DeResolver: A Decentralized Conflict Resolution Framework with Autonomous Negotiation for Smart City Services

YUKUN YUAN, Stony Brook University, USA
MEIYI MA, Vanderbilt University, USA
SONGYANG HAN, University of Connecticut, USA
DESHENG ZHANG, Rutgers University, USA
FEI MIAO, University of Connecticut, USA
JOHN A. STANKOVIC, University of Virginia, USA
SHAN LIN, Stony Brook University, USA

As various smart services are increasingly deployed in modern cities, many unexpected conflicts arise due to various physical world couplings. Existing solutions for conflict resolution often rely on centralized control to enforce predetermined and fixed priorities of different services, which is challenging due to the inconsistent and private objectives of the services. Also, the centralized solutions miss opportunities to more effectively resolve conflicts according to their spatiotemporal locality of the conflicts. To address this issue, we design a decentralized negotiation and conflict resolution framework named DeResolver, which allows services to resolve conflicts by communicating and negotiating with each other to reach a Pareto-optimal agreement autonomously and efficiently. Our design features a two-step self-supervised learning-based algorithm to predict acceptable proposals and their rankings of each opponent through the negotiation. Our design is evaluated with a smart city case study of three services: intelligent traffic light control, pedestrian service, and environmental control. In this case study, a data-driven evaluation is conducted using a large dataset consisting of the GPS locations of 246 surveillance cameras and an automatic traffic monitoring system with more than 3 million records per day to extract real-world vehicle routes. The evaluation results show that our solution achieves much more balanced results, i.e., only increasing the average waiting time of vehicles, the measurement metric of intelligent traffic light control service, by 6.8% while reducing the weighted sum of air pollutant emission, measured for environment control service, by 12.1%, and the pedestrian waiting time, the measurement metric of pedestrian service, by 33.1%, compared to priority-based solution.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; • **Computing methodologies → Modeling and simulation;**

Additional Key Words and Phrases: Smart services, conflicts across services, decentralized resolution, multiple services negotiation

This work was partially supported by NSF 1849238, 1932223, 1951890, 1952096, 2003874, 2047822, and NSF CNS-1553273 (CAREER). A conference version of this work was published in Proceedings of the 12th International Conference on Cyber-Physical Systems, May 2021 [54].

Authors' addresses: Y. Yuan and S. Lin, Stony Brook University, 100 Nicolls Rd, Stony Brook, NY, 11794; emails: {yukun.yuan, shan.x.lin}@stonybrook.edu; M. Ma, Vanderbilt University, Nashville, TN, 37240; email: meiyi.ma@vanderbilt.edu; S. Han and F. Miao, University of Connecticut, 371 Fairfield Way, Unit 4155, Storrs, CT, 062683; emails: {songyang.han, fei.miao}@uconn.edu; D. Zhang, Rutgers University, 57 US-1, New Brunswick, NJ, 08901; email: desheng@cs.rutgers.edu; J. Stankovic, University of Virginia, 85 Engineer's Way, Charlottesville, VA, 22903; email: stankovic@cs.virginia.edu.

2378-962X/2022/11-ART29 $15.00
https://doi.org/10.1145/3529096

ACM Reference format:
Yukun Yuan, Meiyi Ma, Songyang Han, Desheng Zhang, Fei Miao, John A. Stankovic, and Shan Lin. 2022. DeResolver: A Decentralized Conflict Resolution Framework with Autonomous Negotiation for Smart City Services. *ACM Trans. Cyber-Phys. Syst.* 6, 4, Article 29 (November 2022), 27 pages.
https://doi.org/10.1145/3529096

1 INTRODUCTION

The number of smart services has been increasing in modern cities. These services aim to improve the quality of urban lives, e.g., safety, wellbeing, and environmental quality. Examples of smart services include intelligent traffic light control [16, 51], air quality control [7], electric taxi scheduling [55, 56], ambulance management [18], and so on. However, city managers are facing more and more potential conflicts across the growing number of deployed services [26, 31, 41]. For example, services may have different actions on the same devices due to self-interested objectives. Another example is that when acting alone, some city services are fine, but when combined they may be detrimental, e.g., to the environment. Such conflicts have significant impacts on the citizens.

Importantly, how to deal with potential conflicts across services is still under-explored. There exist several papers on resolving conflicts across smart services [24, 26, 32, 34, 47]. Reference [47] uses a client-server architecture to choose one conflict resolution considering each application's specific performance requirements, e.g., resource consumption, and quality of services. Reference [32] proposes a centralized conflict resolution for multiple city services by using an operation center to determine which actions are approved. References [24, 34] resolve conflicts in the smart home by assigning different priorities to smart applications based on their domains. However, these solutions have their intrinsic limitations: Most of them require abundant knowledge of each service to determine the priority/weight, which is usually difficult to achieve in practice due to the private implementations of services; the rapid evolution of services makes keeping the decision center updated for all changes impractical; and it is hard for the center to understand and encode the complex operating logic of all services.

In this article, we propose a novel decentralized negotiation and conflict resolution framework called DeResolver. Unlike the centralized conflict resolutions [24, 32, 34], DeResolver allows the services to resolve conflicts by communicating and negotiating with each other to automatically achieve a Pareto-optimal agreement. The decentralized design has several advantages. First, compared with the centralized resolution, the decentralized conflict resolution can ensure the privacy of service providers, because it does not require the service to transmit the private information of providers, e.g., objectives, service state, and actuator information, and it avoids the single point failure. Second, the decentralized design targets a new setting with competing services rather than the collaborative setting studied by the centralized resolutions. Finally, most conflicts have the spatiotemporal locality. The spatial-temporal locality of conflicts means a conflict may only influence a local area of the city, and it may repeat multiple times during the upcoming time period after the first occurrence. Therefore, multiple conflicts may exist simultaneously but they usually influence different local areas of a city or exist in different time periods of a day. It is natural and efficient to use a decentralized way to resolve each conflict independently.

The service negotiation problem provides a unique setting for an autonomous negotiation design. Services have access to local sensor data but do not know each other's objectives and utility functions. The services that compete against a service in a negotiation are called the opponents or opponent services of this service. We design an autonomous negotiation agent consisting of an opponent-strategy learning module and a negotiation module. To conduct efficient negotiation, a

service should have some knowledge of its opponent services' possible future actions. Therefore, we utilize machine learning in the negotiation agent to estimate the future actions of opponent services. There exists the synergy between the two modules, i.e., the learning module is built upon the self-supervised Bradley-Terry model to provide opponents' possible ranking of different actions for the negotiation module, and the negotiation module provides opponents' real actions as training labels for the learning module to further enable more efficient learning. Since the learning module may estimate opponents' preferences of different actions with errors, we also design a robust negotiation module under uncertain ranking to improve the negotiation performance. This design is evaluated with a case study of services from the domain of transportation and environment with real-world data-driven simulations using the Simulation of Urban MObility.

In summary, the **contributions** of this article are as follows:

- To the best of our knowledge, we are the first to propose a decentralized negotiation framework, called DeResolver, for conflict resolution among city services. As service conflicts demonstrate high spatiotemporal locality in the physical world, the decentralized resolution allows smart services to mitigate cross-domain conflicts in an efficient and scalable manner.
- We design a two-step self-supervised learning module for estimating an opponent's rankings of configurations. The configuration ranking problem is different from state-of-the-art page ranking algorithms, as it is essential to classify proposals into acceptable and unacceptable sets under different states of the city besides providing a quantitative ranking estimation.
- We design a smart automated negotiation module to perform automated negotiation and achieve a Pareto-optimal agreement, which is based on the estimation of how opponent services rank the configurations. The significance of the two modules are not only their novelty but also their synergy. The learning module provides the estimation of opponents' acceptable configurations ranked by utility for the negotiation module. The negotiation module provides opponents' real actions as training labels for the learning module.
- A robust negotiation module is also proposed to handle the estimation errors of how an opponent service ranks the different configurations. Different from the previous works that assume a service has complete knowledge of opponents' preferences [11, 35] or learn a robust policy to conduct decision-making [46], this robust negotiation module optimizes the worst-case performance of a service when the ranking estimations with errors is used for negotiation.
- Our data-driven evaluation is based on a dataset for vehicles that consists of the GPS locations of 246 surveillance cameras and an automatic vehicle capture system with more than 3 million records per day. The results show that compared to a priority-based solution, our resolution can achieve a more equitable solution, i.e., only increasing the average waiting time of vehicles, the measurement metric of intelligent traffic light control service, by 6.8% while reducing the weighted sum of air pollutant emission, measured for environment control service, by 12.1%, and the pedestrian waiting time, the measurement metric of pedestrian service, by 33.1%.

2 CONFLICTS ACROSS CITY SERVICES

2.1 Motivating Example

Modern cities have already implemented smart services to enhance the quality of citizens' lives. These services may be provided by the different companies or departments to the city government. For example, the city bike company dispatches bikes around the city to provide the last-mile transit service [48], the taxi company provides the ride-sharing service [57], and the public safety department schedules patrols to defend against potential attackers [53]. However, conflicts across

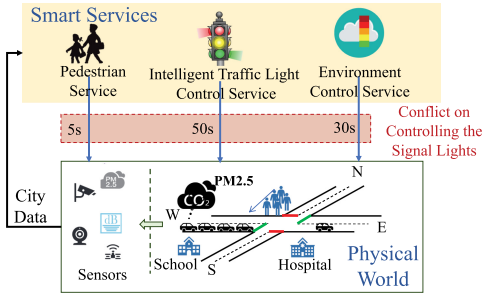

Fig. 1. Demonstration of example.

services arise when two services cannot perform actions simultaneously due to undesirable and harmful effects. In this work, we introduce and use the following example to better illustrate the definition and real-world application scenario of conflicts across services, the scope of the problem that this work addresses, and the system design.

Example 2.1. It shows the inconsistent configurations on traffic lights by three decentralized services. Intelligent traffic light control service [16, 50]: It configures the traffic lights to minimize the average traffic delay at the road intersections. This service can be a decentralized service [50] configuring a traffic light for an intersection independently due to the high computation complexity of coordinating multiple traffic lights simultaneously. It is in the transportation domain.

Pedestrian service [38]: It sets up the traffic lights that show pedestrian crossing signals to minimize the average pedestrian waiting time. This service implements a controller to configure the traffic lights for pedestrians at a road intersection independently. The reason is the setup of a traffic light for pedestrians has little influence on the setup of another one at a nearby road intersection due to the limited walking distance of pedestrians. It is in the transportation domain. Environment control service [7]: It controls the traffic lights to raise environmental quality, e.g., increasing air quality and decreasing noise levels, of road segments. It is a decentralized service and in the environment domain.

These three services run concurrently to achieve their respective objectives. However, potential conflicts may exist among them at runtime. Three services determine their configurations of the green light interval of the **West-East (W-E)** or **North-South (N-S)** directions. The configurations of traffic lights for pedestrians and vehicles should be consistent. The intelligent traffic light control wants a long green light duration due to the high waiting traffic from W-E, whereas the pedestrian service wants to configure a short green light interval because of few pedestrians in the N-S direction. The environment control service does not desire a long green light duration, because the accumulated vehicles around a hospital would increase the air pollution, nor desire a short interval due to the increment of noise level from congested vehicles near a school. Therefore, to meet individual service performance requirements, the conflicts exist among these services.

