SAND2020-4103C

Probing a Set of Trajectories to Maximize Captured Information

Sándor P. Fekete
Department of Computer Science, TU Braunschweig, Germany
s.fekete@tu-bs.de

Alexander Hill
Department of Computer Science, TU Braunschweig, Germany
a.hill@tu-bs.de

Dominik Krupke
Department of Computer Science, TU Braunschweig, Germany
d.krupke@tu-bs.de

Tyler Mayer
Decision Management Systems, Charles River Analytics Inc., Boston, USA.
tmayer@cra.com

Joseph S. B. Mitchell
Department of Applied Mathematics and Statistics, Stony Brook University, USA.
joseph.mitchell@stonybrook.edu

Ojas Parekh
Sandia National Laboratories, USA.
odparek@sandia.gov

Cynthia A. Phillips
Sandia National Laboratories, USA
caphill@sandia.gov

Abstract

We study a trajectory analysis problem we call the Trajectory Capture Problem (TCP), in which, for a given input set $\mathcal{T}$ of trajectories in the plane, and an integer $k \geq 2$, we seek to compute a set of k points ("portals") to maximize the total weight of all subtrajectories of $\mathcal{T}$ between pairs of portals. This problem naturally arises in trajectory analysis and summarization.

We show that the TCP is NP-hard (even in very special cases) and give some first approximation results. Our main focus is on attacking the TCP with practical algorithm-engineering approaches, including integer linear programming (to solve instances to provable optimality) and local search methods. We study the integrality gap arising from such approaches. We analyze our methods on different classes of data, including benchmark instances that we generate. Our goal is to understand the best performing heuristics, based on both solution time and solution quality. We demonstrate that we are able to compute provably optimal solutions for real-world instances.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Algorithm engineering, optimization, complexity, approximation, trajectories

Digital Object Identifier 10.4230/LIPIcs.SEA.2020.2

Related Version A full version of the paper is available at `https://arxiv.org/abs/2004.03486`arXiv:2004.03486 [6].

18th International Symposium on Experimental Algorithms (SEA 2020).
Editors: Simone Faro and Domenico Cantone; Article No. 2; pp. 2:1–2:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In recent years, the progress in technical capabilities has resulted in massive amounts of trajectory data for cars, trucks, trains, aircraft, ships, people, and animals being collected at increasing rates. This presents major challenges for storing and evaluating this ever-growing data, as well as for extracting useful information; this motivates the search for data structures and algorithms that capture some of the most important and useful aspects of such trajectories. At the same time, the availability of large volumes of data makes it possible to consider useful aspects that were previously unavailable due to the lack of data or algorithmic evaluation methods, such as collecting useful information along the traveled trajectories.

One such means of analyzing a set $\mathcal{T}$ of trajectories is to determine "popular pairs" (p_1, p_2) of locations (or location/time pairs) for which there is a significant "value" of the trajectories $\mathcal{T}$ going between those points. This value can arise from aggregated data between checkpoints, such as the total passenger-distance or the accumulated total pollution along the way; it also comes up through the use of tomographic methods (i.e., determining physical phenomena by measuring aggregated effects along the path between two sensors), which are highly important in the context of many other application areas, such as in astrophysics [10]. A limiting factor is usually the need to pick a set of locations of finite cardinality, i.e, placing a limited number of toll booths, cameras for average speed measurement, various other types of sensors, or abstract collections of focus points for sampling trajectories. For an example, consider the scenario shown in Figure 1, which corresponds to more than 500,000 data points that arise from the trajectories of over 250 taxi cabs in San Francisco. Our goal is to identify a small subset of locations that allow us to capture as much of the movement information, in terms of weighted distance between checkpoints, as possible.

Figure 1 A set of taxi trajectories after preprocessing in San Francisco Bay Area. (Satellite images courtesy of Planet Labs Inc.)

In this paper, we study the Trajectory Capturing Problem (TCP), a generalization of "popular pair" computation: Given a set $\mathcal{T}$ of trajectories and an integer $k \geq 2$, determine a set of k *portals* (points) to maximize the sum of the weights of the inter-portal subtrajectories in $\mathcal{T}$; such subtrajectories are said to be "captured" by the set of portals. After establishing that the TCP is NP-hard, and giving some first approximation bounds, we focus on algorithm engineering methods for attacking the problem practically.

1.1 Our Results

We provide results both for algorithmic theory and algorithm engineering aspects of the Trajectory Capturing Problem.

- We prove NP-hardness for the TCP, even for instances with trajectories consisting of individual axis-parallel segments.
- We establish two approximation algorithms. One has approximation factor K if the input trajectory set decomposes into K subsets where within each subset no two segments cross, though they can overlap, (K noncrossing subsets). The other has an approximation factor Δ, the maximum number of input trajectories hit by any single point.
- We develop an Integer Linear Program (IP) to solve TCP instances to provable optimality.
- We show that, in general, the IP formulation has unbounded integrality gap[1]. For inputs decomposing into 2 noncrossing subsets (e.g., arising from axis-parallel segments), we show that the integrality gap is at most $\frac{k}{\lceil k/2 \rceil}$ (Theorem 7).
- We develop methods for generating challenging benchmark instances for experimentation. For geometric instances, based on segment arrangements, this requires care to address geometric robustness and accuracy.
- We compute provably optimal solutions for general instances up to thousands of candidate capture points.
- We give provably optimal solutions for even larger instances, with up to 7000 possible capture points, for instances based on axis-parallel segments, where we find the integrality gap to be quite small.
- Using the IP solutions as a reference, we perform a thorough computational study using heuristic algorithms (a greedy algorithm, iterated local search, simulated annealing, and an evolutionary algorithm), with various settings, to understand how heuristics perform on various instances, in terms of time and solution quality.
- We demonstrate our methods on real-world instances, including a provably optimal solution for taxi-cab data on 250 trajectories, with more than 500,000 individual geographic data points.

