Drone Air Traffic Control: Tracking a Set of Moving Objects with Minimal Power

Chek-Manh Loi[1,†], Michael Perk[1,†], Malte Hoffmann[1], Sándor Fekete[1,2]

***Abstract*— A common sensing problem is to use a set of stationary tracking locations to monitor a collection of moving devices: Given n objects that need to be tracked, each following its own trajectory, and m stationary traffic control stations, each with a sensing region of adjustable range; how should we adjust the individual sensor ranges in order to optimize energy consumption? We provide both negative theoretical and positive practical results for this important and natural challenge. On the theoretical side, we show that even if all objects move at constant speed along straight lines, no polynomial-time algorithm can guarantee optimal coverage for a given starting solution. On the practical side, we present an algorithm based on geometric insights that is able to find optimal solutions for the $\min\max$ variant of the problem, which aims at minimizing peak power consumption. Runtimes for instances with 500 moving objects and 25 stations are in the order of seconds for scenarios that take minutes to play out in the real world, demonstrating real-time capability of our methods.**

I. Introduction

Keeping track of a (potentially large) collection of moving objects is a fundamental problem with a long history. With the rise of air traffic, developing automated systems for this important task became even more critical: In *air traffic control*, a typical scenario involves a set of moving planes, as well as a number of stationary control centers, which are equipped with powerful tracking devices such as long-range radar and various other communication and control capabilities to coordinate overall motion, without much regard for the power consumption of tracking devices. With rapidly increasing number and ubiquity of small and light drones, this type of air traffic control needs further development [8], [20], such as the use of less powerful tracking stations: Given a collection of moving objects that must be tracked by a set of stationary sensors with adjustable sensing regions, how should we adjust the sensing radius over time to optimize power consumption?

Formally, this is the Kinetic Disk Covering Problem (KDC), see Figure 1 for an illustration. Given a set $\mathcal{P} = \{p_1, \ldots, p_n\}$ of n objects, each moving in space along a trajectory $p_i(t)$ over the time interval $t \in [0, 1]$ as well as a set $\mathcal{Y}$ of m centers. The goal is to assign a radius $r_i(t)$ to each center $y_i \in \mathcal{Y}$ at each time t such that each point is covered by at least one disk, i.e., $\forall t \in [0, 1], \forall p_j \in \mathcal{P}(t), \exists y_i \in \mathcal{Y}$: $||y_i - p_j(t)|| \leq r_i(t)$ and the maximum total area of all disks at any time is minimized, i.e., $\min\max_{t\in[0,1]} \sum_{i=1}^{m} \pi r_i(t)^2$. Another variant of the KDC problem is to minimize the total area of all disks over time, i.e., $\min \int_0^1 \sum_{i=1}^{m} \pi r_i(t)^2 dt$.

The problem of finding an area-optimal assignment for a stationary set of points and centers is called the Disk Covering Problem (DC) and is known to be NP-hard [2]. We show that the KDC problem is also NP-hard, even if we are given an optimal solution at time $t = 0$. This means that extending a solution over time in an optimal way is computationally infeasible. We therefore study heuristic approaches for extending stationary solutions over time and provide an exact algorithm as well as efficient heuristics for the $\min\max$ variant of the KDC problem in Section VI. The proposed algorithms are then evaluated on several classes of instances of the KDC problem in Section VII. Despite the theoretic complexity, our results show that the proposed methods can find optimal solutions for realistically sized instances of the KDC problem in a matter of seconds.

II. Preliminaries

We study the case in which each point $\mathcal{P} = \{p_1, \ldots, p_n\}$ moves along a given linear trajectory $p_i(t) = (1 - t) \cdot p_i^0 + t \cdot p_i^1 \in \mathbb{R}^2$ over the time interval $t \in [0, 1]$. We denote the set of points at time t as $\mathcal{P}(t)$ and the stationary instance $I(t) = (\mathcal{P}(t), \mathcal{Y})$ of the DC at time t. For simplicity, we sometimes use $\mathcal{P}$ to denote either a set of moving points or stationary points, with the intended meaning clarified by context.

In the KDC problem, we assign a radius $r_i(t)$ to each center $y_i \in \mathcal{Y}$ at each time t such that all points are covered by at least one disk. In an optimal solution, the radius of each disk is determined by the farthest point assigned to it. We call these *supporting points*. A solution to the KDC is a list of k assignments $\{A_{t_0}, A_{t_1}, \ldots, A_{t_k}\}$, with $0 = t_0 \leq t_1 \leq \ldots \leq t_k = 1$. Each assignment $A_t : (\mathcal{Y} \to \mathcal{P})$ assigns *supporting points* to centers over time. The radius of the disk centered at y_i at time $t \in [t_j, t_{j+1})$ is $r_i(t) = \left\|y_i - A_{t_j}(y_i)(t)\right\|$.

III. Related Work

Given two point sets $\mathcal{P}$, $\mathcal{Y}$ and a coverage function κ, the non-uniform minimum-cost multi-cover (MCMC) problem seeks disk radii so each $p \in \mathcal{P}$ is covered at least $\kappa(p)$ times. Abu-Affash et al. [1] gave a $23.02 + 63.95(\kappa_{\max} - 1)$-approximation for the uniform case, while [3] and [7] provided polynomial-time approximations for the non-uniform case. Huang et al. [17], [18] presented the first PTAS for non-uniform MCMC, achieving solutions arbitrarily close

† These authors share lead authorship

[1] Department of Computer Science, TU Braunschweig, Germany; {loi,perk}@ibr.cs.tu-bs.de, {m.hoffmann,s.fekete}@tu-bs.de; supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) as part of project Computational Geometry: Solving Hard Optimization Problems (CG:SHOP) - 444569951.

