PickUp Planner: A Scheduling and Dynamic Routing Service for Connected Vehicles

Aneta Vulgarakis Feljan, Azadeh Bararsani
Ericsson Research
Email: {aneta.vulgarakis, azadeh.bararsani}@ericsson.com

Nicholas Got
Ericsson Research and Uppsala University
Email: kcingot@gmail.com

***Abstract*—Scheduling is needed in a multitude of smart city domains, such as public transportation, logistics, organization of events and parts delivery in assembly lines. In this paper, we present the architecture and implementation of PickUp Planner, a service that enables scheduling and dynamic routing of connected vehicles for picking up number of users distributed in the city. The service dynamically reconfigures the routes of the vehicles based on real-time traffic condition information, which in turn is gathered from the road infrastructure and other connected vehicles registered to the service. The crux of the service is the usage of a novel vehicle routing and scheduling algorithm that combines clustering algorithms and Answer Set Programming. Our evaluation shows that the algorithm decreases the total distance traveled by the vehicles, and therefore reduces fuel consumption and emissions from the vehicles. We also conclude that Answer Set Programming solvers may not be the best choice for scenarios in which there is a high number of users to be served frequently.**

***Keywords*—vehicle routing and scheduling; answer set programming; clustering; implementation; service.**

I. Introduction

The Internet of Things (IoT) refers to a set of physical objects or devices connected – typically through wireless – to the Internet. According to the IDC research group, there will be 30 billion Internet-connected devices by 2020 [1]. The connected vehicles and intelligent transportation systems build one of the fronts where IoT is already making headway. As such, the vehicles are collecting data from their sensors, provide information to drivers and upload filtered sensor data (e.g., GPS location, road conditions, etc.) to the cloud. This data can in turn be used in different applications or services to enable intelligent transportation systems.

In this paper, we present an architecture and an implementation of a PickUp Planner service that solves the scheduling and dynamic routing problem of a set of connected vehicles in a smart city which serve a number of users who want to travel to different destinations. This problem belongs to the general group of logistic scheduling and route optimization problems, which is a generalization of the traveling salesman problem that is known to be NP-hard [2]. There have been a number of industrial applications addressing a similar problem, i.e., scheduling and logistic optimization [3]. While various algorithms have been developed to solve this type of a problem [4], [5], [6], [7], [8], to the best of our knowledge, no implementation exists that combines clustering techniques (such as *k*-means [9]) and Answer Set Programming [10] for scheduling the vehicles on the roads. Moreover, the PickUp Planner service uses real-time traffic data from the vehicles and the road infrastructure in order to dynamically reconfigure the routes of the vehicles. For demonstration, in this paper we use traffic data from Trafiklab [11], which is an open data marketplace distributing transportation data. The benefits of such a service for the fleet management companies and the city stakeholders are obvious: decreased fuel consumption and optimized routes, as well as increased costumer satisfaction by decreasing travel time. In order to make the problem more tractable we make a set of simplifying, but realistic assumptions, as described in Section II. The implementation of the PickUp Planner service (see Section IV) partially builds on components from the CityPulse framework [12] for large-scale IoT data analytics that provide information in (near-) real-time. The goal of the framework is to transform raw data into actionable information and as such to support the development of different smart city services that rely on the underlying framework layers.

In brief, our contribution is twofold:

- A scheduling and dynamic route optimization algorithm for connected vehicles incorporating clustering techniques and Answer Set Programming;
- A system architecture and a prototype of a PickUp Planner service implementing the above algorithm.

The remainder of this paper is structured as follows. Section II identifies the problem that the PickUp Planner service addresses and the assumptions we make to narrow the scope of the problem. Section III gives an overview of a high level system design using the PickUp Planner service, whereas Section IV outlines a prototype implementation of the service. Our implemented algorithm is described in Section V followed by the obtained evaluation results in Section VI. Finally, Section VII concludes the paper, and gives possible directions for future work.

