Routing Multiple Cars in Large Scale Networks: Minimizing Road Network Breakdown Probability*

Hongliang Guo[1,2], Zhiguang Cao[1,2], Jie Zhang[1], Dusit Niyato[1] and Ulrich Fastenrath[3]

***Abstract*— Traffic has become a universal metropolitan problem. This paper aims at easing the traffic jam situation through routing multiple cars cooperatively. We propose a novel distributed multi-vehicle routing algorithm with an objective of minimizing the road network breakdown probability. The algorithm is distributed, and hence highly scalable, making it applicable for large scale metropolitan road networks. Our algorithm always guarantees a much faster convergence rate than traditional distributed optimization techniques such as dual decomposition. Additionally, the algorithm always guarantees a feasible solution during the optimization process. This feature allows for real time decision making when applied to scenarios with time limits. We show the effectiveness of the algorithm by applying it to an arbitrarily large road network in simulation environment.**

Keywords: ***cooperative routing, distributed optimization, large scale network, road network breakdown***

I. INTRODUCTION

As the number of vehicles grows rapidly each year, traffic congestion has become a big issue for civil engineers in almost all metropolitan cities. Statistics tell an alarming story: the Americans spend 4.2 billion hours a year stuck in traffic, trying to commute or transport goods to market [1]. The Annual Mobility Report released by the Texas Transportation Institute tracks the costs of traffic immobility. Its latest study reported that travelers in 68 urban areas spent more than $72 billion in lost time and wasted fuel, or about $755 annually per driver. That is more than the cost of auto insurance in many places [2]. After stating those statistics, our research initiative becomes quite straightforward; we are trying to save money through reducing the occurrence of traffic jams in transportation networks.

In this paper, we consider the problem of routing multiple cars in an urban area. Our objective is to ease the traffic jam situation through routing some of the cars away from congested regions. In particular, we rely on the optimal principle for traffic and transportation networks with road bottlenecks introduced by Kerner recently in [3]. Then we formulate the optimization problem of routing multiple cars as minimizing the road network breakdown probability. Specifically, the road network breakdown minimization principle [3] states that the network optimum is achieved, when dynamic traffic optimization and/or control are performed in the network in such a way that the probability for spontaneous occurrence of traffic breakdown at one of the network bottlenecks during a given observation time reaches the minimum possible value. The statement is equivalent to maximizing the probability that traffic breakdown occurs at none of the network bottlenecks. Here, it is worth pointing out clearly that in this paper, we treat the terms like "routing multiple cars" and "multi-vehicle routing" as the same problem.

*This work is supported by BMW

[1] Hongliang Guo, Zhiguang Cao, Jie Zhang and Dusit Niyato are with the School of Computer Engineering, Nanyang Technological University, Singapore ({guohl, zhangj, dniyato}@ntu.edu.sg).

[2]Hongliang Guo and Zhiguang Cao are with the Energy Research Institute @ NTU (ERI@N), Interdisciplinary Graduate School, Nanyang Technological University, Singapore (guohl@ntu.edu.sg, caoz0005@e.ntu.edu.sg).

[3]Ulrich Fastenrath is head of traffic information management and routing optimization at BMW Group, Munich, Germany (Ulrich.Fastenrath@bmw.de).

A. Related Work

The research area of multi-vehicle routing has gained increasing interest over the past decades. There are, in general, two perspectives to look into the multi-vehicle routing problem. One is from individual's perspective, and the other is from network's perspective.

1) Agent Perspective: Since vehicles and urban traffic lights can be naturally modelled as agents, most of the researchers are applying multi-agent/agent-based technologies to solve the multi-vehicle routing problem [4], [5], [6], [7].

Agents may represent drivers, vehicles, or other traffic participants. They are explicitly present as active entities in an environment representing the road network where they may exhibit arbitrary complex information processing and decision making. Their behaviour, especially those resulting in simulated movement, can be visualized, monitored, and validated at individual level, leading to new possibilities for analysing, debugging, and illustrating traffic phenomena [8]. We can divide the agent-based approaches within the application domain of traffic management into three subareas, namely, agent-based vehicle control [4], [9], [10], [11], [12], agent-based traffic light control [6], [13], [14], and hybrid agent-based control [7], [15].

Literatures in agent-based vehicle control [4], [9], [10], [11], [12] model vehicles as agents. In this case, multiple vehicles are naturally mapped to multiple agents, and agents negotiate with each other and reach optimal policies for routing. Agents may follow certain behavior rules [9], or they may be able to learn [10], [11] and react to changing environments.

Another direction is to solve the multi-vehicle routing problem from traffic light control point of view [6], [13], [14]. They model traffic lights (in contrast to vehicles) as agents, and agents coordinate and achieve optimal policies for effective multi-vehicle routing.

Recently, there is a new trend of modeling both traffic lights and vehicles as agents [7], [15]. In this case, it is a hybrid agent-based control. Jiang et al. [15] proposed a co-evolutionary strategy to control both the traffic lights and vehicles. Appealing results are demonstrated in simulation.

2) Network Perspective: Compared to agent-based approaches, there is not so much literature reported from network perspective. Literatures in this direction tend to find a certain network equilibrium [16], [17], [18] (i.e., Wardrop equilibrium [19]) which ensures system-level optimum. Recently, Kerner [16] points out that there is a general problem in traffic network theories. Principles for network optimization are usually with the objective function associated with the minimization of some travel cost (travel time, fuel consumption, etc.). These principles do not take into account the empirical features of traffic breakdown, which occurs with some probability in the network. Traffic breakdown usually leads to complex spatiotemporal congestion propagation, and hence changes the dynamics of travel cost (travel time, fuel consumption) dramatically. Therefore, Kerner proposes a novel network optimization objective, which is to minimize the road network breakdown probability. We adopt this objective in the paper.

Agent-based approaches are, in general, scalable with the road network size and the number of participating agents, because each agent is trying to maximize its own objective. However, it is almost impossible for agent-based approaches to guarantee achieving a global optimum. On the other hand, network-based approaches will take the system's overall performance as the evaluation metric. Therefore, the global optimum is well defined and achievable. However, the network-based approaches suffer from the scalability issue, both in terms of the road network size and the number of agents.

B. Contributions

This paper aims at bridging the gap between the two kinds of approaches by developing a distributed network optimization algorithm. We introduce several matrix decomposition techniques into the network optimization process, and make the network optimization process always satisfy the constraints. The algorithm makes use of the second order derivative of the objective function (Hessian matrix), hence it is faster than canonical distributed algorithms which only make use of the first order derivative.