Given the broad range of potential applications, scenarios and assumptions, we do not claim (or even aim) to provide a final set of methods for the general problem. Instead, we focus on demonstrating that a number of modeling and optimization approaches can provide a promising range of insights and tools for future work from various directions.

1.2 Related Work

Related to our problem is the well-studied Geometric Hitting Set Problem (GHS), in which one seeks a smallest cardinality set of points to hit a given set of lines, segments, or trajectories in the plane; as a single point suffices to capture all of a trajectory it lies on, achieving large objective values for the GHS is easier than for the TCP. The GHS is known to be NP-hard, hard to approximate (below a threshold), and some natural geometric cases have constant-factor approximation algorithms; see, e.g., [5], and the references therein.

There is a vast literature on problems of analyzing, clustering, mining, and summarizing a set of trajectories. For an extensive survey of trajectory data mining methods, see Zheng [22]

[1] The integrality gap is $\max_{\mathcal{I}} \frac{\mathrm{LP}(\mathcal{I})}{\mathrm{IP}(\mathcal{I})}$, where $\mathcal{I}$ is a TCP instance, $\mathrm{IP}(\mathcal{I})$ is the optimal IP solution value and $\mathrm{LP}(\mathcal{I})$ is the optimal solution value to the linear programming (LP) relaxation.

and Zheng and Zhou [23]. Notions of "flocks" and "meetings" have been formalized and studied algorithmically [2, 7, 11, 21]. Gudmundsson, van Kreveld, and Speckmann [8] define *leadership*, *convergence*, and *encounter* and provide exact and approximate algorithms to compute each. Andersson, Gudmundsson, Laube, and Wolle [1] show that several *Leader-Problem* (LP) variants (*LP-Report-All, LP-max-Length, LP-Max-Size*) are all polynomially solvable and provide exact algorithms. Buchin et al. [3] present a framework to fully categorize trajectory grouping structures (grouping, merging, splitting, and ending of groups). Other approaches to trajectory summarization naturally include cluster analysis, of which there is a large body of related work. Li, Han, and Yang [14] consider rectilinear trajectories and show how to cluster with bounding rectangles of a given size. Several approaches (e.g., [9, 12, 13, 17]) consider density-based methods for clustering sub-trajectories. Lee, Han, Li, and Gonzalez [12] take it one step further by considering a two-level clustering hierarchy that first accounts for regional density and then considers lower-level movement patterns. Li, Ding, Han, and Kays [15] consider a problem (related to [8]) in which they seek to identify all *swarms* or groups of entities moving within an arbitrary shaped cluster for a certain, possibly disconnected, duration of time. Also, Uddin, Ravishankar, and Tsotras [20] consider finding what they call *regions of interest* in a trajectory database.

In motivating tomographic applications, the number of checkpoints is an important constraint in the use of discrete tomography, e.g., in astrophysics (Korth et al. [10]).

2 Preliminaries

We are given a set of trajectories $\mathcal{T}$, each specified by a sequence of points, e.g., in the Euclidean plane. We seek a set $P = \{p_1, \ldots, p_k\}$ of k *portals*, i.e., selected points that lie on some of the trajectories. While our practical study focuses on instances in which the trajectories $\mathcal{T}$ are purely spatial, e.g., given as polygonal chains or line segments in the plane, our methods apply equally well to more general portals and to trajectories that include a temporal component and live in space-time. More generally, we are given a graph $\mathcal{G}$, with length-weighted edges, and a set of paths within $\mathcal{G}$. We wish to determine a subset of k of the nodes of $\mathcal{G}$ that maximizes the sum of the (weighted) lengths of the subpaths (of the input paths) that link consecutive portals along the input paths.

We seek to compute a P that maximizes the total *captured weight* of subtrajectories between pairs of portals. For a trajectory $\tau \in \mathcal{T}$, if there are two or more portals of P that lie along τ, say $\{p_{i_1}, \ldots, p_{i_q}\}$ (for $q \geq 2$), then the subtrajectory, $\tau_{p_{i_1}, p_{i_q}}$, between p_{i_1} and p_{i_q} is *captured* by P, and we get credit for its weight $f(\tau_{p_{i_1}, p_{i_q}})$. (For many of our instances, $f(\tau_{p_{i_1}, p_{i_q}})$ corresponds to the Euclidean distances, denoted by $|\tau_{p_{i_1}, p_{i_q}}|$, but our methods generalize to other types of weights.) Let $f_P(\tau)$ denote the captured weight of trajectory τ by the portal set P. The Trajectory Capture Problem (TCP) is then to compute, for given $\mathcal{T}$ and k, a set of k portals $P = \{p_1, \ldots, p_k\}$ to maximize $\sum_{\tau \in \mathcal{T}} f_P(\tau)$.

3 Analytical Results

The TCP is NP-hard and hard to approximate for general graphs even when all trajectories have weight 1. Given a graph, let each edge be a trajectory. Then an optimal solution to TCP gives the densest k-node subgraph. Manurangsi [16] showed that, assuming the exponential time hypothesis, there cannot be an $n^{\frac{1}{\log \log n^c}}$-approximation algorithm for the Densest k-Subgraph problem for any constant $c > 0$.

In the following, we give more specific results for a range of geometric TCP versions.

