APPENDIX

Here, we provide additional details for certain aspects of the paper. These information should be perceived as further clarifications and are not necessary in order to follow the paper.

A THE USER EQUILIBRIUM

In this section, we provide a more formal definition of the user equilibrium and some background on how to calculate it. To this end, we first introduce the following function that redistributes flow between paths.

Definition A.1. Let $G = (V, E)$ be a graph, f a route flow, and $(s, t) \in V^2$. For $P_1, P_2, P \in \mathcal{P}_{s\text{-}t}$ and $\delta \in [0, f(P_1)]$, we define the flow redistribution function as

$$\widetilde{f}_{\delta}^{(P_1,P_2)}(P) := \begin{cases} f(P_1) - \delta, & \text{if } P = P_1; \\ f(P_2) + \delta, & \text{if } P = P_2; \\ f(P), & \text{otherwise.} \end{cases}$$

A user equilibrium is then defined as follows.

Definition A.2 (User Equilibrium, [32]). Let $G = (V, E)$ be a graph with latency functions for the edges, f a route flow, and $(s, t) \in V^2$. Then, f is in a user equilibrium if and only if for all $P_1, P_2 \in \mathcal{P}_{s\text{-}t}$, $\delta \in (0, f(P_1)]$, $\tau_{P_1}(f) \leq \tau_{P_2}(\widetilde{f}_{\delta}^{(P_1,P_2)})$.

A.1 Calculation of the User Equilibrium

There are different methods of calculating a user equilibrium [12]. For the MREA, we optimize a convex program. To this end, the Frank–Wolfe algorithm, a general-purpose iterative optimization algorithm for convex optimization problems, is employed [21]. We first outline the framework of the Frank–Wolfe algorithm and afterwards deploy it to the context of user equilibria.

In order to apply the Frank–Wolfe algorithm, there are two requirements: the function to be optimized as well as the set of possible solutions needs to be convex. Following Patriksson [30], the Frank–Wolfe Algorithm works as described in Algorithm 2.

Algorithm 2: The Frank–Wolfe algorithm [21] for optimizing a convex program

Input: Convex set $\mathcal{D}$, $f\colon \mathcal{D} \to \mathbf{R}$ convex, differentiable function, $x_0 \in \mathcal{D}$
Output: $x \in \mathcal{D}$ s.t. $f(x)$ is minimal

$q \leftarrow 0$;
while *not converged* **do**
  Find p_q minimizing the following linear program
    minimize $\quad p_q^T \nabla f(x_q)$
    subject to $\quad p_q \in \mathcal{D}$
  $\gamma \leftarrow$ Line-Search Determination of step size;
  $x_{q+1} \leftarrow x_q + \gamma(p_q - x_q)$;
  $q \leftarrow q + 1$;
return x_q;

The algorithm performs a kind of gradient descent. In each step, it solves a linear program that approximates the convex program and then moves towards the minimizer of this program. The optimal step size is chosen according to the objective function via a line-search.

We apply the Frank–Wolfe algorithm to user equilibria. As shown by Beckmann et al. [6], the user equilibrium can be expressed as a convex program. For a flow u, let

$$z(u) = \sum_{e \in E} \int_0^{u(e)} \tau_e(x)\,\mathrm{d}x.$$

Furthermore, let $(s, t) \in V^2$; then the flow u corresponding to the user equilibrium is the solution of the following equation (2).

$$\begin{aligned} \text{minimize} \quad & z(u) \\ \text{subject to} \quad & \sum_{P \in \mathcal{P}} u(P) = k \\ & \forall P \in \mathcal{P}_{s,t} \quad u(P) \geq 0 \end{aligned} \tag{2}$$

Note that every flow satisfies the constraints of the linear program. To prove that the set of solutions is convex as well, we introduce a new concept called *flow vectors*, allowing interpolation between flows. For every flow f, we derive a flow vector $\overline{f}$. Every s-t path maps to one index in the flow vector. The vector element at the according index is equivalent to the amount of traffic flow $f(P)$ assigned to the corresponding route $P \in \mathcal{P}$. Similar to flow functions, we use $\overline{f}(P)$ for the amount of traffic flow assigned to route P by the flow vector $\overline{f}$. Similar to route flows, flow vectors induce edge flow vectors.