II. Problem Formulation

The scheduling and routing problem refers to picking up a number of users[1] using multiple agents. The users continuously communicate their desires for commuting to different destinations (such as concert venues, shopping malls and airports). It is assumed that there is a sufficient number of vehicles available to provide the service, each vehicle having a specific capacity given in terms of the number of passengers it can carry.

[1] In this paper we use the terms users and travelers interchangeably.

The user may have different travel needs; the system should provide a facility to group requests based on some measure of similarity, such as the distance between users' locations and shared destination. We consider that all vehicles will depart from a same *holding depot* and each vehicle will take the assigned users to a common *destination*. Upon serving assigned trips, vehicles must return to the holding depot to await next trip assignments or end the day. Thus, to serve a trip, each vehicle must depart from the holding depot, pick up all the users within the trip, transport them to their destination and go back to the holding depot i.e., traverse the Hamiltonian cycle defined by the holding depot, starting locations of the travelers in the trip and the destination location.

The system considers users' requests until before the vehicle leaves the holding depot. Once a vehicles is in transit, the system does not assign new users to be picked up by this very vehicle. There is almost certainly some traffic in different parts of the city that will affect the choice of shortest or less-crowded routes for the vehicles, but the impact on the chosen routes, whether positive or adverse, is not known beforehand.

Given the above assumptions, the problem addressed in this paper is to generate an optimal schedule and route for fleet management companies in order to pick up users and get them to their desired destinations. The dual optimization objectives are:

- minimize the transportation cost based on the global travel distance and the fixed costs associated with vehicle use and drivers;
- minimize the number of vehicles required to serve the trips.

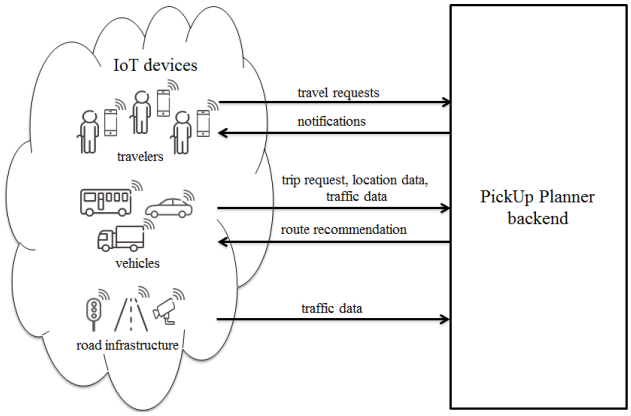

Fig. 1. High level system design using the PickUp Planner service.

III. System Design

This section provides an overview of the system that is centered around the PickUp Planner service, as depicted in Figure 1. The system consists of IoT devices represented as travelers with their mobile phones, vehicles that are supposed to pick up the travelers and road infrastructure sensors sending traffic sensor data. The travelers send travel requests containing information regarding pickup location, desired arrival time and destination, and special needs if any (e.g., a wheelchair for a disabled user). The PickUp Planner backend, which is software hosted in the cloud, processes the requests and allocates travelers to vehicles. When this is done, it sends back notifications to the travelers informing them about the pickup time. Before leaving the holding depot, available vehicles send trip requests to the PickUp Planner backend and receive back a recommended route for picking up their assigned travelers. While driving, the vehicles send traffic data to the PickUp Planner backend. This data and the data gathered from the road infrastructure is used in the dynamic reconfiguration of the routes of the vehicles.

IV. Implementation

In this section, we present a prototype implementation of PickUp Planner[2]. Figure 2 illustrates the functional view of the implemented architecture. We describe each of the components in the following subsections. Please note that the Decision Support component and the Geospatial Data Infrastructure component are off-the-shelf components coming from the CityPulse framework [12]. In our prototype of the PickUp Planner service we include as well the Client Application and Vehicle Application components. However, PickUp Planner can expose Application Program Interfaces (APIs) against which other implementations of the frontend can be built.

