Multi-Robot Dynamic Swarm Disablement

Ori Fogler[1] and Noa Agmon[2]

***Abstract*— Motivated by the use of robots for pest control in agriculture, this work introduces the *Multi-Robot Dynamic Swarm Disablement* problem, in which a team of robots is required to disable a swarm of agents (for example, locust agents) passing through an area while minimizing the cumulative time of the swarm members (equivalent to the cumulative damage they cause) in the area. Showing that the problem is hard even in naive settings, we turn to examine algorithms seeking to optimize the robots' performance against the swarm by exploiting the known movement pattern of the swarm agents. Motivated by the poor performance when a weak group of robots attempts to catch a large swarm of agents, whether it is a significant numerical minority or poor speed gaps, we suggest the use of *blocking lines*: the robots form lines that block the agents along their movement in the environment. We show by both theoretical analysis and rigorous empirical evaluation in different settings that these algorithms outperform common task-assignment-based algorithms, especially for limited robots versus a large swarm.**

I. Introduction

Teams of robots can be used for various purposes, among them cleaning [24], delivery [14] and security [20]. Recently, the use of robots has become more common in agriculture applications, for harvesting, packing and pest control [10]. Motivated by the use of robots to prevent a locust swarm from causing severe damage to crops, we introduce in this paper the *Dynamic Swarm Disablement* (DSD) problem.

In the DSD problem, a team of robots, each equipped with a disabling tool, is faced against a swarm of agents (such as individual locusts) in an attempt to disable them while minimizing the accumulated damage they cause as they pass through an area (feeding on crops). Once a locust agent is placed within the sensing/disablement range of some robot, it is disabled and removed from the system. While the robots act as a coordinated, knowledgeable team, the agents follow a swarm-model, having very simple local behavior (specifically, they follow a fixed course) and limited knowledge of the world. Although having the advantage of knowledge, coordination and computation, the robots are extremely outnumbered by the agents, raising the interesting question of determining their optimal behavior facing this mass of agents. Hence, our goal is to define paths for the robots such that they jointly minimize the accumulated damage caused by the agents before they are disabled (or leave the area), equivalent to minimizing the sum of times all agents are active in the area.

[1]Ori Fogler is with the Department of Computer Science, Bar-Ilan University, Israel (email: foglerori@gmail.com)

[2]Prof. Noa Agmon is with the Department of Computer Science, Bar-Ilan University, Israel (email: agmon@cs.biu.ac.il)

The problem draws similarities to several canonical robotic and optimization problems, such as *multi-robot patrolling* [2], [7], [26], *coverage* [4], [16], *task-assignment* [19] and *TSP* [1], [13], yet it raises complex and innovative challenges due to the mixture of the various fields it touches.

We prove that if we treat the DSD problem as an assignment problem, in which we allow robots to move anywhere in the area and intercept agents individually, the problem is *NP-hard*. Because of this, and inspired by the poor performance of current task-assignment approaches [8] when it comes to a limited group of robots facing a large swarm (that is, many tasks), whether there are large numerical gaps or low capabilities of the robots, we consider a new approach in which we construct *blocking lines* based on the agents' known direction of progress. We suggest several algorithms for placing those lines under different settings, and theoretically prove their optimality within their context in terms of yielding minimal accumulated damage caused by the agents. We then perform an extensive empirical evaluation, comparing the performance of the blocking-line algorithms to benchmark assignment-based algorithms, and show their effectiveness especially when the robots are inferior in terms of numbers or velocity compared to the agents.

II. Related Work

The Dynamic Swarm Disablement (DSD) problem shares common characteristics with some canonical problems from the worlds of robotics and optimization.

Recent work with close motivation to ours is the *Multi-robot Containment and Disablement* (CAD) problem by Maymon et al. [22]. In the CAD problem, a team of robots attempts to contain a swarm of agents preventing any single agent from expanding and then reduces the size of the contained area while preserving an enclosure. The motivation for both problems is very similar, as we try to disable the swarm and stop their progress in order to avoid extensive damage, but still there are significant differences between the problems. In the CAD problem, the agents' movement model is a random walk, so the robots enclose the agents from all directions before moving inside and disabling them. In contrast, in the DSD problem the movement model is in a known direction, and thus the solution concept utilizes those to perform blocking only on one side.

A possible approach to deal with the problem of swarm disablement is to place the robots on a closed polygon that surrounds the agents in their current locations or in the direction of their movement, and then apply methods from *multi-robot perimeter patrol* in order to disable agents that try to pass through the perimeter [6], [30]. Indeed, our

problem seems related to the patrol problem, although there are differences that need to be addressed. Regarding the optimization criteria, while patrolling attempts to maximize the (expected) number of blocked agents, in our case we try to minimize the amount of damage that the swarm causes. In addition, unlike patrolling a fence, we want to allow robots to move in the space to prevent more damage.