[2] L3S, Germany

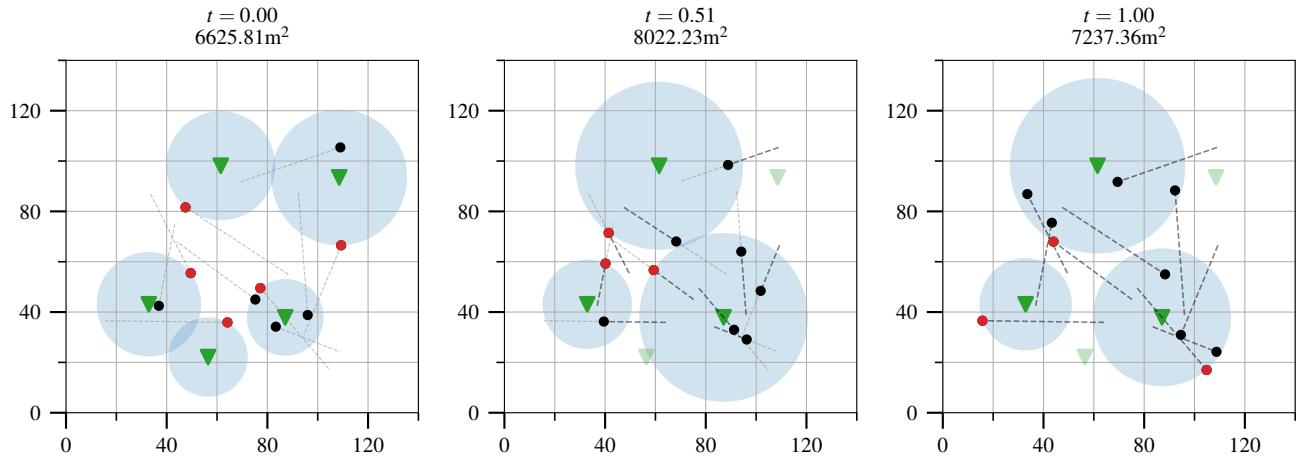


Fig. 1: Solution of the KDC problem for $n = 10$ moving points and $m = 5$ stations. The points move along the dotted lines. The stations are shown as green triangles and are opaque, if the radius is zero. We start with an IP solution at time $t = 0$. Our algorithm can find a solution over time, that uses at most $8022.23\,\mathrm{m}^2$ of area over the whole time interval. This solution is matched by a lower-bound computed by solving the static DC problem at this time step.

to optimal. These works optimize disk costs using dynamic programming and recursive techniques.

Recently, there has been some work on multi-covering fixed points with disks of varying sizes [15]. Another line of closely related work examines multi-covering points with the minimum number of unit disks, where centers are arbitrary but radii are fixed. Gao et al. [14] gave a 5-approximation algorithm running in $\mathcal{O}(n + \kappa_{\max})$, and a 4-approximation with $\mathcal{O}(n^2)$ time. Filipov and Tomova [13] studied coverage with equal disks and proposed a stochastic optimization algorithm. These approaches focus on minimizing disk count rather than disk area.

Basch et al. [4] developed fundamental concepts for kinetic data structures. Based on these results, Bespamyatnikh et al. [6] studied the k-center and k-means problem for both kinetic stations and objects. Crevel et al. [5] studied a closely related problem of transmissioning messages to mobile receivers where the sensors are only activated at certain times to transmit a single message.

IV. Hardness

Due to the known NP-hardness of the static Disk Covering (DC) problem [2], the Kinetic Disk Covering (KDC) problem is NP-hard as well. Suppose, however, that an optimal solution for the static instance at time t_0 is given. One may ask whether it is possible to optimally adapt this solution to a later time $t_1 > t_0$. The following lemma demonstrates that this is, in general, computationally intractable.

Theorem 1: Given a KDC instance $I = (\mathcal{P}, \mathcal{Y})$, a cost bound c, and an optimal solution for the static instance at time t_0, determining whether there exists an assignment of radii at a later time $t_1 > t_0$ that covers all points with total cost at most c is NP-hard.

Proof: Given an instance $I_s(\mathcal{P}_s, \mathcal{Y}_s)$ of the static DC problem, we can construct an instance $I_k(\mathcal{P}', \mathcal{Y}')$ of the KDC problem as follows. For each static point $p \in \mathcal{P}_s$ we add a point $p'(t) = ((1-t) \cdot (0,0) + t \cdot p)$ to $\mathcal{P}'$ and keep the centers, i.e., $\mathcal{Y}' = \mathcal{Y}_s$.

Now at time $t_0 = 0$ the optimal solution is a single disk from the center closest to the origin y_0 with radius $\|y_0\|$. A solution with cost at most c at time $t_1 = 1$ exists if and only if a solution with cost at most c exists for the static instance. ■

V. Stationary Disk Cover

In the stationary Disk Cover (DC) problem, we consider an instances of the KDC at a specific point t in time, i.e., we consider the stationary instance $I(t) = (\mathcal{P}(t), \mathcal{Y})$. We want to find an assignment of radii to the stations $\mathcal{Y}$ such that each point in $\mathcal{P}(t)$ is covered by at least one disk and the total area of the disks is minimized. For this, we provide a heuristic as well as an Integer Programming (IP) formulation that can solve the DC problem to provable optimally.

A. Nearest-Neighbor Heuristic

A simple heuristic to solve the DC problem is to assign each point to its nearest station. In particular, we start by sorting all objects according to their distance to their nearest station in descending order and maintain a set of uncovered objects. Then, we iterate over yet uncovered objects and assign each object to its nearest station in order to cover it with the smallest increase in area. We then update the covered objects according to the new assignment and repeat until all objects are covered.

