Evolutionary Minimization of Traffic Congestion

Maximilian Böther, Leon Schiller, Philipp Fischbeck, Louise Molitor, Martin S. Krejca, and Tobias Friedrich

***Abstract*—Traffic congestion is a major issue that can be solved by suggesting drivers alternative routes they are willing to take. This concept has been formalized as a strategic routing problem in which a single alternative route is suggested to an existing one. We extend this formalization and introduce the multiple-routes (MRs) problem, which is given a start and destination and aims at finding up to *n* different routes that the drivers strategically disperse over, minimizing the overall travel time of the system. Due to the NP-hard nature of the problem, we introduce the MRs evolutionary algorithm (MREA) as a heuristic solver. We study several mutation and crossover operators and evaluate them on real-world data of Berlin, Germany. We find that a combination of all operators yields the best result, reducing the overall travel time by a factor between 1.8 and 3, in the median, compared to all drivers taking the fastest route. For the base case $n = 2$, we compare our MREA to the highly tailored optimal solver by Bläsius et al. (2020), and show that, in the median, our approach finds solutions of quality at least 99.69% of an optimal solution while only requiring 40% of the time.**

***Index Terms*—Evolutionary algorithm, optimization, strategic routing, traffic congestion.**

I. Introduction

TRAFFIC congestion is an increasing problem for urban areas across the world [3]. A solution is to route drivers by proposing them routes that reduce the overall travel time of the system, e.g., by navigation systems. Generally, proposing the *same* route to all drivers is not reasonable, as this rather *causes* traffic congestion if the number of drivers is too high. Instead, drivers need to disperse over *different* routes, with some of them taking suboptimal options into consideration [4]—a cost that some drivers are willing to take [5]. We refer to this setting as *strategic* routing. A well-studied domain that meets some of these requirements is route planning [6]. Most results consider a time component of each route, e.g., by considering flow over time [7] or predicted congestion [8], [9], [10], [11], or they consider multiple routes, where the alternative route needs to be substantially different [12], [13]. However, none of these results take the overall travel time of the system or psychological factors of the drivers into account.

A problem that does consider road capacities and psychological models for route choices by drivers is the recently introduced single-alternative-path (SAP) problem [2], a strategic-routing problem that aims to find an optimal alternative route to a given route for a group of drivers. Still, the SAP problem is restricted to a *single* alternative route and requires one route to be given as an input. In this article, we naturally extend the SAP problem to the more general multiple-routes (MRs) problem, which aims to minimize the overall travel time of all drivers in a system by proposing a set of routes to them, with the number of routes being controlled by a parameter. In order to account for bounded rationality and differing preferences by the drivers [14], we assume they form a user equilibrium (UE) on the given routes, i.e., a state in which no single driver can improve their travel time by choosing a different route. Since the MR problem is **NP**-hard, we introduce the MRs evolutionary algorithm (MREA) to heuristically solve it. The MREA belongs to the class of evolutionary algorithms—nature-inspired metaheuristics that have been applied to great success to hard problems in various domains [15], [16], including nonstrategic routing problems, e.g., the VEHICLE ROUTING PROBLEM [17], [18]. The MREA has a population size of μ, uses four different mutation operators (changing a single solution), and employs crossover (combining different solutions) to find good solutions to the MR problem.

Using real-world data for the city of Berlin, Germany, provided by TomTom Germany, we evaluate all operators of the MREA for different route scenarios and compare them to the naive solution of all drivers taking the fastest route. Our results (Table I) show that using more mutation operators and a larger population size yields better solutions. All three crossover operators that we suggest perform almost equally well, such that one can choose the fastest. Depending on the route scenario, a best configuration of the MREA reduces the overall travel time of the system by factors between 1.8 and 3, in the median. Even using a single mutation operator (a population size of 1 and no crossover) improves the solution by factors between 1.5 and 2.8. We adapt the MREA to the SAP problem and compare its solution quality to the deterministic, highly problem-specific exact solver of Bläsius et al. [2]. We find (Fig. 8) that the best configuration of the MREA, in the median, achieves a solution quality of at least 99.69% in only 40% of the runtime. Overall, our results suggest that the MREA is a heuristic well-suited for solving the MR

Manuscript received 25 January 2022; revised 3 August 2022 and 4 November 2022; accepted 3 December 2022. Date of publication 13 December 2022; date of current version 1 December 2023. The work of Martin S. Krejca was supported by the European Union's Horizon 2020 Research and Innovation Program through the Marie Sklodowska-Curie under Grant 945298-ParisRegionFP. This article extends our results published at GECCO'21 [1]. *(Corresponding author: Martin S. Krejca.)*

Maximilian Böther is with the Department of Computer Science, ETH Zurich, 8092 Zürich, Switzerland.

Leon Schiller, Philipp Fischbeck, Louise Molitor, and Tobias Friedrich are with the Hasso Plattner Institute, University of Potsdam, 14482 Potsdam, Germany.

Martin S. Krejca is with LIX, CNRS, École Polytechnique, Institut Polytechnique de Paris, 91764 Palaiseau, France.

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TEVC.2022.3228750.

Digital Object Identifier 10.1109/TEVC.2022.3228750

TABLE I

MEDIAN BEST FITNESS (LOWER IS BETTER) OF THE 75 RUNS (SECTION IV-B) FOR EACH OF THE SETTINGS FROM SECTIONS IV-C–IV-E FOR ALL 11 SCENARIOS. THE COLUMN k-DIJKSTRA STATES THE BEST POSSIBLE FITNESS IF ALL DRIVERS CHOOSE THE FASTEST ROUTES, ACCOUNTING FOR DELAYS CAUSED BY ALL DRIVERS USING THE SAME STREET, USING DIJKSTRA'S ALGORITHM. THIS FITNESS IS BEATEN BY ANY OF THE MREA CONFIGURATIONS. ALREADY THE FITNESS VALUES OF THE WEAKEST CONFIGURATION `rponly` ARE BETWEEN 35% AND 63% OF THOSE OF k-DIJKSTRA. IN GENERAL, THE FITNESS IMPROVES WITH BETTER CONFIGURATIONS (THAT IS, WITH ENTRIES FURTHER TO THE RIGHT, FOR THE COLUMNS MUTATION OPERATORS AND POPULATION SIZE). THE COLUMN CROSSOVER OPERATORS SHOW THAT THE CHOICE OF THE CROSSOVER OPERATOR HAS ALMOST NO IMPACT ON THE MEDIAN. NOTE THAT THE COLUMN `wexseg` IS THE SAME AS $\mu = 1$ AND THAT $\mu = 4$ IS THE SAME AS `no_heur` DUE TO HOW WE CONDUCT THE EXPERIMENTS. BOLD NUMBERS INDICATE A SIGNIFICANT CHANGE TO THE PREVIOUS COLUMN (IGNORING THE DETERMINISTIC k-DIJKSTRA), USING THE MANN–WHITNEY U TEST [31] WITH A p-VALUE OF 0.05

			Mutation Operators (see Section IV-C)				Population Size μ (see Section IV-D)				Crossover Operators (see Section IV-E)		
ID	k-Dijkstra	rponly	wnewroute	wlinkp	wexseg	$\mu = 1$	$\mu = 2$	$\mu = 4$	$\mu = 8$	no_heur	heur-all	heur-greed	heur-greed-rand
0	153530661	68566307	**64240948**	64662372	64240948	64240948	**61980563**	**59016819**	**58128546**	59016819	58369109	58128546	59016819
1	186995924	90878267	**74610652**	74610652	74610652	74610652	**74602749**	**74541545**	**74541545**	74541545	74541545	74541545	74541545
2	28713342	12364817	12292827	12237029	**12223856**	12223856	**12201922**	**12201922**	**12201922**	12201922	12201922	12201922	12201922
3	92721673	58477361	**50558429**	51197745	**52541858**	52541858	**49713176**	**49071681**	**48625397**	49071681	49071681	49071681	49044459
4	86902859	50225754	**44211386**	**44910249**	44188815	44188815	**43540844**	**41699314**	**41699314**	41699314	41699314	41699314	41699314
5	103023491	38395379	**37639244**	37344703	37387321	37387321	37174180	**37174180**	**37174180**	37174180	37174180	37174180	37174180
6	101643446	36504804	**36353121**	36266833	36347074	36347074	36259504	**36259504**	36259504	36259504	36259504	36259504	36259504
7	40930113	18904046	**18764355**	18764355	18764355	18764355	**18764355**	**18636335**	**18558820**	18636335	18636335	18716682	18636335
8	12014974	6860945	**6407381**	6407381	6407381	6407381	6407381	6407381	6407381	6407381	6407381	6407381	6407381
9	394317060	160818584	**141236603**	140966107	142441589	142441589	**134170060**	**131325658**	**130535412**	131325658	131157891	131088667	131325658
10	57461353	24477839	**22899049**	22876414	22899049	22899049	**22876414**	**22876414**	**22876414**	22876414	22876414	22876414	22876414

problem and, thus, reducing traffic congestion in strategic scenarios.