An additional task that draws similarities with the DSD is the *multi-robot pursuit-evasion* problem. Pursuit-evasion is a game between (one or more) pursuers that attempt to capture (one or more) evaders, where those wish to evade being captured for as long as possible. The pursuit-evasion problem is examined vastly in the literature, discussing, among others, efficient pursuing strategies for the pursuers, seeking the evaders (e.g., [18], [25]). However, in addition to the difference in the optimization criteria, these works differ from ours in the fact that they do not deal with swarms of weak robots and do not utilize any knowledge regarding the movement model of the agents.

One of the most challenging problems of multi-robot systems is how to optimally assign a set of tasks in such a way that optimizes the overall system performance. This problem is known as *multi-robot task allocation* (MRTA) problem. There are few deterministic and nondeterministic optimization methods that have been used in order to solve task allocation problems [3], [23], in addition to market-based methods that gained considerable attention within the robotics research community [17], [29] because of their efficiency, robustness and scalability. While our problem seems at first glance as a case of the MRTA problem, there are difficulties that prevent us from using the common solutions to the problem. The costs of the tasks in the DSD problem are accumulated, since the goal is to minimize the cumulative damage. Furthermore, the costs are also dependent, as the cost of some robot to perform an agent disablement varies if it is moving and disabling another agent meanwhile. This dependency makes our problem more difficult.

III. The DSD Problem

In this section we present the *Multi-robot Dynamic Swarm Disablement* problem and discuss its hardness.

A. Problem Definition

We are given an $2D$-environment W, a team of k homogeneous robots $R = \{r_1, \ldots, r_k\}$ and a group of n agents $A = \{a_1, \ldots, a_n\}$. Robot $r_i \in R$ is located at time t in the Cartesian coordinates $p_i^t := (x_i^t, y_i^t)$, and the set of R-robots' positions at time t is denoted by $P_t = \{p_1^t, \ldots, p_n^t\}$. Similarly, an agent a_i is located at time t in $\tilde{p}_i^t$, and the set of all A-agents' locations at time t is denoted by $\tilde{P}_t = \{\tilde{p}_1^t, \ldots, \tilde{p}_k^t\}$. Following these notations, the initial positions of the robots and the agents are $P_0 = \{p_1^0, \ldots, p_n^0\}$ and $\tilde{P}_0 = \{\tilde{p}_1^0, \ldots, \tilde{p}_n^0\}$, respectively. Denote also the function $dist(a, b)$ as the euclidean distance between two points, a and b,g and $dist_w(a, b) = dist((a, b) \cap W)$ as the distance within the environment W.

The robots are homogeneous, equipped with disablement tool of range d, s.t. robot r_i disables an agent a_j at time t if and only if $dist(p_i^t, \tilde{p}_j^t) \leq d$. In addition, there is a central planner that plans the paths of the robots given the initial world state so that each robot knows to track its planned movement. However, the swarm agent model assumes limited capabilities of sensing and communication of the agents. Therefore, agents make their decisions with limited knowledge of the world.

In the current version of the problem, we assume the agents' behavior model to be a straight movement with constant velocity v and movement direction which is, without loss of generality, always upwards (similar to a locust-swarm movement). The robots instead can move in any direction and they are faster than the agents by a factor of $f > 1$.

The *damage* caused by an agent a_i up to time t is formally defined as:

$$d_i^t := \int_{j=0}^{t-1} dist_w(\tilde{p}_i^j, \tilde{p}_i^{j+1})\, dt$$

Similarly, the damage caused by the entire swarm agents up to time t is defined by:

$$C_w(t) := \sum_{i=1}^{n} d_i^t = \sum_{i=1}^{n} \int_{j=0}^{t-1} dist_w(\tilde{p}_i^j, \tilde{p}_i^{j+1})\, dt$$

Since the swarm moves up a vertical line, the damage caused by an agent a_i can also be written as:

$$d_i^t := \tilde{y}_i^t - \tilde{y}_i^0$$

Similarly, again, the damage caused by the entire swarm can be written as:

$$C_w(t) := \sum_{i=1}^{n} d_i^t = \sum_{i=1}^{n} (\tilde{y}_i^t - \tilde{y}_i^0) = \sum_{i=1}^{n} \tilde{y}_i^t - \sum_{i=1}^{n} \tilde{y}_i^0$$

The damage model assumes independence in the damage amount for different visits by agents at the same location.

Therefore, the *DSD* problem is defined as follows:

> Given an area of interest W, a team of k robots $R = \{r_1, \ldots, r_k\}$ with their initial locations P_0, and a swarm of n agents $\{a_1, \ldots, a_n\}$ with initial locations $\tilde{P}_0$ and a specific movement model, find a path for each robot in R such that the accumulated damage caused by A in W is minimized, that is, $\min \lim_{t\to\infty} C_w(t)$.