B. Integer Programming

An effective method for finding optimal solutions to NP-hard problems is Integer Programming (IP). Although

solving an IP can take exponential time in the worst-case scenario, using meticulously designed mathematical models, specialized algorithm engineering, and existing IP solvers allows for solving considerably large instances to provable optimality. To formulate the DC as an IP, we define a discrete set of candidate disks C.

1) Computing the Candidate Set: The radius of each station is determined by its *supporting point*. This give rise to a remarkably simple enumeration scheme. For each station, we consider every point $p_i \in \mathcal{P}$ as a potential *supporting point*. Specifically, we first sort all points $\mathcal{P}$ in ascending order by their distance to the center y. Then, for each point $p_i \in \mathcal{P}$, we add a disk centered at y with radius $\|y - p_i\|$ to the candidate set $\mathcal{C}$; this disk covers all points up to and including p_i in the ordering.

This yields $\mathcal{O}(mn)$ possible disks, as for each of the m stations we consider n potential support points. Accounting for the sorting of the points for each station, the overall time to compute the candidate set is $\mathcal{O}(nm \log n)$.

2) IP Formulation: For every disk d_i in the candidate set C, we define a binary variable x_i that encodes if the disk is used in the solution. The constraint ensures that every point $p_j \in \mathcal{P}$ is covered by at least one station.

$$\begin{aligned} \text{minimize} \quad & \pi \cdot \sum_{d_i \in C} r_i^2 x_i \\ \text{subject to} \quad & \sum_{\substack{d_i \in C \\ p_j \in d_i}} x_i \geq 1, \qquad \forall p_j \in \mathcal{P} \\ & x_i \in \{0,1\}, \quad \forall d_i \in C \end{aligned}$$

VI. Algorithmic Approaches

Here we discuss algorithmic approaches to solve the KDC problem. The idea is to extend every solution to the (stationary) DC problem at time t to a solution for the KDC problem, see Section VI-A. Further geometric insights allow us to improve extended solutions, see Section VI-B. Finally, we can use these insights to design an iterative algorithm to solve the KDC problem, see Section VI-C.

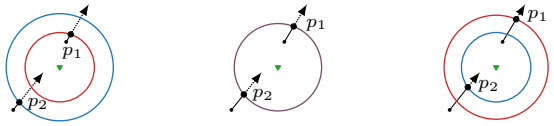
A. Ensuring Feasibility

Every solution to the (stationary) DC problem at time t can be extended to a solution for the KDC problem over time by maintaining the same assignment of objects to stations and altering the *supporting point* of each station as needed to ensure coverage. This yields a feasible solution for the KDC problem, but not necessarily an optimal one. Figure 2 shows, how initially the *supporting point* of the station y is p_2. At time $t \approx 0.41$ the disks defined by p_1 and p_2 have the same radius. After this point in time, the *supporting point* of the disk changes to p_1.

Lemma 1: In $\mathcal{O}(mn^2)$ time, all $\mathcal{O}(mn^2)$ possible *supporting point* changes when extending a stationary solution to $t \in [0,1]$ can be computed.

Proof: We can compute the time when the *supporting point* changes, by solving the equation

$$\|y - p_1(t)\|^2 = \|y - p_2(t)\|^2 . \tag{1}$$



(a) $t = 0.2$ (b) $t \approx 0.41$ (c) $t = 0.6$

Fig. 2: The *supporting point* of station y changes from p_1 to p_2, to ensure coverage of the objects at all times.

for two objects p_1 and p_2 assigned to the same station y. This yields a quadratic equation with at most two solutions, solvable analytically in $\mathcal{O}(1)$ time.

As each station can have at most n different *supporting points* and each *supporting point* can change to at most $n-1$ other objects, this yields $\mathcal{O}(mn^2)$ possible changes of *supporting points*. ∎

Note that in degenerate cases the *supporting point* might not change or multiple objects might have the same distance to a station, see Figure 3. We can resolve these cases, by choosing the object that is moving fastest away from the station as the new *supporting point*. This can be achieved by computing the derivative of the function defining the radius of y for each object p that has the same distance to the station at the event point.

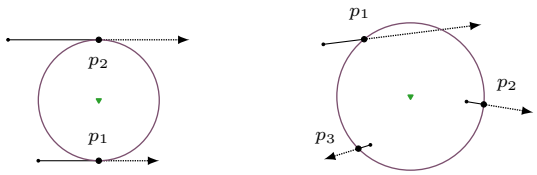


Fig. 3: Degenerate cases after which the assignment of *supporting point* might change. In both cases we choose p_2 as the new *supporting point*.

B. Improving Extended Solutions

Section VI-A shows that stationary solutions can be extended by computing points in time where the *supporting point* of a station changes. Naturally, even when starting with an optimal solution for the stationary variant, this can lead to suboptimal solutions. To reduce the overall cost, we consider a second type of event that changes the assignment of objects to stations, see Figure 4. Intuitively, we want to identify points in time in which it is cheaper to handover a point from one station to another.

Lemma 2: In $\mathcal{O}(m^2n^3)$ time all $\mathcal{O}(m^2n^3)$ possible handovers of objects between two stations (when extending a stationary solution to $t \in [0,1]$) can be computed.