Traditional distributed optimization algorithms, like dual decomposition [20], suffer from two main drawbacks: slow convergence rate and infeasible points during iteration. The latter implies that we have to wait until the end of the algorithm to get a feasible point, making the algorithm inefficient. In our algorithm, we decompose the global problem after the interior point method (IPM) [21]. In this way, we always guarantee a feasible solution during the update. Moreover, since we make use of the second order derivative during the update, the convergence rate of our algorithm is much faster than that of dual decomposition. Unlike the typical multi-vehicle routing problem, in which all vehicles in the road network are controllable, we consider the case that there are uncontrollable default traffic loads in the road network. Our contribution can then be summarized as follows:

- We propose a distributed approach to solve the network optimization problem, which achieves both scalability and global optimum.
- We consider more realistic cases that there are default road network loads which are uncontrollable.
- Our distributed approach always guarantees a much faster convergence rate than traditional distributed optimization techniques, e.g., dual decomposition, as will be shown in the simulation section.
- We guarantee to achieve an always-feasible solution during the iteration process, which is unachievable for canonical distributed optimization techniques.

C. Paper Structure

The rest of the paper is organized as follows: we first pose the road network breakdown probability problem as a convex optimization problem in Section II. Then, we present the distributed algorithm in Section III. Section IV shows the simulation results and finally, Section V states the conclusion of the current work and proposes our future research directions.

II. Problem Formulation

We want to minimize the road network breakdown probability. That is, we will minimize:

$$1 - \prod(1 - Prob(r_i + x_i)) \tag{1}$$

where r_i is the default road network load. Prob$(\cdot)$ is the road breakdown probability function, and the input parameter, x_i, is the controllable load of the road.

We have an equality constraint which is the network flow conservation equation defined as follows:

$$A\boldsymbol{x} = \boldsymbol{b} \tag{2}$$

where $\boldsymbol{A}$ is the road network topology description, which is assumed to be known, and $\boldsymbol{b}$ is a column vector specifying the origin and destination of the vehicles. $\boldsymbol{A}$ is a $m \times n$ matrix, where m is the number of nodes in the road network, and n is the number of edges in the network. Additionally, the controllable load should be nonnegative defined as follows:

$$\boldsymbol{x} \succeq \boldsymbol{0}. \tag{3}$$

We take logistic regression [22], [23] as the mathematical model for road network breakdown probability, which can be expressed as:

$$p(x) = \frac{\mathrm{e}^{wx+c}}{1 + \mathrm{e}^{wx+c}} \tag{4}$$

where p is the probability that there occurs a traffic breakdown in the road segment and x is the load of the road segment, and w and c are model parameters, where w gauges the unit impact of road load to road breakdown, and c quantifies miscellaneous other factors (e.g., weather and road

width) which influence road breakdown. In this paper, we assume that we already know the default road load r_i in Eq. 1 and the mathematic model of the road breakdown probability (w and c).

Applying log transformation of Eq. 1, and taking into account of Eq. 4, we can transform the objective in Eq. 1 to be:

$$\begin{aligned} &\underset{\boldsymbol{x}}{\text{minimize:}} \\ &\ln(-\prod(1-Prob(r_i+x_i))) \\ =&\sum_i \ln(1+\mathrm{e}^{w_i(x_i+r_i)+c_i}). \end{aligned} \tag{5}$$

Summarizing the transformed objective in Eq. 5 and the original constraints, we can formally pose our problem as:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \sum_i \ln(1+\mathrm{e}^{w_i(x_i+r_i)+c_i}) \\ \text{subject to} \quad & \boldsymbol{Ax}=\boldsymbol{b} \\ & \boldsymbol{x}\succeq \boldsymbol{0}. \end{aligned} \tag{6}$$

III. Methodology

Before going into the details of the methodology, we will first lay down the algorithm's logical flow. We apply the canonical interior point method (IPM) [21], and use matrix decomposition techniques to make the key matrix inversion process distributed. When IPM is finished, our algorithm reaches global optimum. We are, in essence, providing a distributed IPM for large scale convex optimization problems.

A. Problem Characteristic Analysis

Apparently, the problem as defined by Eq. 6 is a convex optimization problem [21]. However, different from the canonical convex optimization problem, we have to solve a large scale problem. Matrix $\boldsymbol{A}$ is large (i.e., m and n are large), which prohibits us from applying the traditional interior point method (IPM) [21].

B. Distributed Interior Point Method and Newton-Step Level Matrix Decomposition

In this subsection, we directly apply IPM and decompose the core computation step into small computation units. For representation simplicity, we define

$$f(x,r,w,c)=\ln(1+\mathrm{e}^{w(x+r)+c}) \tag{7}$$

Apply the interior point method [21]: initialize $t_0=1$, and set $\mu_0=10$. At the kth optimization step, we solve the following transformed optimization problem (with log-barrier function) with $\mu=k\mu_0$:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \mu t\sum_{i=1}^{n} f(x_i,r_i,w_i,c_i)-\sum_{i=1}^{n}\ln(x_i) \\ \text{subject to} \quad & \boldsymbol{Ax}=\boldsymbol{b}. \end{aligned}$$

The Newton step can be calculated by solving the following equation:

$$\begin{bmatrix} \boldsymbol{H} & \boldsymbol{A}^\top \\ \boldsymbol{A} & 0 \end{bmatrix} \begin{pmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{w} \end{pmatrix} = -\begin{pmatrix} \boldsymbol{\nabla f(x)} \\ \boldsymbol{Ax-b} \end{pmatrix}. \tag{8}$$

where

$$H=\mathrm{diag}(\mu t\nabla^2 f(x_i)+\frac{1}{x_i^2}). \tag{9}$$

Here, we wish to note that all ∇ operations are performed with respect to x, i.e., $\nabla f(x_i)=\frac{\partial f}{\partial x}|_{x=x_i}$.

Solving Eq. 8 directly is very difficult when matrix $\boldsymbol{A}$ is of high dimension. Here, we solve it via elimination, and the key computation step is:

$$\boldsymbol{AH^{-1}A^\top}\Delta \boldsymbol{w}=\boldsymbol{Ax-b-AH^{-1}}\bigtriangledown \boldsymbol{f(x)}\,. \tag{10}$$

Now, the key step is to calculate the inverse of the term $\boldsymbol{AH^{-1}A^\top}$ in a distributed and efficient way. Because after the computation of the inverse of $\boldsymbol{AH^{-1}A^\top}$, we can calculate $\boldsymbol{w}$. Then, $\boldsymbol{\Delta x}$ can be calculated as follows:

$$\boldsymbol{H\Delta x}=-(\nabla \boldsymbol{f(x)}+\boldsymbol{A}^\top \Delta \boldsymbol{w}). \tag{11}$$

Since $\boldsymbol{H}$ is diagonal, its inverse can be obtained with a reasonable computation effort, i.e., we only need to compute the inverse of its diagonal element. Therefore, the solution of Eq. 11 is straightforward. In the next subsection, we will display the algorithm for the distributed solution of the inverse of the term $\boldsymbol{AH^{-1}A^\top}$.

