Evolutionary Minimization of Traffic Congestion

Maximilian Böther
Hasso Plattner Institute
University of Potsdam
maximilian.boether@student.hpi.de

Leon Schiller
Hasso Plattner Institute
University of Potsdam
leon.schiller@student.hpi.de

Philipp Fischbeck
Hasso Plattner Institute
University of Potsdam
philipp.fischbeck@hpi.de

Louise Molitor
Hasso Plattner Institute
University of Potsdam
louise.molitor@hpi.de

Martin S. Krejca*
Sorbonne University, CNRS, LIP6
Paris, France
martin.krejca@lip6.fr

Tobias Friedrich
Hasso Plattner Institute
University of Potsdam
tobias.friedrich@hpi.de

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 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 Multiple-Routes 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, improving 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. [ATMOS 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.

CCS CONCEPTS

• **Computing methodologies → Heuristic function construction**; *Randomized search.*

KEYWORDS

Strategic routing, traffic congestion, optimization, evolutionary algorithm

ACM Reference Format:
Maximilian Böther, Leon Schiller, Philipp Fischbeck, Louise Molitor, Martin S. Krejca, and Tobias Friedrich. 2021. Evolutionary Minimization of Traffic Congestion. In *2021 Genetic and Evolutionary Computation Conference (GECCO '21), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3449639.3459307

*This work was supported by the Paris Île-de-France Region.

1 INTRODUCTION

Traffic congestion is an increasing problem for urban areas across the world [10]. 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 infeasible, 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 sub-optimal options into consideration [35] – a cost that some drivers are willing to take [24]. We refer to this setting as *strategic* routing.

A well-studied domain that meets some of these requirements is route planning [4]. Most results consider a time component of each route, e.g., by considering flow over time [25] or predicted congestion [16–18, 27], or they consider multiple routes, where the alternative route needs to be substantially different [1, 29]. 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 [9], 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 paper, we naturally extend the SAP problem to the more general Multiple-Routes (MR) 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 [38], we assume they form a user equilibrium 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 Multiple-Routes 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, 33], including non-strategic routing problems, e.g., the Vehicle Routing Problem [7, 31]. 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 1) 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 improves 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 as well as no crossover) already improves the solution by factors between 1.5 and 2.8. Further, 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. [9]. We find (see Figure 5) that the best configuration of the MREA, in the median, achieves a solution quality of at least 99.69 % but only requires 40 % of the run time. Overall, our results suggest that the MREA is a heuristic well-suited for solving the MR problem and thus reducing traffic congestion in strategic scenarios.

The remainder of this paper is organized as follows. In Section 2, we formalize the MR problem, and in Section 3, we introduce the MREA. We then analyze the performance of the MREA and the effect of its operators and population size in Section 4. In Section 5, we apply the MREA to the SAP problem and compare it against the algorithm of Bläsius et al. [9]. Finally, we conclude our work in Section 6. For supplementary material, we refer to our repository [8].

2 THE 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 such that no driver can choose a quicker route as long as no other driver cooperatively changes their route as well and the total number of different routes exceeds n. We call such a state an *n-restricted user equilibrium* (n-UE). The Multiple-Routes (MR) problem aims to find an optimal set of n routes such that the overall travel time of drivers in an n-UE is minimized.

2.1 Problem Modeling

We follow the formalization by Roughgarden and Tardos [32] 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 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 (see Section 2.2), we call the traffic flow *stable.*

The travel time of drivers on an edge $e \in E$ is determined by a latency function $\tau_e\colon \mathbf{R}_{\geq 0} \to \mathbf{R}_{\geq 0}$. 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. We assume τ_e to be monotonically increasing and continuous. For a traffic flow f, the flow f_e over e is then $\sum_{P \in \mathcal{P}_{s,t}: e \in P} f(P)$, 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 $C(f)$ that denotes the overall travel time of all drivers. Formally,

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

The MR problem aims to find a valid and stable traffic flow with minimum cost among all valid and stable traffic flows. We show in the supplementary material [8] that the MR problem is **NP**-hard.

2.2 The User Equilibrium

In routing games, a *user equilibrium* (UE), also known as *Wardrop equilibrium* [36], is a game state where no player has anything to gain by changing only their own strategy [28]. 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 [6, 13, 32]. However, note that 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, this means that 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 [6], which we approximately solve with the Frank–Wolfe algorithm [21], adjusted such that it makes sure to satisfy the constraint of at most n routes. The supplementary material [8] contains a detailed description of the algorithm.

