IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 15 November 2023, accepted 2 January 2024, date of publication 6 February 2024, date of current version 16 February 2024.

*Digital Object Identifier 10.1109/ACCESS.2024.3363426*

RESEARCH ARTICLE

# Differentiable Optimization for Orchestration: Resource Offloading for Vehicles in Smart Cities

**THILO STRAUSS[1], MICHAEL OECHSLE[2], AND UWE BAUKNECHT[3]**
[1]School of AI and Advanced Computing, Xi'an Jiaotong-Liverpool University, Taicang, Suzhou 215412, China
[2]Google, 8002 Zürich, Switzerland
[3]Bosch Group, ETAS Research, 70469 Stuttgart, Germany

Corresponding author: Thilo Strauss (thilo.strauss@xjtlu.edu.cn)

This work was supported in part by the German Ministry for Economic Affairs and Energy (BMWK) in the Project IPCEI-CIS CUBE-C under Grant ID 13IPC021.

**ABSTRACT** Connected and Autonomous Vehicles (CAV) which interact with Roadside Units (RSU) as part of a smart city infrastructure are currently seeing first real-world deployments. Not only can CAVs benefit from access to a cities' infrastructure by obtaining data from various sensors (e. g., Video or Lidar), but they can also leverage the broad network coverage to offload complex computation tasks from their limited on-board hardware to scalable cloud resources. Furthermore, a smart city supporting multi-access edge computing (MEC) can even provide safety-relevant and time-critical services thanks to reduced latency and increased reliability. This requires an algorithm to determine which vehicle offloads computation to which computation resource in the city. This orchestration task is a challenging combinatorial problem subject to resource and quality of service constraints. We present a novel and powerful, yet surprisingly simple algorithm that provides a good and fast approximation to this problem. This Differentiable Orchestrator converts a combinatorial problem into a soft-constrained differentiable analog, which can be solved very quickly. We compare the proposed method with other heuristic methods and conclude that it significantly outperforms most competing methods in artificial examples and realistic scenarios. In order to make the method as reproducible as possible and serve as a baseline for future research we make our data and simulations publicly available.

**INDEX TERMS** Connected and autonomous vehicle (CAV), multi-access edge cloud (MEC), orchestration, optimization, smart city.

## I. INTRODUCTION

Self-driving or fully autonomous vehicles are expected to become a central pillar of future mobility due to their potential for efficient and safe transport [1]. Such vehicles are equipped with a wide variety of heterogeneous sensors and actors that enable various applications ranging from passenger entertainment to safety-critical driving functions. Such functionality, however, necessitates performing a large number of complex computation tasks. Furthermore, these vehicles are equipped with broadband communication devices that enable vehicle-to-vehicle (V2V) as well as vehicle-to-infrastructure (V2I) connectivity. Such so-called Connected and Autonomous Vehicles (CAV) communicate using wireless transmission technologies either based on IEEE 802.11p [2] like DSRC and ITS-G5 or C-V2X [3] based on the LTE or 5G cell-based communication standards. This enables advanced cooperative functions such as platooning [4] or hazard warnings, e. g., for vulnerable road users [5].

A smart city environment is an urban area equipped with sensors, data processing units and a communication infrastructure. Such cities are therefore uniquely equipped to support mobility-related applications. Distributed sensors like LIDAR or stereo camera systems capture data on vehicle positions and road usage which can then be analyzed and correlated by the processing units to provide

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Wei.

supporting functions to CAVs via their wireless transmission devices. To provide sufficient geographical coverage while also keeping all the individual components manageable, sensors, computing and communication can be integrated into so-called Roadside Units (RSU), which are connected to a wired communication network to facilitate information exchange between these units.

Smart cities can provide a variety of functions to CAVs ranging from disseminating traffic information, e. g., to avoid routes with congestion, to taking control of the vehicle itself, e. g., to facilitate driving in complex environments such as parking structures. Implementations of the latter use case, which is known as automated valet parking (AVP) [6], are currently transitioning from prototypes to actual deployments such as at the airport of Stuttgart in Germany [7]. Effectively combining a set of supporting functions as part of an overarching intelligent transportation system (ITS) [8] is currently explored in first deployments, e. g., New York City's Connected Vehicle Pilot Project [9], London's Smart Mobility Living Lab [10] or in various cities in China through their country-wide innovation strategy programs [11].

Apart from access to information beyond the individual vehicle's information horizon, a CAV can benefit from this interaction by moving complex computation tasks from its resource-limited onboard control units to the city's processing units for more efficient computation. This principle of task offloading can be generalized and applied to other applications as well. For example, there is a tremendous number of results on offloading smart phone applications onto various cloud architectures. However, for CAVs there exist safety-critical tasks such as the driving functions in the case of AVP, that cannot afford the volatility in terms of diminished quality-of-service (QoS) figures such as the increased latency and jitter typically associated with connecting to a potentially far-away cloud data center. This necessitates running such tasks on compute units in close proximity to the respective CAV on the smart city's network edge, which can be realized by organizing RSUs as part of a multi-access edge computing (MEC) [12] architecture.

An important factor to consider is the fixed distribution of computation units in the geography of the city. The time-variant density of vehicle traffic flows may lead to problematic traffic hotspots, where the number of CAVs in a hotspot may grow large enough that the closest RSU becomes overloaded, i. e., its compute resources are insufficient for the number of required offloading tasks. To avoid this situation the city may employ an offloading orchestrator that can decide not to run offloaded tasks on the closest RSU, but rather on any RSU with available resources in sufficiently close proximity to ensure QoS. For latency this equates to limiting the transmission distance whereas jitter requires limiting the number of relaying units between computation and vehicle. Furthermore, the orchestrator needs to continuously determine which computation unit in the environment is optimal for each vehicle at each point in time.

We focus on solving an abstract view of this orchestration problem where we consider all computation units to be identical in type and all vehicles to be identical in all parameters relevant to the problem. Each computation unit can only handle a limited amount of vehicles and each vehicle is associated to exactly one computation unit for offloading. To ensure QoS we require that a vehicle can only be associated to a computation unit within a certain number of relaying nodes. We present an algorithmic optimization approach to solve this problem and analyze it in several scenarios.

The remainder of this work is structured as follows. First, we will introduce related works on resource orchestration for CAVs in smart cities in Section II. Section III will introduce our model of the smart city environment and our assumptions regarding the computational tasks. In Section IV we will describe our novel solution approach including the objective function to be solved. In order to assess the performance of this approach we compare it against two baseline algorithms. These simple dedicated heuristics will be introduced in Section V. We will detail the chosen scenario for the performance comparison and the corresponding results in Section VI and provide our conclusions in Section VII.

## II. RELATED WORK

The interaction of smart city environments with CAVs is an active research topic with a variety of diverse subtopics as the surveys of Mach and Becvar [13] and Kahn et al. [14] show. Among the works dealing with questions of assigning compute resources for offloading, there are different areas of focus ranging from analyzing the feasibility of applications and determining the required communication characteristics to planning and configuring the offloading assignment pattern. While our work falls in the last category, there are -to the best of our knowledge- no publications matching our scenario exactly. However, there are many works with somewhat similar scenarios.

Premsankar et al. [16] determine optimal locations to install RSUs with processing units based on expected average load scenarios extracted from an elaborate city scape and radio simulation. They determine the minimum number and locations of RSUs required to provide a given coverage and sufficient computation capabilities based on an integer linear programming (ILP) formulation.