Example 2.2. Another example is three services desire different amount of traffic on certain road segments and cause conflicts. Event service: It blocks the lanes nearby the event to reduce traffic near a city event. Parking service: It navigates the drivers to parking lots near the event. Detour service: It navigates traffic around a road segment under emergency repairs near the event. A conflict across the three services occur when these three services decide how much traffic can be directed to a certain road segment. Coordinated actions are needed to minimize the local congestion.

Based on the examples, we define the conflicts across services as *if two or more services have inconsistent actions on the shared resources due to incompatible individual goals, then they have a*

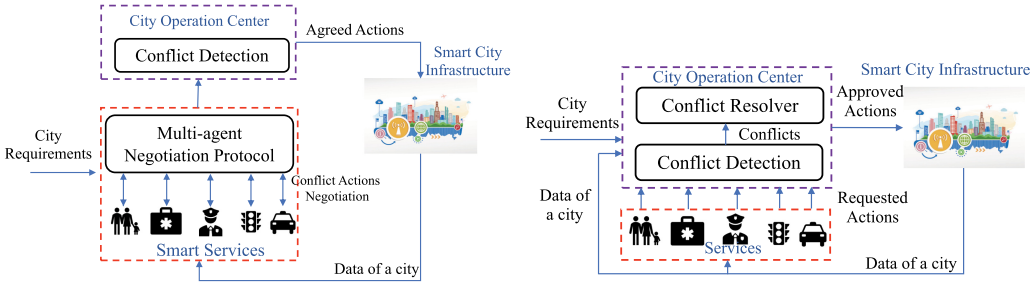

Fig. 2. Comparison of centralized and decentralized conflict resolution.

conflict. If conflicts are not resolved and managed equitably, then they can affects citizens' daily lives. We note that such conflicts do not happen very often for-well designed smart services; they usually occur because services do not consider or know each other's actions when (1) new services and requirements are deployed, (2) city environment changes, and (3) disruptive and unexpected events happen.

2.2 DeResolver Framework

In this work, we consider the following setting of services that result in the conflicts across services: Each service is provided by a stakeholder and has the self-interested and private control objective, which is usually not completely known by the other services. For instance, any service in Example 2.1 does not know the exact control models of the other services. The services can access data from the deployed sensors to check the specific state of the city [29, 36, 37]. Services can reliably communicate and share information with each other, since services managed by different stakeholders have already communicated with the city center to report the operational data in the existing city systems.

To resolve the conflicts across services under the above setting, we design a decentralized negotiation-based conflict resolution, DeResolver. Figure 2(a) shows an overview of DeResolver. It works in three steps: First, smart services collect the data of the city using deployed sensing devices to determine the control decisions and send them to the city operation center. After receiving the control decisions, the operation center uses a conflict detector, e.g., CityGuard [30], to check whether a conflict exists. If a conflict is detected, then the center notifies all the involved services that their collective control decisions result in a conflict and they should resolve the conflict by DeResolver; otherwise, the operation center applies the received control decisions.

Second, with DeResolver, the *services that result in the conflicts* are organized to negotiate an *agreement on the configuration of the shared resources* based on a carefully designed multi-agent negotiation protocol. In each negotiation period, a randomly selected service makes its proposal of the action (i.e., new control decisions) to the other services. Please refer to Section 4.2.1 for how to make the proposal. Then, each of the other services answers acceptance or rejection for the proposal and their answers are broadcasted to all the services in the negotiation. Please refer to Section 4.2.2 for how to make the acceptance or rejection decision. If a proposal is agreed upon by all the services, then they reach an agreement and the negotiation terminates; otherwise, the negotiation continues until the deadline is reached. Section 3.1 introduces how to define the deadline of the negotiation based on the application scenario.

Finally, when an agreement is achieved, it is sent to the operation center, which will detect whether the agreement results in a new conflict considering potential new control decisions from

other services that were not in the previous negotiation process. If not, then the operation center applies the agreement; otherwise, the center notifies all the services resulting in the new conflict that they need to repeat the second step to resolve the new conflict. If the services do not reach an agreement, then the city operation center will execute the default action on the shared resources, which is unknown to the services that result in the conflict.

Existing work [32] proposed CityResolver, a centralized resolution for conflicts across services. Figure 2(b) shows the overview of it. Services send their requested actions to the city operation center. Then, the center detects whether conflicts exist using CityGuard [30]. If so, then the conflict resolver approves part of requested actions to generate a group of actions without conflicts based on its objectives and then apply the approved actions. If no conflict is detected, then all requested actions are applied to the actuators. There are two features of CityResolver. The first one is that it introduces a centralized arbitration to resolve the conflicts by predefined tradeoffs. The second one is that it simultaneously addresses all the conflicts using an integer linear programming-based method if these conflicts happen at the same time.

We summarize the reasons for using the decentralized negotiation-based solution rather than the centralized arbitration to address the conflicts across services as follows:

- The key assumption of the centralized conflict resolution is that these services are collaborative. In contrast, we study a new setting, where services compete with each other to maximize their own utility. These competing services may not be willing to share their private information, e.g., internal logic and objectives, with a centralized arbitration.
- Some cities have the distributed services that are operated by the different departments, e.g., the police, transportation, and environment departments. These services are separated already and it may not be practical to feed all these distributed services into a centralized site, which costs economic and other efforts.
- This decentralized approach is significant for conflicts across competing services, because it allows competing services to negotiate in a scalable and reliable fashion, instead of letting a centralized entity resolve conflicts. In general, it is not necessary to consider many simultaneous conflicts together in a centralized optimization, which is computationally challenging and resource-demanding. Because these conflicts may happen in the different local areas of a city and they do not affect each other.

3 DERESOLVER FRAMEWORK DESIGN

3.1 Formulation of DeResolver

We propose that multiple services that result in a conflict can play a negotiation to resolve this conflict, which is the main idea of DeResolver. In this section, we provide a general mathematical formulation of DeResolver, showing how to formulate a negotiation for a conflict across services.

Negotiation agent: A negotiation is organized for resolving a conflict, and it consists of N services whose control decisions result in a conflict. For example, if N services have inconsistent configurations of an actuator, then these services play a negotiation to resolve the inconsistency. A negotiation agent represents a service. These N services negotiate the issue under discussion with a time horizon of H periods. We assume that every service is honest, because there exist city regulations and mechanisms to ensure the city services are honest and guarantee the city service performance. Meanwhile, the city government has already collected the real-time data of services and deployed the monitoring systems to detect a variety of possible dangerous actions of services.

Proposal: It is a tentative suggestion about a solution to the issue under discussion. Let O_i^h be the proposal that is made by service i to the other $N-1$ services during a negotiation period h. In a negotiation, the issue under discussion can be a configuration of an actuator for a direct conflict or

the distribution of a shared common resource, e.g., the upper bound of noise and the air pollutant emission budget, for an environmental conflict.

Agreement: We define that all the negotiation agents (N services) achieve an agreement if there exists a proposal O_i^h that is accepted by the other $N-1$ services.

Utility: It represents the benefit that a service i can receive by applying a proposal $O_{i'}^h$, denoted as $r_i(O_{i'}^h)$. The utility function is defined according to the objective of each service i. For example, this function may represent the number of packages that are delivered on time for package delivery service or the inverse value of total vehicle waiting time for intelligent traffic light control service.

Multi-agent negotiation protocol: A key issue in designing a negotiation among multiple services is to determine a protocol that these services obey.

In this work, we set up that the services conduct a negotiation using the alternating offer protocol.

Alternating offer protocol: The main idea is that only a service proposes its solution to the issue under discussion during a negotiation period and N services make the proposals in a circular order. If a service makes a proposal during period h, then it should make another proposal during the negotiation period $h+N$ as long as neither an agreement is reached nor the negotiation terminates. If there exists a negotiation agent that rejects the proposal O_i^h, then the negotiation moves to the period $h+1$, and another service makes its proposal. During the negotiation, any service i knows the negotiation behaviors of the other services, i.e., the proposal made during each period, and how the service responses to the proposal, i.e., acceptance or rejection. The agents will follow a predefined order to make proposals. In practice, the city government can determine the order. Alternatively, a random order can be generated to avoid any agent taking advantage of the order.

The negotiation process terminates if N negotiation agents reach an agreement within the H time periods or they cannot find an agreement after H time periods. The length of a time period can be set as a static value, e.g., a second. The maximum number of time periods (H) is determined according to the specific application scenario. How to set H will be introduced in Section 3.2, where we exemplify the formulation of negotiations with the motivating examples of conflicts. If the services cannot reach an agreement, then the device will execute the default configurations that may be determined by the city managers to avoid the failure.

Figure 3 shows an example of how services address the conflict on configuring a traffic light by negotiation. There are three services that want to propose actions on a traffic light. Timeline is divided into fixed-length periods with a default setting if no agreement was reached by a deadline. In Period 1, service A proposes the traffic light configuration, and then service B (C) accepts (rejects) it. Therefore, there is not an agreement. In Period 2, B proposes an action. Again, no agreement was reached because C rejects it. In Period 3, C proposes an action but A and B reject it. Finally, it is A's turn to make a proposal, and service A proposes 15 seconds, which are accepted by the other two services. An agreement is reached before the deadline.

3.2 Case study

In this part, we show the formulation of the negotiation for addressing the conflict in Example 2.1.

The traffic lights for pedestrian crossing signals or vehicle traffic coexist in a road intersection. The signals provided by these two types of traffic lights should be consistent to avoid traffic accidents. Therefore, we assume that there is a traffic light at the intersection of two roads. To simplify the notation, north, south, west, and east are represented by "N," "S," "W," and "E," respectively, and "Green" and "Red" are used to describe the green and red lights. Since two roads' traffic cannot pass the intersection at the same time, there are two states of a traffic light, i.e., (1) Green-WE (Red-NS) and (2) Red-WE (Green-NS).

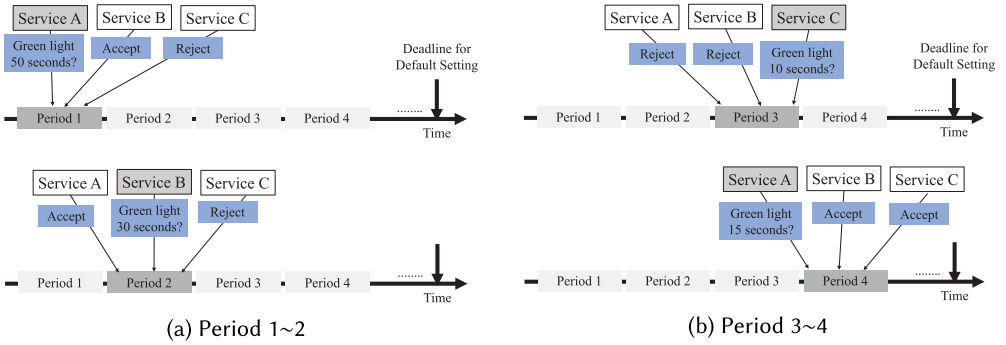

Fig. 3. An example of addressing a conflict by multi-agent negotiation.

In a real-world scenario, such two states exist alternatively, i.e., $1 \rightarrow 2 \rightarrow 1 \rightarrow \ldots$, meaning the schedule of a traffic light is a sequence of phases, where a phase represents several consecutive time slots when a traffic light has the same state. In the first motivating example, each service sets up the length of each traffic light phase, e.g., the number of seconds of each traffic light phase. To simplify the problem description, let t be the traffic light phase that a traffic light is within, and services negotiate the configurations of $(t+1)$-th traffic light phase.