Proof: For two stations y_1 and y_2 with *supporting points* p_1 and p_2, respectively, we consider the case in which y_2 takes over the point p_2 from y_1 at time t, see Figure 4. When y_2 takes over a point from a station y_1, the radius of the disk centered at y_1 will shrink, as the *supporting point* of y_1 changes to the next furthest point p_3. At the same time the radius of the disk centered at the other station y_2 will grow, as its *supporting point* changes to the point it takes over

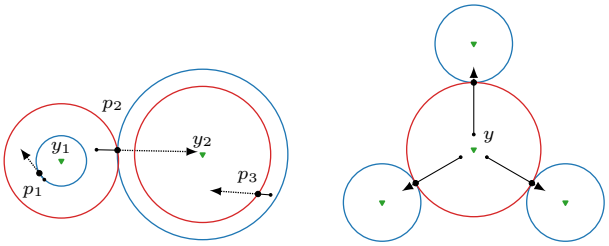


Fig. 4: (Left) Station y_2 takes over a point p_2. The objective values of the red and blue disks are equal at this point in time. We change the *supporting point* of both y_1 and y_2 at the same time. (Right) Generalization of the case depicted on the left to more than two stations.

from y_1. Again, we can compute the time t at which both solutions have the same cost, by solving a set of equations that can be solved analytically.

$$\|y_1 - p_2(t)\|^2 + \|y_2 - p_3(t)\|^2 = \|y_1 - p_1(t)\|^2 + \|y_2 - p_2(t)\|^2$$

As each station can have at most n different *supporting points* and each *supporting point* can change to at most $n-1$ other objects, this yields $\mathcal{O}(m^2n^3)$ possible handovers of objects between two stations. ■

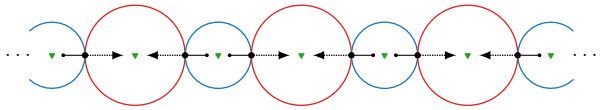


Fig. 5: Multiple stations and objects in the same event point. Either the blue or the red disk cover the objects, but not both.

The concept of handing over objects between stations can be generalized to three or more stations, see Figure 4. Furthermore, a chain of stations can successively take over objects from each other as time progresses, see Figure 5. Initially, the optimal solution consists of blue disks covering all objects. As time progresses, there is a transition point at $t = 0.5$ for which the optimal solution switches to red disks, and the previously used blue disks are no longer part of the solution. This example can be extended to an arbitrarily long chain of objects and stations. In the worst case, detecting these types of events requires comparing every subset of stations with every other subset, resulting in $\Omega(2^m)$ possible events. Thus, we cannot hope to find an efficient algorithm that considers all possible handovers.

C. Iterative Algorithm

We present an algorithm that uses the observations from Sections VI-A and VI-B to compute heuristic and optimal solutions for the $\min\max$ KDC problem. The algorithm maintains a solution over time that contains intervals in which the assignment of objects to stations does not change.

1) Initial Solution: The algorithm starts by computing a stationary solution for $t = 0$ using any of the methods presented in Section V. In the case of the integer programming formulation, the algorithm can also provide a valid lower bound on the optimal solution for the $\min\max$ KDC problem which can be used to determine the quality of the solution during the solving process.

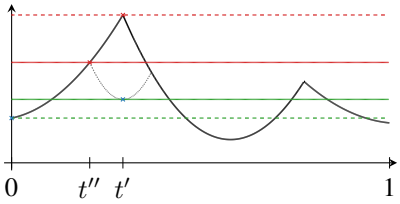


Fig. 6: Iterative approach to improve the $\min\max$ power consumption over time. Green and red lines denote lower and upper bounds on the optimal solution.

2) Extending the Solution: Using a static solution at time t, we extend the solution over time by computing the necessary adjustments from Lemma 1 to obtain a feasible solution. To speed up the computation, we cache the next *supporting point* change for each station y and update the cache whenever the assignment of y changes.

3) Iterative Approach: The algorithm then proceeds by identifying the point $t' \in [0, 1]$ where the current solution has the maximum total area. This point must lie at the intersection of two intervals in which the assignment of objects to stations changes because the objective value within each interval is an upward-opening parabola. At this point in time we compute a new stationary solution for the DC problem using a method from Section V. If the stationary solution has equal or higher objective value than the current solution, we terminate as there is no hope of improving the current solution. If the integer programming formulation is used, we can also terminate if the lower bound is within a desired optimality gap of the current solution. Initially, the (stationary) IP is solved only 1% optimality, which is then gradually decreased to 0.01% when the current KDC solution is within 1.5% of the lower bound.

If the stationary solution has a smaller objective value than the current solution, we can improve our current solution at time t' and potentially over the whole interval $[0, 1]$ or any set of subintervals. This is done by extending the solution at time t' with the methods from Section VI-C.2 both in the positive and negative time direction. Afterwards the solutions are merged into a single solution that is valid over the entire time interval, see Section VI-C.4. This process is repeated until no further improvement is possible.

4) Combining with Previous Solutions: Given a new solution for an interval $[t_1, t_2]$, we can combine it with the current best solution. This can be done by computing the intersection points of the objective values of the two solutions. As each solution consists of intervals in which the assignment of objects to stations does not change, the objective value of each solution is piece-wise quadratic and thus this intersection can be computed efficiently. We then compute the lower envelope of the two solutions in $\mathcal{O}(k_1+k_2)$ time, where k_1 and k_2 are the number of intervals in the two solutions. This yields a new solution that is valid over the entire time, see Figure 6.

Termination: The algorithm terminates when no further improvement is possible, or when the lower bound is within a desired optimality gap of the current solution. We can show that the algorithm always terminates.

Lemma 3: The algorithm terminates after a finite number of iterations.

Proof: There are only finitely many assignments of objects to stations. For any fixed assignment, the objective value as a function of time is a quadratic polynomial. However, the assignment may only be feasible on a subset of the time interval. By Lemma 1, feasibility can only change finitely many times. Thus, each quadratic function can be partitioned into a finite number of feasible pieces. Consequently, there are only finitely many assignment–interval pieces to consider across all assignments.