Salahuddin et al. consider the case of vehicular networks incorporating RSUs and have developed an approach to assign RSU resources such that either the delay for services or the number of required resources is minimized by an ILP formulation in [15]. They also present a reinforcement learning-based approach to minimize the network reconfiguration effort between RSUs caused by an adaption to time-variant vehicle traffic behavior. In contrast to our work, they consider the network resources to be the primary bottleneck, whereas we focus on the processing capabilities.

Vondra and Becvar [17] utilize an intricate simulation environment gathering information on the future availability

and performance of compute resources and choose the computation site based on a ranking of these parameters with the goal of ensuring a timely execution of computation tasks.

Zhanget al. [18] assign computation resources to vehicles to minimize processing delays while also providing load balancing to the system at large. Since their primary figure of merit are delays, they model several different communication channels and their transmission characteristics in detail as part of a multi-tiered network structure. Similar to our work, they utilize a discrete time simulation and aim to provide a game theory-based solution approach that is capable of solving large scenarios very quickly.

The vastly different methodologies employed in the previous works can be ascribed to their intended usage scenarios. While ILP formulations can yield exact optima, they tend to be less scalable for large scenarios such that they are most useful for offline processing in long-term planning. Approaches based on ranking heuristics and game theory may not always be able to determine a global optimum, but can result in fairly good solutions very quickly. They are most suitable for situations with incomplete information or where decision making is subject to timing limitations.

## III. PROBLEM ILLUSTRATION

### A. SYSTEM MODEL

The principal elements given in our model are illustrated in Fig. 1(a), which shows three vehicles in a smart city environment with three RSUs attached to traffic lights and lamp posts. The RSUs have a limited radio range as indicated by the dashed circles, such that they can only connect to cars within their respective circles. The vehicle will connect to the RSU with the locally best wireless transmission characteristics, which in our model is the one with the least distance. To abstract this complex setting into a simple graph, we represent the RSUs (blue nodes denoted $A$, $B$, $C$) and vehicles (purple nodes denoted $p$, $q$, $r$) as vertices as shown in Fig. 1(b). The edges of the proposed graph represent communication channels. We assume fixed wired connections between two RSUs whenever they are directly connected by street segment, such that there is an edge between the RSUs at vertices $A$ and $B$ and another edge between $B$ and $C$. For the vehicles we assume wireless connections to the closest RSU, such that $p$ and $q$ have edges to $A$, whereas $r$ is connected to $C$. All edges are undirected since we expect bidirectional communications channels.

Having defined an abstraction of the given parameters of the smart city environment, we can now focus on the aspects pertaining to the problem itself. The goal is to find an association to an RSU node for each vehicle node such that the resulting set of associations is optimal for the given point in time. While we will provide a formal definition of the optimization goal in Section IV, we will for now merely state that it combines avoiding node overloads to ensure sufficient processing capacity, minimizing the distance to the associated nodes to improve latency and minimizing the number of association switches to increase stability. The associations themselves are represented as dashed lines between the blue and purple vertices in Fig. 1(b). While $p$ and $r$ are associated to the respective node that they also have a direct connection to, the vehicle $q$ is connected to RSU $A$, but associated to node $B$. This means that node $B$ performs the computation tasks for vehicle $q$, whereas node $A$ simply serves as a relay between the two. For the present scenario this achieves perfect load balancing between the three RSUs and also has the added advantage that $q$ will leave the radio coverage of $A$ towards $B$ such that no change in its association will be required in the next time step. Avoiding unnecessary changes in association is beneficial, because every change requires additional resources for the handover of task-relevant data between RSUs. Hence, avoiding unnecessary changes can add more stability to the system at large.

Efficiently minimizing the number of changes, however, requires prior knowledge of the future whereabouts of vehicles. When considering CAVs and RSUs, this becomes possible due to the fact that they exchange navigation information. Beyond local knowledge about vehicles and their associations, RSUs can further disseminate this information among each other and even to a central controller thanks to their wired connections such that the entire state information of all RSUs can be available to decision algorithms.

As previously established, the set of RSUs eligible for offloading by a vehicle is limited by distance and number of relaying nodes. Given the regular intervals at which RSUs need to be deployed for geographical reasons, we simplify our problem description by enforcing an upper bound $k$ on the number of hops between a vehicle and the RSU selected for association.

### B. ABSTRACT REPRESENTATION OF A SMART CITY

In order to illustrate the benefits of our proposed approach, we provide an example for a smart city topology consisting of 50 RSUs and 57 road segments/wired connections between them as illustrated in Fig. 2. Vehicles can move between RSU locations along these road segments and enter or exit the city at specific nodes with the numbers 0, 3, 5, 6, 11, 20, 22, 28, 33, 39, 42, 48 and 49, which are highlighted in the figure with green outlines.

This structure is representative for typical road networks found in Manhattan, New York, where roads are built in rectangular patterns with different lengths of housing blocks between them. This includes imperfections to the otherwise regular layout, e. g., missing street segments between nodes 33 and 42, as well as an entry/exit at node 20 in the middle of the grid which represents a connection to a tunnel or overpass, similar to how the Queens-Midtown tunnel or Ed Koch Queensboro bridge connect to Manhattan's road network.

We represent this example city as a graph $G_{\text{city}} = (C, S)$, where each node $c \in C$ represents a computation unit. All of these units are identical and can provide limited computational resources to the vehicles. Following the definition in the previous Section, the edges $s \in S$ between

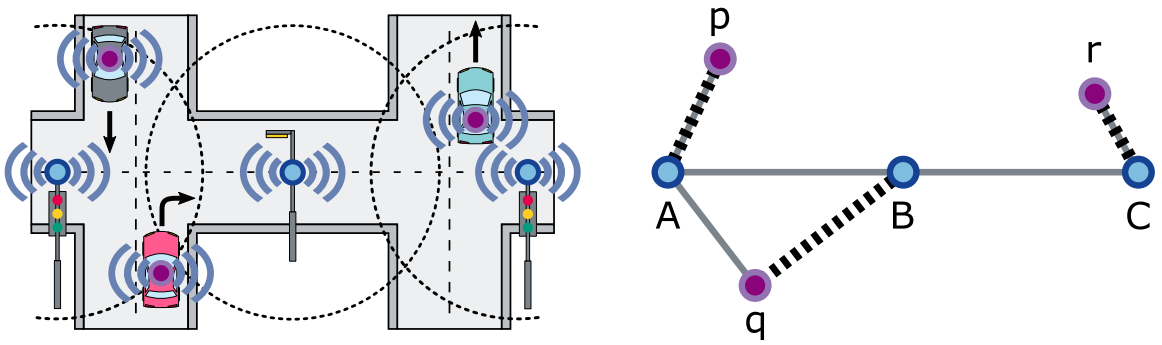



**FIGURE 1.** Vehicles and RSUs in a smart city environment and the corresponding graph representation. Dashed lines in the graph indicate offloading associations and solid lines represent direct communication channels.