B. Problem Hardness

The most general form of the optimization problem, i.e. sending the k robots to disable the n moving agents with minimum total delay, requires solving a kinematic version of the well-known *Traveling Repairman Problem* (TRP) [1]. In the TRP we are given a finite set of points and the travel times between any two of them, and we wish to find the route through the points which minimizes the sum of the delays for reaching each point. This problem is closely related to the canonical *Traveling Salesman Problem*, and it is not surprising that the TRP is also NP-complete [28].

Many variants of TSP have been well studied, discussing their hardness and the possibility to approximate them, including kinetic variants. For example, [12] study the approximation complexity of kinetic TSP with a fixed constant speed in a fixed direction and prove some bounds on the approximation factor of them. In particular, they prove that the *kinetic TSP* or the *moving-targets TSP* is hard. We use a similar reduction as seen in [5] to prove that general *kinetic-TRP* and *fixed-velocity and fixed-direction kinetic-TRP* are NP-Hard. This important result is summarized below[1].

Theorem III.1. *The general kinetic-TRP and the fixed-velocity and fixed-direction kinetic-TRP are both NP-hard.*

Following the hardness of the problem, we suggest arranging the robots as a blocking structure in the swarm's movement direction instead of sending them to conduct chases separately. When we come to block the swarm, we make a distinction between two types of scenario, the *full-blockage* case and the *partial-blockage* case. In the full blockage scenario, the number of the robots k and their range of disablement d are sufficient to fully cover the world in the horizontal axis. This condition occurs when $2dk$ is greater than or equal to the greatest distance between agents on the horizontal axis. In the partial blockage scenario, on the other hand, the promise of the possibility of a complete blockage on the horizontal axis is not fulfilled. As a result, such a scenario must be considered separately and other methods need to be developed for it.

IV. The Full-blockage Case

In this section we consider the DSD problem for the scenario in which the dispersion of the agents and the number of robots are enough to cover the relevant range on the horizontal axis. We want to perform the optimal assignment of the robots that creates the blocking line, and then find the optimal movement of the robots to their positions and possibly their later movement with the line.

Denote by $M := \{m_1, \ldots, m_k\}$ the set of *movements* (paths) for each robot, such that m_i is a set of locations and times: $< p_{i_1}, t_{i_1} >, < p_{i_2}, t_{i_2} >, \ldots$ dictating the path for robot r_i. Correspondingly, the optimization of DSD can be described as $\arg\min_M \lim_{t\to\infty} C_w(t)$.

A. Static Line

The most basic configuration for a blocking line is a *static line* that covers the entire range of agents' positions on the horizontal axis in the direction of their movement, and once the robots reach their positions on the line they remain stationary. In this case, we determine the optimal movement M of the robots to the optimal line.

We assume that the agents are initially placed above the robots, that is, $\max(y_1^0, \ldots, y_k^0) \leq \min(\tilde{y}_1^0, \ldots, \tilde{y}_n^0)$. This is reasonable since in many cases the robots start their mission when they are off the field, and it is important to ensure the optimality of the assignment as discussed later. Denote b as the height of the northern border of the world W. Figure 1 illustrates a possible instance of a static line solution.

[1]Full Proofs are omitted due to space constraints; can be found in [9]

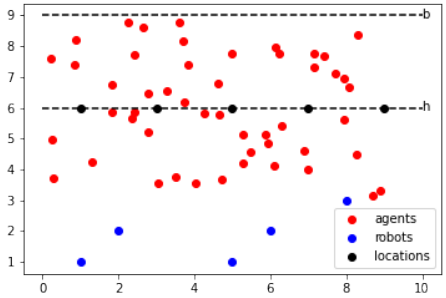


Fig. 1: A DSD instance with a full blocking line, and the locations of the robots along the blocking line.

Denote the x value of the leftmost and rightmost agents by $\tilde{x}_{min} = \min(\tilde{x}_1^0, \ldots, \tilde{x}_n^0)$ and $\tilde{x}_{max} = \max(\tilde{x}_1^0, \ldots, \tilde{x}_n^0)$. These values represent the relevant range to consider on the x-axis. We also define $\tilde{y}_{min}$ as the agents' minimal y value and y_{max} as the robots' maximal y value. To ensure complete blockage, the blocking line should be between $\tilde{x}_{min}$ and $\tilde{x}_{max}$. For simplicity we neglect fractions and say that the length of the blocking line is $\tilde{x}_{max} - \tilde{x}_{min}$ and the needed number of robots to construct the blocking line is $\frac{\tilde{x}_{max} - \tilde{x}_{min}}{2d}$. In this section, as stated above, we refer only to the case where the number of robots is sufficient for the construction of a full blocking line ($k \geq \frac{\tilde{x}_{max} - \tilde{x}_{min}}{2d}$).