The lower envelope $\mathcal{L}$ of these pieces yields the optimal solution that minimizes the total area over time, i.e., it is optimal at every $t \in [0, 1]$. Whenever we compute an optimal static solution using our integer program, our combined solution includes a piece of $\mathcal{L}$. Once our algorithm computes an optimal solution on the same piece of $\mathcal{L}$ twice, i.e., it first proves a lower bound and later finds the maximum in the same piece, no better solution can be found and the algorithm terminates. Because there are only finitely many pieces, the algorithm must terminate after finitely many iterations. ■

Improvements to the Algorithm: We propose further improvements to the above algorithm. The first improvement is to the extend solution routine from Section VI-C.2 by considering handovers of objects between two stations, see Lemma 2. Again an efficient caching scheme can be used to speed up the computation.

The second improvement targets assignments that are obtained while extending the solution. Given a suboptimal solution for the DC problem, we can improve it by altering the assignment of objects to stations. If the *supporting point* of a center y_i is also covered by another disk y_j, we assign the second furthest object of y_i as the new *supporting point* of y_i. This reduces the radius of y_i and therefore the total area of the disks. We can repeat this process until no further improvement is possible.

Finally, we reduce the time spent for extending and combining solutions. This can be done by simultaneously extending and combining the stationary solution (which is better than the current solution) only until it intersects the current solution. The influence of both improvement strategies is evaluated in Section VII.

VII. Evaluation

Experiments were carried out on Linux desktop workstations with AMD Ryzen 9 7900 ($12\times3.7\,\mathrm{GHz}$) CPUs and $96\,\mathrm{GB}$ of RAM running Ubuntu 24.04.3 LTS. Code and data with benchmark problems and results are available [19]. We use the Gurobi [16] solver for the integer program, with a default optimality gap of 0.01% and a time limit of $600\,\mathrm{s}$. To detect numerical issues for degenerate cases, we added checks to ensure correctness of our implementation. A version that uses exact number types for finding roots of the quadratic equations was also tested, see Section VII-D for details. We formulate the following research questions:

- RQ1 Which of the proposed improvement strategies is effective in practice?
- RQ2 How does the heuristic and IP perform for different values of m and n?
- RQ3 How does the difficulty of instances change when considering degenerate cases?
- RQ4 How does the algorithm perform on instances from literature?

For the instance generation, we used line segments with lengths uniformly distributed in $[25, 50]$ in a 100×100 grid. In real-world applications where objects are drones have velocities up to $100\,\mathrm{km\,h^{-1}}$, this resembles scenarios that take between 15 and 30 minutes to play out on a $100\,\mathrm{km}^2$ canvas. In total, we have the following instances sets.

- `fix` We generate 25 instances with $m = 25$ and $n = 500$.
- `fix_n` We fix the number of objects to $n = 500$. For each number of sensors $m \in \{5, 10, \ldots, 50\}$ we generate 10 instances.
- `fix_m` We fix the number of sensors to $m = 25$. For each number of objects $n \in \{50, 100, \ldots, 500\}$ we generate 10 instances.
- `pub` Instances from well-known publicly available benchmarks used in [12]. These include point sets from the CG:SHOP challenges [9], [10], TSPLIB instances [21], instances from a VLSI dataset [22] and point sets from the Salzburg Database of Polygonal Inputs [11]. We scaled point sets to $1\,\mathrm{m}$ resolution, such that they have a diameter of $100\,\mathrm{km}$ and sampled $m = 25$ centers uniformly at random. The minimum and maximum length of trajectories was set to $25\,\mathrm{km}$ and $50\,\mathrm{km}$. This yields a graph G with all possible trajectories as edges. We compute a random maximum cardinality matching in G, yielding $\lfloor \frac{N-25}{2} \rfloor$ objects in almost all cases. As this yielded some degenerate cases in which many trajectories had the same length, we added a small random perturbation to the trajectories to break ties.

A. RQ1: Improvement Strategies

We evaluate the influence of three proposed improvement strategies to the algorithm from Section VI. We denote the removal of duplicates as `NoDup`, the improved extension according to handovers from Lemma 2 as `ImpExt` and partial extension as `PartExt`. Our baseline is the exact algorithm that uses the IP solver from Section V-B. We compare the total runtime of the different components of the algorithm, see Figure 7, i.e., the time to compute the stationary IP solutions and the time to extend and combine solutions. We see that all improvement strategies have a positive effect on the IP solving time. Our experiments show however, that both `ImpExt` and `NoDup` have a negative effect on the time to extend and combine solutions which in

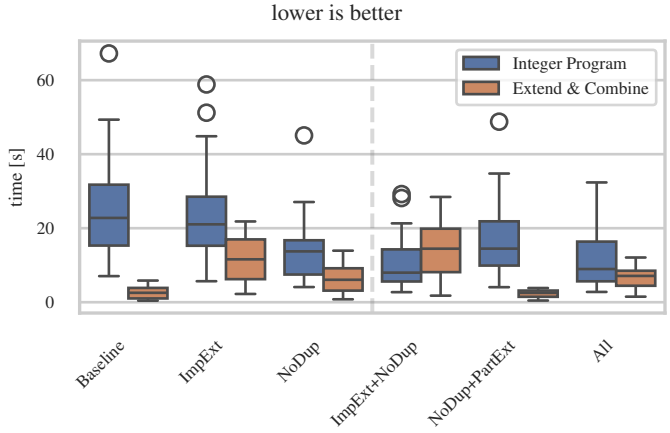


Fig. 7: Evaluation of different improvement strategies on the `fix` benchmark set. The results show that enabling all three solution strategies yields the best performance.

case of `ImpExt` results in a higher overall runtime compared to the baseline.

The use of `PartExt` however reduces the time spent during extension and combination significantly, which mitigates the negative effect of the other two strategies while still reducing the time spent in the IP solver. This results in the best configuration that enables all three improvement strategies and achieves an overall runtime reduction of about 39% compared to the baseline.