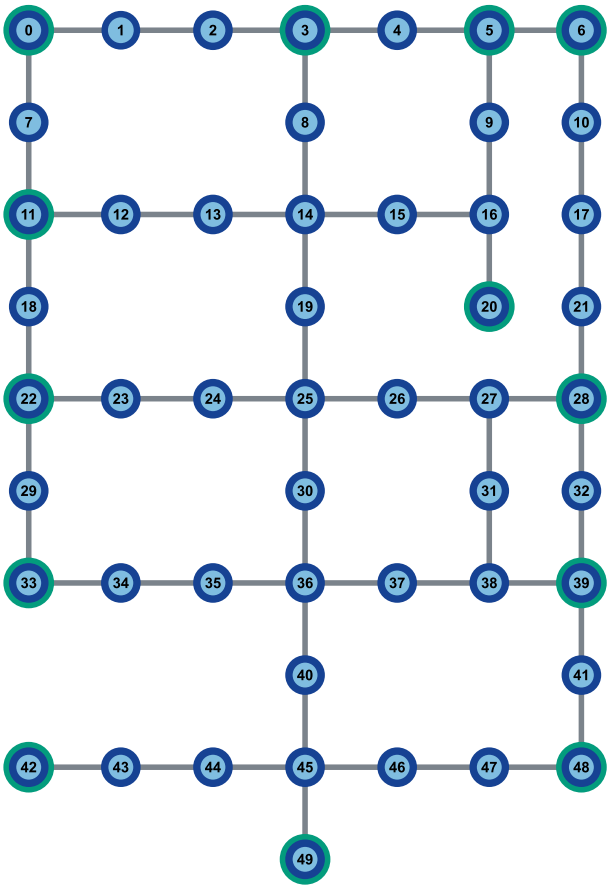



**FIGURE 2.** Graph of RSUs used for experiments. Nodes with green outlines are entry/exit nodes.

computation nodes signify that there is a road segment and that there is a wired communication connection between them. Fig. 2 therefore shows the graph $G_{\text{city}}$.

To simulate traffic, we represent the vehicles within the city as nodes $v \in V$ with edges $(v, c) \in W$. These edges indicate that a vehicle $v$ has a wireless connection to RSU $c$ as it is the closest computation unit. Note that this implies that we consider a dynamic graph model, since $V$ and $W$ are in fact time-dependent as connections change according to the current positions of the vehicles during their journey. Finally, we define a graph $G = (N, E)$ with $N = C \cup V$ and $E = S \cup W$ which therefore contains the entire data for a given moment in time.

## IV. DIFFERENTIAL ORCHESTRATION METHOD (DOM)

### A. SOFT-CONSTRAINED PROBLEM FORMULATION

The key idea for our proposed method is to soft-constrain an otherwise expensive combinatorial problem. Given a graph $G = (N, E)$ with $N = V \cup C$ as defined in the previous Section, we achieve this by assigning each node $n \in N$ a feature vector $f_n \in \mathbb{R}^p$ with $p \geq 1$. Let's consider the problem of computing the probability that a vehicle $n_{\text{car}} \in V$ should associate itself with an element $n_{\text{comp}}$ of the set of computation nodes $\{n^*_{\text{comp}}(j)\}_{j \in N^{G,k}(n_{\text{car}})}$ for task offloading.

Here, $n^*_{\text{comp}} : \mathbb{N} \to C$ is an index function whereas $N^{G,k}(n_{\text{car}})$ is the index list of all computation nodes in the local k-neighborhood of the car $n_{\text{car}}$, i. e., the indexes of all $n_{\text{comp}} \in C$ which can be reached by traversing at most $k$ edges in graph $G$ starting from the node $n_{\text{car}}$. We can now compute

$$P(n_{\text{car}} \text{ offload on } \{n^*_{\text{comp}}(j)\}_{j \in N^{G,k}(n_{\text{car}})}) = \text{softmax} \begin{bmatrix} \boldsymbol{f}^{\boldsymbol{T}}_{\boldsymbol{n}_{\text{car}}} \cdot \boldsymbol{f}_{\boldsymbol{n}^*_{\text{comp}}(N^{\boldsymbol{G},\boldsymbol{k}}_{\boldsymbol{1}}(\boldsymbol{n}_{\text{car}}))} \\ \boldsymbol{f}^{\boldsymbol{T}}_{\boldsymbol{n}_{\text{car}}} \cdot \boldsymbol{f}_{\boldsymbol{n}^*_{\text{comp}}(N^{\boldsymbol{G},\boldsymbol{k}}_{\boldsymbol{2}}(\boldsymbol{n}_{\text{car}}))} \\ \vdots \\ \boldsymbol{f}^{\boldsymbol{T}}_{\boldsymbol{n}_{\text{car}}} \cdot \boldsymbol{f}_{\boldsymbol{n}^*_{\text{comp}}(N^{\boldsymbol{G},\boldsymbol{k}}_{\boldsymbol{m}}(\boldsymbol{n}_{\text{car}}))} \end{bmatrix} \quad (1)$$

where softmax refers to the commonly used softmax function. With the above probability distribution, we can now represent the offloading problem as a continuous optimization problem. In the following we will define a differentiable loss function for each desired property of our proposed offloading setup.

### B. ORCHESTRATION LOSS

This loss term addresses the fact that no computation node should have to process more offloaded computation tasks

than it can handle. We define a maximum number of $m$ vehicle nodes that can be served simultaneously by any single computation node. Therefore, we define the loss function for a computation node $n_{\text{comp}} \in C$ as follows.

$$\begin{aligned}&\text{Loss}_{\text{Orchestration}}(n_{\text{comp}})\\&= \max\left(\sum_{i \in A^{G,k}(n_{\text{comp}})} P(n^*_{\text{car}}(i) \text{ offload on } n_{\text{comp}}) - m, 0\right)\end{aligned} \tag{2}$$

In an analogy to the previous definitions, $n^*_{\text{car}} : \mathbb{N} \to V$ is another index function, whereas $A^{G,k}(n_{\text{comp}})$ refers to an index list of vehicle nodes in the local k-neighborhood of $n_{\text{comp}}$ on graph $G$, such that it contains all possible candidates for association.

### C. PROXIMITY LOSS

This loss term aims to associate a vehicle to the closest eligible computation node. We define it for any vehicle node $n_{\text{car}} \in V$ as

$$\begin{aligned}&\text{Loss}_{\text{Proximity}}(n_{\text{car}})\\&= \sum_{j \in N^{G,k}(n_{\text{car}})} \text{dist}_G(n_{\text{car}}, n^*_{\text{comp}}(j))\\&\quad \cdot P(n_{\text{car}} \text{ offload on } n^*_{\text{comp}}(j)),\end{aligned} \tag{3}$$

where the operator $\text{dist}_G : V \times C \to \mathbb{N}$ returns the minimum distance on the graph $G$ as the number of hops, i. e., the number of edges to be traversed. Note that this is a loss function for our initial problem that does not contain the time component. In order to include that a vehicle should remain associated to the same computation node as long as possible, we will reformulate the $\text{dist}_G$ function. To this end, it makes sense to consider the nodes on the future path of the vehicle. As a heuristic we select a node for computation that is still in permissible distance from the current position, but as close to the expected future position as possible. Hence, we modify the previous functions as follows:

$$\begin{aligned}&\text{dist}^{+\text{t}}_G(n_{\text{car}}, n^*_{\text{comp}}(j))\\&= \begin{cases} 0 & \text{if } n^*_{\text{comp}}(j) \text{ is last node} \\ \text{dist}_G(n^{+t}_{\text{car}}, n^*_{\text{comp}}(j)) & \text{otherwise,}\end{cases}\end{aligned} \tag{4}$$

where $n^{+t}_{\text{car}}$ refers to the expected position of the car node after $t$ hops on the graph. Hereby, we assume that the local neighborhood considers up to $t$ hops with $t \leq k$. Or in other words, the furthest position away from $n_{\text{car}}$ that lies in the driving direction and is in its local neighborhood. Clearly, the formulation of $n^*_{\text{comp}}(j)$ is only meaningful for indexes $j \in N^{G,k}(n_{\text{car}})$. Finally, we like to point out that the trick of setting the distance to 0 at the last computation node is sufficient to let the system switch from one computation node to the next one if it is really necessary. This implies that there is no cost if the car continues to be associated to the current computation node. With this new distance we can now define the new loss function for the time-dependent dynamic graph:

$$\begin{aligned}\text{Loss}_{\text{Proximity}}(n_{\text{car}}, t) = &\sum_{j \in N^{G,k}(n_{\text{car}})} \text{dist}^{+\text{t}}_G(n_{\text{car}}, n^*_{\text{comp}}(j))\\&\cdot P(n_{\text{car}} \text{ offload on } n^*_{\text{comp}}(j)).\end{aligned} \tag{5}$$