Negotiation agent: A negotiation is organized for resolving a conflict, and it consists of N services whose control decisions result in a conflict. A negotiation agent represents a service. For Example 2.1, the negotiation is played by three services ($N = 3$). These N services negotiate the issue under discussion within H periods.

State: The three services access data from the deployed sensors to check the specific state of the city that they are interested in. Figure 1 shows the specific sensors that three services use to collect the data of a city. We list the information that the different sensors can provide as follows: road surveillance camera: videos of traffic around the road intersections; vehicle loop detector: vehicles count; air quality sensor: air quality value; noise sensor: noise level; pedestrian crossing surveillance camera: videos of pedestrians close to the pedestrian crossing. Let $s_{i,k}(t)$ be the states of the city around the traffic light k at the beginning of phase t that service i is interested in, and we assume the above states around a road intersection are stable during a negotiation. $s_{i,k}(t)$ is defined as follows:

- Intelligent traffic light control service: The state component includes the number of waiting vehicles, the vehicle arriving rate in each direction, the vehicle throughput of each direction, the updated waiting time of vehicles, and the states of the traffic light in current phase t and next phase $t+1$.
- Pedestrian service: The state component includes the number of waiting pedestrians, the pedestrian arriving rate, and the pedestrian throughput of each direction, the updated waiting time of pedestrians, and the state of the traffic light in current phase t and next phase $t+1$.
- Environment control service: The state is defined as the combination of the number of vehicles on the adjacent road segments, the number of waiting vehicles $V_{l''}$, the vehicle arriving rate and the throughput of each direction, and the state of the traffic light in current phase t and next phase $t+1$.

Proposal: The issue under discussion is defined as the configuration of a traffic light. The proposal is the length of the next traffic light phase. During a phase t, three services negotiate the

traffic light configurations for $(t+1)$-th phase. Service i proposes $O^h_{i,k}(t) \in D$ during the negotiation period h to configure the phase t of traffic light k. The domain of a traffic light's phase length is defined as $D = \{d |\ d \in [T_{min}, T_{max}]$ and $d \in \mathbb{Z}_+\}$, where T_{min} and T_{max} correspond to the extreme values of the phase duration.

Agreement: In this example, the agreement means there exists a proposal, $O^h_{i,k}(t)$, of traffic light k's configuration, which is approved by all other services during the negotiation period h.

Utility: Let $r_i(O^h_{i',k}(t))$ be the immediate utility that service i can get if the proposal $O^h_{i',k}(t)$ is applied to the tth traffic light phase. The utility functions are formulated as follows:

- Intelligent traffic light control service: The objective is to minimize the total waiting time of vehicles around the intersection k, where waiting vehicles include taxis, bikes, buses, and private cars. Let i be 1 to represent this service and the immediate utility function is: $r_{1,k}(O^h_{i',k}(t)) = -\sum_{t'=1}^{O^h_{i',k}(t)} \sum_{l \in I_k} W_{1,l}(t')$. I_k is the set of approaching lanes of intersection k. t' is a time slot of phase t, and a phase consists of several time slots, e.g., a time slot is one second and there are five time slots in a phase. $W_{1,l}(t')$ is the total waiting time of waiting vehicles in approaching lane l at time slot t'. The inner sum represents all vehicles' waiting time by the end of slot t' and the outer sum is the sum of all vehicles' waiting time over the phase t. To minimize the waiting time, additive inverse of total waiting time is used when maximizing the immediate utility.
- Pedestrian service: Its objective is similar with that of intelligent traffic light control service, stated as minimizing the waiting time of pedestrians in a road intersection k. Then, the utility $r_2(O^h_{i',k}(t))$ of applying configuration $O^h_{i',k}(t)$ at intersection k for pedestrian service is formulated as: $r_{2,k}(O^h_{i',k}(t)) = -\sum_{t'=1}^{O^h_{i',k}(t)} \sum_{l' \in I'_k} W_{2,l'}(t')$. I'_k denotes the set of pedestrians' walking directions of road intersection k and $W_{2,l'}(t')$ are the total waiting time of waiting pedestrian in direction l' at time slot t'. We also maximize the additive inverse.
- Environment control service: This service aims at minimizing the weight sum of environment quality, e.g., noise level and air pollutant emission in all road segments. For given a time slot t' during traffic light phase t and one road segment l'', let $f(V_{l''}(t'))$ denote the value of environment quality, where $V_{l''}(t')$ is the number of vehicles on road segment l'' during time slot t' that connects with road intersection k. Then, the immediate utility of environment control service is formulated as: $r_{3,k}(O^h_{i',k}(t)) = -\sum_{l'' \in I''_k} \omega_{l''} \times f(V_{l''}(t'))$. $\omega_{l''}$ is the weight for road segment l'' that can be defined by the environment around each road segment, e.g., a road segment has high weight if there are hospitals around it. I''_k is the set of adjacent road segments for intersection k and $f(\cdot)$ is a function calculating the environment quality for given number of vehicles.

Multi-agent negotiation protocol: If these three services have multiple simultaneous conflicts on the configurations of n traffic lights, then they play n negotiations simultaneously, where a negotiation is organized for resolving a conflict on one traffic light's configurations. Finally, an agreement is achieved for each negotiation, and multiple simultaneous configurations are agreed among these services.

The negotiation process terminates when an agreement is reached or the number of negotiation periods is over H. We use an example to demonstrate how to define H in traffic light control. For instance, the configuration of the green light phase duration of the N-S direction should be determined before the green traffic light phase of E-W direction ends. Then, according to the starting time of the negotiation and the deadline, the maximum duration of the negotiation is obtained. H is equal to the maximum duration of the negotiation over the length of a time period, where

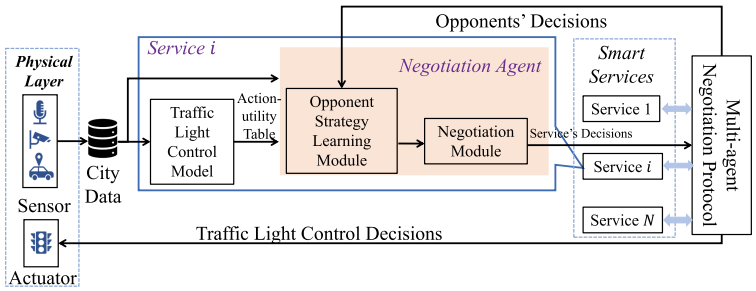

Fig. 4. Design of a service under DeResolver framework.

the denominator can be a static value, e.g., a half second. If no agreement is achieved, then the default configurations is applied to traffic light k, which are determined by city transportation authorities. A negotiation may be organized by the previously described alternating offer protocol or simultaneous offer protocol.

4 DESIGN OF A SMART SERVICE UNDER DERESOLVER FRAMEWORK

It is essential for a service to optimize its negotiation strategy to maximize its utility decided by the agreement. We design an automated negotiation agent for each service that determines the negotiation actions by an opponent strategy learning module and a negotiation module.

Definition 4.1 (Automated Negotiation Problem). Given services with conflicts and the negotiation protocol formulated in Section 3.1, the problem is how any service i determines its action at any negotiation period h, i.e., accepting or rejecting the proposal from other services and making its proposal to maximize its utility.

Figure 4 shows the design of a service with an automated negotiation agent under the DeResolver framework. We take the traffic light control as an example. Given the data of a city, the traffic light control model is used to estimate the action-utility table based on the utility function of service i. There are two columns of each row in the action-utility table, where the first column represents a possible traffic light configuration (action), and the second column is the long-run utility that service i receives if applying the action to the traffic light. Then, service i determines its action at each negotiation period h using the opponent-strategy learning module and the negotiation module. The first module outputs a sequence of acceptable configurations to each opponent service i', and this sequence has the ranking information to the negotiation module. The second module determines the proposals that service i makes, and the acceptance/rejection to opponent services' proposals. The negotiation behaviors of the opponent services are used as training labels for the learning module to improve the learning accuracy and efficiency.

4.1 Opponent-strategy Learning Module

Service i should have some beliefs of its opponent services through the negotiation to maximize its utility that negotiation result introduces. We propose a two-step learner for service i to estimate how any opponent service i' ranks the acceptable configurations. In detail, service i needs to learn $N-1$ models, where a model corresponds to an opponent service i'. Figure 5 illustrates how service A learns B's actions ranked by the utility to B. The first step is filtering, which takes B's all possible actions as an input and outputs a set of B's acceptable actions. The second step is ranking, which ranks the acceptable actions based on their utility to B. For step 1, since we do not have the labels of whether an action is acceptable to B for supervised learning, we design a self-supervised model that uses B's previously proposed or accepted actions as training labels. This is based on

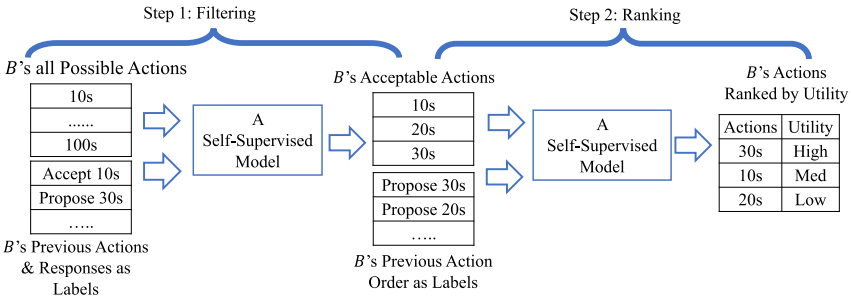

Fig. 5. An example of how service A learns B's action ranked by utility.

the intuition that B only proposes or accepts actions that are acceptable to itself. It is very likely that B will also accept these actions in the future. For step 2, due to lacking the labels of how service B ranks two actions, the self-supervised model uses B's previous order of proposed actions as ranking labels, which is based on the intuition that a service proposes an action with higher utility earlier.

4.1.1 Estimating Acceptable Configurations. The task in the first step of the learner is to learn a function $f_{i'}^1$(state during a period, configuration) $\in [0, 1]$ showing the probability that opponent service i' accepts configuration $d \in D$ during period h given the state during period h. The first-step of the learner takes the state during a negotiation period and a configuration as the input. This function works as a binary classifier to estimate whether a configuration is accepted or rejected by service i'.

First, we define the state during a negotiation period h, denoted as $ns_{i'}^{1,h}$. $P_{i'}^h = \{0, 1\}^{1\times|D|}$ represents whether the configurations are accepted or not by service i' before period h. If configuration d_l is proposed or accepted by service i' before period h, then $P_{i',l}^h = 1$; otherwise, it is 0. We define $ns_{i'}^{1,h} = (s_{i'}(t), P_{i'}^h)$, which is a concatenation of the state of a city that service i' takes and the indicator matrix showing whether configurations are accepted or not.

Second, we discuss how to generate the labeled training data denoted by $U_{i'}^1$ with a form of <state, configuration, label> for each data sample to learn the binary classifier. Given the records of a past negotiation, we generate $ns_{i'}^{1,h}$ based on its definition during each period h. Given $ns_{i'}^{1,h}$, if a configuration d is accepted, i.e., the label y is 1, then we have a data sample, i.e., $< ns_{i'}^{1,h}, d, 1 >$; otherwise, the data sample is $< ns_{i'}^{1,h}, d, 0 >$, i.e., the label y is 0.