3.1 One-Dimensional TCP

In the one-dimensional setting, the underlying graph $\mathcal{G}$ is a path, and the input trajectories $\mathcal{T} = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ are a set of subpaths of $\mathcal{G}$, specified by pairs of integers, a_i, b_i. A solution to the TCP then consists of k points, $P = \{p_1, \ldots, p_k\}$, w.l.o.g. indexed in sorted order, $p_1 < p_2 < \cdots < p_k$.

▶ **Theorem 1.** *The one-dimensional TCP can be solved exactly in polynomial time.*

Proof. For $i = 1, 2, \ldots, k-1$, let $V_i(x)$ be the maximum possible length of $\mathcal{T}$ captured by points $(p_i, \ldots, p_k)$, with $p_i = x \in \{a_1, \ldots, a_n, b_1, \ldots, b_n\}$; let $V_k(x) = 0$, for any x. Then, the value functions V_i satisfy the following dynamic programming recursion, for $i = 1, 2, \ldots, k-1$, and each $x \in \{a_1, \ldots, a_n, b_1, \ldots, b_n\}$:

$$V_i(x) = \max_{x' \in \{a_1, \ldots, a_n, b_1, \ldots, b_n\}, x' > x} \{V_{i+1}(x') + \sum_{j:(x,x') \subseteq (a_j, b_j)} (x' - x)\}.$$

The summation counts the length $(x' - x)$ once for each input interval that contains the interval (x, x'). We can compute the $O(nk)$ values $V_i(x)$ in time $O(n^2 k)$ by incrementally updating the summation as we consider values of x' in increasing order. ◀

3.2 Two-Dimensional TCP

3.2.1 Complexity

▶ **Theorem 2.** *The TCP is NP-hard, even for an input $\mathcal{T}$ of n line segments in the plane.*

The proof of Theorem 2 (see full version [6]) uses a construction involving segments of *many* orientations whose pairwise intersections may only be single points. The following shows that the TCP is already NP-hard for segments of *two* orientations, provided that two intersecting segments may be collinear, and three different segments can intersect in a single point.

▶ **Theorem 3.** *The TCP is NP-hard, even for an input $\mathcal{T}$ of n line segments of at most two orientations in the plane, with any two segments intersecting in at most a single point (there are no overlapping pairs of segments) but possible collinearity.*

3.2.2 Approximation Algorithms

Consider first the case in which the set $\mathcal{T}$ of input trajectories is the (disjoint) union of K subsets of trajectories, $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_K$, with each subset $\mathcal{T}_i$ having the following *path property*: For any connected component of the intersection graph of $\mathcal{T}_i$, the trajectories in that component are all subpaths of some path in the union of $\mathcal{T}_i$. This condition holds, for example, if $\mathcal{T}$ is a set of line segments of K distinct orientations.

▶ **Theorem 4.** *The TCP for an input set $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_K$, with each subset $\mathcal{T}_i$ having the path property, has a polynomial-time K-approximation algorithm.*

Proof. Within each class $\mathcal{T}_i$, the path property implies that the trajectories behave like intervals on a line, so our one-dimensional dynamic programming solution applies. Selecting the best solution that uses all k points for one of the $\mathcal{T}_i$ gives a polynomial-time algorithm with approximation ratio K. ◀

▶ **Theorem 5.** *The TCP for an input set $\mathcal{T}$ of arbitrarily overlapping/crossing trajectory paths in the plane having bounded depth Δ (i.e., no point of $\Re^2$ lies in more than Δ input trajectories) has a polynomial-time Δ-approximation algorithm, for any even number, k, of portals. (If k is odd, the approximation factor is at most $\Delta(1+\frac{1}{k-1})$.)*

Proof. Consider an optimal set, H^*, of k hit points, capturing total trajectory length L^*. For each hit point $h \in H^*$, which lies on $\delta \leq \Delta$ trajectories of $\mathcal{T}$, we replace h with δ copies ("clones") of the point h, with one clone associated with each of the δ trajectories that h hits. In total there are at most $k\Delta$ clones. Consider any trajectory $\tau \in \mathcal{T}$, and consider the clones (copies of hit points) that lie along τ. If there are at least 2 clones on τ, then the portion of τ that lies between the extreme clones on τ is captured; the length of this portion is at most the length of τ. The $k\Delta$ clones capture portions of at most $\lfloor k\Delta/2 \rfloor$ trajectories, resulting in total captured length $L^* \leq \ell_1 + \ell_2 + \cdots + \ell_{\lfloor k\Delta/2 \rfloor}$, where ℓ_i denotes the length of the ith longest trajectory of $\mathcal{T}$ ($\ell_1 \geq \ell_2 \geq \ell_3 \geq \cdots$).

Now consider the simple greedy algorithm that places hit points at the 2 endpoints of the $\lfloor k/2 \rfloor$ longest trajectories of $\mathcal{T}$, using at most k total hit points. This algorithm captures length $L = \ell_1 + \ell_2 + \cdots + \ell_{\lfloor k/2 \rfloor}$. The approximation ratio is at most

$$\frac{L^*}{L} \leq \frac{\lfloor k\Delta/2 \rfloor}{\lfloor k/2 \rfloor}.$$

Thus, $\frac{L^*}{L} \leq \Delta$ for even k. For odd k, the denominator is exactly $(k-1)/2$, while the numerator is either $k\Delta/2$ (if Δ is even) or $(k\Delta-1)/2$ (if Δ is odd); thus, $\frac{L^*}{L} \leq \frac{k\Delta/2}{(k-1)/2} = \Delta(1+\frac{1}{k-1})$. ◀

4 Algorithm Engineering

As the TCP can be considered an optimization problem on a weighted graph, we can use approaches such as Integer Linear Programming and local search heuristics. Given the geometric origins of the TCP, we consider geometric aspects; in addition, dealing with geometric data involved a number of other aspects of algorithm engineering, such as accuracy and correctness when handling locations, coordinates, and intersections.