### D. INTEGRITY LOSS

This loss term aims to prevent vehicles from associating to multiple computation units, which in our model would be an invalid state. To achieve this, we determine the largest association probability in the distribution of all nodes in the local neighborhood and subtract it from 1. Therefore, the optimization will attempt to maximize a single association probability and minimize all others. For a vehicle node $n_{\text{car}} \in V$ we define the loss term as follows.

$$\begin{aligned}&\text{Loss}_{\text{Integrity}}(n_{\text{car}})\\&= 1 - max_{j \in N^{G,k}(n_{\text{car}})}(P(n_{\text{car}} \text{ offload on } n^*_{\text{comp}}(j)).\end{aligned} \tag{6}$$

### E. TOTAL LOSS

From the above loss functions we can derive the optimization objective of our offloading problem

$$\begin{aligned}\text{Loss}_{\text{total}}(G) = &\quad \alpha \sum_{n_{\text{comp}} \in C} \text{Loss}_{\text{Orchestration}}(n_{\text{comp}})\\&+ \beta \sum_{n_{\text{car}} \in V} \text{Loss}_{\text{Proximity}}(n_{\text{car}}, t)\\&+ \gamma \sum_{n_{\text{car}} \in V} \text{Loss}_{\text{Integrity}}(n_{\text{car}})\end{aligned} \tag{7}$$

where $\alpha, \beta, \gamma \in \mathbb{R}$ are weighting coefficients. Now the problem can be easily optimized with any gradient descent-based optimization method.

## V. BASELINE METHODS

### A. GREEDY ALGORITHM

This algorithm is the simplest baseline approach where associations are determined by greedy selection based on distance. This means that each vehicle is always associated to the RSU that is the closest to itself in terms of number of hops on the graph. There is neither any consideration of existing associations of other vehicles nor of any processing limits. This method is very fast to compute, but clearly leads to sub-optimal results since RSUs in areas of dense traffic on main streets will be overloaded quickly.

### B. GREEDY ALGORITHM WITH ASSOCIATION KNOWLEDGE (G/AK)

This second baseline algorithm remediates the obvious problem of the standard greedy algorithm by incorporating information about the preexisting node associations. This method iteratively assigns each vehicle to a computation node based on analyzing all such nodes located within the permissible distance from the vehicle. It determines for each node how many vehicles are already associated to it and

selects the subset of all computation nodes with the least amount of associations. Should this subset consist of several computation nodes of identical number of associations, it will choose the one that has the least distance to the vehicle.

Note that this approach does not consider the maximum number of associations explicitly, but rather exhibits a general load balancing behavior. This is done on purpose since in times of unavoidable overload, an equal distribution of the computation load will be fairer towards the vehicles experiencing diminished QoS figures and also give equal chance for each affected computation node to recover from the overload situation.

### C. GREEDY ALGORITHM WITH PATH AND ASSOCIATION KNOWLEDGE (G/PAK)

This baseline algorithm is an enhancement to the previous algorithm and is explicitly designed to compete with the differential orchestration approach in the smart city scenario. This algorithm has the same knowledge as the differential orchestration approach, i. e., it can utilize the same distance metric, which was defined in equation (4). This enables the algorithm to also consider association switches of vehicles between computation nodes and enhance the system stability by reducing the number of changes.

The only drawback of this algorithm is that its decision making remains limited in scope to iteratively treating individual vehicles. This precludes an immediate optimization of the global system state such that solutions cannot be expected to be optimal for complex cases. The details for this approach are given in algorithm 1. Due to its two main loops the runtime behavior scales mostly with the number of cars in the set $V$ and the number of computation nodes in its local k-neighborhood.

**Algorithm 1** G/PAK

**input** : $G = (V \cup C, W \cup S); k \in \mathbb{N}; t \in \mathbb{N}$
**output:** $A \in (V \times C)^{\mathbb{N}}$ `// Vehicle-to-Node Associations`
**begin**
  $A \leftarrow \emptyset$
  **foreach** $v \in V$ **do**
    $d_{\min} \leftarrow \infty$
    $a_{\min} \leftarrow \infty$
    $c_{\text{cur}} \leftarrow \emptyset$
    **foreach** $j \in N^{G,k}(v)$ **do**
      $c \leftarrow n^{*}_{\text{comp}}(j)$
      $a \leftarrow |\{x \mid (x, c) \in A\}|$
      $d \leftarrow \text{dist}_{\text{G}}{}^{+t}(v, c)$
      **if** $a < a_{\min}$ **or** $(a = a_{\min}$ **and** $d < d_{\min})$ **then**
        $c_{\text{cur}} \leftarrow c$
        $d_{\min} \leftarrow d$
        $a_{\min} \leftarrow a$
    $A \leftarrow A \cup (v, c_{\text{cur}})$
  **return** $A$

## VI. EXPERIMENTS

In this Section we study the performance of the proposed algorithm and compare its results to those obtained from the baseline algorithms.

We chose the weighting coefficients explained in Section IV-E to be $\alpha = 5$, $\beta = 1$ and $\gamma = 0.3$. Furthermore, we use 20 iterations of our gradient descent method in the presented experiments. All coefficients have been found by testing many combinations of different values via grid search. We repeated every run for a total of 10 times in order to control for statistical effects and the result tables in the following Sections therefore show mean values and corresponding standard deviations. Except where explicitly noted otherwise, we use the same parameters for the algorithms and scenarios.

### A. INTUITIVE EXAMPLES AND ABLATION STUDY

Here we test the proposed algorithm on several intuitive cases outlined in Fig. 3. Depicted on the left-hand side are suboptimal solutions to the problems, whereas the right-hand side represents the optimal solution to the respective problems. For this analysis, optimal is to be understood in the following sense: No computation unit has more than one vehicle associated to itself as the primary condition with the secondary condition being that vehicles shall offload to units as close as possible to their location.

As before, the purple nodes in these figures represent the vehicles denoted by minuscules, whereas the blue nodes correspond to computation units denoted by majuscules. The gray edges indicate communication channels and the dashed line signifies that a vehicle is associated to a computation unit. Furthermore, all computation nodes show a number which corresponds to the sum of all its associations. If the number exceeds the maximum of one association, it is shown in red with circular highlight around the node.