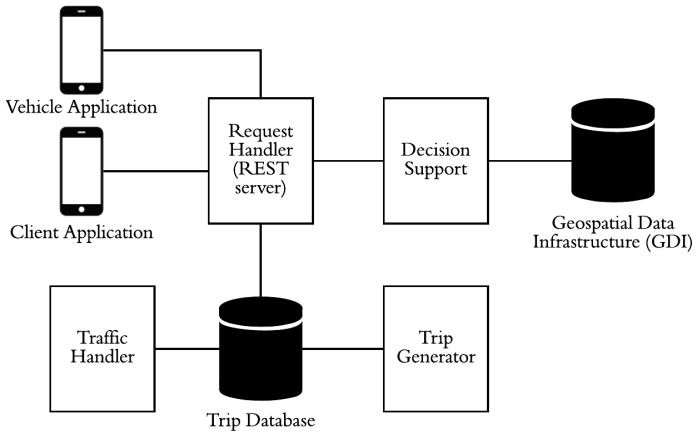

Fig. 2. The functional view of the architecture of our prototype implementation of Pickup Planner.

1) Client Application: Client Application is an Android application that allows users to register to the system and subsequently send travel requests that contain desired arrival time, pickup location and destination, and special needs if any (e.g., a wheelchair). These travel requests are categorized into two groups based on the time constraints specified by the user. The first group contains requests with hard-time constraints that should be served before a specified time, e.g., a flight at 20:00 implies that the passenger should arrive at the airport no later than 18:00 for international flights. The second group encompasses requests with soft-time constraints which are not as time sensitive as the previous group of requests. For instance, attending a social event that takes place for an entire day has a soft-time constraint by this definition. Travel requests are sent to the system through the Request Handler component and are stored in Trip Database. For each travel request the

[2]The source code is available here: https://github.com/Tiglas/pickup-planner

system sends a notification once there is a vehicle assigned to serve it. Figure 3 shows a screenshot of Client Application.

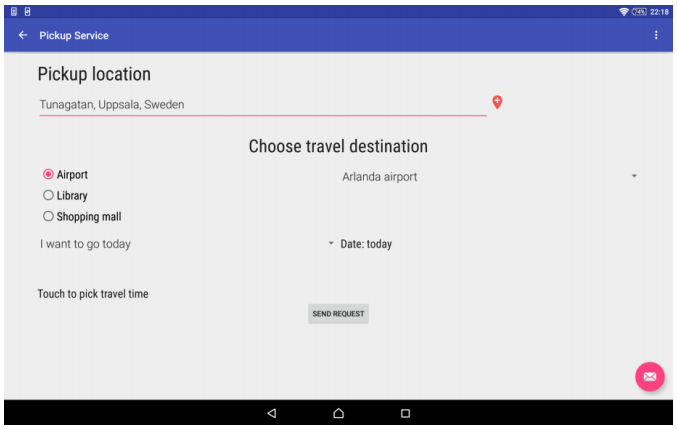

Fig. 3. Screenshots of some of the functionalities of Client Application.

2) Vehicle Application: Vehicle Application is also an Android application, resident in each vehicle in the fleet. It shows the *trip*, i.e., pickup order and locations that each vehicle should follow in order to serve a given set (i.e., a cluster) of travel requests, their destination, together with a variable denoting the next user that should be picked. The next user variable is updated as the users within a trip are picked. For more details we refer the reader to Section V. Note that each cluster is made of travelers sharing a common destination. The route to visit the pickup locations is dynamically generated, and depends on traffic data. The driver, through Vehicle Application, begins a trip by requesting a trip assignment. Each instance of Vehicle Application could send a location data and traffic related data (e.g., average speed and other sensor data) that can be stored in Trip Database. This data is intended to be used to dynamically update the routes of other affected vehicles. Figure 4 shows a screenshot of Vehicle Application.