3 THE MULTIPLE-ROUTES EA

The MREA (Algorithm 1) 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, and each of which is evaluated as described in Section 2.2. The MREA generates offspring in two different (and exclusive) ways: by **(1)** expanding the current population via a crossover operation provided as an input and **(2)** by employing a random number of mutation operators to a copy of each individual. Then, the MREA reduces the population size to μ via truncation selection, breaking ties uniformly at random. 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 it contains a strict improvement over the parent population. 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 is used, which finds a shortest path on G with randomly perturbed edge weights.

3.1 RandDijkstra

The RandDijkstra (RD) is a randomized variant of Dijkstra's shortest-path algorithm [19]. 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 $w \sim \mathrm{N}(\tau_e(x), 0.8 \cdot \tau_e(x))$, where τ_e is the latency of e and where x is the traffic flow routed from s to t. Due to its extensive use, RD contributes the most to the run time of the MREA.

3.2 Mutation Operators

In total, the MREA has four different mutation operators: *NewRoute*, *RandomP*, *LinkWP*, and *ExSegment*, each with its own weight. When

Algorithm 1: The 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 $\mu$,
       crossover strategy $cStra$, termination criterion
Output: Set of $n$ routes from $s$ to $t$
$P \leftarrow \emptyset$;
repeat $\mu$ times
    $ind \leftarrow$ new individual;
    repeat $n$ times $ind$.addRoute(RandDijkstra($s, t, k$));
    $P$.add($ind$);
while termination criterion not met do
    $C \leftarrow \emptyset$;
    repeat $\sqrt{\mu^2 - \mu/2}$ times
        $ind_1$, $ind_2 \leftarrow$ two uniformly at random chosen
          individuals in $P$;
        $C$.add($cStra$($ind_1$, $ind_2$));
    $P' \leftarrow$ copy of $P$;
    for every individual $ind$ in $P'$ do
        $mutations \leftarrow \max(1, \text{Pois}(1.5))$;
        $ops \leftarrow \emptyset$;
        repeat $mutations$ times
            $ops$.add(randomly weighted selected operator in
              {NewRoute, RandomP, LinkWP, ExSegment});
        if $ops$ contains ExSegment then
            $ops \leftarrow$ {ExSegment};
        apply operators in $ops$ to $ind$;
    if no individual in $C$ is better than the best in $P$ then
        $C \leftarrow \emptyset$;
    $P \leftarrow$ the $\mu$ best individuals in $C \cup P' \cup P$;
return the best individual in $P$;
```

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. Afterward, for each mutation to apply, a mutation operator is chosen randomly proportionally to its weight. If ExSegment is chosen, then all other operators are discarded for this mutation. Last, all chosen operators are applied to the individual.

3.2.1 NewRoute. The operator 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 10 iterations, and then lowered linearly such that it reaches 1 in iteration 200.

3.2.2 RandomP. The operator replaces subsegments of a randomly selected subset of routes via RD. The routes to be modified are chosen randomly proportionally to their inverse traffic flow. For each such route, it chooses a start node uniformly at random and a destination node by advancing a number of steps according to the Gaussian distribution $N(0.25r, 0.5r)$, where r is the length of the route. Then, RD is applied to replace the route segment between these two nodes. As the operator should 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. RandomP has a constant weight of 60.