Algorithm 1: StaticLine($P_0, \tilde{P}_0, b$)

Result: movement M

```
// minimal makespan assignment on bottom line
```
$loc_i := (\tilde{x}_{min} + d + 2d \cdot i, \tilde{y}_{min}), 0 \leq i < \frac{\tilde{x}_{max} - \tilde{x}_{min}}{2d}$
$Distances := \{dist(loc_i, p_j^0) : 0 \leq i < \frac{\tilde{x}_{max} - \tilde{x}_{min}}{2d}, j \in [k]\}$
$Costs := \{2^{\text{index of dist in } Sorted(Distances)} : dist \in Distances\}$
$(p_1^l, \ldots, p_k^l) \leftarrow HungarianMethod(P_0, \{loc_i\}, Costs)$

```
// farthest robot from its position
```
$m := \arg\max_{i \in [k]} dist(p_i^0, p_i^l)$

```
// lines' heights to consider
```
$H := \{\frac{f^2\tilde{y}_i^0 - y_m^0 + \sqrt{(f\tilde{y}_i^0 - fy_m^0)^2 + (x_m^l - x_m^0)^2(f^2-1)}}{f^2-1} : i \in [n]\}$

```
// disabled agents for a given line
```
$A_{disabled}(h) := \{i : \tilde{y}_i^0 \leq h - \frac{\sqrt{(x_m^l - x_m^0)^2 + (h - y_m^0)^2}}{f}, i \in [n]\}$

```
// minimizing damage score
```
$D(h) := \sum_{i \in [n]} (b - \tilde{y}_i^0) - |A_{disabled}(h)|(b - h)$
$h^* \leftarrow \arg\min_{h \in H} D(h)$

$M \leftarrow ((x_1^l, h^*), \ldots, (x_k^l, h^*))$

StaticLine algorithm gets the initial positions of the robots and the agents, P_0 and $\tilde{P}_0$, in addition to a *boundary* line b that limits the area, and determines the movement M, that is, the movement of the robots to their optimal assignment on the optimal blocking line h^*.

The steps of the algorithm are as follows. First, the algorithm determines the optimal assignment of the robots, up to the height of the line, according to some line above the robots (e.g., $h = \tilde{y}_{min}$). The optimal assignment is the one that minimizes the organization time, a.k.a. *makespan*, which is the maximal time it takes for a robot to reach its place. This can be done by solving an augmentation of the *assignment problem* (explained below). After the assignment we know the identity of the farthest robot that determines the makespan time. The algorithm accordingly defines a set of relevant lines H that include the optimal solution, selects the line h^* that minimizes the damage function $D(h)$ and returns the movement of the robots to that line with the optimal x values that were previously calculated.

Below we mention lemmas that prove the algorithm's correctness (some details are omitted due to space constraints).

The claim that the optimal assignment is the same for any line above the robots is justified by the following lemma.

Lemma IV.1. *The optimal assignment of the robots on a line is preserved for all lines that are higher than the robots.*

Proof: (sketch) As long as all the lines are on one side of the robots (up or down), the order between the distances between the robots and the locations on the lines is maintained, and therefore the optimal assignment is maintained.

The assignment task itself is a fundamental combinatorial optimization problem, with a well-known polynomial-time algorithm to solve it, the *HungarianMethod* [19]. The time complexity of the original algorithm is $O(n^4)$, but was later reduced to $O(n^3)$ by a few modifications [15]. The assignment costs are the distances between the robots and the target locations on the given line, however, we do not want to minimize the sum of the distances but only the maximal distance, as it determines the construction time. This gap is bridged, similar to the solution concept presented in [7], by a mapping between the sorted distances and rising exponents of 2, exploiting the useful property of $2^{n+1} > 2^{n+1} - 1 = \sum_{i=0}^{n} 2^i$ which causes the order of the sums to be determined by the greatest element in them.

Once we have determined the optimal assignment, it remains to analytically find the finite set H, which contains the lowest possible line to catch the agent a for each $a \in A$. The lines are examined by $D(h)$, a function that measures the damage for a given line h by the damage of the blocked agents and the damage of the escaping agents. An optimal line is guaranteed to be in H as the following lemma proves.

Lemma IV.2. *H contains the globally-optimal static horizontal line that minimizes the agents' accumulated damage.*

Proof: (sketch) Suppose that h^* is an optimal solution that captures agents $A_{disabled}(h^*)$. There is a minimal line $h' \in H$ that captures the same group of agents $A_{disabled}(h^*)$, and since $h' \leq h^*$, as h' is the lowest line to catch $A_{disabled}(h^*)$, we have that $|A_{disabled}(h')|(b - h') \geq |A_{disabled}(h^*)|(b - h^*)$ and thus $D(h') \leq D(h^*)$.

The correctness of *StaticLine* algorithm follows directly from the lemmas.