C. Distributed Solution for the Inverse of $\boldsymbol{AH^{-1}A^\top}$

Problem: Calculating the inverse of $\boldsymbol{AH^{-1}A^\top}$

Input: $\boldsymbol{A}\in \boldsymbol{R}^{m\times n}$; $\boldsymbol{H}\in (\mathrm{diag})^{n\times n}\cap \boldsymbol{S^n_{++}}$. Here, the term $\boldsymbol{R}^{m\times n}$ refers to all $m\times n$ real matrices [24]; the term $(\mathrm{diag})^{n\times n}$ refers to all $n\times n$ diagonal real matrices; the term $\boldsymbol{S^n_{++}}$ refers to all $n\times n$ symmetric positive definite matrices.

Solution:

Step (1): Represent matrix $\boldsymbol{A}$ in a *column* distributed manner, which is:

$$\boldsymbol{A}=\left(\boldsymbol{A_1},\boldsymbol{A_2},\cdots,\boldsymbol{A_N}\right) \tag{12}$$

where N is the number of computers (computation entities) that we have, $\boldsymbol{A_i}\in \boldsymbol{R}^{m\times n_i}$, and

$$\sum_{i=1}^{N} n_i=n. \tag{13}$$

Step (2): Calculate $\boldsymbol{H^{-\frac{1}{2}}}$ and represent it ($\boldsymbol{H^{-\frac{1}{2}}}$) in a *row* distributed manner as expressed in Eq. 15.

Since $\boldsymbol{H}\in (diag)^{n\times n}\cap \boldsymbol{S^n_{++}}$, the computation of $\boldsymbol{H^{-1}}$ is fairly straightforward, which is just to obtain every diagonal element of matrix $\boldsymbol{H}$ and take the inverse of it. We define $\boldsymbol{H}^{-\frac{1}{2}}$ as a matrix, which makes $\boldsymbol{H}^{-\frac{1}{2}}(\boldsymbol{H}^{-\frac{1}{2}})^\top=\boldsymbol{H}^{-1}$.

The computation of $\boldsymbol{H}^{-\frac{1}{2}}$ is to take every diagonal element of the $\boldsymbol{H^{-1}}$ and compute the square root of it. Now, we have the representation of $\boldsymbol{H}^{-\frac{1}{2}}$, and define it as follows:

$$\boldsymbol{\Lambda}=\boldsymbol{H}^{-\frac{1}{2}}. \tag{14}$$

Now, we represent $\mathbf{\Lambda}$ as:

$$\mathbf{\Lambda} = \begin{pmatrix} \Lambda_1 \\ \Lambda_2 \\ \vdots \\ \Lambda_N \end{pmatrix} \tag{15}$$

where $\Lambda_i \in \boldsymbol{R}^{n_i \times n}$

Step (3): Transform the representation of $\boldsymbol{AH^{-1}A}^\top$.

Combining Eq. 12 and Eq. 15, we have:

$$\begin{aligned} &\boldsymbol{AH^{-1}A}^\top \\ &= \big(\boldsymbol{A_1}, \boldsymbol{A_2}, \cdots, \boldsymbol{A_N}\big) \begin{pmatrix} \Lambda_1 \\ \Lambda_2 \\ \vdots \\ \Lambda_N \end{pmatrix} \begin{pmatrix} \Lambda_1 \\ \Lambda_2 \\ \vdots \\ \Lambda_N \end{pmatrix}^\top \big(\boldsymbol{A_1}, \boldsymbol{A_2}, \cdots, \boldsymbol{A_N}\big)^\top \\ &= (\sum_{i=1}^{N} \boldsymbol{A_i}\boldsymbol{\Lambda_i})(\sum_{i=1}^{N} \boldsymbol{A_i}\boldsymbol{\Lambda_i})^\top \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} \boldsymbol{A_i}\boldsymbol{\Lambda_i}\boldsymbol{\Lambda_j}^\top\boldsymbol{A_j}^\top \end{aligned} \tag{16}$$

Since the matrix Λ is a diagonal matrix, it provides an orthogonal decomposition basis over the space of $\boldsymbol{R}^n$. It means that:

$$\boldsymbol{\Lambda_i}\boldsymbol{\Lambda_j}^\top = \begin{cases} \mathbf{0} & \text{if } i \neq j \\ \boldsymbol{\Lambda_i}\boldsymbol{\Lambda_i}^\top & \text{if } i = j \end{cases}$$

Therefore, we can continue transforming Eq. 16 as:

$$\begin{aligned} \boldsymbol{AH^{-1}A}^\top &= \sum_{i=1}^{N}\sum_{j=1}^{N} \boldsymbol{A_i}\boldsymbol{\Lambda_i}\boldsymbol{\Lambda_j}\boldsymbol{A_j} \\ &= \sum_{i=1}^{N} \boldsymbol{A_i}\boldsymbol{\Lambda_i}\boldsymbol{\Lambda_i}^\top\boldsymbol{A_i}^\top \end{aligned} \tag{17}$$

Now, we need to do one more transformation over the term $\boldsymbol{\Lambda_i}\boldsymbol{\Lambda_i}^\top$. Recall the definition of $\mathbf{\Lambda}$ in Eq. 15. We have $\boldsymbol{\Lambda_i} \in \boldsymbol{R}^{n_i \times n}$. Since $\mathbf{\Lambda}$ is a $n \times n$ diagonal matrix, we know that $\boldsymbol{\Lambda_i}\boldsymbol{\Lambda_i}^\top$ is a $n_i \times n_i$ diagonal matrix. Define:

$$\boldsymbol{\Sigma_i} = \boldsymbol{\Lambda_i}\boldsymbol{\Lambda_i}^\top \tag{18}$$

and define $\boldsymbol{\Sigma_i^{\frac{1}{2}}} \in \boldsymbol{R}^{n_i \times n_i}$ as a matrix, which calculates the square root of the diagonal elements of $\boldsymbol{\Sigma_i}$. We have:

$$\boldsymbol{\Sigma_i^{\frac{1}{2}}}(\boldsymbol{\Sigma_i^{\frac{1}{2}}})^\top = \boldsymbol{\Sigma_i}. \tag{19}$$

Now, Eq. 17 can be transformed to:

$$\begin{aligned} \boldsymbol{AH^{-1}A}^\top &= \sum_{i=1}^{N} \boldsymbol{A_i}\boldsymbol{\Lambda_i}\boldsymbol{\Lambda_i}^\top\boldsymbol{A_i}^\top \\ &= \sum_{i=1}^{N} \boldsymbol{A_i}\boldsymbol{\Sigma_i^{\frac{1}{2}}}(\boldsymbol{\Sigma_i^{\frac{1}{2}}})^\top\boldsymbol{A_i}^\top \end{aligned} \tag{20}$$

Here, we wish to highlight that the computation of $\boldsymbol{A_i}\boldsymbol{\Sigma_i^{\frac{1}{2}}}$ is simple, because $\boldsymbol{\Sigma_i}^{\frac{1}{2}}$ is a diagonal matrix and we only need to perform a column-wise product over matrix $\boldsymbol{A_i}$.