We train $f_{i'}^1$(state during a period, configuration) using the labeled data to minimize the following loss function:

$$\mathcal{L}^1 = -\frac{1}{|U_{i'}^1|} \sum_{<ns_{i'}^{1,h}, d, y>} y \log\left(f_{i'}^1\left(ns_{i'}^{1,h}, d\right)\right) + (1-y)\log\left(1 - f_{i'}^1\left(ns_{i'}^{1,h}, d\right)\right). \quad (1)$$

This cross-entropy loss function is widely used for binary classification problems. This function calculates a score that summarizes the average difference between the actual and predicted probability distributions. If the actual classification value is 0, then the corresponding loss value is $-\log(1 - f_{i'}^1(ns_{i'}^{1,h}, d))$; otherwise, the loss value is $-\log(f_{i'}^1(ns_{i'}^{1,h}, d))$. The optimal cross-entropy loss value is 0. There are multiple binary classifiers widely used in the related work, e.g., neural network, K-nearest neighbors, and support vector machines. In the evaluation, we set their loss functions as Equation (1) and then evaluate their performance. The classifier that generates the best results empirically is used in the data-driven evaluation. The set of acceptable configurations

to an opponent service i' may change under the different state of the city, e.g., dynamic traffic volume in each direction, and our training function adapts to such changes, since the state of a city is a part of its input.

4.1.2 Estimating Ranking of Acceptable Configurations. Given the set of acceptable configurations to service i', service i still needs to estimate how service i' ranks these estimated acceptable configurations. An intuition is that the opponent service i' ranks the configurations based on the utility that they introduce. The task of the second step of the learner is to learn a function $f^2_{i'}$(state, configuration) that ranks the estimated acceptable configurations. We assume that the ranking function assigns a score $f^2_{i'}(\cdot)$ to each configuration, where a large score represents a high ranking. The inputs to this learner are the state of the city that service i' is interested in and a configuration.

To learn the above score function, we first generate our training data including data with ranking information $U^2_{i'}$ and data without ranking information $L^2_{i'}$ of service i'. During any past negotiation, service i' may propose multiple configurations. We assume that for any two configurations, service i' proposes the one with higher utility at first. Suppose a past negotiation is associated with a stable state of the city, denoted as $ns^2_{i'}$. In a negotiation, if any two configurations d_1 and d_2 are proposed by service i' during two different periods and d_1 is proposed earlier, then we add a data sample, i.e., $< l_1 \succ l_2 >$ where $l_1 = < ns^2_{i'}, d_1 >$ and $l_2 = < ns^2_{i'}, d_2 >$ to $U^2_{i'}$. In the same negotiation, if any two configurations d_1 and d_2 are not proposed by service i', then we add a data sample $< l_1, l_2 >$ to $L^2_{i'}$. The ranking information is included in any sample in $U^2_{i'}$. However, it is not contained in $L^2_{i'}$.

We use a semi-supervised learning method to train the function $f^2_{i'}(\cdot)$ by these two datasets. For the training dataset $U^2_{i'}$, we consider the probability models that assign a probability of $< l_1 \succ l_2 >$, based on the score difference $f^2_{i'}(l_2) - f^2_{i'}(l_1)$.

Bradley-Terry model [22, 45] is widely used to estimate the probability $P(< l_1 \succ l_2 >)$ that $< l_1 \succ l_2 >$ is true given a pair of individuals l_1 and l_2. This model associates a score $f^2_{i'}(l_1)$ with each individual configuration, l_1. Then, it defines the probability that l_1 is preferred to l_2 as the logistic function of their score difference: $P(< l_1 \succ l_2 >) = \frac{1}{1+e^{f^2_{i'}(l_2)-f^2_{i'}(l_1)}}$. Therefore, the objective of training dataset $U^2_{i'}$ is to maximize the following likelihood function:

$$\sum_{<l_1 \succ l_2> \in U^2_{i'}} \log(P(< l_1 \succ l_2 >)) = \sum_{<l_1 \succ l_2> \in U^2_{i'}} \log \frac{1}{1 + e^{f^2_{i'}(l_2)-f^2_{i'}(l_1)}}. \tag{2}$$

Since ranking information is not included in $L^2_{i'}$, we would like to tie the similarity of configurations to the score similarity. Let r_{l_1,l_2} represent the similarity between l_1 and l_2, defined as $r_{l_1,l_2} = \frac{|d_1-d_2|}{T_{max}-T_{min}}$. Then, we would like to penalize the function $f^2_{i'}(\cdot)$ if similar configurations have quite different scores, formulated as:

$$\sum_{<l_1,l_2> \in L^2_{i'}} - r_{l_1,l_2} \log \Big(P(< l_1 \not\succ l_2 >) P(< l_2 \not\succ l_1 >) \Big) = \sum_{<l_1,l_2> \in L^2_{i'}} - r_{l_1,l_2},$$

$$* \log \Big((1 - P(< l_1 \succ l_2 >)) * (1 - P(< l_2 \succ l_1 >)) \Big) = \sum_{<l_1,l_2> \in L^2_{i'}} -r_{l_1,l_2} \log \frac{0.5}{1 + cosh(f^2_{i'}(l_1) - f^2_{i'}(l_2))}.$$

The intuition of the above equation is that for two configurations without preference information, the probability function of preference should not show that $< l_1 \not\succ l_2 >$ or $< l_2 \not\succ l_1 >$. The first part of the above equation ensures that if there is no preference between l_1 and l_2, then the penalty is minimized when $P(< l_1 \succ l_2 >) = 0.5$. If $P(< l_1 \succ l_2 >)$ is close to 0 or 1, then the penalty is maximized. Then, we apply the Bradley-Terry model to generate the right side.

In summary, we would like to train the function $f_{i'}^2(\cdot)$ to maximize following function with a negative weight β to balance the importance of fitting two datasets:

$$\mathcal{L}^2 = \sum_{<l_1>l_2>\in U_{i'}^2} \log \frac{1}{1+e^{f_{i'}^2(l_2)-f_{i'}^2(l_1)}} + \beta \sum_{<l_1,l_2>\in L_{i'}^2} -r_{l_1,l_2} \log \frac{0.5}{1+cosh(f_{i'}^2(l_1)-f_{i'}^2(l_2))}. \quad (3)$$

The trained function $f_{i'}^2(\cdot)$ assigns a score to these estimated acceptable configurations, and then service i estimates how service i' ranks them. We do not assume that the ranking of configurations is stable for service i', since the ranking may change with the state of a city, e.g., dynamic number of waiting vehicles or pedestrians in each direction. Our learning functions take the dynamic state of a city as a part of its input, so they can estimate the new ranking of configurations for service i' when the state of a city changes.

4.2 Negotiation Module

4.2.1 Strategy for Making Proposals. In this part, we introduce our strategy for making proposals designed for service i. This strategy takes the action-utility table, past negotiation behaviors of current negotiation, and the estimation that how opponent services rank the estimated acceptable configurations as input to determine the proposal.

A service wants to achieve an agreement to get as much utility as possible by using a strategy to propose a configuration that not only introduces the highest utility to itself but also is acceptable to the other services based on the estimation of opponents. If such a configuration does not exist, then the service lowers its lowest acceptable utility to make a proposal, and the amount of utility that is given up depends on its opponent services' last two proposals.

When the learner of opponents is used by service i to estimate how service i' ranks the estimated acceptable configurations during period h, let $A_{i,i'}^h$ denote the output of learner and it is a sequence of acceptable configurations.

Since service i's last proposal is rejected by at least one of other $N-1$ services, service i should concede its lowest acceptable utility to make its proposal be acceptable to other services. We use the reactive concession strategy for service i to update its lowest acceptable configuration based on the previous proposals of other $N-1$ services. Service i computes the ranking difference, which represents how much any opponent service i' concedes between its last two proposals, denoted as $\Delta u_{i,i'}^h$ based on the estimation of how service i' ranks the configurations. The maximum ranking difference that service i can decrease is equal to $\min_{1\le i'\le N, i'\neq i} \Delta u_{i,i'}^h$. Thus, based on service i's last proposal, maximum ranking decrease, and its action-utility table, service i updates its list of the acceptable configurations during period h, denoted as $A_{i,i}^h$. It is noted that service i only concedes during the negotiation periods when this service makes the proposal. We assume $A_{i,i}^h$ only includes the configuration with the highest utility when service i makes the first proposal of current negotiation.

Proposal generation: Since the negotiation may terminate with different agreements, we use the Pareto-optimal agreement to measure them and the definition is shown as follows:

Definition 4.2 (Pareto-optimal Agreement). One agreement d is Pareto-optimal if there is no other agreement d' such that for utility function U_i for agent i, $\forall i \in \{1,\ldots,N\}, U_i(d') \ge U_i(d)$ and $\exists\, i, U_i(d') > U_i(d)$.

In other words, a Pareto-optimal agreement is able to make any individual service's performance better off without making at least one individual service's performance worse off. There may be multiple Pareto-optimal agreements of a negotiation, and we do not measure which one is

the best. Service i also wants to reach a Pareto-optimal agreement, since such a result can maximize its performance, and it does not make opponents miss any benefit.

During period h, if service i makes its first or second proposal, then it can choose the configuration with the highest or $(S+1)$-th highest utility, respectively, where S is the initial concession rate. Otherwise, it first lowers its lowest acceptable utility using the reactive concession strategy to get the new list of acceptable configurations, $A^h_{i,i}$. Let $\mathcal{I}^h_i$ be the set of configurations that exist in all configuration lists $A^h_{i,i'}$ $(1 \le i' \le N)$. If the set $\mathcal{I}^h_i$ is not empty, then service i's the following configuration during period h: $d = \text{argmax}_{d \in \mathcal{I}^h_i \cap Pe} U_i(d)$, where Pe is the set of Pareto-optimal agreements calculated by service i based on its estimation of other services' ranking of configurations. Although service i does not know the actual utility functions of opponent services, the estimated ranking of configurations is enough to compute the set of Pareto-optimal solutions. If I^h_i is empty, meaning there is no configurations that are acceptable to all services, then service i proposes the configuration introducing the lowest acceptable utility, since it has already conceded when updating the set of acceptable configurations, and the proposed configuration is defined as $\text{argmin}_{d \text{ in } A^h_{i,i}} U_i(d)$,

4.2.2 Acceptance Strategy. In this part, we describe the strategy that service i uses to determine acceptance or rejection of a proposal from an opponent service i' during period h. Our negotiation agent uses a utility-based condition to make the decisions. Given the proposal from another service i' during period h denoted by $O^h_{i',k}$, service i first checks whether $O^h_{i',k}$ is an element of $A^h_{i,i'}$. If not, then service i rejects this. Otherwise, this service detects whether $O^h_{i',k}$ can be improved, meaning there is another configuration d that can improve the utility of service i and does not decrease the performance of service i' according to $A^h_{i,i'}$ and the action-utility table of service i. If $O^h_{i'}$ can be improved, then service i should reject it, since its utility can be improved while not sacrificing other services' performance. Otherwise, it is accepted.

4.3 Negotiation Agent Design

According to the strategy for making proposals and the acceptance strategy, we summarize the automated negotiation agent as follows: If it is service i's turn to make a proposal during period h, then service i will use strategy for making proposals to determine its proposal; otherwise, service i decides to accept or reject a proposal from another service using the acceptance strategy.