Lemma A.3. *For a graph $G = (V, E)$, let $s, t \in V$ and k be the traffic flow traveling from s to t. Let $\mathcal{D}$ be the set of flow vectors between s and t. Then, $\mathcal{D}$ is a convex set.*

Proof. Let u and u' be two flow vectors between s and t. Furthermore, let $\gamma \in [0, 1]$. We now prove that the interpolated vector $u'' = \gamma \cdot u' + (1 - \gamma) \cdot u$ is a flow vector between s and t as well. Therefore, we already showed that the demand k is exactly fulfilled, i.e., $\sum_{P \in \mathcal{P}} u''(P) = k$, and that for all $P \in \mathcal{P}$, $u''(P) \geq 0$. Since u and u' are flow vectors, for all $P \in \mathcal{P}$, $u(P), u'(P) \geq 0$. With $0 \leq \gamma \leq 1$ and the definition of u'' we get, for all $P \in \mathcal{P}$, $u''(P) \geq 0$.

We now show that $\sum_{P \in \mathcal{P}} u''(P) = k$. Note that $\sum_{P \in \mathcal{P}} u(P) = \sum_{P \in \mathcal{P}} u'(P) = k$ since u and u' are flow vectors that fulfill the demand k exactly.

$$\begin{aligned} \sum_{P \in \mathcal{P}} u''(P) &= \sum_{P \in \mathcal{P}} ((1 - \gamma) \cdot u(P) + \gamma \cdot u'(P)) \\ &= (1 - \gamma) \sum_{P \in \mathcal{P}} u(P) + \gamma \sum_{P \in \mathcal{P}} u'(P) \\ &= (1 - \gamma) \cdot k + \gamma \cdot k \\ &= k. \end{aligned}$$

□

As described in Algorithm 2 the Frank–Wolfe algorithm solves a linear program in each iteration. For calculating user equilibria, we substantiate the abstract term $p_q^T \nabla z(x_q)$ by calculating the gradient of z and simplifying to $p_q^T \nabla z(x_q) = \sum_{e \in E} \tau_e(x_q(e)) \cdot p_q(e)$. Furthermore, the constraint $p_q \in \mathcal{D}$ is equivalent to p_q being a flow vector. Hence, the linear program in equation (3) needs to be solved in every iteration in order to obtain p_q based on the current flow vector x_q.

$$\begin{aligned} \text{minimize} \quad & p_q^T \nabla z(x_q) = \sum_{e \in E} \tau_e(x_q(e)) \cdot p_q(e) \\ \text{subject to} \quad & \sum_{P \in \mathcal{P}} p_q(P) = k \\ & \forall P \in \mathcal{P} \qquad p_q(P) \geq 0 \end{aligned} \tag{3}$$

We now show that solving this linear program is equivalent to assigning all drivers to the shortest route in a graph where each edge e has a fixed cost of $\tau_e(x_q(e))$.

Lemma A.4. *When calculating a user equilibrium on a graph $G = (V, E)$ for $(s, t) \in V^2$ using the Frank–Wolfe algorithm, in iteration q, the solution p_q to the linear program is the flow vector x that assigns all k drivers to the shortest s-t path of an adjusted graph G' where every edge e has a cost of $\tau_e(x_q(e))$.*

Proof. Let P be the shortest path from s to t. Assume the contrary, i.e., that the optimal assignment x' assigns flow to another path P' such that $x'(P') > 0$. As all edges have constant costs and P is the shortest path, $\tau_P(x_q) < \tau_{P'}(x_q)$. Hence, according to the definition of the system cost C, assigning all drivers using P' to P yields another feasible assignment with lower overall costs which is a contradiction to x' being the optimal assignment. □

In the case of the Multiple-Routes problem, we are only allowed to assign drivers to a fixed set of routes as discussed in Section 2.2. To approximate a user equilibrium, the drivers are assigned to the shortest route in the set instead of the graph. This may break the convergence of the Frank–Wolfe algorithm but allows to approximate the user equilibrium on the routes quite well.