4.1 Integer Linear Programming

4.1.1 An IP Formulation

As a problem of combinatorial optimization, the TCP can be modeled as an Integer Linear Program (IP), for which solutions can be computed with the help of powerful IP solvers. The following IP models the TCP.

$$\max \textstyle\sum_{\tau\in\mathcal{T}, e\in E(\tau)} f(e) x_{\tau,e}$$

$$\textstyle\sum_{v\in V} y_v \leq k \qquad \text{(Constraint 1)}$$

$$\forall \tau = (v_0, \ldots, v_l) \in \mathcal{T}:$$

$$\forall i \in 0, \ldots, l-1: \quad \begin{cases} x_{\tau,v_i v_{i+1}} \leq y_i & \text{if } i = 0, \\ x_{\tau,v_i v_{i+1}} \leq y_i + x_{\tau,v_{i-1} v_i} & \text{else} \end{cases} \qquad \text{(Constraint 2)}$$

$$\forall i \in 1, \ldots, l: \quad \begin{cases} x_{\tau,v_{i-1} v_i} \leq y_i & \text{if } i = l, \\ x_{\tau,v_{i-1} v_i} \leq y_i + x_{\tau,v_i v_{i+1}} & \text{else} \end{cases} \qquad \text{(Constraint 3)}$$

$$\forall v \in V, \tau \in e \in \tau: \quad x_{\tau,e}, y_v \in \{0,1\}$$

We have two types of Boolean variables: y_v, for $v \in V$, which indicates if node v is one of the k selected portals, and $x_{\tau,e}$, for edge $e \in E$ on trajectory τ, which indicates if the

portion e of trajectory τ is captured by selected portals. For an edge e, there are distinct variables, $x_{\tau,e}, x_{\tau',e}$, for trajectories $\tau \neq \tau'$, because e can be captured in τ but not in τ'.

Our objective function maximizes the weighted sum of captured trajectory edges, where $E(\tau)$ denotes the edges of τ in $\mathcal{G}$, and $f(e)$ is the weight (i.e. length) of edge e. (Optionally, we could have trajectory-dependent weights on edges.) Constraint 1 limits the number ($\leq k$) of selected portals. Constraints 2 and 3 enforce that, in order for an edge to be captured as part of trajectory τ, there must be a selected portal in each direction; either there is a selected portal at the next node, or the following trajectory edge is also captured. In the latter case, because τ has no cycle (it is a simple path), there must be a selected portal on τ at some point in that direction if any portion of τ is to be captured.

For an example, see Section A.5 in the full version [6].

4.1.2 Fractional Solutions

Relaxing the integrality constraints of the IP may result in fractional solutions. We show (in the full version [6]) that the gap (ratio) between the best fractional and the integral (optimal) solution objective functions can be arbitrarily large, for any fixed k.

▶ **Theorem 6.** *The integrality gap for the TCP IP can be arbitrarily large for any k.*

For instances arising from non-overlapping (i.e., no parallel segments may share more than one point) axis-parallel segments, we can bound the integrality gap, because the particularly bad "clusters" of the general case cannot occur.

▶ **Theorem 7.** *For trajectories $\mathcal{T}$ arising from non-overlapping axis-parallel line segments, the integrality gap is at most $\frac{k}{\lfloor k/2 \rfloor}$, for $k \geq 2$.*

Proof. We can easily get an integral solution by simply capturing the $\lfloor \frac{k}{2} \rfloor$ longest trajectories (segments) by selecting their at most k endpoints as portals.

We create a new LP instance, called LP_2, by including two copies v_1, v_2 of each portal variable v, so that one copy lies only on horizontal segments, while the other lies only on vertical segments. We constrain $y_{v_1} = y_{v_2} = y_v$ and allow a budget of $2k$ portals for LP_2. Any feasible values for the y_i in the original LP solution are still feasible in LP_2 (setting both copies), so the optimal solution of LP_2 is an upper bound for the original LP. Because segments do not overlap, every portal now lies only on a single segment. Thus the optimal solution for LP_2 covers the $\frac{2k}{2} = k$ longest segments. This shows the integrality gap is at most $\frac{k}{\lfloor k/2 \rfloor}$. ◀

The bound of Theorem 7 is tight for $k = 2$: Consider four segments that are edges of a unit square; then, $k = 2$ portals can capture at most length 1, while a fractional value of $1/2$ at each of the four corners yields objective value $4/2 = 2$ for the LP. For $k \geq 4$, it becomes increasingly difficult to build instances with a high integrality gap.

4.2 Heuristics

Integer Linear Programming solvers can provide provably optimal or near-optimal solutions for relatively large instances. However, eventually runtime and memory requirements become a limiting factor for large enough instances, so it becomes important to develop effective heuristics. We considered a spectrum of heuristics: *Greedy*, which constructs solutions from scratch by locally optimal choices; *Iterated Local Search*, which iteratively improves a current solution by finding a better one in its local neighborhood; *Simulated Annealing*, which uses a "temperature" function that governs the probability of temporarily accepting a worse solution

during a local search; and *Genetic Algorithms*, which maintain a selection of solutions that are locally modified and combined to achieve gradually better solutions.