3.2.3 LinkWP. The operator is identical to RandomP except for the choice of delimiting nodes of the subsegment to replace, which are chosen more specifically to the MR instance. For each node v on a route in which a subsegment should be replaced, RandomP 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 the metric. Then, the destination node is chosen randomly by selecting one of the nodes on the original route that comes after the start node, weighted using 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.

3.2.4 ExSegment. The operator swaps subsegments between two routes of the same individual. First, it chooses a pair of different routes from the route set uniformly at random. Then, all cycles are removed from both routes. Afterward, the nodes occurring in both routes are determined, which we call *shared points*. Among the shared points, let the *divergence points* be the nodes who have a different successor in both routes. Further, let the *goto points* be those shared points who have a different predecessor in both routes. ExSegment chooses one divergence point v_s uniformly at random. Then, it chooses a node v_t uniformly at random from the set of all goto points that appear after v_s. The route segments between v_s and v_t from both chosen routes are then swapped.

The weight of ExSegment is determined dynamically. If ExSegment was applied within the last 6 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 6 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 4 for more details).

3.3 Crossover Operators

We present three different binary crossover operators. Note, in contrast to the mutation operators, the MREA only uses a single crossover operator, specified by the input. This is due to there being a large tradeoff between run time 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 score D that reflects how similar the routes of an individual are. The assumption is that

Table 1: The median best fitness (lower is better) of the 75 runs (see Section 4.1) for each of the settings from Sections 4.2 to 4.4 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 (i.e., with entries further to the right, for the columns Mutation Operators and Population Size). The column Crossover Operators shows that the choice of the crossover operator has almost no impact regarding 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 [26] with a p-value of 0.05. We refer to the respective sections for more information.

		Mutation Operators (see Section 4.2)				Population Size μ (see Section 4.3)				Crossover Operators (see Section 4.4)			
ID Scenario	k-Dijkstra	`rponly`	`wnewroute`	`wlinkp`	`wexseg`	$\mu = 1$	$\mu = 2$	$\mu = 4$	$\mu = 8$	`no_heur`	`heur-all`	`heur-greed`	`heur-greed-rand`
0 Babelsberg – Lichterfelde	153530661	68566307	**64240948**	64662372	64240948	64240948	**61980563**	**59016819**	**58128546**	59016819	58369109	58128546	59016819
1 Griebnitzsee – Ahrensfelde	186995924	90878267	**74610652**	74610652	74610652	74610652	**74602749**	**74541545**	**74541545**	74541545	74541545	74541545	74541545
2 KaDeWe – East Side Gallery	28713342	12364817	12292827	12237029	**12223856**	12223856	**12201922**	**12201922**	**12201922**	12201922	12201922	12201922	12201922
3 Lichterfelde – Prenzlauer Berg	92721673	58477361	**50558429**	51197745	**52541858**	52541858	**49713176**	**49071681**	**48625397**	49071681	49071681	49071681	49044459