A.1.1 Step Size Determination. There are various approaches for choosing the factor $\gamma \in [0, 1]$ used for interpolating between x_q and p_q. We employ a line-search, i.e., we find $\gamma \in [0, 1]$ minimizing

$$\tilde{z}(\gamma) = z(x_q + \gamma(p_q - x_q))$$

as this is the best step towards the global minimum of z that can be made from one iteration to the next. To this end, consider the derivatives w.r.t. γ,

$$\tilde{z}'(\gamma) = \sum_{e \in E} \tau_e(x_q(e) + \gamma(p_q(e) - x_q(e))) \cdot (p_q(e) - x_q(e)), \text{ and}$$
$$\tilde{z}''(\gamma) = \sum_{e \in E} \tau_e'(x_q(e) + \gamma(p_q(e) - x_q(e))) \cdot (p_q(e) - x_q(e))^2.$$

Since the cost functions τ_e are monotonically increasing, for all $\gamma \in [0, 1]$, $z''(\gamma) \geq 0$. For a concrete edge latency function τ_e, we now set the first derivative to zero in order to calculate the minimum. For the US Traffic Model used in Section 4, we obtain

$$\begin{aligned} \tilde{z}'(\gamma) &= \sum_{e \in E} (a_e \cdot (x_q(e) + \gamma(p_q(e) - x_q(e))^2 + b_e) \cdot (p_q(e) - x_q(e)) \\ &= \sum_{e \in E} a_e (p_q(e) - x_q(e))^3 \cdot \gamma^2 + 2a_e x_q(e)(p_q(e) - x_q(e))^2 \cdot \gamma \\ &\qquad + (a_e x_q(e)^2 + b_e)(p_q(e) - x_q(e)) \end{aligned}$$

which is a second-order polynomial whose roots can be calculated efficiently.

B NP-HARDNESS OF MULTIPLE-ROUTES

In this section, we show the **NP**-hardness of the Multiple-Routes problem. To this end, we define a decision problem variant of Multiple-Routes which adds a comparison factor C to the instances.

Definition B.1. An instance (G, s, t, k, n, C), where G subsumes a graph and the associated latency functions, is in Multiple-Routes, if and only if there is an n-user equilibrium flow that distributes k drivers over a set of n routes on G from s to t such that the overall travel time is at most C.

We now show that this decision problem is **NP**-complete via a reduction from the **NP**-complete 2 Directed Disjoint Paths (2DDP) problem [20]. This problem decides, given a directed graph $G = (V, E)$ and four nodes $s_1, s_2, t_1, t_2 \in V$, whether there is an s_1-t_1 path P_1 and an s_2-t_2 path P_2 such that P_1 and P_2 are edge-disjoint.

Theorem B.2. *Multiple-Routes is* **NP***-complete.*

Proof. Multiple-Routes is in **NP**, as the graph can be traversed non-deterministically starting at s and to construct n paths from s to t. The resulting flow and its overall travel time can be calculated in polynomial time.