Theorem 4.1. *Consider N services in a negotiation. They negotiate to determine a configuration from the configuration space $D = \{T_{min}, T_{min}+1, \ldots, T_{max}\}$. If all N services use our negotiation algorithm, and the estimation of acceptable configurations and their ranking to any service i is accurate, then the result of our algorithm is guaranteed to reach a Pareto-optimal agreement.*

Proof. Suppose there are N services playing a negotiation during period $1, 2, \ldots, H$. The period from $N(r-1)+1$ to rN is called round r. All services take turns to propose an action in each round until they reach an agreement. Let $A^r_{i,i'}$ be the set of acceptable configurations of service i' that is estimated by service i in round r. Specially, $A^r_{i,i}$ is service i's true acceptable configuration when $i = i'$. According to our algorithm, at any round r service i first expands its acceptable configuration $A^r_{i,i}$ with a lower utility, that is to say,

$$A^r_{i,i} = A^{r-1}_{i,i} \cup c^r_i, \tag{4}$$

where $c^r_i < min\{A^{r-1}_{i,i}\}$. Then, it computes $C = \bigcap_{i=1}^{N} A^r_{i,i}$.

If $C \neq \emptyset$, then service i selects the candidate introducing the largest utility from $C \cap Pe$, where Pe is the set of Pareto-optimal solutions that are calculated by the estimated ranking of configurations for other services i'; if $C = \emptyset$, then c_i^r is service i's proposal at round r.

Notice that $|A_{i,i}^r| > |A_{i,i}^{r-1}|$ based on Equation (4); also, we have $|A_{i,j}^r| > |A_{i,j}^{r-1}|$, because all services use our negotiation algorithm. Therefore, $\exists r \forall j, |A_{i,j}^r| = |D|$. In this case, $C \neq \emptyset$, because $\forall j, A_{i,j}^r = D$. In other words, all services must reach an agreement at round r.

Finally, we show that the agreement is a Pareto-optimal solution. Under the assumption that both the estimation of acceptable configurations and ranking of configurations are accurate, the proposal determined by $\text{argmin}_{d \in A_{i,i}^r} U_i(d)$ is not accepted by the other $N-1$ services. The reason is $C = \emptyset$. Only the proposal generated by $d = \text{argmax}_{d \in C \cap Pe} U_i(d)$ can be an agreement, and the agreed proposal is Pareto-optimal, since it is an element of the set of Pareto-optimal agreements according to $d \in C \cap Pe$. □

4.4 Robust Negotiation Module under Uncertain Ranking

As described in Section 4.2, a service uses the estimation of how an opponent service ranks the different configurations to make proposals and reply to others' proposals efficiently. However, it is challenging to predict the ranking accurately [5, 8]. The incorrect estimation of ranking could mislead the actions of a service, resulting in the sub-optimal agreement or being exploited by opponent services. For example, if a service i incorrectly computes how much an opponent service i' concedes between the last two proposals due to the incorrect estimation, then service i may concede a lot when making the proposal, resulting in the missing of utility. In this part, we design the robust negotiation module to address the inaccurate estimation of ranking.

Given the list of acceptable configurations for service i' estimated by service i in Section 4.1.1, a score function $f_{i'}^2(\cdot)$ is used (designed in Section 4.1.2) to rank the different estimated acceptable configurations. If any two estimated acceptable configurations, l_1 and l_2, have the similar score, i.e., $|f_{i'}^2(l_1) - f_{i'}^2(l_2)|$ is close to 0 and $P(< l_1 \succ l_2 >)$ is close to 0.5, then the possibility of incorrectly ranking l_1 and l_2 is high. Based on $P(< l_1 \succ l_2 >)$, we define the probability that l_1 is preferred by service i' than l_2 as:

$$g(l_1, l_2) = 2 * |P(< l_1 \succ l_2 >) - 0.5| = 2 * |P(< l_2 \succ l_1 >) - 0.5| \in [0, 1]. \tag{5}$$

It represents the probability that the preference of l_1 and l_2 can be estimated correctly, e.g., if $P(< l_1 \succ l_2 >)$ is close to 0.5, then the probability that estimates the pairwise ranking of l_1 and l_2 correctly is close to 0. Based on the pairwise ranking function $f_{i'}^2(\cdot)$ and the set of acceptable configurations for service i', we define the set of possible rankings for service i' estimated by service i, and it is denoted by $\mathcal{S}_{i,i'}$. For any pair of configurations, l_1 and l_2, if $g(l_1, l_2) \geq \alpha$, then we say that the estimated preference of l_1 and l_2 has high confidence, and the pairwise ranking is determined by $f_{i'}^2(l_1)$ and $f_{i'}^2(l_2)$; otherwise, the ranking could be $l_1 \succ l_2$ or $l_2 \succ l_1$. Then, based on the different pairwise ranking of any two configurations, l_1 and l_2, we can obtain the set of all possible rankings, denoted as $S_{i,i'}$. For example, there are three configurations l_1, l_2, and l_3. If $g(l_1, l_2) \geq \alpha$, $g(l_1, l_3) \geq \alpha$, and $g(l_2, l_3) < \alpha$, then the set of all possible rankings is $\{< l_1, l_2, l_3 >, < l_1, l_3, l_2 >\}$.

Based on the set of all possible rankings, we change the strategy for making proposals in Section 4.2 a little bit. When making proposals, since service i's last proposal is rejected by at least an opponent service, it needs to update its lowest acceptable configuration using the reactive concession strategy, which depends on how much the opponent services concede during the previous negotiation periods. Let $H(A_{i,i'}^h, \mathbf{O}_{i'})$ represent how much the service i' concedes during its last two proposals given the estimated ranking of configurations, i.e., $A_{i,i'}^h$. We propose the robust

reactive concession strategy and use the following equation to obtain how much the service i concedes: $\min_{\Delta u^h_{i,i}} \max_{A^h_{i,i'} \in S_{i,i'}} |\Delta u^h_{i,i} - \min_{1 \le i' \le N, i' \ne i} H(A^h_{i,i'}, \mathbf{O}_{i'})|$.

The main idea is the ranking difference that service i can decrease, i.e., $\Delta u^h_{i,i}$, should be close to the minimum ranking difference that $N-1$ opponent services decrease within their previous proposals, i.e., $\min_{1 \le i' \le N, i' \ne i} H(A^h_{i,i'}, \mathbf{O}_{i'})$. Given the set of estimated rankings, service i needs to determine $\Delta u^h_{i,i}$ to minimize the worst difference between $\Delta u^h_{i,i}$ and $\min_{1 \le i' \le N, i' \ne i} H(A^h_{i,i'}, \mathbf{O}_{i'})$.

Making proposals: After updating the lowest acceptable configurations, service i has a new set of acceptable configurations $A^h_{i,i}$. Let $\mathcal{I}^h_i$ be the set of configurations that are acceptable to all services. Since any ranking in $S_{i,i'}$ has the same set of acceptable configurations, $\mathcal{I}^h_i$ is deterministic. If $\mathcal{I}^h_i = \emptyset$, then service i only proposes the acceptable configuration with the minimum utility to itself, i.e., $\min_{d \in A^h_{i,i}} U_i(d)$. Given the different estimation of ranking in $\forall 1 \le i' \le N, i' \ne i$, $S_{i,i'}$, there are the different sets of Pareto-optimal solutions. Each element in $\mathcal{I}^h_i$ might be the Pareto-optimal solution in several combinations of ranking estimations, i.e., $< A^h_{i,1}, \ldots, A^h_{i,N} >$, where $A^h_{i,i'} \in S_{i,i'}$. If $\mathcal{I}^h_i \ne \emptyset$, then service i will propose the action that appears as the Pareto-optimal solution in the different combinations of ranking estimations with the most times.

Acceptance Strategy: Given the proposal from another service i' during period h denoted by $O^h_{i',k}$, service i first checks whether $O^h_{i',k}$ is acceptable to service i' according to $S_{i,i'}$. If not, then service i rejects this. Otherwise, this service detects whether $O^h_{i',k}$ is the action that appears as the Pareto-optimal solution in the different combinations of ranking estimations with the most times. If not, then service i should reject, since it does not maximize the possibility that a service's performance can be improved while not sacrificing other services' performance. Otherwise, it is accepted.

4.5 Action-utility Table Computation

It is essential for any service i to know the utility that different configurations can introduce. Even without conflicts, each service also needs to compute such a table to choose the optimal configuration for optimizing the quality of service. The action-utility table computation is beyond the scope of this study, i.e., addressing the conflicts across services.

For the traffic light control example, determining optimal control actions has already been well studied in the previous work, classified into two categories: conventional methods and reinforcement learning-based solutions. Conventional methods [11, 49] configure fixed schedule or changing rules according to previous knowledge, which are vulnerable to the dynamic traffic condition. Reinforcement learning-based methods [21, 51, 52] take real-time traffic conditions as input and aim at selecting the action resulting in the maximum reward. Based on the related work [51], we design a reinforcement learning-based agent to control any traffic light k.

The state, action, and reward (utility) of an RL agent for three services are defined in Section 3.1, respectively. Given the real-time state, the task of an agent is to find the action (length of the next traffic light phase) that maximizes the long-term reward, following the Bellman Equation [43]: $U_{i,k}(s_{i,k}(t), a) = r_i(a) + \gamma \max U_{i,k}(s_{i,k}(t+1), a')$. s_t is the state of the city used by service i at the beginning of traffic light phase t. The long-term action reward is the summation of the reward of the next traffic light phase $t+1$ and the maximum potential future reward.

5 VALIDATION

5.1 Methodology

The experiments are conducted using SUMO, a simulation platform providing APIs to model traffic systems including vehicles, pedestrians, environment measurement, and traffic light control.

SUMO can simulate vehicles and pedestrian mobility for given routes and traffic light control policies.

We collect the real-world vehicle mobility data by 246 surveillance cameras in Shenzhen, China, over the time period from 05/01/2017 to 05/20/2017. A record is generated when a vehicle is captured by the camera, and each record consists of captured time, camera ID, and other information. We also have the city map to show each road intersection's GPS data and a table to map each camera ID to the actual GPS location. The size of the dataset is 55.0 GB. We use five days' data for the experiment and the remaining data is imported into SUMO for training. We import a $3\ km \times 2\ km$ region of Shenzhen and the corresponding vehicle traffic as the city environment to SUMO, including nine traffic lights that three services want to control. We also generate pedestrian traffic with the setting that one person per four seconds or five seconds for the different directions of a intersection. A traffic light is configured as an integer between 1 and 60, representing the duration of a light phase.

To evaluate the performance of DeResolver, we compare it with four state-of-the-arts centralized conflict resolution frameworks: (i) a priority-based solution based on Reference [34]; it gives a higher priority to intelligent traffic light control service than the other two services, because the primary goal of traffic light control is traffic flow optimization. (ii) A weight-based solution based on Reference [32]; it selects the action that maximizes the weighted sum of three services' utility ratios. The utility ratio of a service is defined as the ratio between the utility of an action and the maximum utility that this service can get if no conflict occurs. The weight is determined by city managers according to their understandings of services. The weights for intelligent, pedestrian, and environmental control services are defined as 1, 2, and 10, respectively. (iii) A round-robin solution [28] that applies the requested actions from three services by cyclic execution. (iv) A Pareto-efficient solution that knows the action-utility tables of all services and selects the Pareto-efficient configuration that maximizes the minimum service utility ratio of all services. This solution ensures that any service cannot improve its performance without reducing any other services' performance. Meanwhile, it provides the max-min fairness for services.