In Section II, we formalize the MR problem, and we introduce the MREA in Section III. In Section IV, we analyze the performance of the MREA and the effect of its operators and population size. In Section V, we apply the MREA to the SAP problem and compare it against the algorithm of Bläsius et al. [2]. We conclude our work in Section VI. For supplementary material, we refer to our repository [19].

II. MULTIPLE-ROUTES PROBLEM

Given a route network graph $G = (V, E)$ and a continuous flow of $k \in \mathbf{R}_{\geq 0}$ drivers per unit of time between an origin $s \in V$ and a destination $t \in V$, we consider routing this flow among $n \in \mathbf{N}^+$ routes, where we assume that drivers distribute among these n routes such that no driver in this flow can choose a quicker route as long as no other driver cooperatively changes their route. We call such a state an n-restricted UE (n-UE). The MRs problem aims to find an optimal set of n routes such that the overall travel time of drivers in an n-UE is minimized.

In the following, we describe how we model the MR problem (Section II-A), prove that it is **NP**-hard (Section II-B), and go into detail about the UE (Section II-C).

A. Problem Modeling

We follow the formalization by Roughgarden and Tardos [20] but add the constraint of n routes. Let $G = (V, E)$ be a directed graph, $s \in V$, $t \in V$, $k \in \mathbf{R}_{\geq 0}$, and $n \in \mathbf{N}^+$. Further, let $\mathcal{P}_{s,t}$ denote the set of all routes from s to t. A traffic flow $f\colon \mathcal{P}_{s,t} \to \mathbf{R}_{\geq 0}$ is a mapping that assigns to each $P \in \mathcal{P}_{s,t}$ a value representing the amount of drivers on each edge of P per unit of time. Note that this value may not be an integer. We call a traffic flow *valid* if and only if $|\{P \in \mathcal{P}_{s,t} \mid f(P) > 0\}| \leq n$ and if $\sum_{P \in \mathcal{P}_{s,t}} f(P) = k$. Further, if and only if f is an n-UE (Section II-C), we call the traffic flow *stable*. The travel time of drivers on an edge $e \in E$ is determined by a latency function $\tau_e\colon \mathbf{R}_{\geq 0} \to \mathbf{R}_{\geq 0} \cup \{\infty\}$. That is, for all $x \in \mathbf{R}_{\geq 0}$, $\tau_e(x)$ defines the time a single driver needs to travel along e assuming there are x agents entering e per unit of time.[1] We assume τ_e to be monotonically increasing and continuous. For a traffic flow f, the flow f_e over e is then $\sum_{P \in \mathcal{P}_{s,t}\colon e \in P} f(P)$, and the overall travel time of drivers on route $P \in \mathcal{P}_{s,t}$ is $\tau_P(f) = \sum_{e \in P} \tau_e(f_e)$.

Last, for each traffic flow f, we associate a cost $\mathcal{C}(f)$ that denotes the overall travel time of all drivers. Formally

$$\mathcal{C}(f) = \sum_{P \in \mathcal{P}_{s,t}} f(P) \cdot \tau_P(f). \tag{1}$$

The MR problem aims to find a valid and stable traffic flow with minimum cost among all valid and stable traffic flows.

B. NP-Hardness of Multiple-Routes

In the following, 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 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-UE 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.

[1]The latency function can also be used to model road capacities by setting it to infinity if too many drivers access a road.

We show that this decision problem is **NP**-complete via a reduction from the **NP**-complete 2 Directed Disjoint Paths (2DDP) problem [21]. 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 1: Multiple-Routes is **NP**-complete.

Proof: Multiple-Routes is in **NP**, as the graph can be traversed nondeterministically starting at s and constructing n paths from s to t. The resulting flow and its overall travel time can be calculated in polynomial time. It remains to show that Multiple-Routes is **NP**-hard.

Given a 2DDP instance, the reduction adds two nodes s and t and four edges e_{s-s_1},e_{s-s_2},e_{t_1-t} and e_{t_2-t} to the graph G. Furthermore, for the two edges e_{s-s_1} and e_{t_1-t}, the latency functions for all x are defined as follows:

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

For the remaining edges e, we define for all x

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

Assume an instance with $n = 2$ routes, $k = 3$ 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 and 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 and P_2. If we construct two new paths $P_1' = (e_{s-s_1}, P_1, e_{t_1-t})$ and $P_2' = (e_{s-s_2}, P_2, e_{t_2-t})$, the UE flow on this route set assigns one driver to path P_1' and two drivers to path P_2'. This is a n-UE 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-UE 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 two agents, as we would have nonzero costs otherwise. Moreover, these two paths may not share an edge, as there would be nonzero 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. ■

C. User Equilibrium

In routing games, a UE, also known as *Wardrop equilibrium* [22], is a game state where no player has anything to gain by changing only their own strategy [23]. This state occurs when all drivers act selfishly and choose their route such that they aim to minimize their travel time, given that other drivers also occupy roads [20]. In the MR problem, we consider n-UEs, where no driver can improve their travel time by unilaterally changing their route while the traffic flow stays valid. Given (G, s, t, k), a UE always exists [20], [24], [25]. However, in contrast to UEs, an n-UE is a traffic flow with a route set of maximum size n where drivers are not allowed to choose a new route if this exceeds the number of n different routes in total. In particular, an n-UE does not have to be unique, as it highly depends on n. Nonetheless, each valid UE is also an n-UE.

We approximate an n-UE by computing a UE under the constraint of using at most n routes. To this end, we model the UE as a convex problem [24], which we approximately solve with the Frank–Wolfe algorithm [26], adjusted such that it makes sure to satisfy the constraint of at most n routes.

In the following, we overview the Frank–Wolfe algorithm as well as its step-size, which is a crucial parameter.

1) Frank–Wolfe Algorithm for User Equilibria: In the following, we provide a more formal definition of the UE and some background on how to calculate it. To this end, we first introduce the following function that redistributes flow between paths.

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

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

Definition 3 (UE, [20]): 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 UE if and only if for all $P_1, P_2 \in \mathcal{P}_{s-t}$, $\delta \in (0, f(P_1)]$, $\tau_{P_1}(f) \leq \tau_{P_2}(\widetilde{f}_\delta^{(P_1,P_2)})$.

The MREA (Algorithm 2) calculates the UE of the MR problem by optimizing a convex program via the Frank–Wolfe algorithm [26]. In order to apply this algorithm, the function to be optimized as well as the set of possible solutions need to be convex. Following Patriksson [27], the Frank–Wolfe Algorithm works as described in Algorithm 1. In each step, it solves a linear program that approximates the convex program and then moves toward the minimizer of this program. The optimal step size is chosen according to the objective function via a line search.

The UE can be expressed as a convex program [24]. For a flow u, let $z(u) = \sum_{e\in E} \int_0^{u(e)} \tau_e(x)\, dx$. For $(s, t) \in V^2$, the flow u corresponding to the UE is the minimum of z, subject to $\sum_{P\in\mathcal{P}} u(P) = k$ and $\forall P \in \mathcal{P}_{s,t} : u(P) \geq 0$. We show that the solution set of this convex program is convex. To this end, we introduce a new concept called *flow vectors*, allowing interpolation between flows. For every flow f, we derive a flow vector $\bar{f}$. 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 $\bar{f}(P)$ for the amount of traffic flow assigned to route P by the flow vector $\bar{f}$. Similar to route flows, flow vectors induce edge flow vectors.

Lemma 1: 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 $\sum_{P\in\mathcal{P}} u''(P) = k$ holds. 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

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

As described in Algorithm 1 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 $p_q^T \nabla z(x_q) = \sum_{e\in E} \tau_e(x_q(e)) \cdot p_q(e)$, subject to $\sum_{P\in\mathcal{P}} p_q(P) = k$ and $\forall P \in \mathcal{P} : p_q(P) \geq 0$. Furthermore, the constraint $p_q \in \mathcal{D}$ is equivalent to p_q being a flow vector. Hence, this linear program needs to be solved in every iteration in order to obtain p_q based on the current flow vector x_q.

We 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 2: When calculating a UE 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 $\mathcal{C}$, 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 II-C. In order to approximate a UE, 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 UE on the routes quite well.

2) 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 toward the global minimum of z that can be made from one iteration to the next. To this end, we consider the derivatives with respect to γ

$$\tilde{z}'(\gamma) = \sum_{e\in E} \tau_e\big(x_q(e) + \gamma\big(p_q(e) - x_q(e)\big)\big)\cdot\big(p_q(e) - x_q(e)\big)$$

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

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 U.S. Traffic Model we introduce in Section IV, we obtain

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

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

Algorithm 1: FRANK–WOLFE Algorithm [26] for Optimizing a Convex Program

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

```
q ← 0;
while not converged do
    Find p_q minimizing the following linear program
        minimize p_q^T ∇f(x_q)
        subject to p_q ∈ D;
    γ ← Line-Search Determination of step size;
    x_{q+1} ← x_q + γ(p_q − x_q);
    q ← q + 1;
return x_q;
```

III. MULTIPLE-ROUTES EA

The MREA (Algorithm 2) is an elitist EA for optimizing the MR problem. Given an MR instance (G, s, t, k, n), it maintains a population of μ route sets (the *individuals*), each of which consists of exactly n (not necessarily different) routes from s to t. Each individual is scored via a value (the *fitness*), which is determined by first approximating the n-UE, as described in Section II-C, and then scoring the resulting traffic flow via (1). Individuals are compared via their fitness, and a lower fitness is considered better.