It now suffices to show that Multiple-Routes is **NP**-hard. Given a 2DDP instance, the reduction adds two nodes s, t and four edges $e_{s\text{-}s_1}$,$e_{s\text{-}s_2}$,$e_{t_1\text{-}t}$ and $e_{t_2\text{-}t}$ to the graph G. Furthermore, for the two edges $e_{s\text{-}s_1}$ and $e_{t_1\text{-}t}$, the latency functions for all x are defined as

$$\tau_{e_{s\text{-}s_1}}(x) = \tau_{e_{t_1\text{-}t}}(x) = \begin{cases} 0, & \text{if } x \leq 1; \\ x - 1, & \text{else.} \end{cases}$$

For all other edges e, we define the latency function for all x as

$$\tau_e(x) = \begin{cases} 0, & \text{if } x \leq 2; \\ x - 2, & \text{else.} \end{cases}$$

In a setting with $n = 2$, 3 different drivers who have to be routed from s to t with overall costs of $C = 0$, we now show that there are two disjoint paths P_1, P_2 if and only if we are able to solve Multiple-Routes on the modified graph with the latency functions defined above.

Assume that there are two disjoint paths P_1, P_2. If we construct two new paths $P_1' = (e_{s\text{-}s_1}, P_1, e_{t_1\text{-}t})$ and $P_2' = (e_{s\text{-}s_2}, P_2, e_{t_2\text{-}t})$, the user equilibrium flow on this route set assigns one driver to path P_1' and two drivers to path P_2'. This is a n-user equilibrium since all used paths have a latency of 0 under this distribution. Thus, the overall travel time is 0 and the constructed instance is in Multiple-Routes.

For the opposite direction of the reduction, let there be a valid n-user equilibrium flow with an overall travel time of 0 on the graph G. By construction, there must be a path from s over s_1 and t_1 to t used by 1 agent and another path from s over s_2 and t_2 to t used by 2 agents, as we would have non-zero costs otherwise. Moreover, these two paths may not share an edge, as there would be non-zero costs otherwise. We have thus found two disjoint paths P_1 from s_1 to t_1 and P_2 from s_2 to t_2.

As the reduction is polynomial-time, this concludes the proof. □

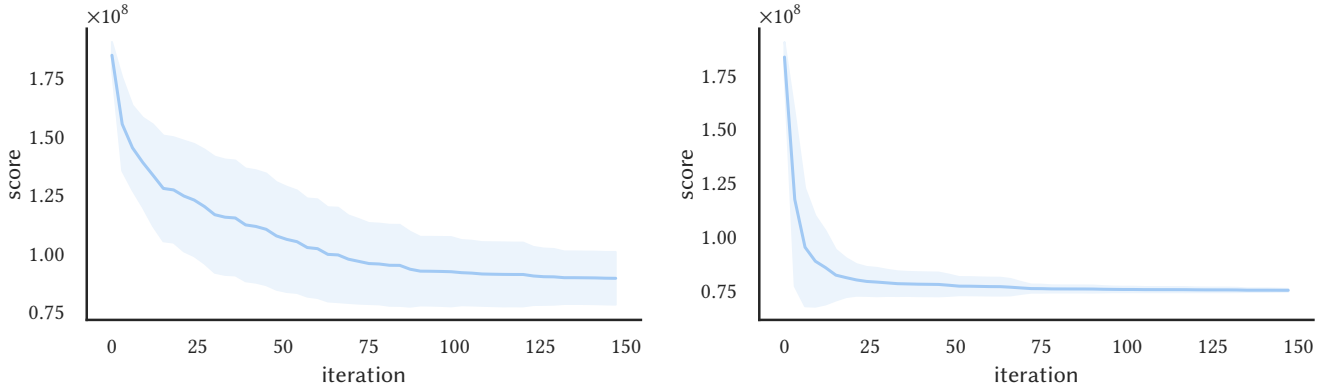

Figure 6: Fitness curves which show the mean absolute score (bold line) as well as the standard deviation (colored band) per iteration, for scenario 1. In the upper curve, the `rponly` setting is considered, while the lower curve uses the `wexseg` configuration.