Greedy begins by selecting the two portals at the ends of the longest trajectory, and then incrementally, greedily selects portals that in each step increase the total captured length as much as possible. *Greedy* can be fooled and give poor solutions; it can, though, serve to give a reasonable starting solution for our other metaheuristics.

Iterated Local Search (ILS) is a basic metaheuristic that, given an initial solution, iteratively replaces the current solution with the best solution found by applying a single local modification, until no further improvement can be achieved. The set of solutions that can be obtained by a single local modification from a specific solution is called its *neighborhood*. For a local modification operator based on changing a single portal, the neighborhood consists of all solutions that differ in exactly one portal. The smaller the neighborhood, the faster the best solution within it can be found; however, a smaller neighborhood also reduces the search space and correspondingly can reduce the quality of the obtained solutions. We considered *global neighborhoods*, based on moving a random portal to an alternative random candidate node, and *local neighborhoods*, based on moving a single portal to positions adjacent to other (unmoved) portals. ILS is initialized with any reasonable solution; after some experimentation with alternatives (e.g., random selection), we settled on using *Greedy* as the starting solution for ILS.

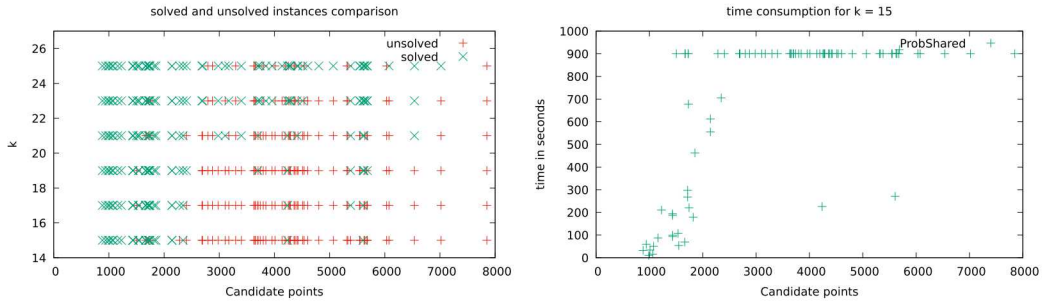
Simulated Annealing is similar to ILS, but instead of searching for the best solution in the neighborhood, it selects a random solution in the neighborhood and moves to it if (1) it is an improvement, or (2) if it is not an improvement, but it passes a random test. The probability of moving to a worse solution is determined by a "temperature function" and decreases with time. Initially, it can easily escape local optima; when the search satisfies a termination criterion, it returns the best solution it found.

We considered three different termination criteria: the total number of iterations, the number of iterations without an improvement, and the total runtime. For temperature regulation, we used a geometric reduction by a constant multiplicative amount; for diversification we "reheated" the temperature to the start temperature when we did not change the solution for a certain number of iterations. For translating the temperature function to a probability function, we used the Bolzmann function: $\text{bolzmann}(s', s'', T) := \exp^{(-\frac{s'-s''}{T})}$, where s' and s'' are the captured weight of two neighboring solutions. In addition, we used parallelization for pursuing multiple searches from different starting points.

Evolutionary algorithms (EAs) are motivated by the way adaption to environmental conditions happens in nature. They maintain a "population" of current solutions. At each step, the EA produces new solutions through mutations (i.e., local changes) and recombination (combining pieces of solutions in the current population to create new ones). Then, the EA keeps the best solutions (previous or new) to maintain a stable population size. We create the initial population by a version of Greedy that starts with a random segment instead of the longest one. For mutations, we used ILS or SA. The probability of selection for recombination is $\frac{f(s)-f(s_{\min})}{\sum_{s' \in S} f(s')}$. We used uniform random crossover.

4.3 Generating Benchmark Instances

Our IP and heuristic methods apply to general sets of trajectories $\mathcal{T}$, given by spatiotemporal or combinatorial data. We focus on geometric instances, most of which are based on line-segment trajectories. Instances based on random segments tend to be very easy to solve because most vertices have degree 2. So we generated instances based on a set of seed points and selected segments linking them, resulting in arrangement graphs with multi-trajectory



(a) Solved and unsolved instances. **(b)** Time used for instances with $k = 15$

Figure 2 Test results for solved and unsolved instances using the IP. The number of seed points varies from 35 up to 55 points. All tests were performed with a time limit of 900 seconds.

intersections and more complicated covering graphs. Alternatively, we tested adding new intersection points to the set of seed points when incrementally constructing the arrangement. For all methods, we used exact intersection point computations from the Computational Geometry Algorithms Library (CGAL) to overcome problems of floating point precision for large instances. We generated seed points randomly, using a variety of spatial distributions, including uniform distributions, point sets from the TSP benchmark library [19], and point sets with density distributions based on light maps, corresponding to population densities (see [4]).

4.4 Experimental Evaluation

All experiments were performed on a single Intel(R) Core(TM) i7-4770 (4×3.4 GHz) with 32 GB and CPLEX (V12.7.1 with default settings), with a time limit of 900 s. The code and data is available at `https://github.com/ahillbs/trajectory_capturing`.