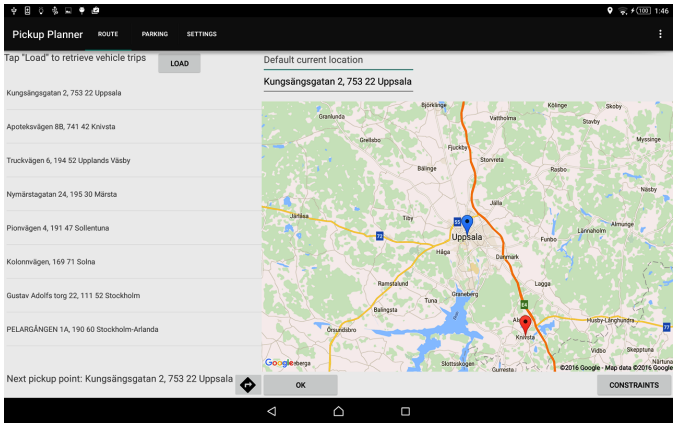

Fig. 4. A screenshot of Vehicle Application.

3) Request Handler: Request Handler is implemented as a REpresentational State Transfer (REST) server that receives travel requests from Client Application, storing them in Trip Database. Additionally, Vehicle Application issues trip requests through Request Handler, which returns the recommended route back to the application. The Request Handler is implemented as a Flask [13] application, providing web service end points for the Vehicle and Client applications. Authentication is provided through HTTP Basic Auth (see RFC 2617 [14]).

4) Trip Generator: The Trip Generator component is responsible for generating the vehicle trips. It pulls the travel requests from Trip Database and processes them using the clustering algorithm (see Section V-A). After clustering, it calls the trip planning and optimization algorithm (see Section V-B) to generate the vehicle trips. The trips are stored back into Trip Database.

5) Trip Database: Trip Database stores user profiles, travel requests, traffic data and generated vehicle trips. It is implemented as a PostgreSQL [15] database.

6) Traffic Handler: Traffic Handler component is a server that is intended to receive traffic data coming from the road infrastructure and from the vehicles. In the current implementation, it pulls road infrastructure data from Trafiklab API [11].

7) Geospatial Data Infrastructure: Geospatial Data Infrastructure (GDI) is a database that is used to store geospatial OpenStreetMap data for a given city. Furthermore, it contains functions that facilitate querying the data. It enables calculation of cost for all the paths on a route. As such, it provides an enhanced routing system by having multidimensional path weighting, e.g., path distance, duration, pollution, events or combined metrics. As a result, it is possible to avoid certain city areas or routes.

8) Decision Support: The Decision Support component takes the possible routes from GDI and determines the route that a vehicle should take between two locations in the vehicle's trip. Section V-C describes this in more details. The reasoning engine is built in Answer Set Programming (ASP) [10] using the ASP solver Clingo [17].

V. Algorithm

The proposed solution is composed of three main stages (i.e., sub algorithms): clustering, trip planning and optimization, and dynamic route update. The first stage groups the travel requests based on destination and arrival time constraints. The clustered requests provide the input for trip planning and optimization. Since travelers are located in diverse locations, an initial grouping before generating a trip is a useful step towards achieving the optimization objective of reducing vehicle travel time, as well as traveled distance. Trip optimization attempts to minimize the trip traveled distance, resulting in trips that have a minimal cost path. The input to the system is a *complete graph* constructed from the road network, with the vertices being the locations of the travelers, the holding (souce) depot and destination locations, and the edges are the shortest path connections among the traveler locations and the holding depot and destination locations. An illustration of such a graph is given in Figure 5. In the following sections, we present a closer examination of each of the three stages.