4 Lichterfelde – Steglitz	86902859	50225754	**44211386**	**44910249**	44188815	44188815	**43540844**	**41699314**	**41699314**	41699314	41699314	41699314	41699314
5 Moabit – Birkenwerder	103023491	38395379	**37639244**	37344703	37387321	37387321	37174180	**37174180**	**37174180**	37174180	37174180	37174180	37174180
6 Olympiastadion – Rotes Rathaus	101643446	36504804	**36353121**	36266833	36347074	36347074	36259504	**36259504**	36259504	36259504	36259504	36259504	36259504
7 Potsdamer Platz – Pergamonmuseum	40930113	18904046	**18764355**	18764355	18764355	18764355	**18764355**	**18636335**	**18558820**	18636335	18636335	18716682	18636335
8 Potsdamer Platz – Tempelhofer Feld	12014974	6860945	**6407381**	6407381	6407381	6407381	6407381	6407381	6407381	6407381	6407381	6407381	6407381
9 Teltow – Hoppegarten	394317060	160818584	**141236603**	140966107	142441589	142441589	**134170060**	**131325658**	**130535412**	131325658	131157891	131088667	131325658
10 Wannsee – Schönefeld	57461353	24477839	**22899049**	22876414	22899049	22899049	**22876414**	**22876414**	**22876414**	22876414	22876414	22876414	22876414

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:

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

3.3.1 Exhaustive Crossover. The operator considers *all* $\binom{2n}{n}$ route sets possible from the routes of the two parents and returns the combination with the highest diversity score.

3.3.2 Greedy Crossover. The operator greedily constructs a new route set, guided by the metric D. It randomly chooses one of the $2n$ routes of the parents, proportionally to their inverse traffic flow. Each of the following $n - 1$ routes is chosen such that it optimizes the diversity metric of the thereby created partial route set.

3.3.3 Randomized Greedy Crossover. This operator takes the same approach as Greedy Crossover, but instead of greedily choosing the route that maximizes the diversity of the route set, it randomly selects one of the $2n$ routes, 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.

4 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 4.2, we begin by evaluating the mutation operators, excluding crossover. In Section 4.3, we analyze the impact of the population size μ. Last, in Section 4.4, we add crossover to the MREA, and we compare the quality achieved by the three different crossover operators with each other.

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 mutation operators RandomP and NewRoute. Nonetheless, adding more operators generally decreases the spread of the results, in addition to improving them. Table 1 shows a summary of the median best fitness of all our parameter settings.

4.1 Experimental Setup

We consider MR instances with the graph G being the street network of Berlin, Germany, provided by TomTom Germany, and with $k = 3\,000$, which is a reasonable choice [9]. Further, 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* (see also Table 1):

(0) outside the inner-city; country road, non-obvious deviation
(1) very long; fastest route uses Autobahn (AB; express highway), second fastest route goes through the inner-city
(2) short, inner-city; lots of possible detours
(3) long, south to north; covering AB and inner-city side streets
(4) very short, inner-city; direct route is using side streets, but highway and AB are nearby
(5) long, start in the city center; choice for highway or AB
(6) long, inner-city; can be driven almost entirely on a highway
(7) short, inner-city; different highways or side streets that are reasonable, in a Manhattan-like layout