C ACCELERATION OF RANDDIJKSTRA

The acceleration of shortest path algorithms is subject of intensive research [4]. Well-known speedup techniques for Dijkstra's algorithm like SHARC [5] often require preprocessing of the graph. In the case of RandDijkstra, we cannot employ acceleration techniques that require preprocessing due to the randomness of the edge weights. Hence, most modern shortest-path variants cannot be used in the Multiple-Routes EA.

The speedup technique proposed by Aviram and Shavitt [3] does not require any preprocessing. The authors describe a priority queue that utilizes an invariant of Dijkstra's algorithm. In Dijkstra's algorithm, once a node of value x has been popped from the queue, no node with distance less than x will be pushed into the queue again. As this invariant also holds for the RandDijkstra, we use their approach.

The basic idea is to represent the queue as an array, where the entry at index i is a linked list of all nodes pushed into the queue with weight i. We note that the weights are required to be integer values. Even though the RandDijkstra deals with floating-point weights, the randomly determined weights can be rounded in the insertion operation. As the weights have been randomized, this does not make any difference. This array allows for O(1) insertion and decrease-key operations.

The queue maintains a pointer to the last index from which an element was removed. According to the invariant, this pointer never decreases. Hence, when pointing to a non-empty cell, pop_min also is in O(1). Whenever the pointer points towards an empty cell, it increases until it finds a non-empty cell or the queue is empty. Hence, if w is the maximum weight of a node pushed to the queue, the runtime of Dijkstra's algorithm using this priority queue is $O(|E|+w)$. Note, that this is not a *real* priority queue anymore as it does not support the insertion of nodes with weight lower than the current pointer.

One important factor that determines the real-world runtime of this approach is the size of the array during initialization. If the array is too small, it needs to be resized often if a large weight gets pushed into the queue. If the array is too big, the initial memory allocation takes too much time. As an estimation, we set the initial queue size to 30 % of the largest weight encountered during the initialization of the population, but at least 65 565. In experiments, this has shown to be a good estimation for our traffic model and scenarios. For details of the implementation, we refer to the original paper [3].

D FURTHER EVALUATION OF THE FITNESS IMPROVEMENT PER ITERATION

For further analysis of `rponly` and `wexseg` configurations, we consider the respective fitness curves of this scenario in Figure 6. These two curves depict the development of the average score as well as the standard deviation throughout the 150 iterations for scenario 1.

First, it is very clear that the additional operators heavily reduce the spread of the scores, not only in the final iteration, but throughout the entire execution of the MREA. It is again shown that the mean score in the `wexseg` setting is lower than in the `rponly` setting. Additionally, the curve of the `wexseg` setting provides the insight that in most runs, the iteration budget of 150 is too high as most runs have converged around iteration 75.

E IMPLEMENTATION DETAILS

We implemented the MREA in C++ 17 and embedded it into the routing framework of Bläsius et al. [9], which allows for compatibility with the standard MATSim traffic simulator [23]. The source code can be found in our repository [8]. With exception of a priority queue, we use data structures from the C++ STL, rely on OpenMP for parallelization [14], and on the GNU Scientific Library [22].

REFERENCES

[1] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. 2013. Alternative Routes in Road Networks. *Journal of Experimental Algorithmics* 18, Article 1.3 (2013), 17 pages. https://doi.org/10.1145/2444016.2444019

[2] Enrique Alba, Gabriel Luque, and Sergio Nesmachnow. 2013. Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research* 20, 1 (2013), 1–48. https://doi.org/10.1111/j.1475-3995.2012.00862.x

[3] Nimrod Aviram and Yuval Shavitt. 2015. Optimizing Dijkstra for Real-World Performance. (2015). arXiv:1505.05033

[4] Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. 2016. Route Planning in Transportation Networks. In *Algorithm Engineering: Selected Results and Surveys*. Springer International Publishing. https://doi.org/10.1007/978-3-319-49487-6_2

[5] Reinhard Bauer and Daniel Delling. 2009. SHARC: Fast and Robust Unidirectional Routing. *Journal of Experimental Algorithmics* 14, Article 4 (2009), 29 pages. https://doi.org/10.1145/1498698.1537599

[6] Martin J. Beckmann, Charles B. MacGuire, and Christopher B. Winsten. 1956. *Studies in the Economics of Transportation*. Yale University Press.