Performing singular value decomposition (SVD) over the result of $\boldsymbol{A_i}\boldsymbol{\Sigma_i^{\frac{1}{2}}}$, we can obtain:

$$\boldsymbol{A_i}\boldsymbol{\Sigma_i^{\frac{1}{2}}} = \boldsymbol{U_i}\boldsymbol{\Lambda_i'}\boldsymbol{V_i}^\top \tag{21}$$

where $\boldsymbol{U_i}$ and $\boldsymbol{V_i}$ are $m_i \times m_i$ unitary matrices, and $\boldsymbol{\Lambda_i'}$ is diagonal matrix.

Now, Eq. 20 can be further transformed to:

$$\begin{aligned} \boldsymbol{AH^{-1}A}^\top &= \sum_{i=1}^{N} \boldsymbol{A_i}\boldsymbol{\Sigma_i^{\frac{1}{2}}}(\boldsymbol{\Sigma_i^{\frac{1}{2}}})^\top\boldsymbol{A_i}^\top \\ &= \sum_{i=1}^{N} \boldsymbol{U_i}\boldsymbol{\Lambda_i'}\boldsymbol{V_i}^\top\boldsymbol{V_i}(\boldsymbol{\Lambda_i'})^\top\boldsymbol{U_i}^\top \\ &= \sum_{i=1}^{N} \boldsymbol{U_i}\boldsymbol{\Lambda_i'}(\boldsymbol{\Lambda_i'})^\top\boldsymbol{U_i}^\top. \end{aligned} \tag{22}$$

The term $\Lambda_i'(\Lambda_i')^\top$ is a diagonal matrix, and we define:

$$\boldsymbol{\Gamma_i} = \boldsymbol{\Lambda_i'}(\boldsymbol{\Lambda_i'})^\top \tag{23}$$

Eq. 22 can be expressed as:

$$\boldsymbol{AH^{-1}A}^\top = \sum_{i=1}^{N} \boldsymbol{U_i}\boldsymbol{\Gamma_i}\boldsymbol{U_i}^\top. \tag{24}$$

Now, the question is how to compute the inverse of the right hand side (RHS) of Eq. 24. The solution can be found in the Appendix.

D. Computation Complexity Analysis

In this section, we analyze the computation complexity of the proposed algorithm. Before analyzing the algorithm's computation complexity, we first lay down the evaluation metric.

1) Flops and baseline computation complexity: The computation cost of an operation can often be expressed through the number of floating-point operations (flops). A flop is defined as an addition, subtraction, multiplication or division of two floating-point numbers [21]. To evaluate the complexity of an algorithm, we count the total number of flops, express it as a function (usually a polynomial) of the dimensions of the matrices and vectors involved, and simplify the expression by omitting all terms except the leading terms.

Following the flop definition, we can express the computation complexity of an SVD (Singular Value Decomposition) operation over a matrix $\boldsymbol{A} \in \boldsymbol{R}^{m \times n}$ as $O(m^2n + n^3)$. The computation complexity of a matrix-matrix product $(C) = \boldsymbol{AB}$, where $\boldsymbol{A} \in \boldsymbol{R}^{m \times n}$ and $\boldsymbol{B} \in \boldsymbol{R}^{n \times p}$, is $O(mnp)$.

2) Worst case computation complexity of the algorithm over a single computation unit: Since we are developing a distributed algorithm, calculating the overall computation complexity is not meaningful. What we concern the most is how the computation effort is distributed across the computation units.

In this subsection, we calculate the computation complexity over one single computation unit. Since the data size is not the same, and the computation process for one computation unit may end before the algorithm exits, i.e., when one computation unit/agent is merged to another computation unit, this agent exits. We consider the computation complexity of the 'worst' computation unit/agent. Here, the term 'worst computation unit' means that the computation unit/agent performs the most complex computation over the whole process.

Examining the algorithm flow process in the Appendix, we can see that we are required to perform SVD over matrices $\boldsymbol{X_i}$, which needs $O(m_i^2 n + n^3)$ flops. Since we are considering the 'worst computation unit', the computation complexity can be expressed as $O(\max_i (m_i^2)n + n^3)$.

Then, the algorithm requires the computation of $\boldsymbol{U_q}$ and $\boldsymbol{P^{-1}}$. The computation of $\boldsymbol{U_q}$ requires three steps of $n \times n$ matrix-matrix multiplication, and one step of SVD over an $n \times n$ matrix, and the computation complexity of the matrix-matrix multiplication and SVD are both $O(n^3)$. Thus, the computation complexity of calculating $\boldsymbol{U_q}$ is $O(n^3)$. Likewise, the computation of $\boldsymbol{P^{-1}}$ requires two steps of $n \times n$ matrix-matrix multiplication, and thus the computation complexity is $O(n^3)$. Multiplying the matrices together involves two steps of $n \times n$ matrix-matrix multiplication, and hence the computation complexity is also $O(n^3)$. Performing SVD over the final $n \times n$ matrix requires $O(n^3)$. Summing together, and dropping the constant terms, we can conclude that the computation complexity is $O(n^3)$.

The above computation complexity is for one matrix merge operation. Now we need to count how many operations of matrix merge we actually need. We have N computation units. After one round of two-computer merge, we are left with $\lceil \frac{N}{2} \rceil$ computers, where $\lceil x \rceil$ is the ceiling operator returning the smallest integer which is larger than x. Therefore, after at most $\lceil \log_2 N \rceil$ operations of matrix merge process, we can finish the algorithm.

In summary, the computation complexity for the worst computer is $O(\max_i (m_i^2)n + n^3 + \lceil \log_2 N \rceil n^3)$. Dropping constant terms, we can reach $O(\max_i (m_i^2)n + \lceil \log_2 N \rceil n^3)$.

IV. Simulation Results

In this section, we will evaluate our proposed algorithm through answering the following three common questions:

- Are we calculating the inverse of $\boldsymbol{AH^{-1}A^\top}$ correctly?
- Are we much faster than canonical distributed optimization methods like dual decomposition and ADMM [20]?
- Are we guaranteeing an always-feasible solution during the optimization process?

A. Simulation Setup

1) Simulation Environment: We consider a realistic box area of the Singapore road network as the simulation environment. The selected area is shown in Fig. 1. The number of nodes is 1703, and there are 3136 connections altogether. As a result, the matrix $\boldsymbol{A}$ in the equality constraint is of size 1703×3136. We select a relatively small network size as the testing environment, because this size of the problem can be solved for a centralized solution which allows us to compare it with the proposed distributed solution.