B. RQ2: Influence of m and n

We evaluate the influence of the number of objects n and the number of stations m on the performance of the algorithm. Our comparison uses the best configuration from Section VII-A. We execute the algorithm from Section VI with both an IP solver and the nearest neighbor heuristic from Section V to solve the DC problem in each iteration. We denote the IP-based algorithm as `IP` and the nearest neighbor based algorithm as `NN`. Additionally, we implemented another heuristic that divides the time $[0, 1]$ into $k = 10$ evenly spaced intervals and computes the nearest neighbor solution at each border. The algorithm `FixedNN` then extends the solutions using the routine from Section VI-C.2 and reports the lower envelope of all computed solutions. We note that one instance of `IP` on the `fix_n` benchmark was restarted with a different seed due to numerical instability in the non-exact extension step that caused our checks to fail.

We evaluate the performance of the different algorithms on `fix_m` and `fix_n` to access the influence of m and n on the overall runtime and quality of solutions, see Figure 8. For the `fix_m` benchmark, we see that `IP` produces optimal solutions for instances with up to 500 objects in less than $30\,\mathrm{s}$. The `NN` heuristic is significantly faster, solving all instances within 6 seconds. Although `NN` is faster, it produces solutions with an average optimality gap of about 20% for smaller instances and up to 45% for larger instances. The `FixedNN` is slightly better in terms of solution quality than `NN` with an average optimality gap of around 20% for all

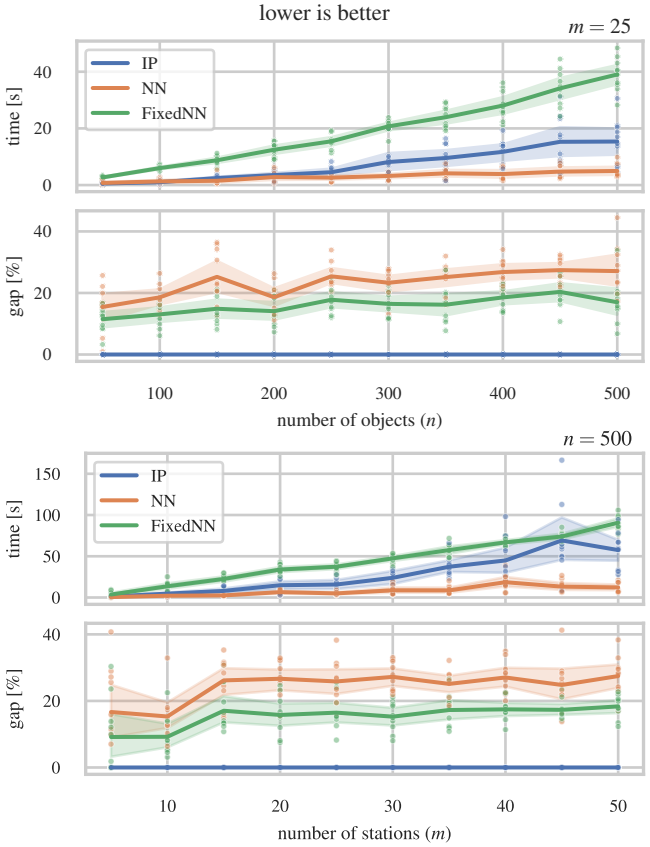


Fig. 8: Performance of `IP`, `NN` and `FixedNN` on instances from the `fix_m` and `fix_n` datasets. While `IP` is significantly slower than `NN`, it produces optimal solutions for all instances. `FixedNN` is slightly better in terms of solution quality than `NN` but significantly slower.

instances but is significantly slower with an average runtime even higher than `IP`. For the `fix_n` benchmark, we see a similar trend but observe a higher influence of m on the performance of the algorithms. In particular doubling the number of stations from $m = 25$ to 50 for $n = 500$ objects more than doubles the maximum runtime of `IP` from about $25\,\mathrm{s}$ to about $70\,\mathrm{s}$.

In summary, `IP` is able to solve all instances to optimality in a matter of seconds, demonstrating real-time capability for scenarios that take minutes to play out in the real world.

C. RQ3: Degenerate Cases

In many real-world scenarios, objects may move in a coordinated manner. Planes often follow established air corridors, and drones may operate in formations or along predefined paths. We evaluate the performance of the best algorithm `IP` from Section VII-B on on four classes of degenerate instances; one with uniformly random trajectories (`fix`), one where all have the same slope, and two where all have the same start or end point, see Figure 9.

We observe that instances with same slope are slighly easier to solve. On the other hand instances where all trajectories start or end at the same point are significantly easier to solve with all instances being solved within seconds.

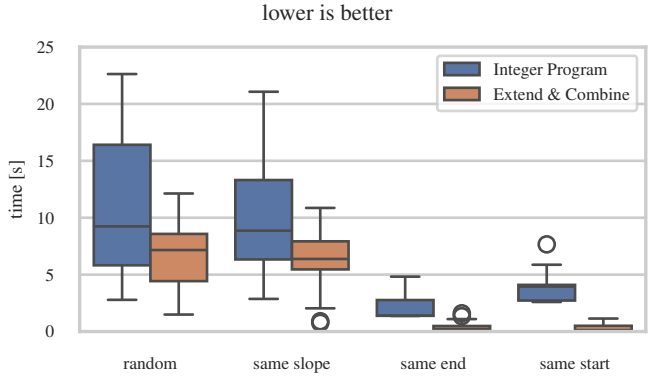


Fig. 9: Runtime comparison between degenerate cases: uniformly random trajectories, same slope, same start point, and same end point. Degenerate instances are consistently easier to solve than random instances, suggesting that real-world instances may be more tractable. Note that this plot is clipped; one random and one same-slope instance of the integer program (with runtimes of 33 s and 38 s) exceed the displayed range.