The MREA generates offspring in two different (and exclusive) ways: by 1) via a crossover operation (lines 8–11) and 2) by employing a random number of mutation operators to a copy of each individual (lines 12–20). Then, the MREA reduces the population size to μ via truncation selection, breaking ties uniformly at random (line 23). Note that to avoid a single good individual being copied via crossover and then taking over the entire population, the offspring generated by crossover is only considered for selection if there is an individual in the offspring population, that is, strictly better than the best individual in the parent generation (lines 21 and 22). The algorithm stops after a user-defined termination criterion. Although the MREA operates on *sets* of routes, many operators also perform changes to *single* routes. To this end, the subroutine RandDijkstra (RD) is used, which finds a shortest path on G with randomly perturbed edge weights.

In the following, we explain the RD subroutine (Section III-A) and then go into detail about the mutation (Section III-B) and crossover (Section III-C) operators of the MREA.

Algorithm 2: MULTIPLE-ROUTES EA. Note That We Use Set Notation, Even Though the Sets Are Multisets

Input: MR instance (G, s, t, k, n), population size μ, crossover strategy *cStra*, termination criterion
Output: Set of n routes from s to t

1 $P \leftarrow \emptyset$;
2 **repeat** μ **times**
3   *ind* $\leftarrow$ new individual;
4   **repeat** n **times**
5   add route RANDDIJKSTRA(s, t, k) to *ind*;
6    $P \leftarrow P \cup \{ind\}$;
7 **while** *termination criterion not met* **do**
8    $C \leftarrow \emptyset$;
9   **repeat** $\sqrt{\mu^2 - \mu/2}$ **times**
10      $ind_1, ind_2 \leftarrow$ chosen u.a.r. from P;
11      $C \leftarrow C \cup \{cStra(ind_1, ind_2)\}$;
12    $P' \leftarrow$ copy of P;
13   **for** *every individual ind in* P' **do**
14   *mutations* $\leftarrow \max(1, \text{Pois}(1.5))$;
15   *ops* $\leftarrow \emptyset$;
16   **repeat** *mutations* **times**
17   *ops* $\leftarrow$ *ops* $\cup$ {randomly weighted selected operator in {NewRoute, RandomP, LinkWP, ExSegment}};
18   **if** *ops contains* ExSegment **then**
19   *ops* $\leftarrow$ {ExSegment};
20   apply operators in *ops* to *ind*;
21   **if** *no individual in C is strictly better than the best in P* **then**
22      $C \leftarrow \emptyset$;
23    $P \leftarrow$ the μ best individuals in $C \cup P' \cup P$;
24 **return** *the best individual in P*;

A. RandDijkstra

The RD is a randomized variant of Dijkstra's shortest-path algorithm [28]. Given two nodes s and t, it returns a random, yet still short route from s to t. RD works like Dijkstra's algorithm, but whenever relaxing an edge e, its weight w is perturbed such that $w \sim N(\tau_e(x), 0.8 \cdot \tau_e(x))$, where τ_e is the latency of e and where x is the traffic flow routed from s to t and where 0.8 was determined a good value in preliminary tests. Due to its extensive use, RD contributes the most to the runtime of the MREA. Hence, we consider in the following possible speed-up techniques.

Acceleration of RANDDIJKSTRA*:* The acceleration of shortest path algorithms is subject of intensive research [6]. Well-known speed-up techniques for Dijkstra's algorithm, like SHARC [29], often require preprocessing of the graph. In the case of RD, 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.

We use the approach of Aviram and Shavitt [30], which does not use preprocessing. It employs a priority queue that utilizes the invariant of Dijkstra's algorithm that once a node of value x has been popped from the queue, no node with a distance less than x is pushed into it again. This invariant also holds for the RD. The approach represents the queue as an array allowing for $O(1)$ insertion and decrease-key operations. The entry at index i is a linked list of all nodes pushed into the queue with weight i, required to be integer values. Thus, the randomly determined floating-point weights of RD are rounded in the insertion operation. The queue maintains a pointer to the last index from which an element was removed. Due to the invariant, this pointer never decreases. Hence, when pointing to a nonempty cell, the pop operation that gives us the minimal element also is in $O(1)$. Whenever the pointer points toward an empty cell, it increases until it finds a nonempty 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 whenever a large weight gets pushed into the queue. If the array is too big, the initial memory allocation takes 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 [30].

B. Mutation Operators

In total, the MREA has four mutation operators: 1) *NewRoute*; 2) *RandomP*; 3) *LinkWP*; and 4) *ExSegment*, each with its own weight. When mutating an individual, the MREA first decides how many mutations to execute consecutively. This number is determined by a Poisson distribution with an expected value of 1.5, but at least one mutation is performed (line 14 in Algorithm 2). Afterward, for each mutation to apply, a mutation operator is chosen randomly proportionally to its weight (line 17). If ExSegment is chosen, then all other operators are discarded for this mutation (lines 18 and 19). Last, all chosen operators are applied to the individual (line 20). In the following, we detail all four mutation operators.

1) NewRoute: Chooses a single route randomly proportionally to its inverse traffic flow and replaces this chosen route with one computed by RD. The weight of NewRoute is determined dynamically. In order to have a good exploration–exploitation tradeoff, it is 30 for the first ten iterations, and then lowered linearly such that it reaches 1 in iteration 200.

2) RandomP: Replaces subsegments of a randomly selected subset of routes via RD. The routes to be modified are chosen proportionally to their inverse traffic flow. For each such route, r denoting its length, RandomP chooses a start node uniformly at random and a destination node by advancing a number of steps according to the Gaussian distribution $N(0.25r, 0.5r)$. Then, RD replaces the route segment between these two nodes. In order to find a different subsegment between these two nodes, RD increases the costs of the edges of the current route. Last, all cycles that may occur in the route after the replacement are deleted, i.e., if a route visits a vertex v twice, the edges between the two visits form a cycle and are removed. RandomP has a constant weight of 60.

3) LinkWP: Is identical to RandomP except for the choice of delimiting nodes of the subsegment to replace. For each node v on a chosen route, LinkWP calculates a metric that describes how likely it is for a meaningful subroute to occur at v. The metric is defined as the sum of the capacities of all outgoing edges of v except for the edge currently used in the route. The start node is chosen randomly proportionally to this metric. The destination node is chosen randomly by selecting one of the nodes on the original route that comes after the start node, proportionally to the same metric.

LinkWP has a constant weight of 30. Note that this weight is lower than the one of RandomP in order to not introduce a too heavy problem-specific bias into the mutation step.

4) ExSegment: Swaps subsegments between two routes of the same individual. First, it chooses a pair of different routes uniformly at random and removes their cycles. Then, it determines the nodes occurring in both routes, which we call *shared points*. Among the shared points, let the *divergence points* be the nodes whose successor is different in both routes, and let the *goto points* be those whose predecessor differs. ExSegment chooses one divergence point v_s uniformly at random and a node v_t uniformly at random from the set of all goto points that appear after v_s. If such nodes exist, the route segments between v_s and v_t from both chosen routes are then swapped. If not, nothing happens (see Fig. 1 in the supplementary material for more details).

The weight of ExSegment is determined dynamically. If ExSegment was applied within the last six iterations, its weight is 0, as this operator is expensive and a too rapid succession of uses is unlikely to change much. If ExSegment was applied more than six iterations ago, its weight is determined as follows. It starts at 15 and is increased linearly to 30, depending on the iterations without improvement. The point in time when it reaches exactly 30 depends on the used convergence criterion (see Section IV for more details).

C. Crossover Operators

We consider three different binary crossover operators. We recall that, in contrast to the mutation operators, the MREA only uses a single crossover operator. This is due to there being a large tradeoff between runtime cost and improvement in solution quality when considering different operators and due to the operators all being versions of the same idea.

Regardless of the operator chosen, the MREA creates $\sqrt{\mu^2 - \mu/2}$ offspring in each iteration. Note that this number is the square root of all possible $\binom{\mu}{2}$ 2-combinations of μ individuals. By the birthday paradox, the possibility of a combination of two individuals being chosen at least twice becomes over 50% once in the order of this value. Thus, when creating $\sqrt{\mu^2 - \mu/2}$ offspring, we aim to create as many individuals as possible without getting many doubles.

All of our proposed operators consider a *diversity score D* that reflects how similar the routes of an individual are. The assumption is that a more disjoint route set usually leads to a lower overall travel time, due to less congestion on single roads. For an individual S and an edge $e \in E$, let c_e^S denote the count how often the edge appears in S. The score D of S is defined such that larger values are worse

$$D(S) = \frac{\sum_{e \in \{e \in E | c_e^S > 1\}} \left(c_e^S\right)^2}{\max\left(1, \sum_{e \in \{e \in E | c_e^S = 1\}} c_e^S\right)}.$$

In the following, we explain how each crossover operator constructs a new solution. In addition, Fig. 2 in the supplementary material provides further details.