Theorem IV.3. *StaticLine algorithm returns the movement (assignment) of the robots on a static horizontal line, minimizing the accumulated damage caused by the agents.*

The complexity of *StaticLine* algorithm is derived from the complexity of *HungarianMethod* on the k robots and their target locations, which is $O(k^3)$ (or less, using other implementations for the *assignment problem*), assuming that the number of the robots on the blocking line is also $O(k)$ and the $O(n)$ complexity of referring to the agents for each of the n lines in H. In total, $O(k^3 + n^2)$.

B. Traveling Line

Now we consider horizontal lines with the possibility of vertical movements. Because the robots move faster than the agents by a factor of f and they are not limited to stay in static positions, the robots can possibly reach all the agents.

Under these settings, we can calculate a non-trivial movement M that is optimal based on a variant of the *LineTRP* [1]. The LineTRP is a case of TRP in which the points are positioned on a straight line, and in [1] the authors present a polynomial *dynamic programming* solution for it. In our case, the blocking line turns the robots' movements to be only in one (vertical) dimension, similar to the LineTRP.

Algorithm 2: TravelingLine($P_0, \tilde{P}_0$)

Result: movement M
$(x_1^l, \ldots, x_k^l), m$ and H are calculated as in algorithm *StaticLine*

`// makespan time`
$makespan(h) := \frac{\sqrt{(x_m^l - x_m^0)^2 + (h - y_m^0)^2}}{fv}$

`// minimizing damage score`
$D(h) := LineTRPv(\tilde{P}_{makespan(h)}, h) + n \cdot makespan(h) \cdot v$
$h^* \leftarrow \arg\min_{h \in H} D(h)$

$M \leftarrow$
$((x_1^l, h^*), \ldots, (x_k^l, h^*)) \oplus LineTRPv(\tilde{P}_{makespan(h^*)}, h^*)$

The algorithm calculates the optimal assignment in terms of the x values, the farthest robot m and the set of relevant lines H exactly as *StaticLine*. The only change is the score function $D(h)$, which is calculated by the modified *LineTRP* solution, in addition to the damage that is caused during the reorganization before the moving line begins to operate.

The modified *LineTRP*, named *LineTRPv*, receives the updated positions of the agents $\tilde{P}_{makespan(h)}$, the line h and returns the optimal path and its damage. The points of the agents are sorted vertically and separated to the agents that below or on a given line h, denoted by $A_{down} = (d_0, d_1 \ldots)$ (descending), and the agents that above h, denoted by $A_{up} = (u_0, u_1, \ldots)$ (ascending). The state of the problem is uniquely represented by $[d_i, u_j]$ ($[u_j, d_i]$), which means that the repairman or the line is currently at d_i (u_j) and the range that was already visited is between d_i and u_j. The speed of the repairman is increased by an additional v when traveling against the velocity of the agent, and the opposite when traveling in the same direction. A table D containing

the optimal values of the paths for each end-state can be calculated by the following equations:

$$D[d_1, u_0] = n \cdot \frac{h - d_1}{fv + v},\ D[u_1, d_0] = n \cdot \frac{h - u_1}{fv - v},$$

$$D[d_i, u_0] = D[d_{i-1}, u_0] + la \cdot \frac{d_{i-1} - d_i}{fv + v},$$

$$D[u_j, d_0] = D[u_{j-1}, d_0] + la \cdot \frac{u_j - u_{j-1}}{fv - v},$$

$$D[d_i, u_j] = \min(D[d_{i-1}, u_j] + la \cdot \frac{d_{i-1} - d_i}{fv + v}, D[u_j, d_{i-1}] + la \cdot \frac{u_j - d_i}{fv + v}),$$

$$D[u_i, d_j] = \min(D[u_{i-1}, d_j] + la \cdot \frac{u_i - u_{i-1}}{fv - v}, D[d_j, u_{i-1}] + la \cdot \frac{u_i - d_j}{fv - v})$$

The indexes i, j are the frontier indexes of A_{down}, A_{up} respectively, and la is the number of living agents, which is $n + 1 - i - j$. The optimal solution is $\max(D[d_{last}, u_{last}], D[u_{last}, d_{last}])$ since the end state of the optimal path is $[d_{last}, u_{last}]$ or $[u_{last}, d_{last}]$. The complexity of filling the table is $O(n^2)$ as its size.

An optimal line is guaranteed to be in H (proof omitted here), and the correctness of *TravelingLine* follows directly.

Theorem IV.4. *TravelingLine algorithm returns a movement of the robots as a horizontal line with a vertical movement minimizing the accumulated damage caused by the agents.*

The complexity of *TravelingLine* is $O(n^3 + k^3)$ due to *HungarianMethod*, as before, and due to the calls of n to *LineTRPv* when implemented in time of $O(n^2)$.