Taking the Example 2.1, we define the metrics to measure three services' performance. Average waiting time of a vehicle: For a vehicle, we calculate its waiting time (speed less than $0.1\ m/s$) and report the average value. Average waiting time of each pedestrian: We calculate the waiting time of each pedestrian and report the average value. The weighted sum of air pollutant emission per hour: We assign a weight for different road segments based on the nearby environment, e.g., large weight for hospitals and schools, and then report the weighted sum of air pollutant emission of all road segments. The measurement unit of waiting time is second, and that of environment control service is kilogram per hour.

5.2 Performance of Learner of Opponents

First, we describe how we collect the data used to train the learners that are proposed in Section 4.1. We simulate that three services operate to control nine traffic lights by feeding the 15-day traffic data into SUMO, and we set up that they play a negotiation if conflicts exist. During the negotiation, each service uses the proposed negotiation agent to play the negotiation, and services also collect the data that are generated from services' negotiation behaviors. In each period of a negotiation, a data sample is generated for each service following the process described in the third paragraph of Section 4.1.1. The data samples collected over all periods of all negotiations are used to train the binary classifier. Two datasets, i.e., one with ranking information and the other one without ranking information, are generated when a negotiation ends with the process described in the second paragraph of Section 4.1.2. Two datasets are used to train the learner.

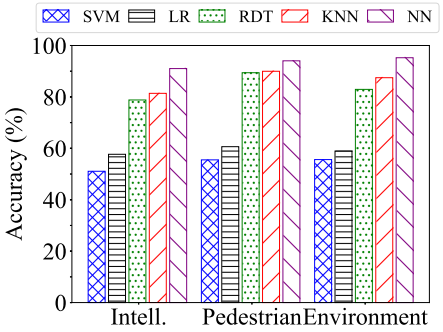

Fig. 6. Performance of classifying acceptance or rejection.

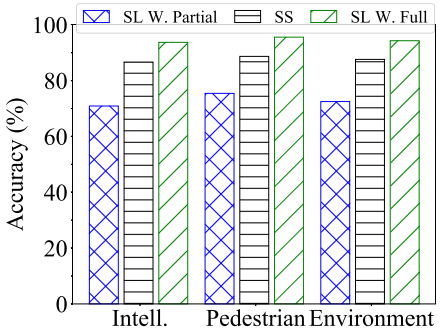

Fig. 7. Performance of learning opponents' preference.

Second, we define the estimation accuracy as the main metrics for evaluating the learning-based algorithm. The accuracy of estimating the acceptable or unacceptable configurations to service i is used to measure the performance of the first level of the learner. The accuracy of estimating how service i ranks the acceptable configurations is used to measure the performance of the second level of the learner. The detailed mathematical description of the estimation accuracy is as follows:

The accuracy of estimating the acceptable or unacceptable configurations to service i is used to measure the performance of the first level of the learner. The metric is defined as: $Acc_i^1 = \sum_{m=1}^{M}\sum_{h=1}^{H_m} NC_{i,m,h} / \sum_{m=1}^{M}\sum_{h=1}^{H_m} C_{i,m,h}$. M is the number of negotiations that are organized during the evaluation. H_m is the number of periods that last in the negotiation m. $NC_{i,m,h}$ is the number of configurations that are correctly classified as acceptable or unacceptable to service i in the period h of mth negotiation. $C_{i,m,h}$ is the number of configurations that are classified as acceptable or unacceptable to service i in the period h of mth negotiation. The accuracy of estimating how service i ranks the acceptable configurations is used to measure the performance of the second level of the learner. The metric is defined as: $Acc_i^2 = \sum_{m=1}^{M}\sum_{h=1}^{H_m} NP_{i,m,h} / \sum_{m=1}^{M}\sum_{h=1}^{H_m} P_{i,m,h}$. $NP_{i,m,h}$ is the number of pairs of configurations, whose ranking order is estimated correctly in the period h of mth negotiation. $P_{i,m,h}$ is the number of pairs of configurations, whose ranking order is estimated in the period h of mth negotiation. The accuracy of a learner shows how often a service correctly estimates its opponent services' behaviors, which is useful to conduct negotiation efficiently.

Finally, we report the evaluation results. We measure the performance of five widely used binary classifiers, and then select the one with the best performance. **Support vector machine (SVM):** a classical algorithm to find a hyperplane for classifying the data. **Logistic regression (LR)** [19] a statistical model estimating the parameters of a logistic model. **Random decision tree (RDT)** [14] a method that constructs multiple trees in randomly selected subspaces of the feature space and uses the combined predictions of the individual trees as the output. **K-nearest neighbors (KNN)** [3] a type of instance-based learning, where the classification of a data point is the same as the class most common among its K nearest neighbors. In this evaluation, we empirically test the value of K and set K as 10 because $K = 10$ has the best results in our tests. **Neural network (NN)** [12] It uses an NN to learn the linear or non-linear combination.

Figure 6 shows the classification accuracy using five different classifiers to estimate whether any one of three services accepts a configuration or not. It can be observed that the neural network-based learner outperforms all other four solutions with more than 90.0% accuracy for all three services, which means that more than 90.0% of configurations are correctly classified as acceptable or unacceptable to an opponent service. The reason is that a neural network can approximate both linear and non-linear hyperplane to partition the feature space. It is also observed that KNN

Table 1. Performances of Different Conflicts Resolutions

	Intell.		Pedestrian		Environment	
	Light	Heavy	Light	Heavy	Light	Heavy
Priority-based	161.48	300.56	22.93	33.75	565.20	3,085.20
Weight-based	293.74	530.86	18.40	26.17	309.60	1,613.88
Round-robin	289.74	518.65	20.05	30.62	334.08	2,177.64
DeResolver	172.39	412.59	20.15	31.11	378.00	1,890.31
Pareto-efficient	172.61	381.96	20.18	29.01	345.20	1,701.72

Heavy traffic means the rush hours of one day and light traffic means non-rush hours of one day.

also achieves the second-best performance with accuracy more than 80.0%, since any two close configurations have a high possibility to receive the same acceptance or rejection decisions. In conclusion, we use a neural network-based classifier to estimate whether configurations can be accepted by a service.

When measuring the **semi-supervised learning algorithm (SS)**, we design two other methods for comparison: **supervised learning with partial order information (SL W. Partial)** and **supervised learning with full order information (SL W. Full)**. The first method learns the opponent ranking model only using the collected data with partial order information and aiming at only minimizing logistic loss function, i.e., Equation (2). The second comparison method assumes that given the state of the city, the full order information of all configurations is known, and this method conducts supervised learning to minimize Equation (2). We use TF-Ranking [40] to implement our semi-supervised learning method, which optimizes the weighted sum of two objectives simultaneously.

Figure 7 shows the estimation accuracy of order between any two configurations by three methods. It is observed that our semi-supervised learning method can achieve more than 86.0% accuracy for estimating the preference of all three services. For pedestrian service, our semi-supervised learning algorithm increases the estimation accuracy by 17.6% compared with the supervised learning method with only partial order information. But it decreases the performance by 7.2% compared with the supervised learning algorithm with full order information. This observation is normal, since our solution takes full use of the distance between two configurations without order information to penalize generating a large score difference for two close configurations. However, a dataset with full order provides the most information.

5.3 Performance of DeResolver

5.3.1 Comparison of Resolutions to Eliminate Conflicts. Table 1 shows the performance of three services when using different solutions to resolve conflicts across them under the light and heavy traffic. We set up that all three services using our negotiation method under the DeResolver framework. There are two observations: The first one is priority or weight-based resolution can improve the performance of one service greatly, meanwhile degrading the other two services' performance significantly. By comparing priority and round-robin solutions, we can see that changing the subjective weight on different services could reduce the weighted sum of air pollutant emission by 40.9% while increasing the average vehicle waiting time by 79.4% with light traffic. The second one is our solution can achieve close performance compared with the Pareto-efficient solution. Compared with applying a Pareto-efficient solution that is generated from the actual action-utility table of three services, DeResolver can achieve close vehicles and pedestrian waiting time, meanwhile increasing the air pollutant emission by 9.5% with light traffic. Meanwhile, with heavy traffic,

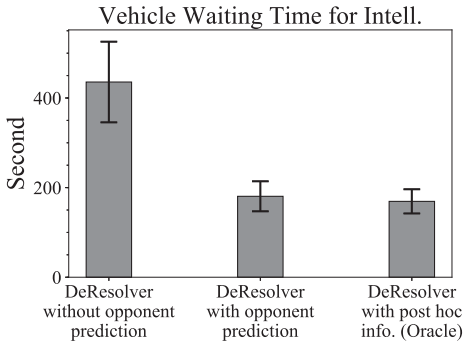

Fig. 8. Sensitivity analysis of learning module.

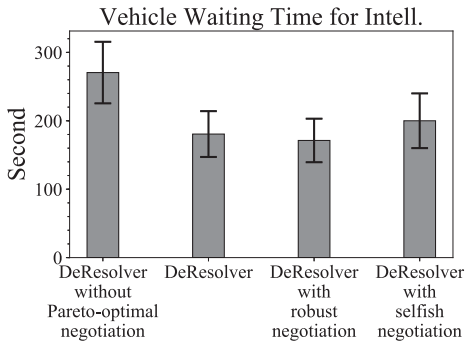

Fig. 9. Sensitivity analysis of negotiation module.

the performance of three services by DeResolver decreases less than 10% compared with that of three services by Pareto-efficient solution. It is because DeResolver misses the action that can maintain the performance of two services and improve that of environment control service due to estimation error.

The action conflicts across services result in a tradeoff among these services' performance when determining the configurations of shared actuators. To be noted, DeResolver's goal is to find a tradeoff for all the services' performance. In some cases, DeResolver cannot guarantee the best performance of each individual service compared with some other algorithms. However, it does provide a balanced performance while resolving the conflicts (as shown in Table 1). Different from the solutions that assign a higher priority or weight to a particular service based on prefixed rules, DeResolver allows smart services to reach agreements under dynamic city states via negotiation, introducing a balanced set of actions for all the services. We believe this is an important step to maintain balanced performances of smart services, especially with an increasing number of services deployed in smart cities. In the future work, we will continue exploring the fairness when resolving conflicts among services.

5.3.2 Module Sensitivity Analysis. In this part, we evaluate the impact of the opponent learning module and negotiation module on the performance of a service. First, we evaluate whether learning services' ranking is useful for improving the performance of one service. We consider two variations. The first one is DeResolver with post hoc information (Oracle). Suppose a service knows the acceptable configurations to opponent services and the ranking. The second variation is DeResolver without opponent prediction. Suppose a service does not learn opponent services' acceptable configurations and preference of different actions. It just proposes the configurations from the one with the highest utility to the one with the lowest utility one-by-one. In this experiment, we assume that the intelligent traffic light control service uses the variations of our negotiation method, and the other two services always use the negotiation agent, i.e., DeResolver.

The results are shown in Figure 8. The main observation is learning the opponent services' ranking can help the service to get more benefit from the service that does not learn this information. Compared to the case that there is not any estimation of two opponent services, the average vehicle waiting time reduces by 58.5% when using DeResolver with opponent prediction. Meanwhile, DeResolver with opponent prediction has the similar performance with an oracle solution, where the post hoc analysis is conducted to directly use other services' utility tables without prediction, showing the effectiveness of our learning module.