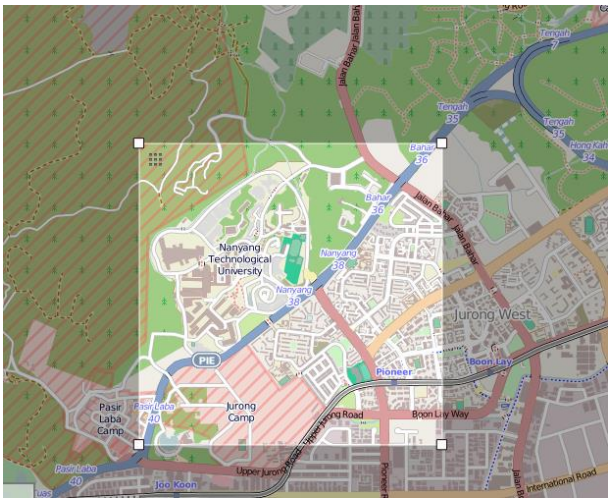

Fig. 1. The targeted road network sample

2) Parameter Setup: We set $w = 0.01$ and $c = -3$ in Eq. 4, and the total number of vehicles is $100/w = 10000$. The uncontrollable road network loads (r_i) are set to be random variables between 0 and 90% of the total number of vehicles. In this setting, we assume that we are able to control 10% of the total vehicles on the road.

For the algorithm parameters: we set IPM-related parameters as: $t = 1$, $u = 10$ and $\epsilon = 10^{-4}$. The parameters for performing Newton's method are set as: $\alpha = 0.3$ and $\beta = 0.8$, the stopping criteria for Newton decrement λ is set as: $\epsilon_{nt} = 10^{-4}$.

3) Results and Analysis: The test road network size is small enough so that we are able to obtain the centralized solution, which is through calculating the inverse of $\boldsymbol{AH^{-1}A^\top}$ directly.

Fig. 2 shows the 'error' as computed by our distributed methods. Here, we calculate the inverse of $\boldsymbol{AH^{-1}A^\top}$ through central direct methods and our proposed distributed methods. Then the Frobenius norm [25] of the difference between the two result matrix is defined as the error.

Although we can observe a slight increase of the error as we increase the number of computers, we are confident that we always achieve the correct inverse the $\boldsymbol{AH^{-1}A^\top}$ because the error is in the scale of 10^{-21}.

Fig. 3 shows the convergence process of our distributed algorithm. Here, we calculate the absolute optimal point using the centralized method directly. Then we implement our distributed methods and compare the result with the optimal solution. As we can see from Fig. 3, after about 21 Newton iterations, we are able to reach the solution with arbitrarily small deviation (the deviation magnitude in the order of 10^{-5}).

Here, we wish to highlight that, empirically, the required number of Newton iterations (21 in our simulation) does not grow with the size of the problem. It means that for

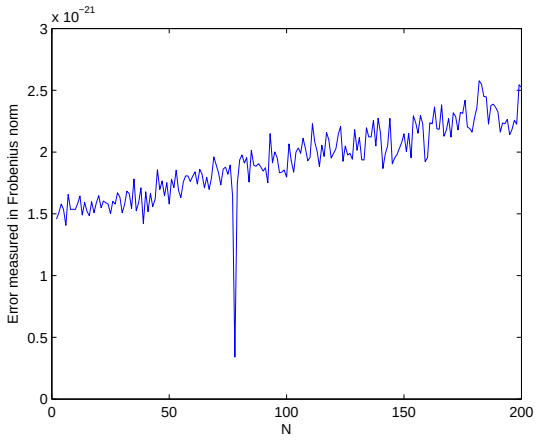

Fig. 2. The computation error versus the number of computers

an arbitrarily large network, the number of required Newton iterations is still around 20. However, the computation complexity per Newton step grows as the the network size increases. Therefore, our aim is to make this part distributed and scalable.

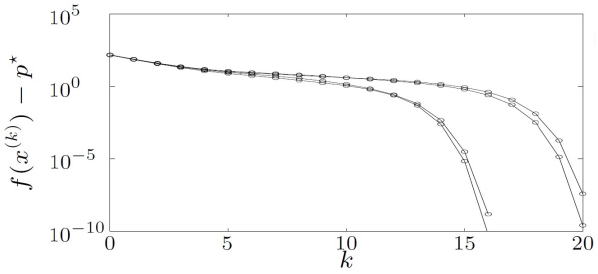

Fig. 3. The difference between the kth-step solution and the optimal solution VS computation steps. (Different curves correspond to different initial points.)

We also implemented dual decomposition and ADMM [20]. The required number of iterations for convergence is around 7 million steps. So far, we still cannot simply conclude that our proposed algorithm is much faster than traditional distributed algorithms, because each iteration in our computation is much more complex than the iterations in dual decomposition or ADMM.

The computation complexity of each iteration in dual decomposition or ADMM is $O(mn)$, where m and n are the nodes and edges in the network, respectively, while the computation complexity of each iteration of our algorithm is $O(mn_i^2)$, where n_i is the number of edges allocated to the computer. In the worst case, $n_i = n$, the computation complexity in each iteration of our algorithm is $n = 3703$ times larger than that of dual decomposition. Multiplying the total steps together, and defining the computation complexity of each iteration in dual decomposition as a unit, we can obtain that the total computation complexity of our algorithm is $21 \times 3703 = 77763$. While the total computation complexity of dual decomposition and ADMM is 7 million. Our algorithm is significantly better in this case study.

Fig. 4 shows the feasibilty metric during the optimization process. We need to define a metric for feasibility. Since we use IPM [21] with log barrier function, we can guarantee that the inequality constraint is always satisfied. Thus, we only consider the equality constraint. We define feasibility as the Frobenius norm [25] of the residual between $\boldsymbol{Ax}$ and $\boldsymbol{b}$.

Both dual decomposition and ADMM unify the equality constraint into the objective (i.e., forming the Lagrangian function) and perform decomposition thereafter. In this case, they can only achieve a feasible point after the whole optimization process is done. In real world applications, we might need a suboptimal solution within the computation time limit.

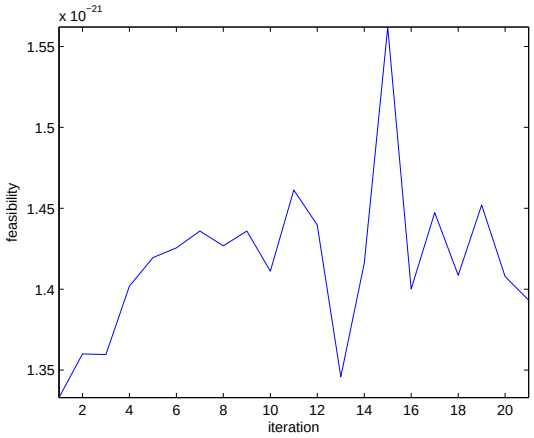

Fig. 4. Feasibility versus the number of Newton iterations

In Fig. 4, we can see that, after every Newton step, the feasibility measurement is always in the scale of 10^{-21}, which means that it is always feasible. This feature is not present in dual decomposition and ADMM.