A. Clustering

The grouping of travel requests is based on the destination and arrival time constraints. The approach taken is to first

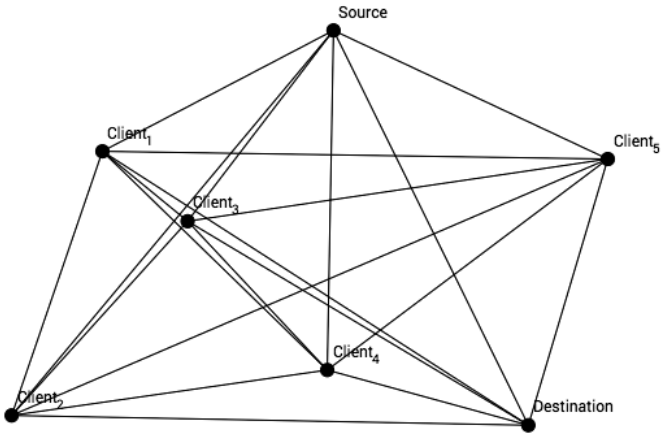

Fig. 5. An example graph derived from the road network.

group together requests that have a common destination and arrival time within a predefined system time interval. For example, travel requests with arrival within one hour can be grouped into same cluster. The vehicle capacity determines the cluster size and the total number of clusters.

To achieve these goals, we considered a solution incorporating both ASP and clustering algorithms, such as *k*-means [9] or DBSCAN [18]. We use ASP to group the users' requests based on common destination and desired arrival time. In addition, we employ the *k*-means algorithm inside the groups to cluster users' requests based on their location. In order to choose a meaningful distance threshold ϵ for DBSCAN, the data and scale must be well understood. The PickUp planner was designed for use in arbitrary locations around cities of Stockholm and Uppsala, making it difficult to choose the parameter ϵ, hence the decision to use the *k*-means algorithm instead. In the following we describe the solution in more details.

a) Grouping using Answer Set Programming: Answer Set Programming (ASP) is a type of logic programming oriented towards NP-hard search problems. An ASP program is composed of a set of *rules*, each rule consisting of a *Head* and *Body*. If the *Body* is evaluated as true then the inference is that the *Head* holds. The *Head* and the *Body* are formed by a set of symbols, called *atoms*. The solution of a problem encoded as an answer set program can be a set of *answer sets*.

In our implementation the users' requests (including location, destination and desired arrival time) are input to ASP as *atoms*. *Rules* are defined for grouping the users' requests based on common destination and desired arrival time. A listing of the rules impacting the grouping of the users' requests is presented in Listing 1. In the listing *nodes()*, *neighbors()*, *cost()* and *destinations()* are external Python functions that we implement to query Trip Database for users' requests, neighbors in the constructed graph, cost of the edges in the graph and desired destinations, respectively.

```
1   % List of destinations
2   destination(@destinations()).
3
4   % Filter nodes on destination
5   node(@nodes(D)) :- destination(D).
6
7   % Directed edges
8   edge(X, @neighbors(X)) :- node(X).
9
10  % Edge costs
11  cost(X,Y,@cost(X,Y)) :- edge(X,Y).
```

Listing 1. Program for returning answer sets for users traveling to a given destination.

b) k-means clustering: The goal of clustering is to partition the users within a group into several clusters. The clusters are formed such that the users within the same cluster are closer together compared to users assigned to another cluster. The "closeness" between two users is calculated as the squared Euclidean distance [19].

$$d(x_i, x_i') = \sum_{j=1}^{p} (x_i j - x_i' j)^2 = ||x_i - x_i'||^2$$

We used the Scikit-learn library implementation of *k*-means [9] to cluster requests from different users. The cluster size is defined based on the capacity of the available vehicles. The clustering is invoked as an external function from the answer set program, and the results from the clustering serve as facts for the trip planning and optimization algorithm. *k*-means clustering function is invoked in line 5 of Listing 1 through the external function $node(D)$.