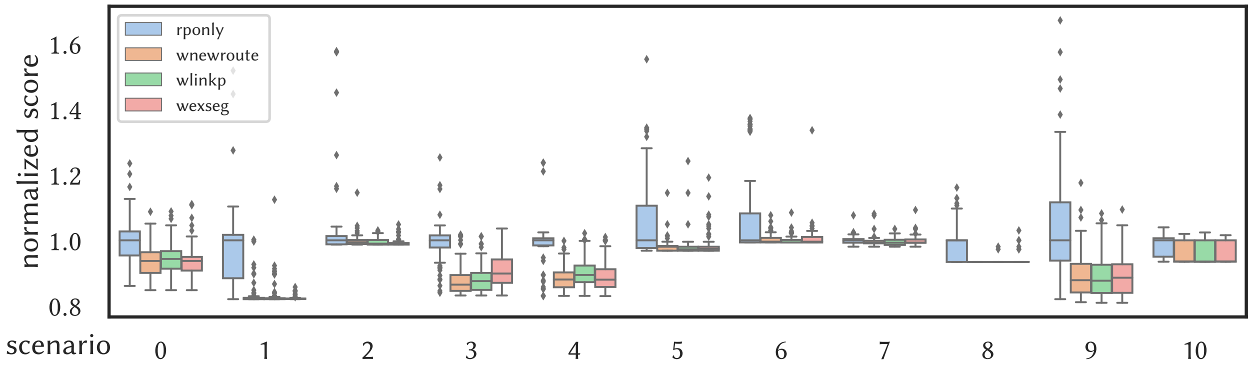


Figure 1: Boxplots (see Section 4.1) of the normalized best fitness of the MREA with $\mu = 1$ after 150 iterations for the 11 scenarios, with 75 runs per scenario. Each of the four colors, from left to right, represents one of the algorithm configurations explained in Section 4.2. Per scenario, the 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 4.2.1 for more details.

(8) medium-long, inner-city; bottleneck at a bridge, but opportunity to split up onto two highways
(9) long, south-west to east; either long detour using AB or a more direct inner-city highway
(10) route where the k-Dijkstra shortest route detours to use AB

For the latency functions, we follow the recommendation of the US Bureau of Public Roads [34], 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 [9].

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 3.2.4) is chosen such that it reaches a value of 30 if there was no improvement in the last $20\,\% \cdot 150 = 30$ iterations. Last, we start 75 independent runs of the MREA on all 11 scenarios per setting. We refer to our repository [8] for details.

Boxplots. When visualizing our results with 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.

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.

4.2 Analysis of the Mutation Operators

We analyze the utility of the MREA's four mutation operators (Section 3.2) by considering how well each operator improves the fitness of the overall best solution (Section 4.2.1) and how quickly the algorithm finds such a solution (Section 4.2.2). 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 mutation only. Further, 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`).

4.2.1 *Best overall fitness.*

We study the impact of each configuration on the best fitness achieved after our termination criterion of 150 iterations. Our results are depicted in Figure 1.

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 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 as well as 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.

4.2.2 *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 that the MREA changed the fitness of its best individual. The results are depicted in Figure 2. 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 4.2.1, which show that, on average, configurations with more operators have a better median performance.

The speed of convergence is dependent on the scenario, and there is no clear trend among the four configurations. The mid-90 %

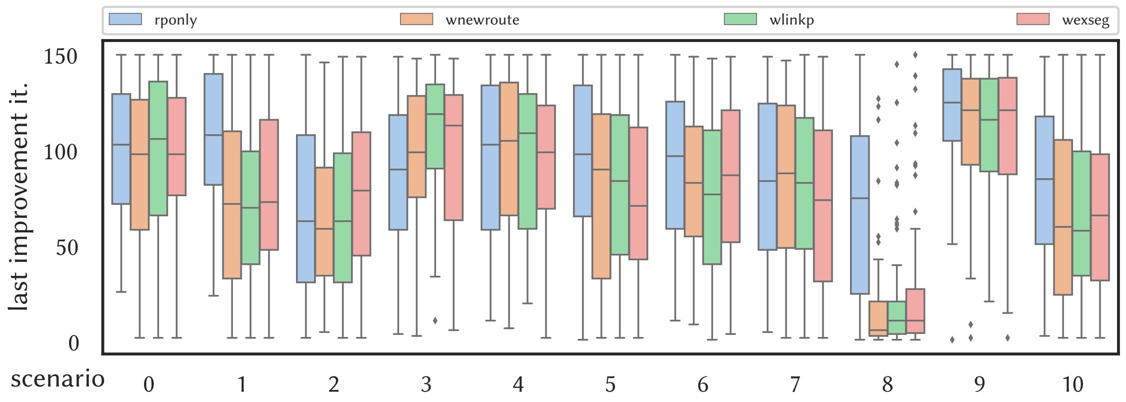


Figure 2: Boxplots (see Section 4.1) of the last of, in total, 150 iterations in which the MREA with $\mu = 1$ improved its best fitness, for the 11 scenarios. Each of the four colors, from left to right, represents one of the algorithm configurations explained in Section 4.2, and each configuration 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 4.2.2 for more details.

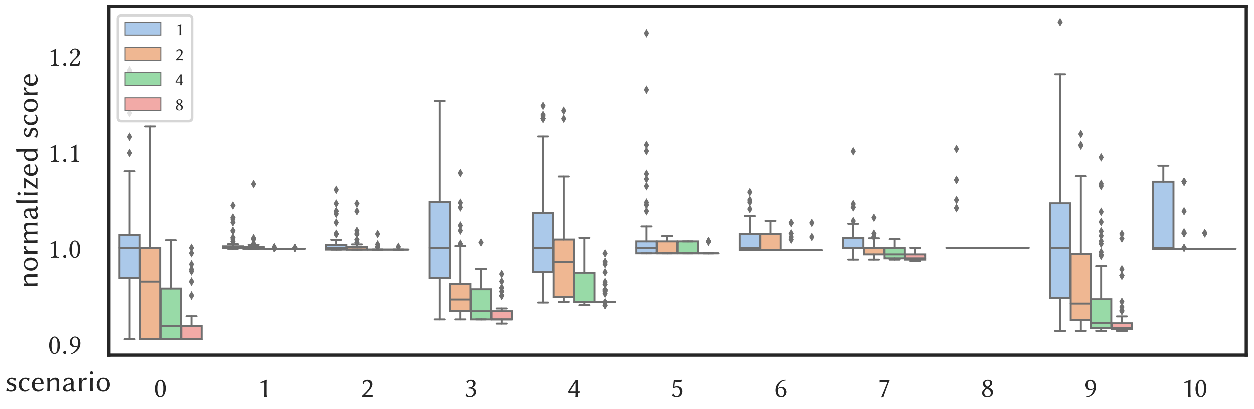


Figure 3: Boxplots (see Section 4.1) of the normalized best fitness of the MREA after 150 iterations for the 11 scenarios, with 75 runs per scenario. Each of the four colors, from left to right, represents a different population size μ. Per scenario, the fitness is normalized to the median of $\mu = 1$. In general, a higher population size seems more beneficial, but the gain is diminishing. Please refer to Section 4.3 for more details.

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 the MREA to converge.