The performance of our method and the competing methods is shown in table 1. We provide values for the number of associations that exceed the maximum, i. e., the overload, and also for distance given by the number of hops between vehicles and computation nodes. We also added the values corresponding to the optimal solutions from the right-hand side in figure 3 for both the overload and the distance metric. For each scenario, i. e., each row of the table, we highlight the value closest to the optimum in bold face. We note that the standard Greedy shows a standard deviation value of 0, because it is deterministic. Overall, we observe that our method outperformed both the Greedy and the G/AK algorithms by a significant margin in the overload metric (by a factor 3–7). Note that the overload metric is the more important metric, because it will have a more immediate and significant impact on the service quality.

Regarding the secondary metric, the average number of hops between a vehicle and its associated computation node,

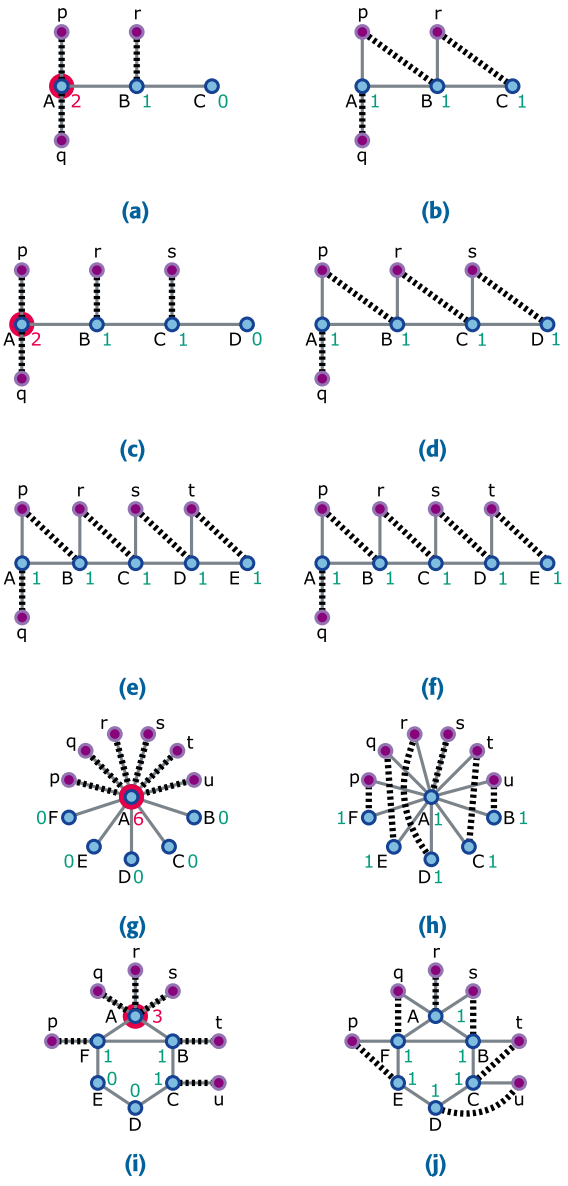



**FIGURE 3.** Experimental graphs illustrating problems. Numbers at RSUs (blue circles) denote associations with vehicles (purple circle). Red highlights overloaded RSUs. Dashed lines indicate offloading associations and solid lines represent direct communication channels.

the standard Greedy works best. This is because it simply uses the closest computation unit. However, we note that our method has values similar to the theoretic values of the optimal solution.

Finally, we notice that the performance of our method strongly depends on the number of parameters it has available. That is, the problem can not be solved easily in case the number of parameters is significantly lower than the number of vehicles in any local neighborhood of any RSU in the system.

### B. SMART CITY EXAMPLE STUDY

Here, we present our results for several scenarios based on the more realistic graph explained in Section III-B. We simulate vehicle traffic using different numbers of vehicles driving though the city over a period of 200 time steps. The vehicles appear in the city at randomized entry nodes at randomized points in time and drive to their randomized target locations where they leave the system. Vehicle speed is chosen uniformly such that each time step marks a transition of the wireless connection from one RSU to another RSU for every vehicle in the system. For reasons of simplicity and to highlight the algorithm behavior in scenarios of significant overload, we chose to limit the computational resources of the RSUs to a maximum of one vehicle association. The paths through the city are also randomized for each vehicle resulting in a minimum length of 2 and a maximum of 29 road segments. On average vehicles establish wireless connections to about 16 different RSUs during their journeys.

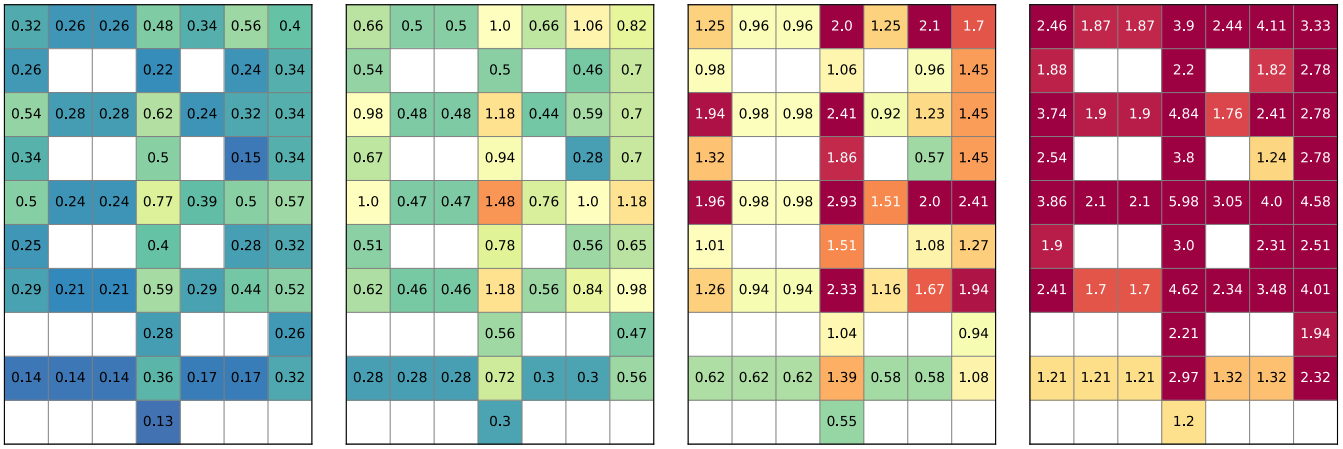
We consider four different traffic scenarios: one with low traffic (200 vehicles), moderate traffic (400 vehicles), heavy traffic (800 vehicles), and intentional overload (1600 vehicles). For the lower traffic scenarios it is typically possible not to overload the system, which does not hold true for the heavier traffic situations up to the intentional overload scenario. Fig. 4 shows the average number of cars per time step that are present and connected at RSUs of the graph shown in Fig. 2 for the different traffic scenarios. Recall that in our setup each RSU can only handle one car. Blue and green colors show nodes where the number of cars is low enough that a computation on the local node is possible in the average case such that the scenarios with 200 and 400 cars are expected to be solvable with only little overload. Yellow and orange show loads exceeding the local computational resources that may still be offloaded to lightly loaded RSUs in the permissible proximity, whereas red indicates that unavoidable overloads are to be expected.

For the evaluation we run each of the algorithms once at every time step such that a vehicle may experience a maximum number of association switches equal to the number of time steps it remains in the city. Like with the simple examples of the previous chapter, we repeat each run for a total of 10 times and present mean values with corresponding standard deviations. Due to the complexity of the problem, it is not possible to compute the optimal ground truth solution. Hence, we compare the results purely numerically.