B. Trip planning and optimization

Assuming that we have the clusters ready, we move to generating the trips, i.e., the order of pickup locations. To find a round-trip that picks up all the users within a cluster (i.e. a Hamiltonian cycle), the problem is defined uniformly by separating the encoding from the problem instances [17]. A problem instance is a complete graph of a given cluster in which the number of the nodes correspond to a given vehicle capacity. On the other hand, the problem encoding is the task of finding a Hamiltonian cycle which we divide into two subproblems in order to find a solution to the original problem. To achieve this we define a number of logic rules, as shown in Listing 2.

```
1   % Cycle must start from source, visit all
2   % nodes, destination and return to
3   % source (ID = 1)
4   cycle(D,1) :- destination(D).
5
6   % Generate
7   { cycle(X,Y) : edge(X,Y) } = 1 :- node(X).
8   { cycle(X,Y) : edge(X,Y) } = 1 :- node(Y).
9
10  % Define
11  reached(Y) :- cycle(1,Y).
12  reached(Y) :- cycle(X,Y), reached(X).
13
14  % Test node whose reachability is not
    demonstrable
15  :- node(Y), not reached(Y).
16
17  % Display
18  #show cycle/2.
```

Listing 2. Program for trip generation.

Once the cycles are generated, the next step finds an optimal cycle with respect to a distance cost. Listing 3 returns answer set(s) with a minimal cost. Note that in order to compute all optimal answer sets, it is necessary to

change Clingo's default optimization mode with the options `'--opt-mode=optN'` and `'--quiet=1'`.

```
1   #minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.
```

Listing 3. Program for finding a minimal cost Hamiltonion cycle.

C. Dynamic route update

The dynamic route update algorithm is implemented as part of the Decision Support component. It covers both initial routes generated to serve a trip, as well as dynamic routes when there are traffic conditions affecting the initial route and the vehicle is serving that route.

VI. Evaluations

In this section, we evaluate the effect of clustering travel requests on the total distance traveled by vehicles serving the requests. The number of clusters is chosen such that the cluster size is at most equal to the capacity of the vehicle. One vehicle serves a single cluster at a time. In order to evaluate if clustering can improve serving the requests, we also consider the case where no clustering is done, by randomly assigning travelers to be served by the vehicles. For both cases, a uniform vehicle capacity of four travelers is assumed.

Trip Generator uses the output from clustering or random assignment to provide an optimal travel path. Table I and Table II show the results obtained for 15 travel requests located in diverse locations around Stockholm, traveling to the same destination, which was ran on a computer with 2.5 GHz Intel Core i5 processor and 8GB 1600 MHz DDR3 memory. The optimized distances in Table II are averaged over five runs of the algorithm.

TABLE I. Clustered requests

Cluster size	Optimized distance (km)
2	128.367
3	161.199
2	88.308
4	88.778
4	145.149
Total distance	611.801

TABLE II. No clustering

Number of assigned travelers	Average optimized distance (km)
3	156.282
4	162.894
4	152.658
4	161.990
Total distance	633.824

Our results show that the random assignment of travelers to the vehicles may result in lower number of vehicles compared to the clustering case (see Table I and Table II). However, the evaluation shows that clustering decreases the distance traveled by the vehicles (for the studied use case the decrease is approximately 22 km). Note that the benefits of clustering over the random assignment case will be even bigger if maximum capacity of the vehicles is not necessarily used. This implies reduction in fuel consumption for the fleet management companies, as the traveling distance is shorter. For the travelers, clustering enables shorter commute time. Thus, there is a clear benefit in using clustering techniques as part of the processing pipeline compared to choosing travelers randomly when determining optimal routes for vehicles serving travel requests.