C. Separate Traveling

Instead of forcing robots to move together in a naive horizontal line structure, it may sometimes be beneficial to treat different areas of the environment differently. For example, it might be useful to divide the space into consecutive vertical "buckets" so that in each of them a single robot is a full blocking line of its own and behaves optimally with respect to its own section. Separation can only yield a better outcome, since it examines the options that are examined without separation, and in addition it adds the ability to perform different movements in different buckets.

The following procedure partitions the range of the x-values of the agents into consecutive buckets of width $2d$, determines the behavior of a single robot in each bucket and chooses the robot-to-bucket assignment.

Algorithm 3 consists of the following steps. The procedure first divides the horizontal range of the agents into consecutive buckets of width $2d$, assuming no remainder for simplicity, and runs *TravelingLine* for each robot for each bucket, that is $k \cdot \frac{\tilde{x}_{max} - \tilde{x}_{min}}{2d}$ runs. Each run of *TravelingLine* computes the movement and the damage cost for the specific robot handling the specific bucket. By doing so we get the costs for each robot-to-bucket assignment, and the optimal combination can be found by running the original *HungarianMethod* on these costs.

The complexity of *SeparateTraveling* is $O(k^2 n^3 + k^3)$ because the $O(k^3)$ of *HungarianMethod* for k robots and k buckets, and the use of *TravelingLine* for the assignment scores that happens k^2 times (suppose that the buckets' number is within the order of the robots' number), multiplied by the complexity of *TravelingLine* itself, that is $O(n^3)$ since there are at most $O(n)$ agents in a bucket. Indeed, by sharing the internal *lineTRPv* optimal path information between the *TravelingLine* runs for the different robots on the same bucket, the complexity can be reduced to $O(kn^3 + k^3)$.

Algorithm 3: SeparateTraveling($P_0, \tilde{P}_0$)

Result: movement M

```
// partition of the agents into 2d-buckets
B_i := {a_j : x̃_j^0 ∈ [x̃_min + 2di, x̃_min + 2d(i+1))},
  0 ≤ i < (x̃_max − x̃_min)/2d

// optimal robot-to-bucket assignment
Tasks := {B_i}
Costs := {cost of TravelingLine(p_i^0, B_j)
  : i ∈ [k], 0 ≤ j < (x̃_max − x̃_min)/2d }
L ← HungarianMethod(P_0, Tasks, Costs)

M ← movements of TravelingLine corresponding to L
```

V. The Partial-blockage Case

In this section we consider the DSD problem for the scenario in which the dispersion of the agents and the number of the robots are not suitable for a full coverage of the agents' range on the horizontal axis, that is, $k < \frac{\tilde{x}_{max} - \tilde{x}_{min}}{2d}$.

A. Static Line Lack

In the current scenario it is not possible to find a full blocking line, and thus we consider static lines that are static only in vertical movements, while the robots can move on them horizontally in order to perform as many blocks as possible (which is equivalent to minimizing the damage since any penetration causes the same amount of damage).

Similar to the *multi-robot perimeter patrol* technique of *reachability graph* [6], we construct a graph that encapsulates all the possible blocking options and run a *min-cost-max-flow* algorithm to obtain the movement of the robots (valid flow) that optimizes the number of caught agents (min-cost). We had to make some adjustments in the graph because in our case the robots are not initially placed on the perimeter where the perceptions happen. Figure 2 shows an example of a modified reachability graph and how to build it.

We call this modified version *FlowMoves*, and it returns the optimal horizontal movement of the robots for a given line, as well as additional information such as the identity of the blocked agents. The modifications are trivial.

In contrast to the full blockage case where the optional lines are derived by the arrival time of the farthest robot which is the same robot r_{max} for all the lines, in our case the robots can have different initial allocations for different lines and thus the identity of the farthest robot is not necessarily preserved. The way we overcome this is by adding more line candidates, which are the minimal height lines that block each single agent for each possibility of choosing a robot to disable this particular agent. This chosen robot determines

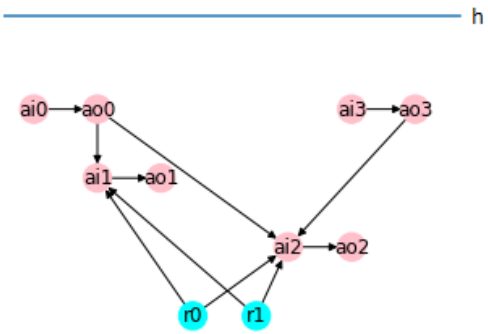


Fig. 2: Reachability graph example (using *networkx* [11] package). The robots r_0 and r_1 need to block the agents $a_0, .., a_3$ on the blue line h. There is an edge between robot r_i and agent a_j if r_i can block a_j on h, and there is an edge between agents if a robot that blocks the first can later move and block the second. The agents are divided into *in* and *out* vertices to simulate a vertex cost.

uniquely the height of the minimal line to catch the particular agent, instead of the farthest robot in the full blocking lines. Therefore, the number of lines in H is $O(kn)$, and optimality is still achieved from some line within it.