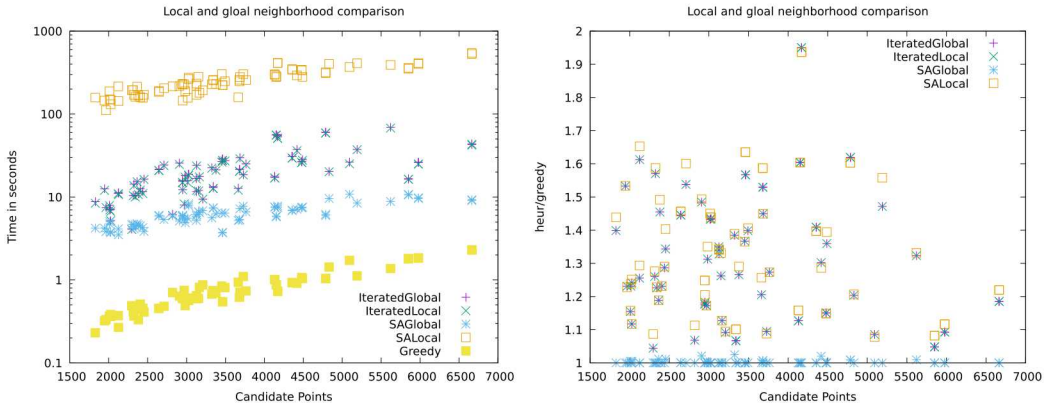
4.4.1 Integer and Linear Programming

We first consider the sizes of instances that our IP can solve to optimality within a 900-second time limit, and we consider which factors contribute to the difficulty of the instance.

We varied the number of seed points between 35 and 55 and varied the (uniform) probability of a segment connecting two seeds between 10% and 20%. In Figure 2a, we see that instances with up to about 2500 candidate points (i.e., nodes at intersection points in the arrangement, where portals can be placed) can be solved for $15 \leq k \leq 23$ to provable optimality within the time limit. Instances with more than 2500 candidate points are most often not solved for $k < 23$. For instances with $k \geq 23$, the problem seems to be easier to solve. In Figure 2b we see that for $k = 15$ and between 1500 and 2000 candidate points, instances start to become very difficult to solve. However, for $k \geq 23$, instances are still solvable for more than 2500 candidate points.

4.4.2 Heuristic Methods

Neighborhoods for Local Methods. For modifying a given solution, we considered global neighborhoods, in which a portal is moved to an arbitrary other position, and local neighborhoods, in which a portal is only moved to positions that connect to another portal. Using global neighborhoods, all solutions are theoretically quickly reachable but they are significantly larger than local neighborhoods and, thus, a meta-heuristic may not work in



(a) Comparison of required time

(b) Comparison of achieved objective values and greedy solutions

Figure 3 Comparison of solutions for local and global neighborhood with Iterated Local Search and Simulated Annealing for $k = 25$.

a focused enough way. Details of this comparison can be found in Section A.4 in the full version [6]; in particular Figure 3 shows our experimental evaluation. Iterated Local Search yields the same solution quality for both neighborhoods; with global neighborhood, only the runtime increases. Simulated Annealing with global neighborhoods barely improves the initial greedy solution, while it gives the best solutions with local neighborhoods.

As a result, we used local neighborhoods for all meta-heuristics.

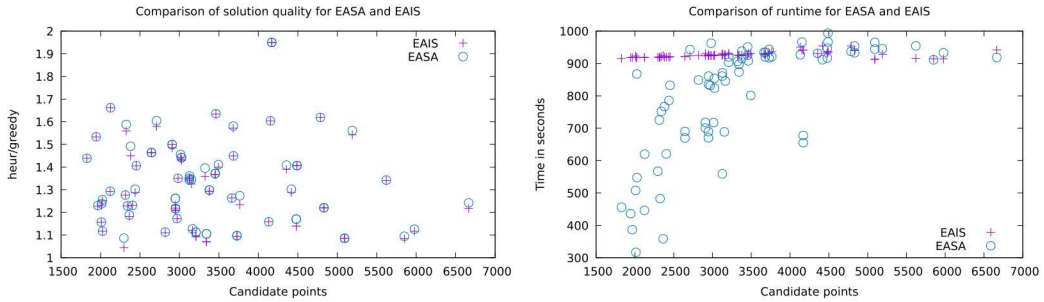
Mutation Strategy for Evolutionary Algorithms. For evolutionary algorithms, choosing the right kinds of mutations is of crucial importance, as these allow reaching solutions that are not achievable only via recombination. Practical usefulness requires focused mutations that have a high probability of being useful, instead of purely random changes. That is why we considered Iterated Local Search and Simulated Annealing as mutation operations. As Simulated Annealing has a longer runtime, we used a faster terminating version (with potentially worse solutions) when using it for mutation. In the following we refer to the version with Simulated Annealing as EASA and with Iterated Local Search as EAIS. We have a start with 100 solutions and keep an ongoing population of 50 solutions. The evolutionary algorithm stops after 15 minutes (but can take slightly longer to finish the last round) or if it has not found an improvement for multiple rounds.

Figure 4 shows the experimental comparison of both mutation variants. One can see that EASA performs slightly better and is significantly faster for smaller instances. This implies that EASA often quickly finds a good solution but is usually not able to improve it further and terminates early. EAIS, on the other hand, is able to improve its quality until the time limit but still remains slightly worse. For the further experiments, we settled on EASA.