It is also observed that the time taken to generate trips and the travel distance reduce with the number of clusters (see Table III). Note that varying the number of clusters implies a corresponding variation in vehicle capacity i.e., all travel requests must be served in any case. While this may lead to the conclusion that the degenerate case of zero clusters would produce the best solving time performance and distance optimization, this is only a theoretical hypothesis. In current practice, the vehicle drivers must decide on the passenger pickup order, provided there is more than one passenger to transport. And since this order is based on heuristics such as first-come, first-served service policy, it is expected that a route planning algorithm that merely uses some heuristic, e.g., placing travel requests in a First-In First-Out queue, would not do any better than the cluster-based approach presented here. It is also noteworthy that the performance of the ASP solver gradually deteriorates with an increase in the number of requests. In the PickUp Planner design, the asynchronous trip generation is done periodically, for instance every two hours. When there are more than 25 travel requests, the Trip Generator is unable to fulfill a two-hour trip generation frequency (see Figure 6). In essence, this means that for scenarios in which there is a high number of requests that must be served frequently, the ASP solver Clingo may not be the best choice in achieving this.

TABLE III. Varying the number of clusters

Number of clusters	Solving time (minutes)	Distance (km)
5	44	566.605
4	35	441.730
3	26	331.338
2	19	238.874

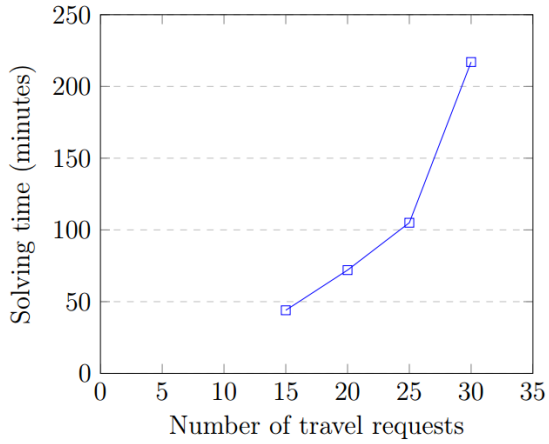

Fig. 6. Relation between the number of travel requests and the Trip Generator solving time.

VII. Conclusion

In this paper, we have introduced PickUp Planner - a service enabling scheduling and dynamic routing for connected vehicles. The studied service is centered around users that continuously send travel requests, vehicles supposed to pick

up the travelers and transport them to their destination, as well as road infrastructure sensors sending traffic sensor data. The essence of the service is the usage of a novel vehicle routing and scheduling algorithm that combines k-means clustering [9] and Answer Set Programming [10]. The traffic sensor data collected from the vehicles and the road infrastructure is used as an input for the dynamic route generation of the vehicles. Our evaluation shows that the algorithm decreases the total distance traveled by the vehicles, and therefore reduces fuel consumption, as well as increases costumer satisfaction by offering decreased travel time. Moreover, the tests that we conducted with 30 users indicate that the time performance for trip generation is not sufficient for use cases where the frequency of trip generation is less than three hours. In such cases in order to achieve a better time performance with a large number of travel requests, it seems imperative to replace the Trip Generator component implemented in Answer Set Programming with for instance off-the-shelf Vehicle Routing Problem solvers such as OptaPlanner [20].

The functionality of the PickUp planner service could later be extended as follows. The Traffic Handler could pull traffic data generated by the vehicles from Trip Database and decide whether the vehicles' current route should be updated. If an update should be made, Traffic Handler could send a message to the affected vehicles using, for example Firebase Cloud Messaging [16], recommending that a new route request should be sent. The routes would be generated on demand by the Decision Support component. Alternatively, the generation of the routes could be triggered by the Traffic Handler component without any action from the driver. This means that the Traffic Handler component would directly send a request to the Decision Support component to generate a new route for the affected vehicle.

For now, in our current prototype, we use the Decision Support component for querying the possible routes connecting two points from a given trip. In the future, we could use the Decision Support component for reasoning in finding possible routes with respect to other parameters than distance, such as pollution, temperature, speed, etc. In order to do this, we could use other components from the CityPulse framework [12] such as the Event Detection- and Context Filtering component.