4) Summary: In this section, we use a simple network to show that (1) our algorithm is able to calculate the inverse of $\boldsymbol{AH^{-1}A^\top}$ correctly; (2) the total computation complexity is much smaller than traditional algorithms; (3) we always guarantee a feasible solution during the optimization process.

V. Conclusion and Future Works

This paper has presented a distributed network optimization algorithm for multi-vehicle routing. We have formulated the multi-vehicle routing problem as a network optimization problem. Different from traditional distributed optimization techniques like dual decomposition and ADMM, we distributize the Newton-step calculation process through a novel matrix decomposition algorithm. In this way, firstly, we make use of the second derivative information, hence speed up our optimization process; secondly, we can always guarantee a feasible solution during the iterations of optimization. Simulation results have shown both advantages of our algorithm.

However, we have not tested our algorithm in traffic simulators like SUMO [26]. In the future work, we will

implement our algorithm in traffic simulators, and further test the performance. We will also apply the proposed algorithm into real world after the simulator step. The road network breakdown probability is assumed to be given in this paper; in the next step, we will adopt different methods to obtain the model of road network breakdown probability.

APPENDIX

A. Matrix Merging Process

We are interested in solving for the inverse of $\sum \boldsymbol{V_i}\boldsymbol{\Lambda_i}\boldsymbol{V_i}^\top$ efficiently and distributedly. We represent the term as:

$$\boldsymbol{M} = \sum_{i=1}^{N} \boldsymbol{V_i}\boldsymbol{\Lambda_i}\boldsymbol{V_i}^\top \tag{25}$$

where $\boldsymbol{V_i}$ is unitary matrix and Λ_i is diagonal matrix. Ideally, if we can represent $\boldsymbol{M}$ as follows:

$$\boldsymbol{M} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top \tag{26}$$

where, $\boldsymbol{V}$ is a unitary matrix, and $\boldsymbol{\Lambda}$ is a diagonal matrix with positive diagonal values, then it is easy for us to compute the inverse of $\boldsymbol{M}$.

Before transforming Eq. 25 into the form of Eq. 26 directly, we start with an easier step by merging two matrices. The problem is to represent $\boldsymbol{V_i}\boldsymbol{\Lambda_i}\boldsymbol{V_i}^\top + \boldsymbol{V_j}\boldsymbol{\Lambda_j}\boldsymbol{V_j}^\top$ into the form of Eq. 26. Define:

$$\boldsymbol{M_{ij}} = \boldsymbol{V_i}\boldsymbol{\Lambda_i}\boldsymbol{V_i}^\top + \boldsymbol{V_j}\boldsymbol{\Lambda_j}\boldsymbol{V_j}^\top. \tag{27}$$

Since both of the two terms in Eq. 27 are symmetric positive definite matrices, according to Theorem 5.1 in the next subsection, there exists an invertible matrix $\boldsymbol{P}$, such that $\boldsymbol{P}^\top\boldsymbol{V_i}\boldsymbol{\Lambda_i}\boldsymbol{V_i}^\top\boldsymbol{P} = \boldsymbol{I}$, and $\boldsymbol{P}^\top\boldsymbol{V_j}\boldsymbol{\Lambda_j}\boldsymbol{V_j}^\top\boldsymbol{P} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$.

Therefore, matrix $\boldsymbol{P}$ satisfies the following equation:

$$\boldsymbol{P}^\top\boldsymbol{M_{ij}}\boldsymbol{P} = \boldsymbol{I} + diag(\lambda_1, \lambda_2, \cdots, \lambda_n). \tag{28}$$

After simple deduction, we can get that:

$$\begin{aligned}\boldsymbol{M_{ij}} &= (\boldsymbol{P^{-1}})^\top(\boldsymbol{I} + diag(\lambda_1, \lambda_2, \cdots, \lambda_n))\boldsymbol{P^{-1}} \\ &= (\boldsymbol{P^{-1}})^\top diag(\lambda_1 + 1, \lambda_2 + 1, \cdots, \lambda_n + 1)\boldsymbol{P^{-1}}.\end{aligned} \tag{29}$$

Continue to apply singular value decomposition to the second half of Eq. 29, we can represent $\boldsymbol{M_{ij}}$ as:

$$\boldsymbol{M_{ij}} = \boldsymbol{V_{ij}}\boldsymbol{\Lambda_{ij}}\boldsymbol{V_{ij}}^\top \tag{30}$$

Now, we have finished the merging process of the matrices. The merged matrix ($\boldsymbol{M_{ij}}$) is also represented in the same form as before the merging. We can continue with the merging process when $\boldsymbol{M_{ij}}$ 'meets' other matrix. Thus, the key computation step is to compute matrix $\boldsymbol{P}^{-1}$.

1) The computation process of $\boldsymbol{P^{-1}}$: Consider Eq. 27, since $\boldsymbol{\Lambda_i}$ is a diagonal matrix with all positive diagonal elements. Suppose $\boldsymbol{\Lambda_i} = diag(\lambda_1, \lambda_2, \cdots, \lambda_n)$; we define:

$$\boldsymbol{\Lambda_i^{-\frac{1}{2}}} = diag(\lambda_1^{-\frac{1}{2}}, \lambda_2^{-\frac{1}{2}}, \cdots, \lambda_n^{-\frac{1}{2}}) \tag{31}$$

and then we define

$$\boldsymbol{P_1} = \boldsymbol{\Lambda_i^{-\frac{1}{2}}}\boldsymbol{V_i}^\top. \tag{32}$$

Now, we have:

$$\boldsymbol{P_1}^{-1} = \boldsymbol{V_i}\boldsymbol{\Lambda_i^{\frac{1}{2}}}. \tag{33}$$

We multiply Eq. 27 by $\boldsymbol{P_1}$ from the left and by $\boldsymbol{P_1}^\top$ from the right, then we can get:

$$\begin{aligned}\boldsymbol{P_1}\boldsymbol{M_{ij}}\boldsymbol{P_1}^\top &= \boldsymbol{P_1}\boldsymbol{V_i}\boldsymbol{\Lambda_i}\boldsymbol{V_i}^\top\boldsymbol{P_1}^\top + \boldsymbol{P_1}\boldsymbol{V_j}\boldsymbol{\Lambda_j}\boldsymbol{V_j}^\top\boldsymbol{P_1}^\top \\ &= \boldsymbol{I} + \boldsymbol{\Lambda_i^{-\frac{1}{2}}}\boldsymbol{V_i}^\top\boldsymbol{V_j}\boldsymbol{\Lambda_j}\boldsymbol{V_j}^\top\boldsymbol{V_i}(\boldsymbol{\Lambda_i^{-\frac{1}{2}}})^\top.\end{aligned} \tag{34}$$

