms for Multi-Objective Optimization on GPUs," in *Proc. of CEC*, 2015, pp. 1567–1574.

[30] N. Papangelopoulos, D. Vlachakis, A. Filntisi, and P. Fakourelis, "State-of-the-Art GPGPU Applications in Bioinformatics," *International Journal of Systems Biology and Biomedical Technologies*, vol. 2, no. 4, pp. 24–48, 2013.

[31] Z. Peng, B. Xiao, Y. Yao, J. Guan, and F. Yang, "U-Safety: Urban Safety Analysis in A Smart City," in *Proc. of IEEE ICC*, 2017, pp. 1–6.

[32] Y. Yao, Z. Peng, B. Xiao, and J. Guan, "An efficient learning-based approach to multi-objective route planning in a smart city," in *Proc. of IEEE ICC*, 2017, pp. 1–6.

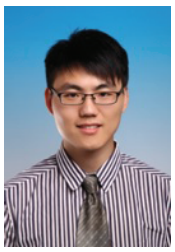
[33] I. Sutskever, O. vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Proc. of IEEE NIPS*, 2014, pp. 3104–3112.

[34] P. E. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, 1968.

[35] E. Manca1, A. Manconi, A. Orro, G. Armano1, and L. Milanesi, "Cuda-Quicksort: An Improved GPU-based Implementation of Quicksort," *Concurrency Computation: Practice and Experience*, vol. 28, pp. 21–43, 2016.

Yuan Yao (S'09-M'13) received the B.S., M.S. and Ph.D. degrees in computer science from Northwestern Polytechnical University, Xian, China, in 2007, 2009 and 2015, respectively. Prior to joining the faculty at NPU, he was a Postdoctoral Researcher in the Department of Computing at Polytechnic University, Hong Kong. He is currently an Associate Professor in the School of Computer Science, Northwestern Polytechnical University. His research interests are in the area of realtime and embedded system, swarm intelligence, cross-layer design in vehicular ad hoc networks, and security in vehicular networks.

Bin Xiao (S'01-M'04-SM'11) is currently an Associate Professor in the Department of Computing, The Hong Kong Polytechnic University. Dr. Xiao received the B.Sc and M.Sc degrees in Electronics Engineering from Fudan University, China, and Ph.D. degree in computer science from University of Texas at Dallas, USA. His research interests include distributed wireless systems, network security, and software-defined networks (SDN). Dr. Xiao has published more than 100 technical papers in international top journals and conferences. Currently, he is the associate editor of the Journal of Parallel and Distributed Computing (JPDC) and Security and Communication Networks (SCN). He is the IEEE Senior member, ACM member and the recipient of the best paper award of the international conference IEEE/IFIP EUC-2011.



Zhe Peng (S'11) is a Ph.D. candidate in Department of Computing, The Hong Kong Polytechnic University. He received his BSc degree in Communication Engineering from Northwestern Polytechnical University, and MSc degree in Electronic Engineering and Information Science from University of Science and Technology of China, in 2010 and 2013 respectively.His research interests include mobile computing, data mining, machine learning, and blockchain.