4.4.3 Comparison of Heuristics with IP as Baseline

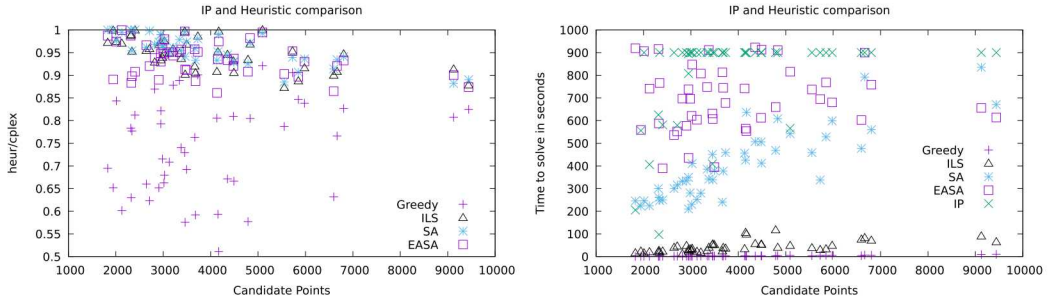
We compared the heuristics in terms of solution quality and runtime against the IP solver, which produces not only solutions, but also guaranteed bounds.

Figure 5 shows the obtained results. The Evolutionary Algorithm produces, on average, the worst solutions of all metaheuristics, while still requiring more time than the others. However, it is the only metaheuristic tested that reliably computes good solutions for instances consisting of several point clusters, for which solutions consist of several connected



(a) Solution quality for Evolutionary Algorithms **(b)** Runtimes for Evolutionary Algorithms

Figure 4 Comparison of solution quality and runtime by Evolutionary Algorithm with Iterated Local Search and with Simulated Annealing.



(a) Solution quality for heuristics and IP solutions. **(b)** Runtimes of heuristics and IP solutions.

Figure 5 Comparisons of solution quality and runtime for all tested algorithms.

components. These instances did not occur with our generation method but only in separate, manually created instances which are not part of this experiment.

The greedy approach never falls below $\frac{1}{2}$OPT for these instances, while being the fastest.

Iterated Local Search (ILS) appears to produce quite good solutions, which are not worse than 10% below the optimal solution, in a very short time frame. While it only finds a local optimum, it seems that the objective function quality of these are quite close to the optimal values. As a consequence, Iterated Local Search can produce good solutions for instances with up to 5000 candidate points and $k = 25$.

The best heuristic algorithm, in terms of solution quality and runtime, appears to be Simulated Annealing. The combination of fast diversification at high temperature and random swaps for improving solutions at low temperature seems to work quite well; in addition, the mechanism of reheating to restart diversification, in combination with multi-threading to cover a larger search space, are characteristics that are not present in the Evolutionary Algorithm. For $k = 25$, Simulated Annealing produces excellent solutions for instances with up to 5000 candidate points; this instance size can be easily increased for smaller k.

In summary, we can produce excellent heuristic solutions for instances with up to 5000 candidate points and $k = 25$. If fast solutions are desired, Iterated Local Search is the method of choice. The best tradeofff between runtime and solution quality is offered by Simulated Annealing. Finally, for cluster-based instances, we recommend Evolutionary Algorithms.

4.4.4 Linear Programming and Integrality Gap

As described in Section 3, if the TCP input trajectories come from K subsets of noncrossing trajectories, we have a K-approximation algorithm based on dynamic programming. In particular, if the input trajectories consist of axis-parallel line segments, $K = 2$, so there is a 2-approximation. This may coincide with better practical solvability of these kinds of instances. We have verified this for some instances for which all segments are axis-parallel and (for collinear segments) non-overlapping. (See Figure 11 in the full paper [6] for such an instance with 1100 segments and roughly 8200-8500 candidate points. We have also solved instances with 2000 segments and 19,000 candidate points.) Furthermore, for instances with up to 20,000 points, the integrality gap was never larger than 20% for $k = 5$, 7% for $k = 10$, 5% for $k = 15$ and less than 2.5% for larger k. See Figures 13–17 in the full version [6].

4.4.5 Application to Taxi Trajectory Data

We have applied our TCP model to solve real-world data sets to optimality. In Figure 6 we show the results of computing $k = 5$ optimal portals for a set of trajectories based on taxi cab routes in the San Francisco Bay Area. The data is based on 375 vehicles, sampled every 5 minutes, 288 times per day, for one week [18]; see the trajectories in Figure 1.

Our experiments included runs on 30 instances, with k ranging from 5 to 11, on sets of 10 to 120 trajectories of varying lengths (comprised of 1300 to 3700 edges, and 600 to 1800 vertices). The trajectories are snapped to a regular grid graph. Solution times of the IP were up to 200 seconds of computation, with most instances taking less than 10 seconds.

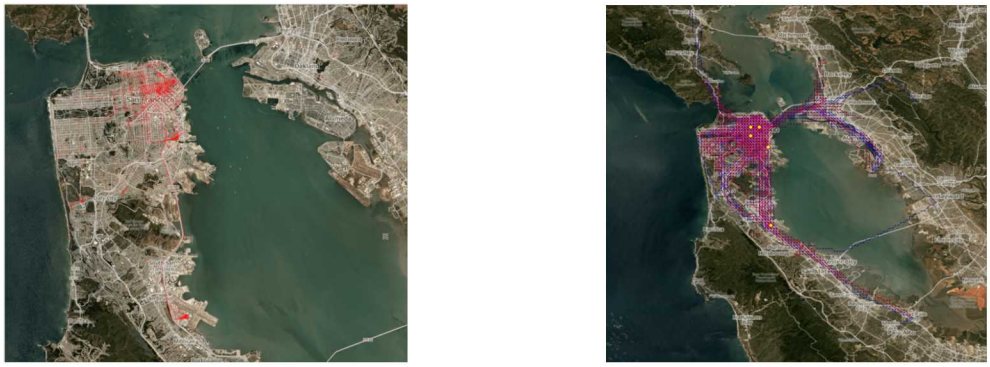


Figure 6 (Left) Candidate points before processing. (Right) Solution to real-world TCP instance: An optimal set of $k = 5$ portals are highlighted. Red trajectory portions (some of which may loop back) are captured, blue ones are not captured. Satellite images are courtesy of Planet Labs Inc.