Algorithm 4: StaticLineLack($P_0, \tilde{P}_0, b$)

Result: movement M

// heights for each agent and robot
$H :=$
$\{\frac{f^2\tilde{y}_i^0 - y_j^0 + \sqrt{(f\tilde{y}_i^0 - fy_j^0)^2 + (x_j^l - x_j^0)^2(f^2-1)}}{f^2-1} : i \in [n], j \in [k]\}$

// disabled agents for a given line
$A_{disabled}(h) :=$
$\{i :$ a_i agent is caught by $FlowMoves(P_0, \tilde{P}_0, h),\ i \in [n]\}$

// minimizing damage score
$D(h) := \sum_{i\in[n]} (b - \tilde{y}_i^0) - |A_{disabled}(h)| \cdot (b - h)$
$h^* \leftarrow \arg\min_{h\in H} D(h)$

$M \leftarrow$ paths of $FlowMoves(P_0, \tilde{P}_0, h^*)$

Algorithm 4 works as follows. *StaticLineLack* first calculates the relevant lines to consider H, that is, the minimum height lines that disable an agent a_i for each choice of robot to reach this agent. Then the *FlowMoves* procedure is run on each line to check which agents are caught and which are not for the best blocking that the line is capable of. The best line h^* is then selected taking into account the damage derived from the disabled and the escaping agents. Finally, once we find the optimal line, we can retrieve the optimal movements by the (cached) answer of *FlowMoves* on h^*.

An optimal line is guaranteed to be in H since it includes all the relevant options (proof omitted here), and the correctness of *StaticLineLack* algorithm follows directly.

Theorem V.1. *StaticLineLack algorithm returns a movement of the robots on a horizontal line that minimizes the accumulated damage caused by the agents.*

Algorithm *StaticLineLack* uses a run of *FlowMoves* method to determine the score of each line among the $O(kn)$ lines in H, which is the runtime complexity of $O(knVE^2) = O(kn(k+n)(kn+n^2)^2)$ assuming the runtime complexity of $O(VE^2)$ of the *min-cost-max-flow* method.

B. Separate and Additive Line Lack

Two other heuristic versions that we briefly mention are the *SeparateStaticLineLack* and *AdditiveLineLack* algorithms. In both versions the computation time is reduced by limiting the time expensive procedures to only a relatively small groups of agents instead of the whole group.

In *SeparateStaticLineLack* we divide the space into k vertical buckets, where in each bucket one robot is assigned and performs a horizontal patrol using a reachability graph (using a linear time *longest-path* query on DAG instead of flow). The assignment is done in the same way as *Separate-Traveling*, examining any robot and bucket combination and choosing the best assignment of robots to buckets.

In the second solution, *AdditiveLackLine*, we perform a *horizontal* division such that each time *StaticLineLack* is applied on a group with a fixed number of agents in the order of the agents from the highest to the lowest. The reason that the blocking starts in a top-down way is to take advantage of the agents' movement by coming towards the blocking line and not moving away from it. Unlike *StaticLineLack*, the robots are not necessary staying on one single horizontal line, but choosing a new line for each group of agents.

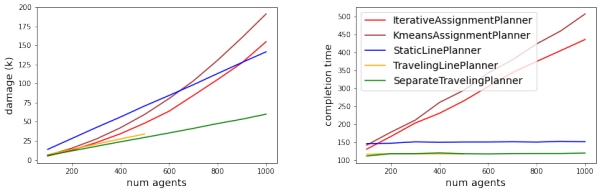
VI. Empirical Analysis

We have implemented the algorithms for the full coverage scenario and the partial coverage and tested their results[2]. In addition, we also implemented baseline algorithms that represent the simple approach to the task-allocation problem based on [8], referred to as *assignment-based* benchmark, in order to emphasize the performance of the blocking lines[3].

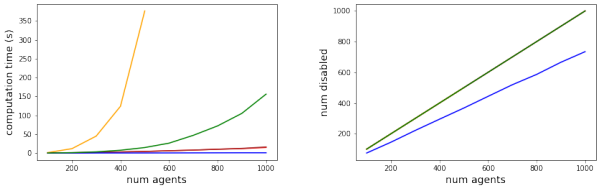
In the first assignment-based algorithm, named *Iterative-Assignment*, we perform an iterative assignment by reapplying *HungarianMethod*. After each assignment of k robots and agents, the states of the remaining robots and agents are updated before assignment in the next iteration. Clearly, the optimal assignment in each iteration is a greedy choice that can differ from the global optimal solution. For example, the optimal solution can assign all agents to a single close robot, which is not the case when iterative assignment is used.