4.2.3 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 using RandomP.

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.

4.3 Analysis of the Population Size

We analyze to what extent the MREA benefits from having a population size larger than 1. Since Section 4.2 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 Figure 3. 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. Although the impact of a larger population size is different among the scenarios, our results suggest that the improvement for $\mu = 2$ and $\mu = 4$ provide a large improvement over

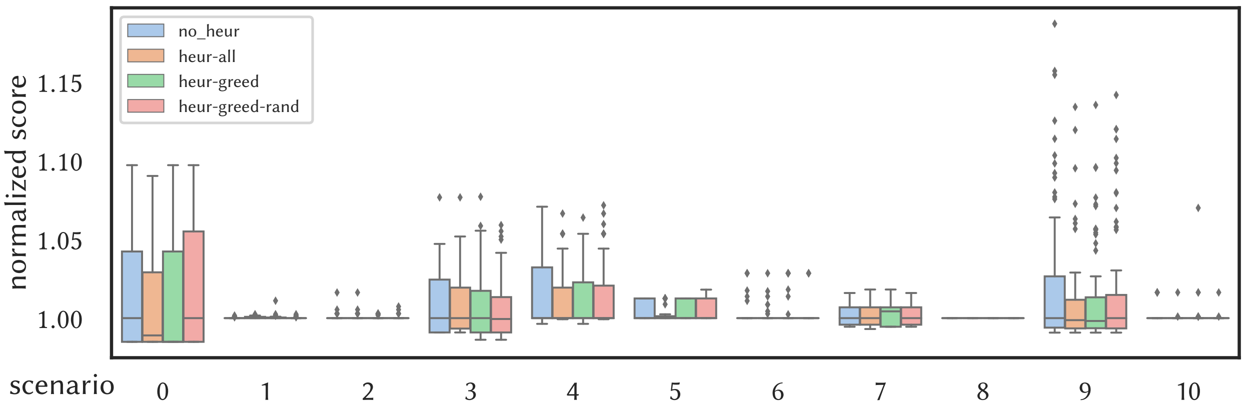


Figure 4: Boxplots (see Section 4.1) of the normalized best fitness of the MREA after 150 iterations for the 11 scenarios, with 75 runs per scenario. Each of the four colors, from left to right, represents one different crossover operator (including no crossover). Per scenario, the fitness is normalized 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 4.4 for more details.

$\mu = 1$. 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 runs 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 %.

4.3.1 Conclusion. Using a larger population size improves the quality of the best fitness and also decreases the spread among the different runs per scenario. However, the computational effort increases with the number of individuals, and the quality gain in fitness from using more individuals varies among the different configurations. Our experiments suggest a sweet spot at 4 individuals.

4.4 Analysis of the Crossover Operators

We analyze the utility of the MREA's three crossover operators (Section 3.3), measuring the overall best fitness for each operator. To this end, we use all mutation operators, choose $\mu = 4$, and consider the following configurations:

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

Our results are depicted in Figure 4. The advantage of crossover strongly depends on the scenario and none are significant. 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.

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

5 APPLICATION TO THE SAP PROBLEM

We apply the MREA to the SINGLE-ALTERNATIVE-PATH (SAP) problem [9] and empirically investigate its performance in terms of solution quality and run time. 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. [9] propose a highly specialized algorithm that solves it optimally – the SAP baseline (SAP-B) –, which we compare the MREA against.

Since the SAP problem is a special case of the MR problem, the complexity of the MREA reduces in certain aspects. In addition, we use the insights gained from Section 4 to further improve the MREA. We call the resulting algorithm the SAP-EA.

5.1 The 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 user equilibrium 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 proposed in Sections 3.2 and 3.3, 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 4.2 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.

5.1.1 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 RandDijkstra. To this end, RandomPwD uses a parameter $\delta \in [0, 1]$, which it automatically adjusts. It determines $k \sim \mathrm{N}(\delta \cdot m, (0.05 \cdot m)^2)$, 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 $\delta \longleftarrow \alpha \cdot \delta$.

5.1.2 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.

5.2 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 run time. Recall that the SAP-B is an optimal algorithm. Thus, the SAP-EA cannot outperform it in regard to best fitness.