1) Exhaustive Crossover: Considers all $\binom{2n}{n}$ route sets possible from the routes of the two parents, and returns the combination with the lowest diversity score.

2) Greedy Crossover: Greedily constructs a new route set, guided by D. It randomly chooses one of the $2n$ routes of the parents, proportionally to their inverse traffic flow. The remaining $n-1$ routes of the new solution are chosen greedily among the remaining routes of both parents, always choosing the first (new) route such that the current diversity score is minimized.

3) Randomized Greedy Crossover: Takes the same approach as Greedy Crossover, but instead of greedily choosing the route maximizing the diversity score of the route set, it randomly selects one of the $2n$ routes, with replacement, proportionally to the inverse of its diversity score. That is, the more diverse the route set with that route is, the more likely the route is to be chosen.

IV. Parameter Evaluation

We empirically analyze the utility of the operators of the MREA on the street network of Berlin, Germany. For each operator, we investigate how much the solution quality of the MREA changes when it is added to the algorithm. In Section IV-C, we begin by evaluating the mutation operators, excluding crossover. In Section IV-D, we analyze the impact of the population size μ. In Section IV-E, we add crossover to the MREA, and we compare the quality achieved by the three different crossover operators with each other. Last, in Section IV-F, we compare the elitist selection strategy of the MREA to tournament selection. Our evaluations show that using more mutation operators, a larger population size, and crossover are all beneficial for improving the best fitness of the MREA. The largest improvement is made by adding the operators RandomP and NewRoute. Further, adding more operators generally decreases the spread of the results, in addition to improving them. Table I summarizes the median best fitness of all our parameter settings.

A. Implementation Details

We implemented the MREA in C++ 17 and embedded it into the routing framework of Bläsius et al. [2], which allows for compatibility with the standard MATSim traffic simulator [32]. The source code is in our repository [19]. With exception of a priority queue, we use data structures from the C++ STL, rely on OpenMP for parallelization [33], and on the GNU Scientific Library [34].

B. Experimental Setup

We consider MR instances with the graph G being the street network of Berlin,[2] Germany, provided by TomTom Germany, and with $k = 3000$, which is a reasonable choice [2]. We choose $n = 2$ in order to model proposing a driver with a small choice of fast routes. Choosing larger values makes this choice more troublesome for the driver, and it makes it also more unlikely to find that many different and fast routes. We choose the following 11 highly diverse *scenarios*.

0) Babelsberg – Lichterfelde not inner city; country road, nonobvious deviation.
1) Griebnitzsee – Ahrensfelde very long; fastest route uses express highway (EH), the second fastest route goes through the inner city.
2) KaDeWe – East Side Gallery short, inner city; many possible detours.
3) Lichterfelde – Prenzlauer Berg long, south to north; EH and inner-city side streets.
4) Lichterfelde – Steglitz very short, inner city; direct route uses. Side streets, but highway and EH are nearby.
5) Moabit – Birkenwerder long, start in the city center; choice for highway or EH.
6) Olympiastadion – Rotes Rathaus long, inner city; possible almost entirely on a highway.
7) Potsdamer Platz – Pergamonmuseum short, inner city; different highways or side streets. That are reasonable, in a Manhattan-like layout.
8) Potsdamer Platz – Tempelhofer Feld medium long, inner city; bottleneck at a bridge, but opportunity to split up onto two highways.
9) Teltow – Hoppegarten long, south-west to east; either long detour using EH or a more direct inner-city highway.
10) Wannsee – Schönefeld the k-Dijkstra shortest route detours to use EH.

For the latency functions, we follow the recommendation of the U.S. Bureau of Public Roads [35], that is, we choose $\tau_e(x) = (\ell_e/s_e) \cdot 1.15(x/c_e)^2$ where s_e, c_e, and ℓ_e denote free-flow speed, capacity, and length of e, respectively, [2].

For the experiments, we consider various *settings.* For each, the termination criterion of the MREA is to stop after 150 iterations. The weight of ExSegment (Section III-B4) is chosen such that it reaches a value of 30 if there was no improvement in the last 20% $\cdot 150 = 30$ iterations. We start 75 independent runs of the MREA on all 11 scenarios per setting.

1) *Boxplots:* The box denotes the mid-50% of the 75 runs, and the whiskers denote the mid-90%. All remaining data points are depicted as diamonds.
2) *Solution Space Size:* Our results indicate that many runs with different settings have equal fitness. This suggests that the solution space is small, highlighting the impact of adding a new operator.

[2]This graph has 158 864 vertices and 342 778 edges.

C. Analysis of the Mutation Operators

We analyze the utility of the MREA's four mutation operators (Section III-B) by considering how well each operator performs on its own (Section IV-C1), how well different combinations of operators perform (Section IV-C2), as well as how quickly the algorithm finds a solution that it does not improve anymore (Section IV-C3). To this end, we do not employ crossover, and we choose a population size of $\mu = 1$ in order to see how much a single solution can be improved by solely mutation.

In Section IV-C1, we consider the operators individually, except for ExSegment (Section III-B4), as it only modifies existing routes with existing segments and does not explore new road segments. The configurations of the MREA that each use a single operator are named as follows.

1) `rponly` only uses RandomP.
2) `wponly` only uses LinkWP.
3) `nronly` only uses NewRoute.

In Sections IV-C1 and IV-C2, we consider four different algorithm configurations, starting with a single operator and then adding more operators.

1) The MREA has only access to RandomP (`rponly`).
2) `rponly` but adding NewRoute (`wnewroute`).
3) `wnewroute` but adding LinkWP (`wlinkp`).
4) Using all four operators (`wexseg`).

We note that the wall clock time for all configurations during these experiments was very similar, with the fitness function evaluation being the most costly operation.

1) Single Best Operator: We study the impact of each configuration on the best fitness achieved after our termination criterion of 150 iterations. The results are depicted in Fig. 1.

For most scenarios, `rponly` performs best with respect to the mean best fitness and the top 75%. Between `wponly` and `nronly`, there is no clear distinction which of both it better in terms of median best fitness. For some scenarios, `wponly` is better, for others, `nronly`. More interestlingly, if `rponly` is outperformed, then by `wponly`. Since both respective mutation operators are similar, with the difference that LinkWP uses more specific information than RandomP, this suggests that it is typically initially better to start with more random choices (as in RandomP). This is also the case why `rponly` is our first configuration in the following experiments.

2) Operator Combinations: We study the impact of the combined configurations on the best fitness achieved after our termination criterion of 150 iterations. Our results are depicted in Fig. 2.

Adding NewRoute yields the largest improvement, with a statistical significance for all scenarios, except for scenario 2. This could be due to it being very short. Thus, RandomP and NewRoute become very similar operations. Averaged over all 11 scenarios, 84% of the `wnewroute` runs are better and 87% are better or equal to the median of the `rponly` runs. For scenarios 4 and 8, all runs of `wnewroute` are better than the median of `rponly`. This is likely a result of scenarios 4

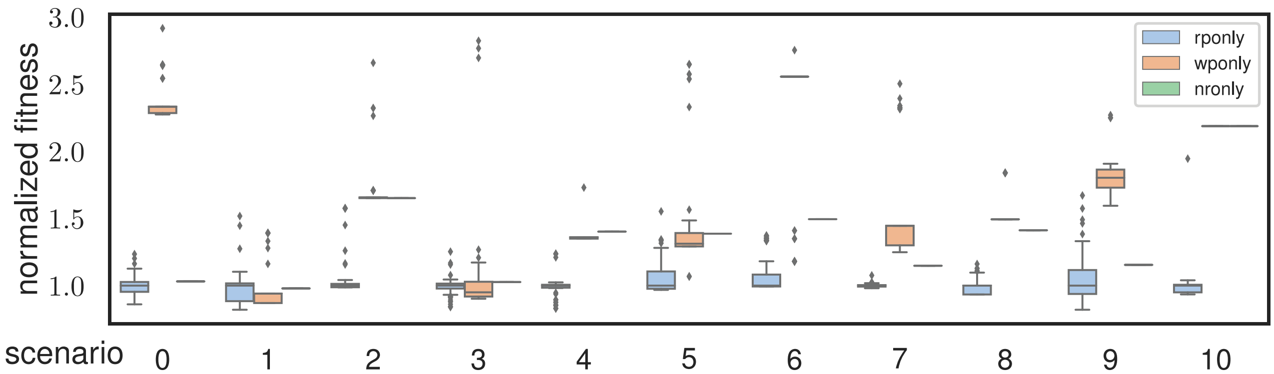

Fig. 1. Boxplots (Section IV-B) of the normalized best fitness of the MREA with $\mu = 1$ after 150 iterations for all 11 scenarios, with 75 runs per scenario. Each of the three colors, from left to right, represents the MREA using exactly one mutation operator from Section III-B. Per scenario, the fitness is normalized to the median of `rponly`. In general, `rponly` performs best. Please refer to Section IV-C1 for more details.