Second, we evaluate how the negotiation module affects the performances. We consider four solutions. The first one is DeResolver without the Pareto-optimal negotiation module. In this

Table 2. Performance of DeResolver with Different Settings of Service Types

Intell.		Pedestrian		Environment	
Type	Perf.	Type	Perf.	Type	Perf.
DeResolver	160.58	Dedicated	23.17	Dedicated	397.04
DeResolver	172.39	DeResolver	20.15	DeResolver	378.00
DeResolver	185.68	Selfish	18.23	Selfish	367.93

variation, a service always proposes the action that has the lowest acceptable utility to itself. The second solution is DeResolver. The third solution is DeResolver with a robust negotiation module, i.e., Section 4.4. The last solution is DeResolver with a selfish negotiation agent. It means when a service makes a proposal, it only proposes the action that is acceptable to all the services based on the estimation and also introduces the maximum utility to itself. We set up that the intelligent traffic light control service uses one of these four solutions for negotiation, and the other two services always use DeResolver.

The evaluation results are shown in Figure 9. It is observed that if replacing our Pareto-optimal-based negotiation module with a selfish local greedy negotiation module, then our Pareto-optimal-based negotiation still slightly outperforms it. The reason is that the opponent services do not concede and sacrifice their utility for a selfish service. The second observation is that the DeResolver with a robust negotiation module can improve the performance of the intelligent traffic light control service by 5.2%. The reason is that when using DeResolver with a robust negotiation module, a service concedes less to be robust to the ranking error, which avoid the service being exploited.

5.3.3 Performance of DeResolver with Different Types of Services. We show the performance of DeResolver when negotiating with different types of services in this part. First, we propose the definitions of service types. Selfish agent: A selfish negotiation agent is not willing to concede during the negotiation. In this experiment, we set up that a selfish agent decreases its worst acceptable configuration ranking by one every two proposing periods. For example, a selfish agent makes an proposal with ranking q at its first proposing period. It will propose another proposal with ranking $q-1$ at its third proposing period. This agent only accepts configurations with ranking no less than q at the periods between its first and third proposing period and still proposals the configuration with ranking q in its second proposing period. Dedicated agent: An agent is willing to reduce its lowest acceptable configuration ranking by two between its two consecutive proposing periods. DeResolver: The agent uses the negotiation algorithm designed in this study.

Table 2 shows the performance of DeResolver with different sets of opponent services' types. The observation is that the performance of DeResolver is stable with different opponent services' types, i.e., DeResolver increases or decreases the performance by 6.9% and 7.7% with dedicated and selfish opponent services, respectively. With dedicated opponent services, DeResolver can take advantage of learning opponent services' acceptable configurations and propose the configuration that is acceptable to all services and introduces the most benefit to itself. When negotiating with selfish agents, the reactive recession strategy makes sure that the DeResolver agent does not miss too much utility according to its estimation and observation of opponents.

5.3.4 Evaluation of DeResolver with Robust Negotiation Module. In this part, we evaluate the performance of DeResolver with a robust negotiation module under the different settings, which is shown in Table 3. In the first setting, we compare the performances of three services when they use DeResolver or DeResolver with robust negotiation. In the second setting, we compare

Table 3. Performance of DeResolver with Robust Negotiation

Intell.		Pedestrian		Environment		Average # of periods to reach an agreement
Type	Perf.	Type	Perf.	Type	Perf.	
DeResolver	172.39	DeResolver	20.15	DeResolver	378.00	17.56
DeResolver with robust negotiation	172.45	DeResolver with robust negotiation	20.08	DeResolver with robust negotiation	360.78	26.85
DeResolver	185.68	Selfish	18.23	Selfish	367.93	28.15
DeResolver with robust negotiation	175.69	Selfish	19.47	Selfish	371.54	37.67

Table 4. Convergence Analysis

Service type			Average # of periods to reach an agreement
Intell.	Pedestrian	Environment	
Dedicated	Dedicated	Dedicated	14.00
DeResolver	Dedicated	Dedicated	16.90
DeResolver	DeResolver	DeResolver	17.56
DeResolver	Selfish	Dedicated	22.06
DeResolver	Selfish	Selfish	28.15

the performance of intelligent traffic light control when it uses DeResolver or DeResolver with robust negotiation, while the other two services always use the selfish negotiation agent. The main observation is that a service can improve its performance using DeResolver with robust negotiation when it experiences the sub-optimal agreement due to estimation errors or it is exploited by selfish negotiation agents. For example, in the first setting, the performance of the environment control service improves by 4.6% and the performance of the intelligent traffic light control service increases by 5.4% in the second setting. It is also observed that the robust negotiation module increases the time cost to reach an agreement, since the service concedes less to be robust to the ranking estimation errors. In conclusion, our design of the robust negotiation module can enhance the performance of a service when using incorrect estimation of opponent services' ranking of configurations while sacrificing the speed of reaching an agreement.

5.3.5 Convergence Analysis with Different Types of Services. In this part, we show the average number of periods needed to reach an agreement for three services with different sets of service types in Table 4. There are multiple observations. The first one is that this negotiation converges quickly when all agents use our negotiation method, e.g., averagely costing 17.56 periods (0.053 milliseconds on a PC) to reach an agreement with 60 possible configurations among three agents. The second one is that DeResolver is willing to concede the lowest acceptable ranking if its opponent services also concede. The third observation is if all agents use our negotiation method, then each agent will concede step-by-step in any two consecutive proposing periods, which is kind of slower compared with the case that all opponent services are dedicated agents. The last observation is that the selfish agent increases the time cost to reach an agreement, since DeResolver is not willing to concede when observing selfish behaviors.

5.3.6 Impact of Minimum Green Light Phase Duration. When configuring the traffic light, high frequency change of direction of green light may leave insufficient time duration for passengers and vehicles to pass the intersection, leading to severe consequences. In this part, we set up the minimum green light phase duration and show the performances of three services in Table 5. The main observation is that the performances of three services decrease with the increase of the minimum green light phase duration. The reason is that the increase of the minimum phase duration

Table 5. Impact of Minimum Green Light Phase Duration

Minimum green light phase (second)	Intell.	Pedestrian	Environment
0	172.39	20.15	378.00
10	180.27	21.05	392.52
20	185.67	23.53	403.11
30	193.28	24.76	426.28

reduces the action space of three services, resulting in that the services cannot take the actions with high utility when the actions violate the requirement.

6 DISCUSSION

Services designed in a decentralized way: A decentralized approach, where a controller is designed for a local area or an individual actuator, could result in a local optimal solution. However, in the current city, e.g., City of Newark [1], many services are provided by the different departments, e.g., the police, transportation, and environment departments, and designed in a decentralized approach to obtain the control decisions of thousands of actuators simultaneously, e.g., traffic light control [50] and online dispatch of taxis to current customer bookings [42]. Therefore, using a decentralized design can reduce the risks of "single point of failure," protect the privacy of service providers, and reduce the cost of feeding all these distributed services into a centralized site.

Malicious services: In the negotiation, some services can be malicious or semi-honest to disrupt consensus. But considering a malicious service is out of scope of this article. It is part of our ongoing work. The city government has already collected the real-time data of services and deployed the monitoring systems, which can be utilized to detect the malicious behaviors of services. Once malicious behaviors of a service are detected, actions will be taken to investigate the specific case and audit the service provider. We also note that if all services are selfish and semi-honest, then it is possible that they cannot reach an agreement and the default configuration is applied, or they reach an agreement that is beneficial to a service provider and reduces the utility of other services, which is not efficient (i.e., not Pareto-optimal). It requires more study on investigating the mechanism design to incentivize services to be honest.

Safety detection: A requirement of the actions is to ensure the safety of a city, such as the actions should not increase the noise level dramatically around the school areas. In our decentralized negotiation-based resolution, each individual service can do the safety check at first to make sure that its proposals or accepted proposals meet the city safety requirements. Then, a checker deployed in the city operation center can check it again to ensure the coexisting actions from multiple services do not violate the requirements due to the effects of accumulation. We envision that the distributed safety checker is better, because the centralized safety checker requires services to do the negotiation again when the agreed action is not safe and the centralized checker rejects it.

Security and Privacy: Compared with the existing centralized resolutions that require the system status, actuator information, and objectives to approve some actions, our decentralized design reduces the risk of leaking private information. This is because each service is only required to share their proposals of actions and answer acceptance or rejection to its opponents' proposals. However, we note that security and privacy are very important for this architecture, which still need to be explored. For example, the communication among services can be attacked in a negotiation. Specific methods may be helpful, such as secure multiparty computation and zero-trust network. However, specific designs still need to be investigated.

Implementation of DeResolver: Based on our design, a service should be able to sense the city state, communicate with other services, and compute the control decisions of actuators. In the existing cities, various types of sensors have already been deployed to collect the real-time city data, which is shared with different types of services. In the existing systems, the services can communicate with the city dashboard and other systems by the interface. The services also have the capability to compute for the optimal control strategy. However, the negotiation protocol and interfaces still need to be regulated and implemented on top of existing infrastructure. We believe that our design can be applied to existing smart services.

7 RELATED WORK

We organize the related work into three categories, i.e., resolving conflicts across services, automated negotiation agent design, and opponent modeling.

Resolving conflicts across services: There exist several papers on resolving conflicts across services [24, 26, 32, 34, 47]. Reference [6] blocks the unsafe state of the target application by forcing monitor code into the app. References [24, 32, 34] resolve conflicts by assigning weight or priority to different services based on their domain and managers' understanding of each smart service. Reference [26] suggests that alternative realizations of users' expected applications can be selected to avoid the conflicts in Internet-of-Things. These centralized solutions may experience "single point of failure" and they require city managers to have abundant knowledge of services [24, 26, 34] for determining the weight of each service [30, 32, 47]; whereas, our decentralized negotiation-based solution does not rely on city managers or a central agent to resolve the conflicts.

Automated negotiation agent design: Multi-agent negotiation has already been widely studied in game theory [9, 13, 17]. However, they cannot be applied to solve our problem directly due to making impractical assumptions, e.g., agents' utility functions have some specified properties [20, 27], agents have complete knowledges of opponents' preference [11, 35], and there exist mediators computing agents' offers [17, 20, 27]. Reference [20] considers finding a Pareto-efficiency solution for multi-attribute negotiation with the assumption that one mediator applies query learning to find near Pareto-efficiency solution and each agent's preference is monotonic. Reference [46] tries to learn the robust negotiation policy for multi-agent policies, which assumes that an agent has the complete knowledge of other agents' objectives. Our work does not rely on these assumptions to conduct negotiation automatically, which are impractical for smart city services.

Opponent modeling: Modeling opponents is essential to improve the performance of negotiation results. The closest related work to this study is learning the acceptance strategy or the preference profile of opponents [4]. To learn the acceptance strategy, existing methods focus on estimating the reservation values or the acceptance probability of different offers. An approach learns opponents' reservation value by anticipating opponents' behaviors with Bayesian learning [44] or non-linear regression [15]. Several methods are proposed to learn the preference profile. Reference [25] uses Bayesian learning to determine the opponent types for given negotiation actions and opponent groups. All these methods make some assumptions of opponents that do not hold in this work or require some detailed information of opponents; whereas, our solution does not make such assumptions to improve the performance of a service under negotiation.

Consensus algorithms: Besides resolving conflicts across smart services by negotiation, there are also some existing works investigating how to reach the consensus in the multi-agent system [2, 10, 23, 33, 39]. Reference [33] designs a concurrency control and consensus protocol, committing conflicted transactions in at most two round-trips. Reference [39] proposes a distributed coordination protocol in each agent to dictate semi-cooperative conflict resolution using Lyapunov-Like barrier functions. Reference [23] designs a distributed consensus algorithm to jointly detect the collaborative events by edge devices. Reference [10] studies the finite-time

consensus control algorithm for higher-order cooperative multi-agent systems with a leader-following structure, and Reference [2] also focuses on investigating the fully distributed consensus algorithm for multi-agents with arbitrary order. However, these works consider the setting of cooperative control and determine the actions of the group/team to maximize their total utility. In this work, we assume that each service wants to maximize its own utility through negotiation.