Here, we define

$$\boldsymbol{\Lambda_j^{\frac{1}{2}}} = diag(\lambda_1^{\frac{1}{2}}, \lambda_2^{\frac{1}{2}}, \cdots, \lambda_n^{\frac{1}{2}}). \tag{35}$$

Then, Eq. 34 can be further transformed to:

$$\boldsymbol{P_1}\boldsymbol{M_{ij}}\boldsymbol{P_1}^\top = \boldsymbol{I} + \boldsymbol{\Lambda_i^{-\frac{1}{2}}}\boldsymbol{V_i}^\top\boldsymbol{V_j}\boldsymbol{\Lambda_j^{\frac{1}{2}}}(\boldsymbol{\Lambda_j^{\frac{1}{2}}})^\top\boldsymbol{V_j}^\top\boldsymbol{V_i}(\boldsymbol{\Lambda_i^{-\frac{1}{2}}})^\top. \tag{36}$$

Define:

$$\boldsymbol{Q} = \boldsymbol{\Lambda_i^{-\frac{1}{2}}}\boldsymbol{V_i}^\top\boldsymbol{V_j}\boldsymbol{\Lambda_j^{\frac{1}{2}}} \tag{37}$$

Then Eq. 36 is simply in the form of:

$$\boldsymbol{P_1}\boldsymbol{M_{ij}}\boldsymbol{P_1}^\top = \boldsymbol{I} + \boldsymbol{Q}\boldsymbol{Q}^\top \tag{38}$$

Applying singular value decomposition on $\boldsymbol{Q}$, we can get:

$$\boldsymbol{Q} = \boldsymbol{U_q}\boldsymbol{\Lambda_q}\boldsymbol{V_q}^\top \tag{39}$$

where $\boldsymbol{U_q}$ and $\boldsymbol{V_q}$ are unitary matrices, and $\boldsymbol{\Lambda_q}$ is diagonal matrix. Replacing $\boldsymbol{Q}$ in Eq. 38 through Eq. 39, we can get:

$$\begin{aligned}\boldsymbol{P_1}\boldsymbol{M_{ij}}\boldsymbol{P_1}^\top &= \boldsymbol{I} + \boldsymbol{U_q}\boldsymbol{\Lambda_q}\boldsymbol{V_q}^\top(\boldsymbol{U_q}\boldsymbol{\Lambda_q}\boldsymbol{V_q}^\top)^\top \\ &= \boldsymbol{I} + \boldsymbol{U_q}\boldsymbol{\Lambda_q}\boldsymbol{V_q}^\top\boldsymbol{V_q}(\boldsymbol{\Lambda_q})^\top\boldsymbol{U_q}^\top \\ &= \boldsymbol{I} + \boldsymbol{U_q}\boldsymbol{\Lambda_q}(\boldsymbol{\Lambda_q})^\top\boldsymbol{U_q}^\top \\ &= \boldsymbol{U_q}\boldsymbol{U_q}^\top + \boldsymbol{U_q}\boldsymbol{\Lambda_q}(\boldsymbol{\Lambda_q})^\top\boldsymbol{U_q}^\top \\ &= \boldsymbol{U_q}(\boldsymbol{I} + \boldsymbol{\Lambda_q}\boldsymbol{\Lambda_q}^\top)\boldsymbol{U_q}^\top\end{aligned} \tag{40}$$

Solving Eq. 40 for $\boldsymbol{M_{ij}}$, and replacing $\boldsymbol{P_1^{-1}}$ by Eq. 33, we can obtain:

$$\begin{aligned}\boldsymbol{M_{ij}} &= \boldsymbol{P_1^{-1}}\boldsymbol{U_q}(\boldsymbol{I} + \boldsymbol{\Lambda_q}\boldsymbol{\Lambda_q}^\top)\boldsymbol{U_q}^\top(\boldsymbol{P_1^{-1}})^\top \\ &= \boldsymbol{V_i}\boldsymbol{\Lambda_i^{\frac{1}{2}}}\boldsymbol{U_q}(\boldsymbol{I} + \boldsymbol{\Lambda_q}\boldsymbol{\Lambda_q}^\top)\boldsymbol{U_q}^\top(\boldsymbol{V_i}\boldsymbol{\Lambda_i^{\frac{1}{2}}})^\top\end{aligned} \tag{41}$$

The inverse of the matrix $\boldsymbol{P}$ as described in Eq. 28 can be directly calculated out as:

$$\boldsymbol{P^{-1}} = (\boldsymbol{V_i}\boldsymbol{\Lambda_i^{\frac{1}{2}}}\boldsymbol{U_q})^\top \tag{42}$$

2) Transforming the Merged Matrix into Standard Form: Now, we know $\boldsymbol{P^{-1}}$, then the merged matrix can be represented as:

$$\boldsymbol{M_{ij}} = \boldsymbol{(P^{-1})}^{\top}(\boldsymbol{I} + \boldsymbol{\Lambda_q}\boldsymbol{\Lambda_q}^{\top})\boldsymbol{P^{-1}}. \tag{43}$$

$\boldsymbol{M_{ij}}$ is a nxn matrix, and applying singular value decomposition over $\boldsymbol{M_{ij}}$, we can represent $\boldsymbol{M_{ij}}$:

$$\boldsymbol{M_{ij}} = \boldsymbol{V_{ij}}\boldsymbol{\Lambda_{ij}}\boldsymbol{V_{ij}}^{\top} \tag{44}$$

where $\boldsymbol{V_{ij}}$ is a unitary matrix and $\boldsymbol{\Lambda_{ij}}$ is a diagonal matrix with all positive diagonal elements.

Here, we are able to merge two matrices, each of which is in the form as represented in Eq. 26, and continue to represent the merged matrix in the form of Eq. 26. As the process continues, we will reach the final representation also in the form of Eq. 26.

B. Theorem: Simultaneously Congruentization and Diagonalization of Two Matrices

Theorem 5.1: $\forall\ \boldsymbol{A} \in \boldsymbol{S^n_{++}}$, $\boldsymbol{B} \in \boldsymbol{S^n}$, $\exists \boldsymbol{P}$, s.t.: (1) $\exists \boldsymbol{P^{-1}}$; (2) $\boldsymbol{P}^{\top}\boldsymbol{A}\boldsymbol{P} = \boldsymbol{I}$; (3) $\boldsymbol{P}^{\top}\boldsymbol{B}\boldsymbol{P}$ =diag$(\lambda_1, \lambda_2, \cdots, \lambda_n)$.

Suppose $\boldsymbol{A}$ and $\boldsymbol{B}$ are both $n \times n$ matrices, $\boldsymbol{A}$ is positive definite, and $\boldsymbol{B}$ is symmetric, then, there exists an invertible matrix $\boldsymbol{P}$, which makes $\boldsymbol{A}$ congruent to identity matrix and in the meanwhile, makes $\boldsymbol{B}$ diagonal. Here, the term $\boldsymbol{I}$ denotes identity matrices.