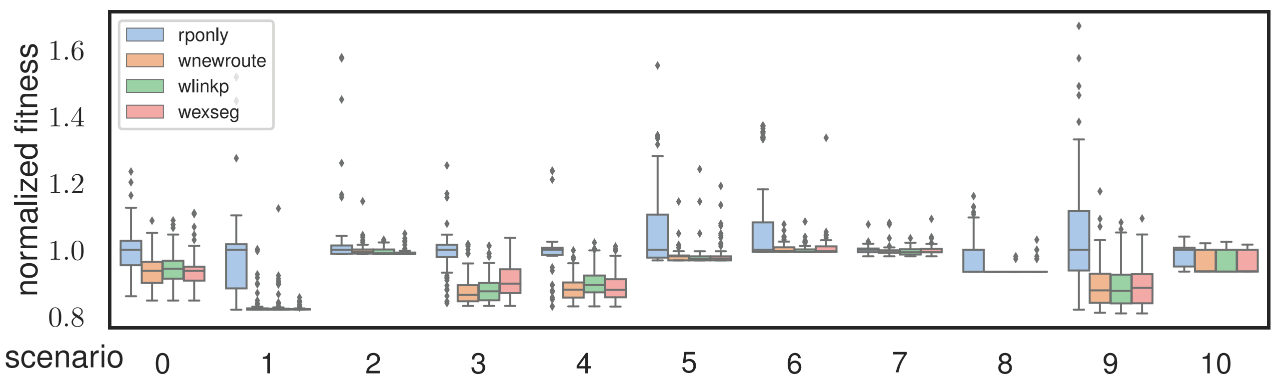

Fig. 2. Boxplots (Section IV-B) of the normalized best fitness of the MREA with $\mu = 1$ after 150 iterations for all 11 scenarios, with 75 runs per scenario. Each of the four colors, from left to right, represents one of the algorithm configurations explained in Section IV-C. Per scenario, the fitness is normalized to the median of `rponly`. In general, configurations with more mutation operators (more to the right per scenario) result in a better final fitness. Please refer to Section IV-C2 for more details.

and 8 requiring two almost disjoint routes, which are more easily found by NewRoute, whereas other scenarios require two nearly identical routes.

Interestingly, in scenarios 3 and 5, LinkWP and ExSegment increase the median best fitness. For LinkWP, recall that it prefers edges with a high capacity. If the best routes do not use such edges, LinkWP has no benefit. Nonetheless, averaged over all scenarios, 84% of the `wlinkp` runs are better and 88% are better or equal to the median of the `rponly` runs. For ExSegment, recall that it swaps segments locally optimally, with respect to the segments randomly chosen. Escaping from such a local optimum can prove hard in certain scenarios, especially, since we only consider a population size of 1. Still, on average, 38% of the `wexseg` runs are better and 56% are better or equal to the median of the `wlinkp` runs, showing a general benefit of ExSegment.

a) Evaluation of the fitness improvement per iteration: We analyze `rponly` and `wexseg` configurations by considering the respective fitness curves. Fig. 3 depicts the development of the average fitness as well as the standard deviation throughout the 150 iterations for scenario 1.

We observe that the additional operators heavily reduce the spread of the fitness, not only in the final iteration but throughout the entire execution of the MREA. The mean fitness in the `wexseg` setting is lower than in the `rponly` setting. Last, the curve of the `wexseg` setting shows that in most runs, the iteration budget of 150 is sufficient as most runs have converged around iteration 75.

3) Speed of Convergence: In order to analyze how quickly the MREA reaches a local optimum from which it cannot escape within its budget of 150 iterations, we consider the last iteration in which the MREA changed the fitness of its best individual. The results are depicted in Fig. 4. Note that this analysis does not consider the fitness of each run, only whether it changed in subsequent iterations or not. For a more complete picture, please also refer to the results from Section IV-C2, which show that, on average, configurations with more operators have a better median performance.

The speed of convergence depends on the scenario, and there is no clear trend among the four configurations. The mid-90% are generally close to the extreme values of 0 and 150. Runs close to 0 show that the scenarios are hard, as the MREA gets stuck very quickly. In contrast, runs close to 150 show that the budget of 150 iterations was insufficient for convergence.

Conclusion: Averaged over all scenarios, more mutation operators lead to a better performance. However, this effect is not very well pronounced for the addition of LinkWP, indicating that it should possibly be merged with the similar operator RandomP. Still, using both operators is overall better than just

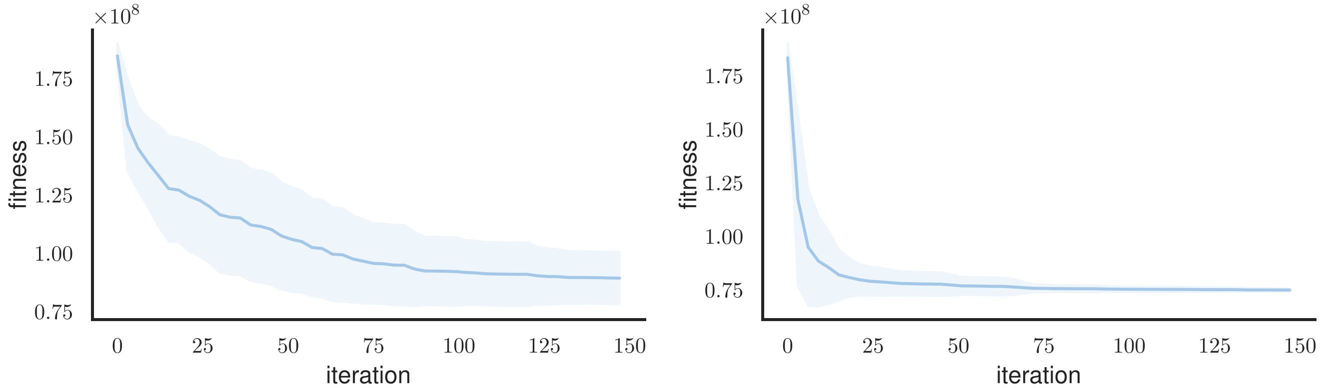

Fig. 3. Fitness curves of the mean absolute fitness (bold line) as well as the standard deviation (colored band) per iteration, for scenario 1. The left plot shows the `rponly` setting, the right plot shows the `wexseg` configuration (see also Section IV-C1).

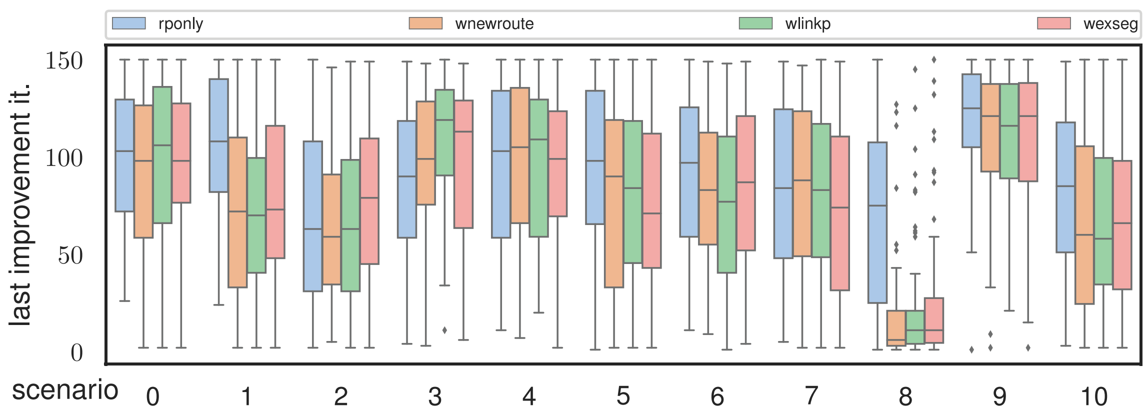

Fig. 4. Boxplots (Section IV-B) of the last of, in total, 150 iterations in which the MREA with $\mu = 1$ improved its best fitness, for all 11 scenarios. Each of the four colors, from left to right, represents one of the algorithm configurations explained in Section IV-C, and each configuration was run 75 times per scenario. Regardless of the scenario, there is a large spread between runs that get stuck quickly and runs that do not converge within 150 iterations. Please refer to Section IV-C3 for more details.

using RandomP. Further, the large spread in the speed of convergence among all configurations and scenarios suggests that the initialization has a large impact on how easy it is to find improvements, more or less regardless of what configuration is run. This indicates that a larger population size may be beneficial, as it increases the initial diversity.

D. Analysis of the Population Size

We analyze to what extent the MREA benefits from having a population size larger than 1. Since Section IV-C suggests that local optima pose a problem for the MREA, a larger population size may help to have alternative solutions to those stuck in local optima. We do not employ crossover but use all four mutation operators, that is, we use the `wexseg` configuration. Our results are depicted in Fig. 5. Note that a higher population size also means more fitness evaluations, as we let each configuration run for 150 iterations. This likely explains the high significances between different configurations.