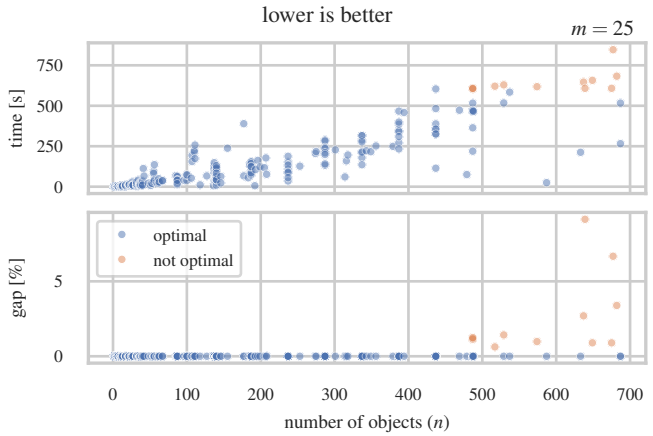


Fig. 10: Performance of the exact version of `IP` on the `pub` dataset. Despite the variety and degeneracies, almost all instances can be solved to provable optimality.

D. RQ4: Instances from Literature

We further assess the algorithm's performance on the `pub` instance set, as shown in Figure 10 and illustrated by Figure 11. The presence of degeneracies originating from real-world data, led to various numerical challenges in our implementation, see Section VII-B. To address these issues, we developed an alternative version of `IP` utilizing exact number types for computations, following Lemmas 1 and 2. The additional overhead of exact number types results in slightly higher runtimes than in the benchmarks reported in Section VII-B. However, our results show that 290 out of 302 instances with up to 700 objects could be solved to provable optimality within 600 s of computation time.

E. Real-World Perspectives

While our results establish both hardness and optimal solvability for the $\min\max$ variant of the KDC under idealized assumptions, several additional factors need to be addressed for real-world scenarios. Overall, our results establish theoretical limits and algorithmic potential under idealized geometric assumptions. Practical deployment requires additional considerations for practical issues such communication delays, uncertainty models, and possibly online adaptation mechanisms.

a) Modeling Abstraction: Our formulation is purely geometric and assumes deterministic trajectories known in advance. We do not model sensor dynamics (e.g., delays when adjusting ranges), communication constraints, or decentralized coordination. In practical UAV or air-traffic systems, range updates may incur latency, bandwidth limitations, or partial communication failures. Such effects may delay handovers or introduce temporary coverage gaps, potentially increasing peak power consumption relative to the idealized optimum. In principle, our algorithm remains applicable under moderate latency if event times are buffered with safety margins; however, practical performance guarantees may degrade accordingly.

b) Real-Time Adaptivity: Although polynomial, the event-based enumeration is evaluated only in an offline setting with pre-known trajectories. In adaptive scenarios with trajectories revealed incrementally or updated online, the recomputation cost may limit scalability. This is where incremental variants are still necessary for large-scale deployments, at the expense of global optimality guarantees.

c) Dimensionality and Physical Effects: Our work focuses on 2D planar coverage and ideal sensing disks to demonstrate basic viability. Real-world systems operate in 3D, for which additional effects like altitude and occlusion may affect coverage. Extending the geometric insights to 3D is possible; practical application needs to address increased computational complexity and further structural properties of optimal solutions.

d) Randomness and Uncertainty: Our experiments demonstrate possibilities for relatively simple trajectories. They do not explicitly model stochastic motion, sensing noise, or localization error. In practice, uncertainty in object position or velocity needs to address robustness margins, effectively enlarging sensing regions and reducing achievable power savings.

VIII. Conclusions

We provide novel insights into tracking large sets of moving objects from a set of fixed observation stations with optimal energy. While this problem is hard in theory, our practical methods can compute provably optimal coverage in a matter of seconds, demonstrating perspectives for real-world capability. There are many promising directions for future work; deployment requires addressing the described practical issues, such as communication delays, online trajectory updates, and three-dimensional coverage. If the stations can be repositioned, the problem becomes even more complex, but also opens up new possibilities for optimization that may further reduce energy consumption. Uncertainty in trajectories and alternative coverage models, such as probabilistic sensing, also present interesting avenues for further research that could enhance the robustness and applicability of our methods in real-world scenarios.