As future work instead of using Trafiklab [11] data, we could use other raw sensor data, thus test the system for vehicle routing and scheduling considering other optimization parameters, such as pollution, temperature and speed. Although the PickUp planner employs only a single holding depot, in the future our service can be extended to incorporate multiple holding depots so as to reflect the fact that fleet management companies typically either have distributed depots, or in the case of taxi companies, several "hot spots" from which prospective travelers are picked.

ACKNOWLEDGMENT

This research has been partially supported by EU FP7 CityPulse Project under grant No.603095. http://www.ict-citypulse.eu. The authors wish to thank Daniel Kümper and Muhammad Intizar Ali for their help on the GDI and the Decision Support component, respectively.

REFERENCES

[1] "IDC: Worldwide Internet of Things Forecast, 2015–2020," https://www.idc.com/.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[3] P. Leitao, S. Karnouskos, L. Ribeiro, J. Lee, T. Strasser, and A. W. Colombo, "Smart Agents in Industrial Cyber–Physical Systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1086–1101, 2016.

[4] E. Alba and B. Dorronsoro, "Solving the vehicle routing problem by using cellular genetic algorithms," in *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, 2004, pp. 11–20.

[5] G. Jeon, H. R. Leep, and J. Y. Shim, "A vehicle routing problem solved by using a hybrid genetic algorithm," *Computers & Industrial Engineering*, vol. 53, no. 4, pp. 680–692, 2007.

[6] I. Cobeanu, B. Târnaucă, S. Nechifor, and V. Comnac, "Real-time scheduling of mobile agents using answer set programming," in *13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*. IEEE, 2012, pp. 1505–1510.

[7] M. Reed, A. Yiannakou, and R. Evering, "An ant colony algorithm for the multi-compartment vehicle routing problem," *Applied Soft Computing*, vol. 15, pp. 169–176, 2014.

[8] R. Baldacci, A. Mingozzi, and R. Roberti, "Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints," *European Journal of Operational Research*, vol. 218, no. 1, pp. 1–6, 2012.

[9] "*k*-means algorithm," http://scikit-learn.org/stable/modules/clustering.html/#k-means (Last Accessed: 2016-06-23).

[10] V. Lifschitz, "What Is Answer Set Programming?" in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, (AAAI), Chicago, Illinois, USA*, 2008, pp. 1594–1597.

[11] "Trafiklab – open transport data," https://www.trafiklab.se/ (Last Accessed: 2016-11-25).

[12] D. Puiu, P. M. Barnaghi, R. Toenjes, D. Kuemper, M. I. Ali, A. Mileo, J. X. Parreira, M. Fischer, S. Kolozali, N. FarajiDavar, F. Gao, T. Iggena, T. Pham, C. Nechifor, D. Puschmann, and J. Fernandes, "CityPulse: Large Scale Data Analytics Framework for Smart Cities," *IEEE Access*, vol. 4, pp. 1086–1108, 2016.

[13] "Flask Documentation website," http://flask.pocoo.org/docs/0.11/ (Last Accessed: 2016-11-25).

[14] "HTTP Authentication: Basic and Digest Access Authentication," https://tools.ietf.org/html/rfc2617 (Last Accessed: 2016-06-23).

[15] "PostgreSQL Documentation-Interfacing Extensions To Indexes," https://www.postgresql.org/docs/current/static/xindex.html (Last Accessed: 2016-11-25).

[16] "Firebase Cloud Messaging," https://firebase.google.com/docs/cloud-messaging/ (Last Accessed: 2016-11-25).

[17] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, and S. Thiele, *Potassco User Guide*, 2nd ed. University of Potsdam, May 2015.

[18] "DBSCAN algorithm," http://scikit-learn.org/stable/modules/clustering.html/#dbscan (Last Accessed: 2016-06-23).

[19] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed., ser. Data Mining, Inference and Prediction. Springer, 2008.

[20] "OptaPlanner web-site," https://www.optaplanner.org/ (Last Accessed: 2016-12-02).