A larger number of individuals improves the median best fitness and reduces the spread. Our results suggest that the improvement for $\mu = 2$ and $\mu = 4$ provide a large improvement over $\mu = 1$. For $\mu = 8$, the improvement in comparison to $\mu = 1$ in median and spread is somewhat smaller. Throughout all scenarios, 61% of the runs with $\mu = 2$ are better and 77% are better or equal to the median of $\mu = 1$. For the runs with $\mu = 4$, these numbers increase to 80% and 93%, respectively. For $\mu = 8$, the increase from $\mu = 4$ is smaller, reaching 88% better and 98% better or equal runs. When comparing to the configuration with $\mu = 2$, 40% of the runs with $\mu = 4$ are better than the median and 85% of the runs are better or equal. For $\mu = 8$, these numbers increase to 49% and 96%.

Conclusion: Using a larger population size improves the quality of the best fitness and decreases the spread among the different runs per scenario. However, the computation cost increases with the population size, and the quality gain in fitness from larger populations varies among the different configurations. Our experiments suggest the sweet spot $\mu = 4$.

E. Analysis of the Crossover Operators

We analyze the utility of the MREA's three crossover operators (Section III-C), measuring the overall best fitness for each operator. To this end, we use all mutation operators, choose $\mu = 4$, and consider the following configurations using.

1) no crossover (`no_heur`).
2) Exhaustive Crossover (`heur-all`).
3) Greedy Crossover (`heur-greed`).
4) Randomized Greedy Crossover (`heur-greed-rand`).

Our results are depicted in Fig. 6. The advantage of crossover strongly depends on the scenario and none are significant.

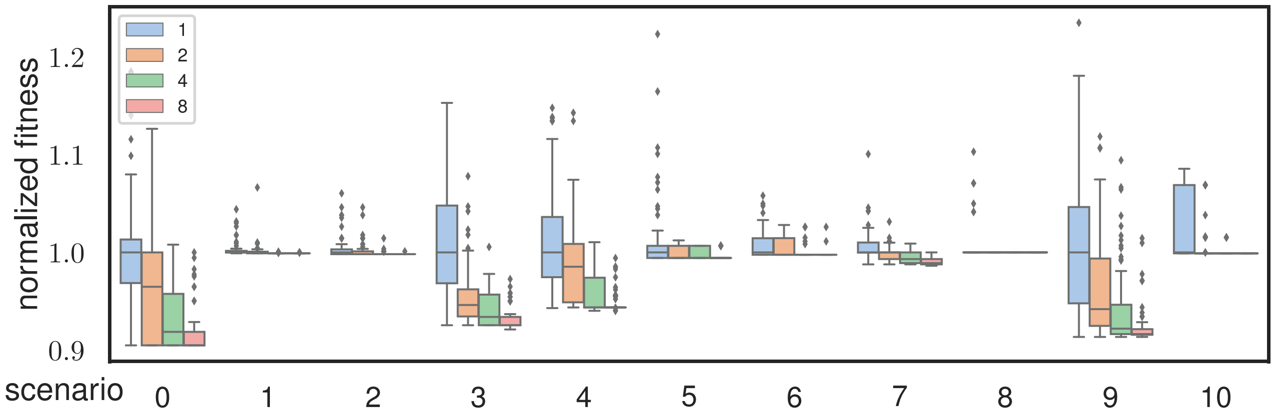

Fig. 5. Boxplots (Section IV-B) of the normalized best fitness of the MREA after 150 iterations for the 11 scenarios, with 75 runs per scenario. Each of the four colors, from left to right, represents a different population size μ. Per scenario, the fitness is normalized to the median of $\mu = 1$. In general, a higher population size seems more beneficial, but the gain is diminishing. Please refer to Section IV-D for more details.

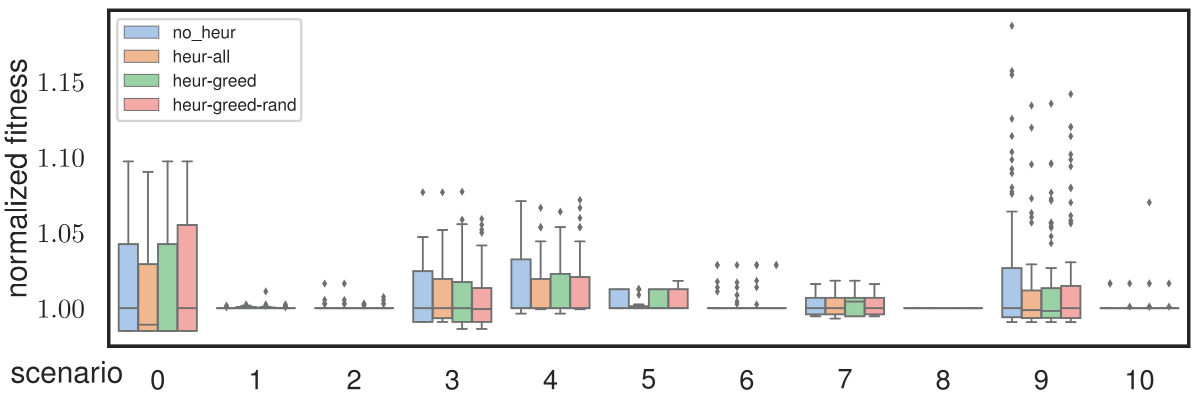

Fig. 6. Boxplots (Section IV-B) of the normalized best fitness of the MREA after 150 iterations for the 11 scenarios, with 75 runs per scenario. Each of the four colors, from left to right, represents one different crossover operator (including no crossover). Per scenario, the fitness is normalized to the median of `no_heur`. The configuration `heur-all` performs best, but only slightly. There is no clear difference between `heur-greed` and `heur-greed-rand`. In general, using crossover reduces the spread of the results. Please refer to Section IV-E for more details.

However, averaged over all 11 scenarios, the median best fitness as well as the spread is always reduced when using a crossover operator in comparison to using no crossover. This is also true for the minimum and maximum normalized fitness, highlighting the reduction of outliers. Interestingly, `heur-all` does not have a large benefit over the two greedy operators. Considering the two greedy strategies, on average, 19% of the `heur-greed` runs are better and 72% are better or equal to the median of `no_heur`; for `heur-greed-rand`, we get 18% and 74%, respectively. There is no clear tendency whether `heur-greed` or `heur-greed-rand` performs better.

Conclusion: In general, crossover improves the result quality of the MREA and reduces its spread. Among the different crossover operators, Exhaustive Crossover performs best but only slightly. Considering its high computation cost compared to the other two operators, it should not be chosen. Greedy Crossover and Randomized Greedy Crossover provide very good alternatives, each performing roughly equally well.

F. Analysis of the Selection Strategy

The MREA uses an elitist selection strategy, known as truncation selection (line 23). Such strategies get trapped in local optima, from which it can be hard to escape. In order to prevent this, a nonelitist strategy could be better. Thus, we exchange the elitist selection of the MREA (line 23) with a nonelitist strategy. To this end, we consider binary tournament selection with a tournament size of 2. This means that, instead of selecting the μ best individuals from the population consisting of parent individuals as well as offspring from mutation and potentially crossover, we repeat the following steps μ times, each time selecting an individual for the parent population of the next iteration: Choose two individuals uniformly at random (with replacement) and select the one with the better (that is, smaller) fitness. Note that the random selection of individuals does not guarantee that the best individuals are going to be part of the parent generation of the next iteration.

The results are depicted in Fig. 7. We note that the experiments return in both cases the best fitness found in any of the 150 iterations. For the elitist selection, an individual of this fitness is in the final population. However, for the tournament selection, such an individual could have been removed in a previous iteration, as the strategy is nonelitist. Still, the elitist selection outperforms the tournament selection with respect to both the mean fitness and the spread, except for scenarios 1 and 8, where they are tied. Interestingly, scenario 10 is also

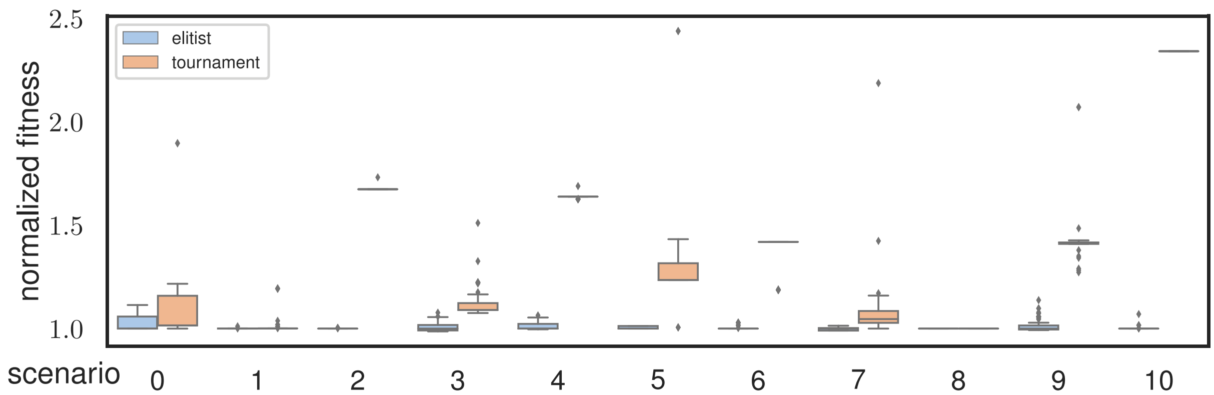

Fig. 7. Boxplots (Section IV-B) of the normalized best fitness of the MREA after 150 iterations for the 11 scenarios, with 75 runs per scenario. The MREA employs all mutation operators (Section III-B), the `heur-greed` crossover (Section III-C), and a population size of $\mu = 4$. For each scenario, the left box (in blue) refers to the MREA with (standard) elitist truncation selection (Algorithm 2), and the right box (orange) refers to the MREA using binary tournament selection. For both selection strategies, the best overall fitness is returned. Per scenario, the fitness is normalized to the median. The elitist selection is never worse than the tournament selection and usually clearly outperforms it. Please refer to Section IV-F for more details.