5 Conclusion

We have shown that the TCP is NP-hard, even for axis-aligned line segment trajectories in the plane, and given approximation algorithms for two cases. Can we improve the approximation factor of K for a set of trajectories that is the union of K subsets, each of which is noncrossing? Can we improve the approximation factor of Δ for a set of trajectories of depth at most Δ?

A focus of our work is the exploration, via algorithm engineering, of practical methods for solving the TCP. Our methods are based on integer programming and on simple heuristic search methods. It will be interesting to develop more specified methods for other, specific classes of instances, such as further geometric instances arising from other types of real-world geographic data.

Acknowledgments

Work by Tyler Mayer was mostly carried out while at Stony Brook University. Joe Mitchell and Tyler Mayer were partially supported by the National Science Foundation (CCF-1526406) and a grant from the US-Israel Binational Science Foundation (BSF project 2016116). Joe Mitchell was also partially supported by the DARPA Lagrange program. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

References

1 Mattias Andersson, Joachim Gudmundsson, Patrick Laube, and Thomas Wolle. Reporting leaders and followers among trajectories of moving point objects. *GeoInformatica*, 12(4):497–528, 2008.

2 Marc Benkert, Joachim Gudmundsson, Florian Hübner, and Thomas Wolle. Reporting flock patterns. *Computational Geometry*, 41(3):111–125, 2008.

3 Kevin Buchin, Maike Buchin, Marc van Kreveld, Bettina Speckmann, and Frank Staals. Trajectory grouping structure. In *Algorithms and data structures*, pages 219–230. Springer, 2013.

4 Sándor P. Fekete, Andreas Haas, Michael Hemmer, Michael Hoffmann, Irina Kostitsyna, Dominik Krupke, Florian Maurer, Joseph S. B. Mitchell, Arne Schmidt, Christiane Schmidt, and Julian Troegel. Computing nonsimple polygons of minimum perimeter. *Journal of Computational Geometry*, 8:340–365, 2017.

5 Sándor P Fekete, Kan Huang, Joseph SB Mitchell, Ojas Parekh, and Cynthia A Phillips. Geometric hitting set for segments of few orientations. *Theory of Computing Systems*, 62(2):268–303, 2018.

6 Sándor P. Fekete, Alexander Hill, Dominik Krupke, Tyler Mayer, Joseph S. B. Mitchell, Ojas Parekh, and Cynthia A. Phillips. Probing a set of trajectories to maximize captured information, 2020.

7 Joachim Gudmundsson and Marc van Kreveld. Computing longest duration flocks in trajectory data. In *Proc. of the 14th Annual ACM International Symposium on Advances in Geographic Information Systems*, pages 35–42. ACM, 2006.

8 Joachim Gudmundsson, Marc van Kreveld, and Bettina Speckmann. Efficient detection of patterns in 2D trajectories of moving points. *Geoinformatica*, 11(2):195–215, 2007.

9 Marios Hadjieleftheriou, George Kollios, Dimitrios Gunopulos, and Vassilis J Tsotras. On-line discovery of dense areas in spatio-temporal databases. In *Advances in Spatial and Temporal Databases*, pages 306–324. Springer, 2003.

10 Haje Korth, Michelle F. Thomsen, Karl-Heinz Glassmeier, and W. Scott Phillips. Particle tomography of the inner magnetosphere. *Journal of Geophysical Research: Space Physics*, 107(A9):SMP–5, 2002.

11 Patrick Laube, Matt Duckham, and Thomas Wolle. Decentralized movement pattern detection amongst mobile geosensor nodes. In *Geographic Information Science*, pages 199–216. Springer, 2008.

12 Jae-Gil Lee, Jiawei Han, Xiaolei Li, and Hector Gonzalez. TraClass: trajectory classification using hierarchical region-based and trajectory-based clustering. *Proceedings of the VLDB Endowment*, 1(1):1081–1094, 2008.

13 Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *Proc. of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 593–604. ACM, 2007.

14 Yifan Li, Jiawei Han, and Jiong Yang. Clustering moving objects. In *Proc. Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 617–622. ACM, 2004.

15 Zhenhui Li, Bolin Ding, Jiawei Han, and Roland Kays. Swarm: Mining relaxed temporal moving object clusters. *Proceedings of the VLDB Endowment*, 3(1-2):723–734, 2010.

16 Pasin Manurangsi. Almost-polynomial ratio eth-hardness of approximating densest k-subgraph. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 954–961, 2017.

17 Mirco Nanni and Dino Pedreschi. Time-focused clustering of trajectories of moving objects. *Journal of Intelligent Information Systems*, 27(3):267–289, 2006.

18 Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. CRAWDAD dataset epfl/mobility (v. 2009-02-24), doi:10.15783/c7j010, February 2009.

19 Gerhard Reinelt. TSPLIB–a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

20 Md Reaz Uddin, Chinya Ravishankar, and Vassilis J Tsotras. Finding regions of interest from trajectory data. In *Mobile Data Management (MDM), 2011 12th IEEE International Conference on*, volume 1, pages 39–48. IEEE, 2011.

21 Marcos R Vieira, Petko Bakalov, and Vassilis J Tsotras. On-line discovery of flock patterns in spatio-temporal data. In *Proc. of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 286–295. ACM, 2009.

22 Yu Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):29, 2015.

23 Yu Zheng and Xiaofang Zhou. *Computing with spatial trajectories*. Springer Science & Business Media, 2011.