8 CONCLUSION

Conflicts across services directly affect users' mobility and health in modern cities. To achieve dynamic resolution, we propose a decentralized negotiation and conflict resolution framework called DeResolver. Under such a framework, a learning-based solution is designed to guide how a service negotiates with other services to maximize its utility. Trace-driven simulations show that our solution achieves much more balanced results, i.e., only increasing the average vehicles' waiting time measured for intelligent traffic light control service by 6.8% while reducing the weighted air pollutant emission measured for environment control service and the pedestrian waiting time measured for pedestrian service by 12.1% and 33.1%, compared to priority-based solutions.

ACKNOWLEDGMENTS

We thank all the reviewers for their insightful feedback to improve this article.

REFERENCES

[1] City of Newark. 2022. Departments & agencies of City of Newark. Retrieved from https://www.newarknj.gov/departments.
[2] A. Abdessameud and A. Tayebi. 2018. Distributed consensus algorithms for a class of high-order multi-agent systems on directed graphs. *IEEE Trans. Automat. Control* 63, 10 (2018), 3464–3470.
[3] N. S. Altman. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *Amer. Statist.* 46, 3 (1992).
[4] T. Baarslag, M. J. C. Hendrikx, K. V. Hindriks, and C. M. Jonker. 2016. Learning about the opponent in automated bilateral negotiation: A comprehensive survey of opponent modeling techniques. *Auton. Agents Multi-agent Syst.* 30 (2016). 849–898.
[5] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li. 2007. Learning to rank: From pairwise approach to listwise approach. In *ICML*.
[6] Z. B. Celik, G. Tan, and P. D. McDaniel. 2019. IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT. In *NDSS*.
[7] F. V. Cespedes, A. M. Ciechanover, and M. Eiran. 2018. BreezoMeter: Making air pollution data actionable. https://www.hbs.edu/faculty/Pages/item.aspx?num=55052.
[8] Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhi-Ming Ma, and Hang Li. 2009. Ranking measures and loss functions in learning to rank. *Adv. Neural Inf. Process. Syst.* 22 (2009).
[9] R. M. Coehoorn and N. R. Jennings. 2004. Learning on opponent's preferences to make effective multi-issue negotiation trade-offs. In *ICEC'04*. ACM.
[10] H. Du, G. Wen, G. Chen, J. Cao, and F. Alsaadi. 2017. A distributed finite-time consensus algorithm for higher-order leaderless and leader-following multiagent systems. *IEEE Trans. Syst. Man, Cyber. Syst.* 47, 7 (2017).
[11] U. Endriss. 2006. Monotonic concession protocols for multilateral negotiation. In *AAMAS'06*. ACM.
[12] J. Friedman, T. Hastie, and R. Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. Springer.
[13] K. Hindriks and D. Tykhonov. 2008. Opponent modelling in automated multi-issue negotiation using Bayesian learning. In *AAMAS*.
[14] T. K. Ho. 1995. Random decision forests. In *ICDAR*. IEEE.
[15] C. Hou. 2004. Predicting agents tactics in automated negotiation. In *IEEE/WIC/ACM IAT'04*.
[16] Intel. 2019. Intelligent Traffic Management System. Retrieved from https://solutionsdirectory.intel.com/solutions-directory/Intelligent_Traffic_Management_System.
[17] T. Ito, H. Hattori, and M. Klein. 2007. Multi-issue negotiation protocol for agents: Exploring nonlinear utility spaces. In *IJCAI*.
[18] S. Ji, Y. Zheng, Z. Wang, and T. Li. 2019. A deep reinforcement learning-enabled dynamic redeployment system for mobile ambulances. *ACM Interact. Mob. Wear. Ubiq. Technol.* 3, 1 (2019).

[19] D. G. Kleinbaum, K. Dietz, M. Gail, and M. Klein. 2002. *Logistic Regression*. Springer.
[20] G. Lai, C. Li, and K. Sycara. 2006. Efficient multi-attribute negotiation with incomplete information. *Group Decis. Negot.* 15 (2006) 511–528.
[21] L. Li, Y. Lv, and F. Wang. 2016. Traffic signal timing via deep reinforcement learning. *IEEE/CAA J. Automat. Sin.* 3, 3 (2016).
[22] M. Li, H. Li, and Z. Zhou. 2009. Semi-supervised document retrieval. *Inf. Process. Manag.* 45, 3 (2009).
[23] S. Li, S. Zhao, P. Yang, P. Andriotis, L. Xu, and Q. Sun. 2019. Distributed consensus algorithm for events detection in cyber-physical systems. *IEEE Internet Things J.* 6, 2 (2019), 2299–2308.
[24] C. Mike Liang, B. F. Karlsson, N. D. Lane, F. Zhao, J. Zhang, Z. Pan, Z. Li, and Y. Yu. 2015. SIFT: Building an Internet of safe things. In *IPSN'15*. ACM.
[25] R. Lin, S. Kraus, J. Wilkenfeld, and J. Barry. 2008. Negotiating with bounded rational agents in environments with incomplete information using an automated agent. *Artif. Intell.* 172, 6–7 (2008), 823–851.
[26] R. Liu, Z. Wang, L. Garcia, and M. Srivastava. 2019. RemedioT: Remedial actions for Internet-of-Things conflicts. In *BuildSys'19*. ACM.
[27] Y. Lou and S. Wang. 2016. Approximate representation of the Pareto frontier in multiparty negotiations: Decentralized methods and privacy preservation. *Eur. J. Operat. Res.* 254, 3 (2016), 968–976.
[28] S. M. Mostafa and H. I. Amano. 2020. Dynamic round robin CPU scheduling algorithm based on K-means clustering technique. *Appl. Sci.* 10, 15 (2020).
[29] M. Ma, S. Preum, M. Ahmed, W. Tärneberg, A. Hendawi, and J. Stankovic. 2019. Data sets, modeling, and decision making in smart cities: A survey. *ACM Trans. Cyber-Phys. Syst.* 4, 2 (2019).
[30] M. Ma, S. M. Preum, and J. A Stankovic. 2017. CityGuard: A watchdog for safety-aware conflict detection in smart cities. In *IoTDI*. ACM.
[31] M. Ma, S. M. Preum, W. Tarneberg, M. Ahmed, M. Ruiters, and J. Stankovic. 2016. Detection of runtime conflicts among services in smart cities. In *SMARTCOMP*.
[32] M. Ma, J. A. Stankovic, and L. Feng. 2018. CityResolver: A decision support system for conflict resolution in smart cities. In *ICCPS'18*.
[33] S. Mu, L. Nelson, W. Lloyd, and J. Li. 2016. Consolidating concurrency control and consensus for commits under conflicts. In *OSDI*. 517–532.
[34] S. Munir and J. A. Stankovic. 2014. DepSys: Dependency aware integration of cyber-physical systems for smart homes. In *ICCPS'14*.
[35] J. F. Nash Jr. 1950. The bargaining problem. *Economet.: J. Economet. Societ.* 18, 2 (1950), 155–162.
[36] NYC. 2020. NYC Open Data. Retrieved from https://opendata.cityofnewyork.us/.
[37] City of Newark. 2020. City of Newark Open Data. Retrieved from http://data.ci.newark.nj.us/.
[38] Graz University of Technology. 2019. New Traffic Light System Automatically Recognizes Pedestrians' Intent to Cross the Road. Retrieved from https://phys.org/news/2019-05-traffic-automatically-pedestrians-intent-road.html.
[39] D. Panagou, D. Stipanović, and P. Voulgaris. 2015. Distributed coordination control for multi-robot networks using Lyapunov-like barrier functions. *IEEE Trans. Automat. Control* 61, 3 (2015), 617–632.
[40] R. K. Pasumarthi, S. Bruch, X. Wang, C. Li, M. Bendersky, Ma. Najork, J. Pfeifer, N. Golbandi, R. Anil, and S. Wolf. 2019. TF-ranking: Scalable TensorFlow library for learning-to-rank. In *KDD'19*. ACM.
[41] A. Piscitello, F. Paduano, A. A. Nacci, D. Noferi, M. D. Santambrogio, and D. Sciuto. 2015. Danger-system: Exploring new ways to manage occupants safety in smart building. In *WF-IoT*.
[42] K. T. Seow, N. H. Dang, and D. Lee. 2010. A collaborative multiagent taxi-dispatch system. *IEEE Trans. Automat. Sci. Eng.* 7, 3 (2010).
[43] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press.
[44] K. Sycara and D. Zeng. 1997. Benefits of learning in negotiation. In *AAAI'97*.
[45] M. Szummer and E. Yilmaz. 2011. Semi-supervised learning to rank with preference regularization. In *CIKM'11*. ACM.
[46] Y. Tang. 2019. Towards learning multi-agent negotiations via self-play. In *ICCVW*.
[47] R. M. B. S. Thais, B. R. Linnyer, and A. F. L. Antonio. 2010. How to conciliate conflicting users' interests for different collective, ubiquitous and context-aware applications? In *IEEE LCN*. 288–291.
[48] S. Wang, T. He, D. Zhang, Y. Shu, Y. Liu, Y. Gu, C. Liu, H. Lee, and S. H. Son. 2018. BRAVO: Improving the rebalancing operation in bike sharing with rebalancing range prediction. *ACM Interact. Mob. Wear. Ubiq. Technol.* 2, 1 (2018).
[49] F. V. Webster. 1958. *Traffic Signal Settings*. Technical Report. Road Research Lab Tech Papers.
[50] H. Wei, N. Xu, H. Zhang, G. Zheng, X. Zang, C. Chen, W. Zhang, Y. Zhu, K. Xu, and Z. Li. 2019. CoLight: Learning network-level cooperation for traffic signal control. In *CIKM'19*.
[51] H. Wei, G. Zheng, H. Yao, and Z. Li. 2018. IntelliLight: A reinforcement learning approach for intelligent traffic light control. In *KDD*.
[52] M. Wiering. 2000. Multi-agent reinforcement learning for traffic light control. In *ICML'00*. 1151–1158.

[53] H. Yang, S. Tsai, K. Liu, S. Lin, and J. Gao. 2019. Patrol scheduling against adversaries with varying attack durations. In *AAMAS'19*.
[54] Y. Yuan, M. Ma, S. Han, D. Zhang, F. Miao, J. Stankovic, and S. Lin. 2021. DeResolver: A decentralized negotiation and conflict resolution framework for smart city services. In *ACM/IEEE ICCPS'21*.
[55] Y. Yuan, D. Zhang, F. Miao, J. Chen, T. He, and S. Lin. 2019. pˆ 2Charging: Proactive partial charging for electric taxi systems. In *ICDCS'19*.
[56] Y. Yuan, Y. Zhao, and S. Lin. 2021. SAC: Solar-aware e-taxi fleet charging coordination under dynamic passenger mobility. In *CDC*.
[57] D. Zhang, Y. Li, F. Zhang, M. Lu, Y. Liu, and T. He. 2013. CoRide: Carpool service with a win-win fare model for large-scale taxicab networks. In *SenSys'13*.

Received 30 June 2021; revised 21 February 2022; accepted 24 March 2022