We consider two metrics. The main metric is the average percentage of nodes that are overloaded over the entire time interval. Hence, the best score is 0% and the worst is 100%. This essentially tests if the system would perform well with respect to the amount of computational load. The numerical study for this metric can be found in table 2. We find that the performance of our method is superior to the two advanced baselines at 200 and 400 vehicles, except for the cases of very large neighborhoods. For larger numbers of vehicles, however, our method consistently outperforms the baseline approaches by a significant margin. We note that our method performs especially well with a low number of parameters.

**TABLE 1. Performance and ablation study.**

| Algorithm | Optimum | Greedy | G/AK | DOM, p=2 | DOM, p=3 | DOM, p=7 |
|---|---|---|---|---|---|---|
| Problem | Overload Metric: mean ± standard deviation | | | | | |
| Fig. 3a | 0.000 | 1.000 ± 0.000 | 0.676 ± 0.470 | **0.010** ± 0.099 | 0.022 ± 0.147 | 0.085 ± 0.279 |
| Fig. 3c | 0.000 | 1.000 ± 0.000 | 0.912 ± 0.283 | **0.133** ± 0.339 | 0.166 ± 0.372 | 0.319 ± 0.466 |
| Fig. 3e | 0.000 | 1.000 ± 0.000 | 0.986 ± 0.117 | **0.376** ± 0.488 | 0.39 ± 0.488 | 0.567 ± 0.495 |
| Fig. 3g | 0.000 | 5.000 ± 0.000 | **0.000** ± 0.000 | 1.641 ± 0.767 | 0.518 ± 0.637 | **0.000** ± 0.000 |
| Fig. 3i | 0.000 | 2.000 ± 0.000 | 0.972 ± 0.165 | 0.266 ± 0.477 | **0.206** ± 0.424 | 0.428 ± 0.526 |
| Problem | Distance Metric: mean ± standard deviation | | | | | |
| Fig. 3a | 0.666 | 0.000 ± 0.000 | 0.215 ± 0.312 | **0.659** ± 0.066 | 0.652 ± 0.096 | 0.610 ± 0.186 |
| Fig. 3c | 0.750 | 0.000 ± 0.000 | 0.124 ± 0.220 | **0.65** ± 0.253 | 0.628 ± 0.274 | 0.526 ± 0.334 |
| Fig. 3e | 0.800 | 0.000 ± 0.000 | 0.092 ± 0.154 | **0.505** ± 0.382 | 0.498 ± 0.381 | 0.420 ± 0.354 |
| Fig. 3g | 0.833 | 0.000 ± 0.000 | **0.833** ± 0.000 | 0.795 ± 0.126 | 0.823 ± 0.070 | **0.833** ± 0.000 |
| Fig. 3i | 0.833 | 0.000 ± 0.000 | 0.301 ± 0.159 | **0.764** ± 0.175 | 0.745 ± 0.190 | 0.650 ± 0.232 |




**(a) 200 Vehicles** **(b) 400 Vehicles** **(c) 800 Vehicles** **(d) 1600 Vehicles**

**FIGURE 4. Average number of vehicles at RSU locations during the simulated time period for different traffic scenarios. The color scale from blue to red indicates increasing overlaod.**

**TABLE 2. Smart city - evaluation.**

| Algorithm | | Greedy | G/AK | G/PAK | DOM, $p = 4$ | DOM, $p = 8$ | DOM, $p = 16$ | DOM, $p = 32$ |
|---|---|---|---|---|---|---|---|---|
| Cars | Hops | Percentage of Nodes that are overloaded: mean ± standard deviation | | | | | | |
| 200 | 2 | 5.36 | 0.654 ± 0.058 | 0.651 ± 0.0255 | **0.135** ± 0.0163 | 0.2 ± 0.0179 | 0.394 ± 0.0258 | 0.796 ± 0.0836 |
| 200 | 3 | 5.36 | 0.062 ± 0.0218 | 0.046 ± 0.0211 | **0.019** ± 0.0094 | 0.04 ± 0.0179 | 0.161 ± 0.0391 | 0.435 ± 0.0552 |
| 200 | 4 | 5.36 | 0.01 ± 0.0045 | **0.003** ± 0.0046 | 0.035 ± 0.0186 | 0.082 ± 0.0303 | 0.235 ± 0.0602 | 0.51 ± 0.0527 |
| 400 | 2 | 15.48 | 7.91 ± 0.1165 | 7.927 ± 0.123 | **3.972** ± 0.1073 | 4.676 ± 0.0544 | 5.8 ± 0.1467 | 7.148 ± 0.1175 |
| 400 | 3 | 15.48 | 3.78 ± 0.0772 | 3.519 ± 0.141 | **2.36** ± 0.1334 | 2.989 ± 0.0984 | 4.324 ± 0.1216 | 5.903 ± 0.1236 |
| 400 | 4 | 15.48 | 1.965 ± 0.0706 | **1.723** ± 0.1041 | 2.518 ± 0.1028 | 3.191 ± 0.1332 | 4.564 ± 0.1177 | 6.13 ± 0.1092 |
| 800 | 2 | 37.18 | 44.289 ± 0.1456 | 44.225 ± 0.1419 | **29.811** ± 0.2033 | 30.604 ± 0.2262 | 31.872 ± 0.1831 | 32.842 ± 0.2289 |
| 800 | 3 | 37.18 | 45.024 ± 0.1003 | 44.998 ± 0.1097 | **28.381** ± 0.2794 | 29.623 ± 0.1855 | 31.103 ± 0.2008 | 32.634 ± 0.3065 |
| 800 | 4 | 37.18 | 44.429 ± 0.0541 | 44.369 ± 0.1089 | **27.33** ± 0.3922 | 29.028 ± 0.232 | 30.543 ± 0.3023 | 32.479 ± 0.2689 |
| 1600 | 2 | 63.57 | 79.102 ± 0.1207 | 78.961 ± 0.0667 | **61.982** ± 0.2948 | 62.643 ± 0.3554 | 63.651 ± 0.2531 | 64.413 ± 0.2834 |
| 1600 | 3 | 63.57 | 82.52 ± 0.0752 | 82.553 ± 0.1 | **57.972** ± 0.2823 | 60.584 ± 0.3404 | 62.525 ± 0.2891 | 64.712 ± 0.2383 |
| 1600 | 4 | 63.57 | 83.613 ± 0.0617 | 83.675 ± 0.0528 | **53.428** ± 0.3659 | 58.332 ± 0.3374 | 60.497 ± 0.2632 | 63.797 ± 0.2614 |
| Cars | Hops | Association Switches per Vehicle: mean ± standard deviation | | | | | | |
| 200 | 2 | 16.78 | 14.466 ± 0.046 | **7.014** ± 0.042 | 7.755 ± 0.065 | 8.57 ± 0.051 | 9.486 ± 0.11 | 10.138 ± 0.091 |
| 200 | 3 | 16.78 | 13.934 ± 0.079 | **4.486** ± 0.029 | 6.468 ± 0.089 | 7.906 ± 0.118 | 9.558 ± 0.073 | 10.969 ± 0.107 |
| 200 | 4 | 16.78 | 13.52 ± 0.099 | **3.381** ± 0.035 | 7.171 ± 0.115 | 9.01 ± 0.146 | 10.914 ± 0.127 | 12.303 ± 0.097 |
| 400 | 2 | 16.42 | 13.861 ± 0.077 | **8.014** ± 0.038 | 8.763 ± 0.079 | 9.48 ± 0.067 | 10.299 ± 0.093 | 10.853 ± 0.096 |
| 400 | 3 | 16.42 | 13.763 ± 0.041 | **5.935** ± 0.062 | 7.322 ± 0.117 | 8.966 ± 0.066 | 10.688 ± 0.111 | 11.774 ± 0.064 |
| 400 | 4 | 16.42 | 13.582 ± 0.046 | **4.726** ± 0.075 | 7.9 ± 0.081 | 9.78 ± 0.084 | 11.64 ± 0.078 | 12.768 ± 0.101 |
| 800 | 2 | 16.42 | 13.745 ± 0.038 | 9.029 ± 0.037 | **7.918** ± 0.028 | 8.495 ± 0.038 | 9.269 ± 0.042 | 9.926 ± 0.072 |
| 800 | 3 | 16.42 | 13.911 ± 0.026 | 7.535 ± 0.036 | **7.378** ± 0.067 | 8.542 ± 0.052 | 10.019 ± 0.076 | 11.297 ± 0.068 |
| 800 | 4 | 16.42 | 13.944 ± 0.054 | **6.853** ± 0.04 | 8.228 ± 0.075 | 9.753 ± 0.044 | 11.308 ± 0.084 | 12.565 ± 0.037 |
| 1600 | 2 | 16.363 | 13.678 ± 0.032 | 9.806 ± 0.033 | **7.225** ± 0.016 | 7.466 ± 0.03 | 7.997 ± 0.022 | 8.681 ± 0.025 |
| 1600 | 3 | 16.363 | 14.009 ± 0.031 | 8.606 ± 0.046 | **7.391** ± 0.05 | 8.078 ± 0.042 | 9.123 ± 0.03 | 10.352 ± 0.032 |
| 1600 | 4 | 16.363 | 14.11 ± 0.02 | **8.16** ± 0.041 | 8.547 ± 0.047 | 9.571 ± 0.039 | 10.801 ± 0.033 | 12.05 ± 0.058 |