5.2.1 Experimental Setup. We use the same setup as in Section 4.1, 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 [37] 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. The SAP-EA terminates after 1000 iterations or whenever it does not improve for 100 iterations. Last, we choose $\mu = 1$. We used a machine with two Intel Xeon Gold 5118 CPUs and 64 GiB of memory.

5.2.2 Experimental Evaluation. Our results are depicted in Figure 5. The maximum of all medians in the score 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 run time ratio increases for higher values of p both in median and spread, as RandomPwD is computationally more expensive

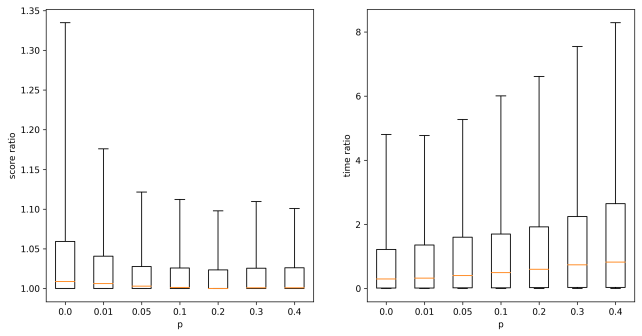


Figure 5: The ratio of the best fitness (left) and the run time (right) of the SAP-EA with $\mu = 1$, $\alpha = 0.4$, $\beta = 35$, and different values of p compared to the SAP-B. The run times of SAP-B range from 0.3 seconds to 30 minutes, with better time ratios for higher SAP-B run times. Each boxplot 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 95 %. A higher value of p, i.e., an increased use of RandomPwD, yields generally better solutions and a smaller spread but also increases the run time. A sweet spot seems to be around $p = 0.05$. Please also refer to Section 5.2.2.

than NewRoute. Since the configuration with $p = 0.05$ is very close to the best configuration both in score and time, we deem it the best configuration out of all.

6 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, and we showed that using all mutation operators yields the best results. Further, each crossover operator reduces the spread of the results. Last, we applied the MREA to a more specific setting, aiming to find a single alternative route to a given route, and compared it to a highly specialized optimal algorithm. Although the MREA is more general, it 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 [11], a parallelization method well suited for EAs [2]. 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.

ACKNOWLEDGMENTS

We thank TomTom Location Technology Germany GmbH for supplying us with the data necessary for our experiments. Further, we thank Galassi et al. [22] for the GNU Scientific Library and Dagum and Menon [14] for OpenMP.

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

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
1 $q \leftarrow 0$;
2 while not converged do
3     Find $p_q$ minimizing the following linear program
                  minimize    $p_q^T \nabla f(x_q)$
                  subject to  $p_q \in \mathcal{D}$
4     $\gamma \leftarrow$ Line-Search Determination of step size;
5     $x_{q+1} \leftarrow x_q + \gamma(p_q - x_q)$;
6     $q \leftarrow q + 1$;
7 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).

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

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

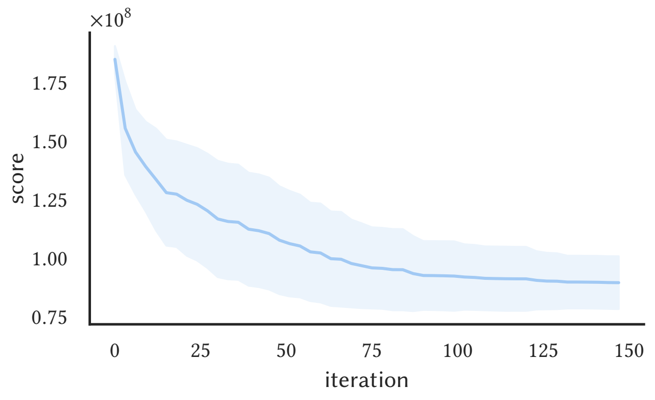
$$\tau_e(x) = \begin{cases} 0, & \text{if } x \leq 2; \\ x - 2, & \text{else.} \end{cases}$$

In a setting withn $n = 2$, 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. □



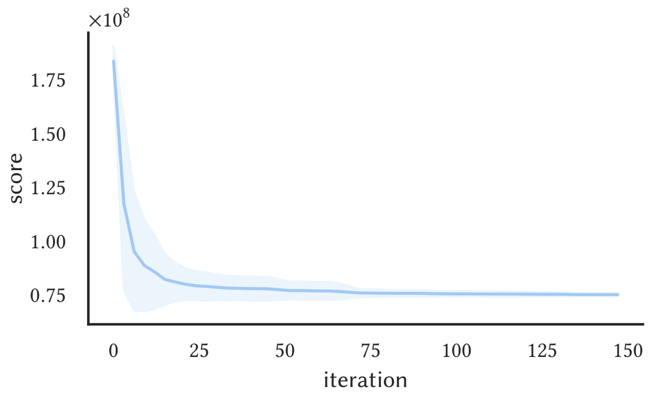


Figure 6: Fitness curves which show the mean absolute score (bold line) as well as the standard deviation (colored band) per iteration, for scenario 1. In the upper curve, the `rponly` setting is considered, while the lower curve uses the `wexseg` configuration.

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].