[7] Jean Berger and Mohamed Barkaoui. 2003. A Hybrid Genetic Algorithm for the Capacitated Vehicle Routing Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2003*. 646–656. https://doi.org/10.1007/3-540-45105-6_80

[8] Thomas Bläsius, Maximilian Böther, Philipp Fischbeck, Tobias Friedrich, Alina Gries, Falk Hüffner, Otto Kißig, Martin S. Krejca, Pascal Lenzner, Louise Molitor, Leon Schiller, Armin Wells, and Simon Wietheger. 2021. *Strategic Routing GitHub Repository*. Retrieved April 03, 2021 from https://github.com/MaxiBoether/strategic-routing

[9] Thomas Bläsius, Maximilian Böther, Philipp Fischbeck, Tobias Friedrich, Alina Gries, Falk Hüffner, Otto Kißig, Pascal Lenzner, Louise Molitor, Leon Schiller, Armin Wells, and Simon Wietheger. 2020. A Strategic Routing Framework and Algorithms for Computing Alternative Paths. In *Proceedings of the 20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS) 2020*. 10:1–10:14. https://doi.org/10.4230/OASICS.ATMOS.2020.10

[10] Nick Cohn. 2019. *The TomTom Traffic Index: An Objective Measure of Urban Traffic Congestion*. Retrieved January 27, 2021 from https://www.tomtom.com/blog/road-traffic/urban-traffic-congestion/

[11] James Patrick Cohoon, Shailesh U. Hegde, Worthy N. Martin, and Dana S. Richards. 1987. Punctuated Equilibria: A Parallel Genetic Algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms and Their Application*. 148–154. https://doi.org/10.18130/V3219P

[12] José R. Correa and Nicolás E. Stier-Moses. 2011. Wardrop Equilibria. In *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc. https://doi.org/10.1002/9780470400531.eorms0962

[13] Stella Dafermos. 1980. Traffic Equilibrium and Variational Inequalities. *Transportation Science* 14, 1 (1980), 42–54. https://doi.org/10.1287/trsc.14.1.42

[14] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: An Industry Standard API for Shared Memory Programming. *IEEE Computational Science and Engineering* 5, 1 (1998), 46–55. https://doi.org/10.1109/99.660313

[15] Kalyanmoy Deb and Christie Myburgh. 2016. Breaking the Billion-Variable Barrier in Real-World Optimization Using a Customized Evolutionary Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2016*. 653–660. https://doi.org/10.1145/2908812.2908952

[16] Daniel Delling. 2009. Time-Dependent SHARC-Routing. *Algorithmica* 60, 1 (2009), 60–94. https://doi.org/10.1007/s00453-009-9341-0

[17] Daniel Delling and Dorothea Wagner. 2009. Time-Dependent Route Planning. In *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*. Springer, 207–230. https://doi.org/10.1007/978-3-642-05465-5_8

[18] Ugur Demiryurek, Farnoush Banaei-Kashani, and Cyrus Shahabi. 2010. A Case for Time-Dependent Shortest Path Computation in Spatial Networks. In *Proceedings of SIGSPATIAL 2010*. ACM, 474–477. https://doi.org/10.1145/1869790.1869865

[19] Edsger Wybe Dijkstra. 1959. A Note on Two Problems in Connexion with Graphs. *Numer. Math.* 1, 1 (1959), 269–271. https://doi.org/10.1007/BF01386390

[20] Steven Fortune, John Hopcroft, and James Wyllie. 1980. The Directed Subgraph Homeomorphism Problem. *Theoretical Computer Science* 10, 2 (1980), 111 – 121. https://doi.org/10.1016/0304-3975(80)90009-2

[21] Marguerite Frank and Philip Wolfe. 1956. An Algorithm for Quadratic Programming. *Naval Research Logistics Quarterly* 3, 1-2 (1956), 95–110. https://doi.org/10.1002/nav.3800030109

[22] Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Patrick Alken, Michael Booth, Fabrice Rossi, and Rhys Ulerich. 2019. *GNU Scientific Library Reference Manual*. Network Theory Ltd.