Proof: Since $\boldsymbol{A}$ is positive definite, $\boldsymbol{A}$ is congruent to the identity matrix, i.e., $\boldsymbol{A} \in \boldsymbol{S^n_{++}} \Rightarrow \exists \boldsymbol{P_1}$, s.t. $\boldsymbol{P_1}^{\top}\boldsymbol{A}\boldsymbol{P_1} = \boldsymbol{I}$.

Since $\boldsymbol{P_1}^{\top}\boldsymbol{B}\boldsymbol{P_1}$ is symmetric, then we can orthogonally diagonalize it to a diagonal matrix. $\boldsymbol{B} \in \boldsymbol{S} \Rightarrow \boldsymbol{P_1}^{\top}\boldsymbol{B}\boldsymbol{P_1} \in \boldsymbol{S} \Rightarrow \exists \boldsymbol{P_2}$ s.t. (1) $\boldsymbol{P_2}^{\top}\boldsymbol{P_2} = \boldsymbol{P_2}\boldsymbol{P_2}^{\top} = \boldsymbol{I}$; (2) $\boldsymbol{P_2}^{\top}\boldsymbol{P_1}^{\top}\boldsymbol{B}\boldsymbol{P_1}\boldsymbol{P_2} =$ diag$(\lambda_1, \lambda_2, \cdots, \lambda_n)$.

Define $\boldsymbol{P} = \boldsymbol{P_1}\boldsymbol{P_2}$, we have: $\boldsymbol{P}^{\top}\boldsymbol{A}\boldsymbol{P} = \boldsymbol{I}$, and $\boldsymbol{P}^{\top}\boldsymbol{B}\boldsymbol{P}$=diag$(\lambda_1, \lambda_2, \ldots, \lambda_n)$.

■

REFERENCES

[1] American Society of Civil Engineers, *Report Card for America's Infrastructure*, 2009.

[2] Texas A&M Transportation Institute, *Annual Urban Mobility Report*, 2012. Funding provided in part by the Southwest Region University Transportation Center (SWUTC).

[3] B. S. Kerner, "Physics of traffic gridlock in a city," *Phys Rev E Stat Nonlin Soft Matter Phys*, vol. 84, no. 4-2, p. 045102, 2011.

[4] J. Auld and A. Mohammadian, "Activity planning processes in the Agent-based Dynamic Activity Planning and Travel Scheduling (ADAPTS) model," *Transportation Research Part A: Policy and Practice*, vol. 46, no. 8, pp. 1386–1403, 2012.

[5] M. Balmer, N. Cetin, K. Nagel, and B. Raney, "Towards truly agent-based traffic and mobility simulations," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '04, (Washington, DC, USA), pp. 60–67, IEEE Computer Society, 2004.

[6] A. Bazzan, "A distributed approach for coordination of traffic signal agents," *Autonomous Agents and Multi-Agent Systems*, vol. 10, no. 1, pp. 131–164, 2005.

[7] A. Bazzan, D. de Oliveira, F. Klgl, and K. Nagel, "To adapt or not to adapt consequences of adapting driver and traffic light agents," in *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning* (K. Tuyls, A. Nowe, Z. Guessoum, and D. Kudenko, eds.), vol. 4865 of *Lecture Notes in Computer Science*, pp. 1–14, Springer Berlin Heidelberg, 2008.

[8] A. L. C. Bazzan and F. Klgl, "A review on agent-based technology for traffic and transportation," *The Knowledge Engineering Review*, vol. 29, pp. 375–403, 6 2014.

[9] A. Bazzan, J. Wahle, and F. Klgl, "Agents in traffic modelling from reactive to social behaviour," in *KI-99: Advances in Artificial Intelligence* (W. Burgard, A. Cremers, and T. Cristaller, eds.), vol. 1701 of *Lecture Notes in Computer Science*, pp. 303–306, Springer Berlin Heidelberg, 1999.

[10] E. Camponogara and J. Kraus, Werner, "Distributed learning agents in urban traffic control," in *Progress in Artificial Intelligence* (F. Pires and S. Abreu, eds.), vol. 2902 of *Lecture Notes in Computer Science*, pp. 324–335, Springer Berlin Heidelberg, 2003.

[11] C. Desjardins, J. Laumnier, and B. Chaib-draa, "Learning agents for collaborative driving," in *Multi-Agent Systems for Traffic and Transportation* (F. Bazzan, A. L. C. & Klugl, ed.), pp. 240–260, 2004.

[12] H. Dia, "An agent-based approach to modelling driver route choice behaviour under the influence of real-time information," *Transportation Research Part C: Emerging Technologies*, vol. 10, no. 56, pp. 331 – 349, 2002.

[13] L. B. de Oliveira and E. Camponogara, "Multi-agent model predictive control of signaling split in urban traffic networks," *Transportation Research Part C: Emerging Technologies*, vol. 18, no. 1, pp. 120 – 139, 2010. Information/Communication Technologies and Travel Behaviour Agents in Traffic and Transportation.

[14] N. Gartner and N. R. C. U. T. R. Board, *OPAC: A Demand-responsive Strategy for Traffic Signal Control*. Transportation Research Board, National Research Council, 1983.

[15] S. Jiang, J. Zhang, and Y.-S. Ong, "A pheromone-based traffic management model for vehicle re-routing and traffic light control," in *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, (Richland, SC), pp. 1479–1480, International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[16] B. S. Kerner, "Optimum principle for a vehicular traffic network: minimum probability of congestion," *Journal of Physics A: Mathematical and Theoretical*, vol. 44, no. 9, p. 092001, 2011.

[17] H. Rakha and A. Tawfik, "Traffic networks: Dynamic traffic routing, assignment, and assessment," in *Encyclopedia of Complexity and Systems Science* (R. A. Meyers, ed.), pp. 9429–9470, Springer New York, 2009.

[18] N. Gartner and C. Stamatiadis, "Traffic networks, optimization and control of urban," in *Encyclopedia of Complexity and Systems Science* (R. A. Meyers, ed.), pp. 9470–9500, Springer New York, 2009.

[19] J. Wardrop, *Wardrop of Some Theoretical Aspects of Road Traffic Research-Road Paper*. No. no. 36, Road Engineering Division, 1952.

[20] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, pp. 1–122, Jan. 2011.

[21] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[22] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[23] O. Bousquet, S. Boucheron, and G. Lugosi, "Introduction to statistical learning theory," in *In , O. Bousquet, U.v. Luxburg, and G. Rsch (Editors*, pp. 169–207, Springer, 2004.

[24] W. Beyer, *C.R.C. Standard Mathematical Tables*. CRC Press, 1984.

[25] R. A. Horn and C. R. Johnson, eds., *Matrix Analysis*. New York, NY, USA: Cambridge University Press, 1986.

[26] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, pp. 128–138, December 2012.