easily solved with the elitist selection, but the tournament selection struggles a lot and always returns the same (bad) fitness. This might indicate that, for this scenario, there is a local optimum which can be escaped via the mutation and crossover operators but only given sufficient time. The elitist selection strategy guarantees that there is always a currently best solution available to improve. For the tournament selection, such a solution can be removed, making it harder to overcome the local optimum via mutation and crossover.

Conclusion: The elitist selection of the MREA appears to be a reasonable choice that might be even well suited to escape local optima.

V. Application to the SAP Problem

We apply the MREA to the SAP problem [2] and empirically investigate its performance in terms of solution quality and runtime (Section V-B). The SAP problem is a special case of the MR problem that fixes a route between s and t and aims to find a *single* alternative route such that the overall travel time is minimized. Although the problem remains **NP**-hard, Bläsius et al. [2] proposed a highly specialized algorithm that solves it optimally—the SAP baseline (SAP-B), which we compare the MREA against.

As the SAP problem is a special case of the MR problem, the complexity of the MREA reduces in certain aspects. Further, we adjust the MREA using the insights from Section IV. We call the resulting algorithm the SAP-EA (Section V-A).

A. SAP-EA

The SAP-EA is a specialization of the MREA for the SAP problem with some modifications to its mutation operators. Since the SAP problem aims to find a single alternative route, an individual in the SAP-EA corresponds to a single route. Further, since determining a UE for the SAP problem simplifies to equalizing the cost functions of the given and the alternative route, which results in solving a quadratic equation, the SAP-EA does not use the Frank–Wolfe algorithm for fitness evaluation.

Regarding the operators from Sections III-B and III-C, the SAP-EA does not employ crossover, as these operators exchange existing routes, which is pointless for a single route. For the same reason, ExSegment is not used. Out of the remaining operators, NewRoute is used unmodified, and RandomP and LinkWP are combined into the new operator RandomPwD. This is due to our results from Section IV-C showing that LinkWP only provides a small benefit when added but still has its merits for certain scenarios. Last, the SAP-EA always performs exactly one mutation on each individual, using a parameter $p \in (0, 1)$ instead of operator weights. With probability p, NewRoute is performed, otherwise RandomPwD.

In the following, we explain the new operator RandomPwD and then compare the SAP-EA to the SAP-B.

1) RandomPwD: Similar to RandomP, given a route R of length m, RandomPwD replaces a segment of R between two nodes a and b that are k apart via RD. RandomPwD uses a parameter $\delta \in [0, 1]$. It determines $k \sim N(\delta \cdot m, (0.05 \cdot m)^2)$, rounding to the closest whole number, chooses a uniformly at random, and chooses b such that it is k nodes after a. If there are fewer than k nodes after a, then $b = t$.

RandomPwD adjusts δ according to two parameters $\alpha \in [0, 1]$ and $\beta \in \mathbf{N}_{>0}$ in the following way: whenever the SAP-EA does not improve for β iterations, we update $\delta \leftarrow \alpha \cdot \delta$.

2) Comparison to the SAP-B: Although the SAP-EA is specialized for the SAP problem, it is still a general heuristic applicable to different fitness functions. In contrast, the SAP-B is explicitly tailored to solving the SAP problem with monotone cost functions per edge, such as the flow of traffic, as in our setting. Thus, the SAP-B fails for other costs, for example, when optimizing for overall low CO_2 emissions of strategic drivers in a street network. In such a setting, the SAP-EA is still applicable without change.

B. Empirical Investigations

We compare the SAP-EA to the SAP-B on the street network of Berlin, Germany, with respect to best fitness as well as runtime. Recall that the SAP-B is an optimal algorithm. Thus, the SAP-EA cannot achieve a better best fitness.

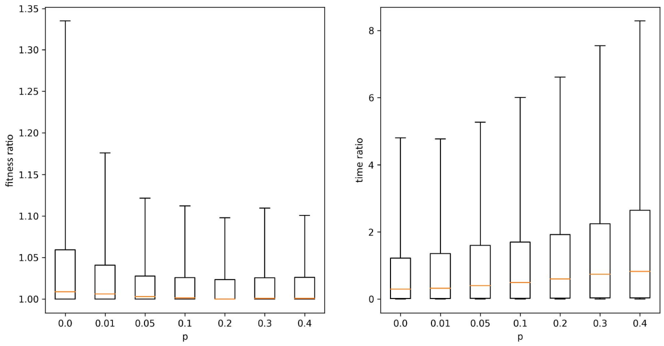

Fig. 8. Ratio of the best fitness (left) and the runtime (right) of the SAP-EA with $\mu = 1$, $\alpha = 0.4$, and $\beta = 35$, and different values of p compared to the SAP-B. The runtimes of SAP-B range from 0.3 s to 30 min, with better time ratios for higher SAP-B runtimes. Each boxplot contains the data of all 20 runs per value of k and per each of the 25 scenarios, totaling to 2000 points per box. The orange line depicts the median, the box the mid-50% of the data, and the whiskers the mid 95A higher value of p, i.e., an increased use of RandomPwD, yields generally better solutions and a smaller spread but also increases the runtime. A sweet spot seems to be around $p = 0.05$. Please also refer to Section V-B2.

In the following, we explain the setup and the evaluation of the experiments we carried out.

1) Experimental Setup: We use the same setup as in Section IV-B, with the following differences. We consider 25 scenarios chosen uniformly at random from the set of cluster centers of s–t pairs, computed by the BIRCH [36] algorithm. The clustering is based on real-world traffic density data provided by TomTom Germany. Per scenario, we choose $k \in \{500, 1000, 1500, 2000\}$ and $p \in \{0.0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4\}$, and we perform 20 runs per value of k and p. For the SAP-EA, we choose $\mu = 1$, and we terminate it after 1000 iterations or whenever it does not improve for 100 iterations. We used a machine with two Intel Xeon Gold 5118 CPUs and 64 GiB of memory.

2) Experimental Evaluation: Our results are depicted in Fig. 8. The maximum of all medians in the fitness ratio is 1.009, for $p = 0$, which is already very close to an optimal fitness. The median decreases up to $p = 0.2$ and increases afterward. Further, the spread is smallest for $p = 0.2$, making this configuration preferable. However, the runtime ratio increases for higher values of p both in median and spread, as RandomPwD is computationally more expensive than NewRoute. Since the configuration with $p = 0.05$ is very close to the best configuration, both in fitness and time, we deem it the best configuration out of all.

VI. Conclusion

We introduced and empirically analyzed the MULTIPLE-ROUTES EA, an evolutionary algorithm designed to suggest alternative routes for street networks with a high flow of traffic with the aim to reduce the overall travel time of all drivers. To this end, we introduced the **NP**-hard MULTIPLE-ROUTES problem, allowing for a precise modeling of our setting. For the MREA, we proposed four mutation and three crossover operators. We found that using all mutation operators yields the best results and that each crossover operator reduces the spread of the results. Last, we applied the MREA to a more specific setting of finding a single alternative route to a given route. We compared it to a highly specialized optimal algorithm and found that the MREA is capable of competing with the tailored algorithm while often being faster.

Overall, our results suggest that the MREA is well-suited for the highly complex problem of distributing traffic. For future work, we propose to extend the MREA to island models [37], a parallelization method well suited for EAs [38]. Another direction is to use data sets that measure other criteria, for example, the emission of cars. We believe that the MREA is well suited for such settings. Last, it would be interesting to see what the impact of the different operators is if one considers MULTIPLE-ROUTES with more than two routes. In this setting, the crossover operators become more expensive but might in turn reduce the spread in the final fitness more drastically. Also, the complexity of single routes might increase, which could affect the runtime of the mutation operators.

Acknowledgment

The authors thank TomTom Location Technology Germany GmbH for supplying them with the data necessary for their experiments. Further, they also thank Galassi et al. [34] for the GNU Scientific Library and Dagum and Menon [33] for OpenMP.

References

[1] M. Böther, L. Schiller, P. Fischbeck, L. Molitor, M. S. Krejca, and T. Friedrich, "Evolutionary minimization of traffic congestion," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2021, pp. 1–9.