[23] Andreas Horni, Kai Nagel, and Kay W. Axhausen. 2016. *The Multi-Agent Transport Simulation MATSim*. Ubiquity Press. https://doi.org/10.5334/baw

[24] Alexander Kröller, Falk Hüffner, Łukasz Kosma, Katja Kröller, and Mattia Zeni. 2021. Driver Expectations towards Strategic Routing. *Transportation Research Record* (2021). To appear.

[25] Ekkehard Köhler, Rolf H. Möhring, and Martin Skutella. 2009. Traffic Networks and Flows over Time. In *Algorithmics of Large and Complex Networks*. Springer, 166–196. https://doi.org/10.1007/978-3-642-02094-0_9

[26] Henry B. Mann and Donald R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18, 1 (1947), 50–60. https://doi.org/10.1214/aoms/1177730491

[27] Giacomo Nannicini, Daniel Delling, Dominik Schultes, and Leo Liberti. 2011. Bidirectional A* search on time-dependent road networks. *Networks* 59, 2 (2011), 240–251. https://doi.org/10.1002/net.20438

[28] John F. Nash. 1950. Equilibrium Points in n-Person Games. *Proceedings of the National Academy of Sciences* 36, 1 (1950), 48–49. https://doi.org/10.1073/pnas.36.1.48

[29] Andreas Paraskevopoulos and Christos D. Zaroliagis. 2013. Improved Alternative Route Planning. In *Proceedings of ATMOS 2013*. Schloss Dagstuhl, 108–122. https://doi.org/10.4230/OASIcs.ATMOS.2013.108

[30] Michael Patriksson. 2003. *The Frank–Wolfe Algorithm*. Retrieved August 16, 2020 from http://www.math.chalmers.se/Math/Grundutb/CTH/tma946/0203/fw_eng.pdf

[31] Jean-Yves Potvin. 2009. State-of-the Art Review—Evolutionary Algorithms for Vehicle Routing. *INFORMS Journal on Computing* 21, 4 (2009), 518–548. https://doi.org/10.1287/ijoc.1080.0312

[32] Tim Roughgarden and Eva Tardos. 2002. How Bad is Selfish Routing? *J. ACM* 49, 2 (2002), 236–259. https://doi.org/10.1145/506147.506153

[33] Dan Simon. 2013. *Evolutionary Optimization Algorithms*. Wiley-Blackwell.

[34] United States Bureau of Public Roads, Office of Planning, Urban Planning Division. 1964. *Traffic Assignment Manual for Application with a Large, High Speed Computer*.

[35] Mariska Alice van Essen. 2018. *The Potential of Social Routing Advice*. Ph.D. Dissertation. University of Twente. https://doi.org/10.3990/1.9789055842377

[36] John Glen Wardrop. 1952. Some Theoretical Aspects of Road Traffic Research. *Proceedings of the Institution of Civil Engineers* 1, 3 (1952), 325–362. https://doi.org/10.1680/ipeds.1952.11259

[37] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *ACM SIGMOD Record* 25, 2 (1996), 103–114. https://doi.org/10.1145/235968.233324

[38] Shanjiang Zhu and David Levinson. 2015. Do People Use the Shortest Path? An Empirical Test of Wardrop's First Principle. *PLOS ONE* 10, 8 (2015), 1–18. https://doi.org/10.1371/journal.pone.0134322