The standard Greedy algorithm is utterly outperformed by most other approaches, but shows seemingly good results at very high system loads. To illustrate the underlying effect, we show the percentage of time every individual RSU was overloaded for one specific scenario in Fig. 5. Since the standard Greedy algorithm cannot perform any load balancing, it drastically overloads RSUs at traffic hotspots impacting a large number of vehicles, whereas a small number of vehicles in lightly loaded areas intermittently receive a much higher service quality. This effect becomes more and more exaggerated the more the system is overloaded in an asymmetric way. Furthermore, we can see that not only do

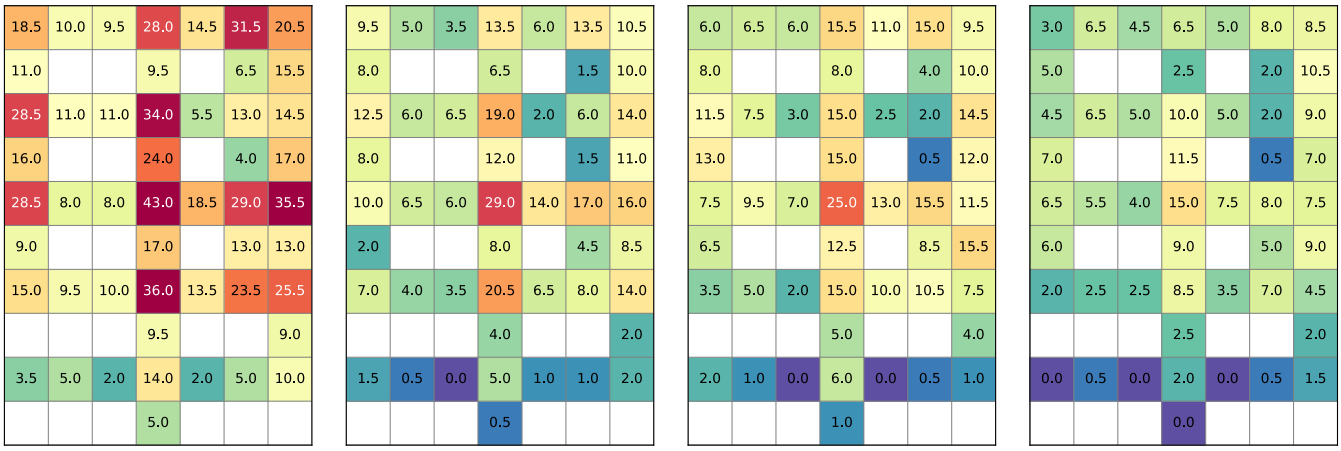



**FIGURE 5.** Percentage of time steps that a node is overloaded for different methods given 400 cars and a 2-neighborhood. The color scale from blue to red indicates an increasing percentage.

all other methods achieve effective load balancing, relieving in particular the nodes with the highest traffic density from Fig. 4, but also that the suggested approach leads to less overloading compared to the other methods.

The second metric is the number of RSU association switches during the journey of a vehicle as shown in table 2. These values were obtained by averaging the number of switches over all cars during the 200 time steps. Here we find that the G/PAK baseline algorithm provides the best results for a low number of cars while performing similar to DOM once the number of vehicles is 800 or 1600. DOM still outperforms the other two baselines regarding this metric. We would like to highlight that this secondary metric is of lower importance, because the association switches only matter when the system is not overloaded.

Summarizing our findings, it can be stated that our method outperformed all baseline approaches in high traffic situations, while performing similarly or better in the simpler cases. Note that in order to provide a fair comparison, we only considered optimization criteria that can also easily be incorporated into Greedy settings. But in general, our method can be easily updated with any other kind of differentiable loss function and even extended to other problems regarding CAVs as well.

Regarding the runtime behavior, we found that DOM primarily scales with the number of vehicles and secondly with the number of hops in the local neighborhood in the same way as G/PAK does. While all reference algorithms require only a few milliseconds to complete their calulcations for one step in time, the actual optimization within DOM takes more time. The smaller scenarios with 200 and 400 vehicles took on average 0.291–0.307 and 0.532–0.563 seconds to complete. For 800 vehicles this increased to a range of 1.04–1.59 and for 1600 vehicles even up to 2.12–3.20 seconds. While the scope and methods were vastly different in the related work, it can be noted that these numbers are very comparable to the results given in [16] where it is stated that the optimization completed in under 2 seconds for 1000 cars, and to the results in [18] where a scenario with 10 RSUs and 120 vehicles required 0.5 seconds to complete. We point out that the evaluation was performed using a single-threaded implementation of DOM. However, the entire process is parallelizable such that every n-hop neighborhood of each RSU in the smart city can run in its own thread. This should allow for a significant increase in speed, as well as scalability to environments of arbitrary size.

Furthermore, in order to make this study easily reproducible and serve as baseline in future research projects we make the smart city graph and car simulations available on GitHub.[1]

## VII. CONCLUSION

In this work, we have presented a novel gradient-based optimization approach and applied it to a computation offloading problem between vehicles and RSUs in a smart city environment. We have formulated this scenario as a combinatorial optimization problem on a dynamic graph. The evaluation has shown that the proposed approach is not only very fast in terms of runtime thanks to its differentiable property, but also that it outperformed several baseline methods consisting of increasingly intelligent heuristics by a significant margin in most cases. Especially for very large and complex vehicle scenarios it is up to 36% better compared to the most advanced baseline.