[2] T. Bläsius et al., "A strategic routing framework and algorithms for computing alternative paths," in *Proc. Symp. Algorithmic Approaches Transp. Model. Optim. Syst. (ATMOS)*, 2020, pp. 1–14.

[3] N. Cohn, "The TomTom traffic index: An objective measure of urban traffic congestion." 2019. [Online]. Available: https://www.tomtom.com/blog/road-traffic/urban-traffic-congestion/

[4] M. A. van Essen, "The potential of social routing advice," Ph.D. dissertation, Fac. Eng. Technol., Univ. Twente, Enschede, The Netherlands, 2018.

[5] A. Kröller, F. Hüffner, L. Kosma, K. Kröller, and M. Zeni, "Driver expectations towards strategic routing," *Transp. Res. Rec.*, vol. 2675, no. 11, pp. 1–10, 2021.

[6] H. Bast et al., "Route planning in transportation networks," in *Algorithm Engineering: Selected Results and Surveys*. Cham, Switzerland: Springer, 2016.

[7] E. Köhler, R. H. Möhring, and M. Skutella, "Traffic networks and flows over time," in *Algorithms of Large and Complex Networks*. Heidelberg, Germany: Springer, 2009.

[8] D. Delling and D. Wagner, "Time-dependent route planning," in *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*. Heidelberg, Germany: Springer, 2009, pp. 207–230.

[9] U. Demiryurek, F. Banaei-Kashani, and C. Shahabi, "A case for time-dependent shortest path computation in spatial networks," in *Proc. Int. Conf. Adv. Geograph. Inf. Syst. (SIGSPATIAL)*, 2010, pp. 474–477.

[10] D. Delling, "Time-dependent SHARC-routing," *Algorithmica*, vol. 60, no. 1, pp. 60–94, 2009.

[11] G. Nannicini, D. Delling, D. Schultes, and L. Liberti, "Bidirectional A* search on time-dependent road networks," *Netw. Int. J.*, vol. 59, no. 2, pp. 240–251, 2011.

[12] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "Alternative routes in road networks," *J. Exp. Algorithmics*, vol. 18, pp. 1–17, Apr. 2013.

[13] A. Paraskevopoulos and C. D. Zaroliagis, "Improved alternative route planning," in *Proc. Symp. Algorithmic Approach. Transp. Model. Optim. Syst. (ATMOS)*, 2013, pp. 1–16.

[14] S. Zhu and D. Levinson, “Do people use the shortest path? An empirical test of Wardrop’s first principle,” *PLoS One*, vol. 10, no. 8, pp. 1–18, 2015.

[15] D. Simon, *Evolutionary Optimization Algorithms*. Chichester, U.K.: Wiley-Blackwell, 2013.

[16] K. Deb and C. Myburgh, “Breaking the billion-variable barrier in real-world optimization using a customized evolutionary algorithm,” in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2016, pp. 653–660.

[17] J. Berger and M. Barkaoui, “A hybrid genetic algorithm for the capacitated vehicle routing problem,” in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2003, pp. 646–656.

[18] J.-Y. Potvin, “State-of-the art review—Evolutionary algorithms for vehicle routing,” *INFORMS J. Comput.*, vol. 21, no. 4, pp. 518–548, 2009.

[19] P. Fischbeck. “Strategic routing GitHub repository.” 2021. [Online]. Available: https://github.com/MaxiBoether/strategic-routing

[20] T. Roughgarden and E. Tardos, “How bad is selfish routing?” *J. ACM*, vol. 49, no. 2, pp. 236–259, 2002.

[21] S. Fortune, J. Hopcroft, and J. Wyllie, “The directed subgraph homeomorphism problem,” *Theor. Comput. Sci.*, vol. 10, no. 2, pp. 111–121, 1980.

[22] J. G. Wardrop, “Some theoretical aspects of road traffic research,” *Proc. Inst. Civil Eng.*, vol. 1, no. 3, pp. 325–362, 1952.

[23] J. F. Nash, “Equilibrium points in n-person games,” *Proc. Nat. Acad. Sci.*, vol. 36, no. 1, pp. 48–49, 1950.

[24] M. J. Beckmann, C. B. MacGuire, and C. B. Winsten, *Studies in the Economics of Transportation*, Yale Univ. Press, New Haven, CT, USA, 1956.

[25] S. Dafermos, “Traffic equilibrium and variational inequalities,” *Transp. Sci.*, vol. 14, no. 1, pp. 42–54, 1980.

[26] M. Frank and P. Wolfe, “An algorithm for quadratic programming,” *Naval Res. Logist. Quart.*, vol. 3, nos. 1–2, pp. 95–110, 1956.

[27] M. Patriksson, “The Frank–Wolfe algorithm.” 2003. [Online]. Available: http://www.math.chalmers.se/Math/Grundutb/CTH/tma946/0203/fw_eng.pdf

[28] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[29] R. Bauer and D. Delling, “SHARC: Fast and robust unidirectional routing,” *J. Exp. Algorithmics*, vol. 14, p. 4, Jan. 2009.

[30] N. Aviram and Y. Shavitt, “Optimizing Dijkstra for real-world performance,” 2015, *arXiv:1505.05033*.

[31] H. B. Mann and D. R. Whitney, “On a test of whether one of two random variables is stochastically larger than the other,” *Ann. Math. Stat.*, vol. 18, no. 1, pp. 50–60, 1947.

[32] A. Horni, K. Nagel, and K. W. Axhausen, *The Multi-Agent Transport Simulation MATSim*. London, U.K.: Ubiquity Press, 2016.

[33] L. Dagum and R. Menon, “OpenMP: An industry standard API for shared memory programming,” *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, pp. 46–55, Jan./Mar. 1998.

[34] M. Galassi et al., *GNU Scientific Library Reference Manual*. Bristol, U.K.: Network Theory, 2019.

[35] *United States Bureau of Public Roads, Office of Planning, Urban Planning Division: Traffic Assignment Manual for Application With a Large, High Speed Computer*, Univ. Wisconsin, Madison, WI, USA, 1964.

[36] T. Zhang, R. Ramakrishnan, and M. Livny, “BIRCH: An efficient data clustering method for very large databases,” in *Proc. Int. Conf. Manag. Data (SIGMOD)*, 1996, pp. 103–114.

[37] J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. S. Richards, “Punctuated equilibria: A parallel genetic algorithm,” in *Proc. Int. Conf. Genet. Algorithms Appl.*, 1987, pp. 148–154.

[38] E. Alba, G. Luque, and S. Nesmachnow, “Parallel metaheuristics: Recent advances and new trends,” *Int. Trans. Oper. Res.*, vol. 20, no. 1, pp. 1–48, 2013.

Maximilian Böther received the M.Sc. degree in IT-systems engineering from the Hasso Plattner Institute, University of Potsdam, Potsdam, Germany, in 2022. He is currently pursuing the Ph.D. degree with the Systems Group, ETH Zurich, Zürich, Switzerland.

His research interests cover the intersection of data management and machine learning, and he is currently building a platform for model training on dynamically growing data sets.

Leon Schiller received the B.Sc. degree in IT-systems engineering from Hasso Plattner Institute, Potsdam, Germany, in 2020, where he is currently pursuing the M.Sc. degree in IT-systems engineering.

His research interests are in the area of probabilistic methods, centered around the analysis of random-graph models, randomized algorithms, and evolutionary optimization techniques.

Philipp Fischbeck received the M.Sc. degree in IT-systems engineering from the Hasso Plattner Institute, University of Potsdam, Potsdam, Germany, in 2018, where he is currently pursuing the Ph.D. degree with the Algorithm Engineering Group.

His research interests are random-graph models and how they can be used to predict and explain real-world graph algorithm behavior, and to build improved algorithms.

Louise Molitor received the M.Sc. degree in computer science from Martin Luther University Halle-Wittenberg, Halle, Germany, in 2017. She is currently pursuing the Ph.D. degree with the Algorithm Engineering Group, Hasso Plattner Institute, University of Potsdam, Potsdam, Germany.

Her research interests are in the area of algorithmic game theory, in particular, in strategic behavior in multiagent systems and game-theoretic analysis of models from residential segregation.

Martin S. Krejca received the Ph.D. degree in computer science from the Hasso Plattner Institute, University of Potsdam, Potsdam, Germany, in 2019.

Since 2022, he has been an Assistant Professor with École Polytechnique, Palaiseau, France. His research interests include the theoretical analysis of random processes, especially, black-box optimization heuristics.

Tobias Friedrich received the Ph.D. degree in computer science from Saarland University, Saarbrücken, Germany, in 2007.

From 2012 to 2015, he was a Full Professor and the Chair of Theoretical Computer Science with the University of Jena, Jena, Germany. Since 2015, he has been a Full Professor with the University of Potsdam, Potsdam, Germany, and the Head of the Algorithm Engineering Group, Hasso Plattner Institute, Potsdam. His current research interests include randomized methods in mathematics and computer science and randomized algorithms (both classical and evolutionary).