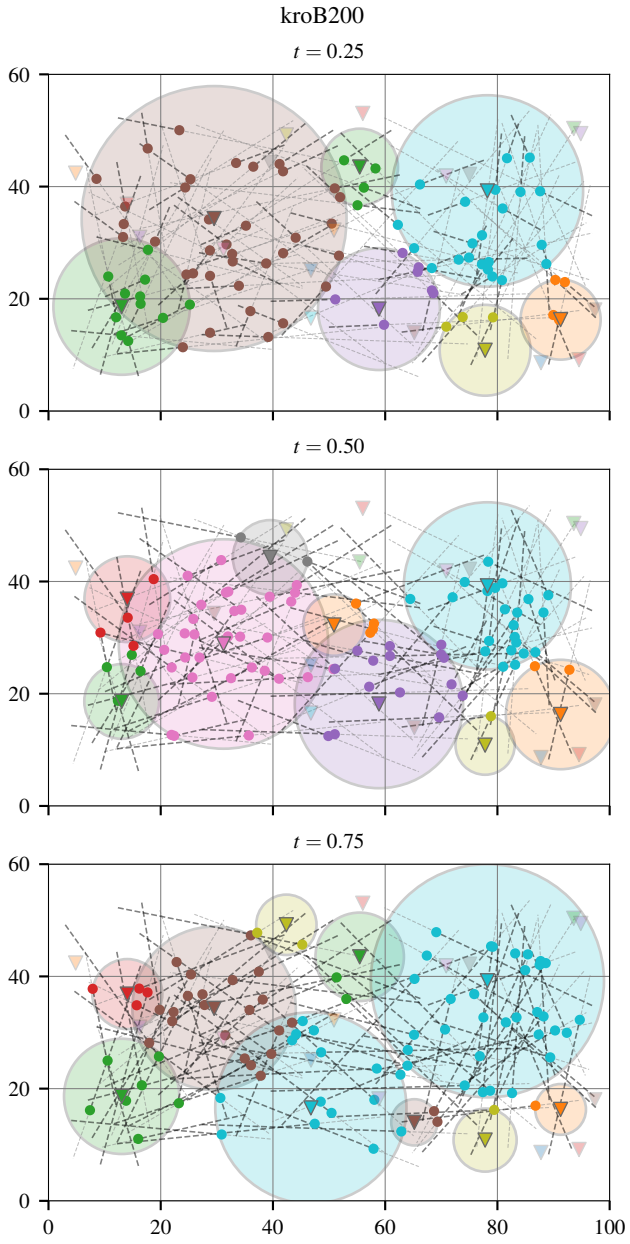


Fig. 11: A solution of an instance from `pub` instance set based on the *kroB200* TSPLIB [21] point set at $t = 0.25, t = 0.5$ and $t = 0.75$.

References

[1] A. K. Abu-Affash, P. Carmi, M. J. Katz, and G. Morgenstern, "Multi cover of a polygon minimizing the sum of areas," *International Journal of Computational Geometry & Applications*, vol. 21, no. 06, pp. 685–698, 2011.

[2] H. Alt, E. M. Arkin, H. Brönnimann, J. Erickson, S. P. Fekete, C. Knauer, J. Lenchner, J. S. B. Mitchell, and K. Whittlesey, "Minimum-cost coverage of point sets by disks," in *Symposium on Computational Geometry (SoCG)*, 2006, pp. 449–458.

[3] R. Bar-Yehuda and D. Rawitz, "A note on multicovering with disks," *Computational Geometry*, vol. 46, no. 3, pp. 394–399, 2013.

[4] J. Basch, L. J. Guibas, and J. Hershberger, "Data structures for mobile data," *J. Algorithms*, vol. 31, no. 1, pp. 1–28, 1999.

[5] C. Bautista-Santiago, J. M. Díaz-Báñez, R. F. Monroy, D. Flores-Peñaloza, D. Lara, and J. Urrutia, "Covering moving points with anchored disks," *Eur. J. Oper. Res.*, vol. 216, no. 2, pp. 278–285, 2012.

[6] S. Bespamyatnikh, B. K. Bhattacharya, D. G. Kirkpatrick, and M. Segal, "Mobile facility location," in *International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M)*, 2000, pp. 46–53.

[7] S. Bhowmick, K. Varadarajan, and S.-K. Xue, "A constant-factor approximation for multi-covering with disks," in *Symposium on Computational Geometry (SoCG)*, 2013, pp. 243–248.

[8] A. P. Cohen, S. A. Shaheen, and E. M. Farrar, "Urban air mobility: History, ecosystem, market potential, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 9, pp. 6074–6087, 2021.

[9] E. D. Demaine, S. P. Fekete, P. Keldenich, D. Krupke, and J. S. B. Mitchell, "Computing convex partitions for point sets in the plane: The CG:SHOP Challenge 2020," *arXiv preprint arXiv:2004.04207*, 2020.

[10] ——, "Area-optimal simple polygonalizations: The CG Challenge 2019," *Journal of Experimental Algorithmics (JEA)*, vol. 27, no. 2, pp. 1–12, 2022.

[11] G. Eder, M. Held, S. Jasonarson, P. Mayer, and P. Palfrader, "Salzburg database of polygonal data: Polygons and their generators," *Data in Brief*, vol. 31, p. 105984, 2020.

[12] S. P. Fekete, P. Keldenich, and M. Perk, "Exact Algorithms for Minimum Dilation Triangulation," in *Symposium on Computational Geometry (SoCG)*, 2025, pp. 48:1–48:18.

[13] S. M. Filipov and F. N. Tomova, "Covering a set of points with a minimum number of equal disks via simulated annealing," in *Numerical Methods and Applications*, I. Georgiev, M. Datcheva, K. Georgiev, and G. Nikolov, Eds., 2023, pp. 134–145.

[14] X. Gao, L. Guo, and K. Liao, "Fast approximation algorithms for multiple coverage with unit disks," in *Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2022, pp. 185–193.

[15] M. Guitouni, C. Loi, S. P. Fekete, M. Perk, and A. T. Becker, "Multi-covering a point set by m disks with minimum total area," in *ICRA*. IEEE, 2025, pp. 3000–3006.

[16] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2025. [Online]. Available: https://www.gurobi.com

[17] Z. Huang, Q. Feng, J. Wang, and J. Xu, "PTAS for minimum cost multi-covering with disks," in *Symposium on Discrete Algorithms (SODA)*, 2021, pp. 840–859.

[18] ——, "PTAS for minimum cost multicovering with disks," *SIAM Journal on Computing*, vol. 53, no. 4, pp. 1181–1215, 2024.

[19] C.-M. Loi and M. Perk, "Drone air traffic control: Tracking a set of moving objects with minimal power," 2025. [Online]. Available: https://doi.org/10.5281/zenodo.17119781

[20] F. Mazzenga, R. Giuliano, and A. Vizzarri, "5g-based synchronous network for air traffic monitoring in urban air mobility," *IEEE Access*, vol. 12, pp. 188 542–188 559, 2024.

[21] G. Reinelt, "TSPLIB–A Traveling Salesman Problem Library," *ORSA Journal of Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[22] Rohe, Andre, "VLSI data set." [Online]. Available: https://www.math.uwaterloo.ca/tsp/vlsi/index.html