While the presented version of the offloading problem was limited in terms of real-world parameters, there is no principal argument not to extend it to arbitrary complexity as long as the objective function can be formulated in a differentiable way. This extensibility and good scaling behavior make the algorithmic approach a very good candidate to solve similar problems in smart city environments that are currently being built in multiple countries (e. g., New York, London, Wuxi IOT city)

[1]The data is publicly available at https://www.github.com/etas/SynTiSC

Furthermore, the algorithmic approach is not just applicable to these scenarios, but can be used to solve all kinds of assignment problems that can be formulated as graphs like coordination of wireless transmissions for vehicular ad-hoc communication networks, software – hardware mapping in vehicle architectures or any generic resource assignment problem. We are looking forward to exploring these options in future works.

## ACKNOWLEDGMENT

The authors are responsible for the content of this article.

The contributions of Michael Oechsle and Thilo Strauss were carried out while employed with ETAS Research.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, views or statements, either expressed or implied, of the affiliated organizations of the authors.

## REFERENCES

[1] P. Coppola and F. Silvestri, "Autonomous vehicles and future mobility solutions," in *Autonomous Vehicles and Future Mobility*. Amsterdam, The Netherlands: Elsevier, 2019, pp. 1–15.

[2] *IEEE Standard for Information Technology—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments*, IEEE Standard 802.11p-2010, Amendment to IEEE Standard 802.11-2007 as amended by IEEE Standard 802.11k-2008, IEEE Standard 802.11r-2008, IEEE Standard 802.11y-2008, IEEE Standard 802.11n-2009, and IEEE Standard 802.11w-2009, 2010, pp. 1–51, doi: 10.1109/IEEESTD.2010.5514475.

[3] B. Toghi, M. Saifuddin, H. N. Mahjoub, M. O. Mughal, Y. P. Fallah, J. Rao, and S. Das, "Multiple access in cellular V2X: Performance analysis in highly congested vehicular networks," in *Proc. IEEE Veh. Netw. Conf. (VNC)*, Taipei, Taiwan, Dec. 2018, pp. 1–8, doi: 10.1109/VNC.2018.8628416.

[4] *Intelligent Transport Systems—Truck Platooning Systems (TPS)—Functional and Operational Requirements*, ISO Standard 4272, Geneva, Switzerland, 2022.

[5] *Vulnerable Road User Safety Message Minimum Performance Requirement*, SAE Standard J2945/9_201703, 2017.

[6] *Intelligent Transport Systems—Automated Valet Parking Systems (AVPS))—Part 1: System Framework, Requirements for Automated Driving and for Communications Interface*, ISO/FDIS Standard 23374-1, Geneva, Switzerland, 2023.

[7] Bosch Media Service. (Dec. 2, 2022). *Stuttgart Airport Set to Welcome Fully Automated Driverless Parking*. Accessed: Apr. 13, 2023. [Online]. Available: https://www.bosch-presse.de/pressportal/de/en/press-release-219648.html

[8] *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Facilities Layer Protocols and Communication Requirements for Infrastructure Services*, ETSI Standard TS 103 301 V2.1.1, 2021.

[9] K. Hartman, K. Wunderlich, M. Vasudevan, K. Thompson, B. Staples, S. Asare, J. Chang, J. Anderson, and A. Ali, "Advancing interoperable connectivity deployment: Connected vehicle pilot deployment results and findings," Dept. Transp., Intell. Transp. Syst. Joint Program Office, Washington, DC, USA, Tech. Rep. FHWA-JPO-23-990, 2023. [Online]. Available: https://rosap.ntl.bts.gov/view/dot/6812

[10] SMLL Limited. *Smart Mobility Living Lab London*. Accessed: Apr. 13, 2023. [Online]. Available: https://smartmobility.london/

[11] R. Zhang, W. Zhong, N. Wang, R. Sheng, Y. Wang, and Y. Zhou, "The innovation effect of intelligent connected vehicle policies in China," *IEEE Access*, vol. 10, pp. 24738–24748, 2022, doi: 10.1109/ACCESS.2022.3155167.

[12] *Multi-access Edge Computing (MEC); Use Cases and Requirements*, ETSI Standard GS MEC 002 V3.1.1, 2023.

[13] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017, doi: 10.1109/COMST.2017.2682318.

[14] L. U. Khan, I. Yaqoob, N. H. Tran, S. A. Kazmi, T. N. Dang, and C. S. Hong, "Edge-computing-enabled smart cities: A comprehensive survey," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10200–10232, Oct. 2020, doi: 10.1109/JIOT.2020.2987070.

[15] M. A. Salahuddin, A. Al-Fuqaha, and M. Guizani, "Software-defined networking for RSU clouds in support of the Internet of Vehicles," *IEEE Internet Things J.*, vol. 2, no. 2, pp. 133–144, Apr. 2015, doi: 10.1109/JIOT.2014.2368356.

[16] G. Premsankar, B. Ghaddar, M. Di Francesco, and R. Verago, "Efficient placement of edge computing devices for vehicular applications in smart cities," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Taipei, Taiwan, Apr. 2018, pp. 1–9, doi: 10.1109/NOMS.2018.8406256.

[17] M. Vondra and Z. Becvar, "QoS-ensuring distribution of computation load among cloud-enabled small cells," in *Proc. IEEE 3rd Int. Conf. Cloud Netw. (CloudNet)*, Luxembourg, Luxembourg, Oct. 2014, pp. 197–203, doi: 10.1109/CLOUDNET.2014.6968992.

[18] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2092–2104, Feb. 2020, doi: 10.1109/TVT.2019.2959410.

**THILO STRAUSS** received the M.Sc. and Ph.D. degrees in mathematical sciences from Clemson University, in 2011 and 2015, respectively. He was a Senior Postdoctoral Research Fellow with the University of Washington, Seattle, in 2015, and he joined ETAS Research (Robert Bosch GmbH), in 2017. He was a Lead Technologies Officer for data science and other positions with ETAS, until 2023. In 2024, he became an Associate Professor with the School of AI and Advanced Computing, Xi'an Jiaotong-Liverpool University, China. Since 2020, he has been an Associate Member of the graduate faculty position with the Department of Mathematics and Statistics, University of North Carolina at Charlotte. His research interests include machine learning, security, the IoT, Bayesian statistics, inverse problems, automotive applications, and computer vision.

**MICHAEL OECHSLE** received the B.Sc. and M.Sc. degrees in physics from the University of Stuttgart. He spent his Ph.D. time with ETAS Research and MPI for Intelligent Systems. Previously, he was with ETAS Research and he is currently a Research Scientist with Google, Zürich. His research interests include computer vision, 3D reconstruction, and neural representations.

**UWE BAUKNECHT** received the Diploma degree in computer science from the University of Stuttgart, Germany, in 2013, and the Ph.D. degree from the Institute of Communication Networks and Computer Engineering (IKR), University of Stuttgart, in 2019. He was a Research Staff Member with IKR, University of Stuttgart, from 2013 to 2019. Since 2019, he has been continued his Ph.D. research with the University of Stuttgart before he joined ETAS Research, in 2023. His research interests include optimization methods, graph theory, and distributed systems with a special focus on communication networks, and cloud computing.