Another assignment-based algorithm we have examined, *KmeansAssignment*, divides the agents into k clusters by the well-known *kmeans* clustering algorithm [21], with the movement within each cluster being greedily done to the nearest agent. The assignment of the robots to the clusters is done by the *HungarianMethod*. Unlike the *IterativeAssignment* procedure, this method tries to delegate separate responsibilities to each robot, which in some cases can be an advantage and in other cases a disadvantage depends on the distribution of the agents, outliers and more.

[2]On i3-4170 CPU, by Python3 without further interpreter optimizations.
[3]The implementations can be found at https://github.com/Elilgo324/dsd.

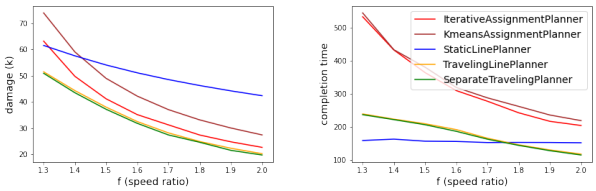


(a) Accumulated damage (b) Task completion time



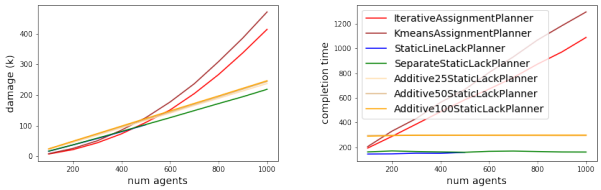
(c) Planner runtime in seconds (d) Number of disabled agents

Fig. 3: Full blockage as a function of the number of the agents.

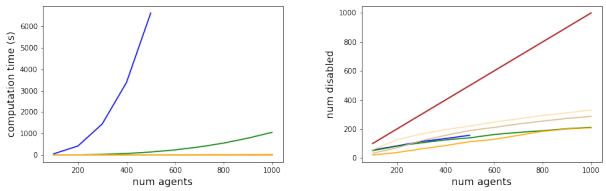


(a) Accumulated damage (b) Task completion time

Fig. 4: Full blockage as a function of the speed ratio.



(a) Accumulated damage (b) Task completion time



(c) Planner runtime in seconds (d) Number of disabled agents

Fig. 5: Partial blockage as a function of the number of the agents.

A. *Full Blockage Evaluation*

The basic configuration of the experiment is a rectangular field with a height that is 3 times the width. The positions of the agents are evenly drawn in the lowest third square in the field and their direction of movement is upwards. The locations of the robots are sampled below the borders of the world. Unless otherwise stated, the number of the agents n is 300 and the speed ratio f is 2. In a full-blockage scenario, the number of the robots is the minimum number of robots sufficient for horizontal coverage, which is the x-range of the agents divided by the disablement range d, rounded.

We have compared the blocking algorithms in the full-blockage scenario, along with the assignment-based algorithms. Interesting stats for comparison include the damage amount, the task completion time (the elapsed time until the last disablement happens), the computation times of the planners and the number of the disabled agents.

1) Agent Number Analysis: We have tested different sizes of swarms. The results are summarized in figure 3.

Figure 3a shows that the damage caused by the assignment-based algorithms, *IterativeAssigment* and *KmeansAssignment*, is close to the damage caused by the other algorithms when the number of agents in relatively small. The more agents that are added to the environment, the gaps between the blocking-based and the assignment-based algorithms become more and more significant. The completion times of the blocking algorithms remain almost the same, as figure 3b shows, while they increase consistently for the assignment algorithms because of the chases that the robots conduct. Among the blocking algorithms, the *SeparateTraveling* algorithm manages to avoid damage the best. The *TravelingLine* shows relatively close performance, but its runtime exceeded the experiment threshold at some point. As seen in figure 3c, the runtime of *SeparateTraveling* also grows significantly with the number of agents, but remains relatively reasonable.

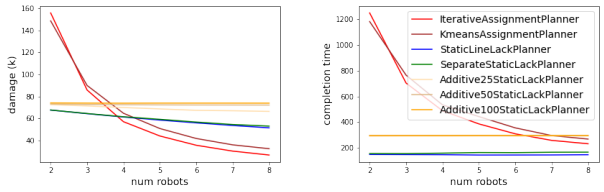
The *StaticLine* algorithm performs worse than the assignment algorithms for a small number of agents but outperforms them as the number of agents increases. Figure 3d shows that indeed all the algorithms eventually capture all the agents, except the *StaticLine* which gives up on some. This fact can disprove a possible misconception that the more agents you disable the more damage you prevent. The fact that *StaticLine* performs well even though it disables fewer agents, can be an advantage if minimizing the number of disablement actions is an objective.

2) Speed Ratio Analysis: We have tested different velocity ratio of the robots compared to the velocity of the agents. The results are summarized in figure 4.

From figures 4a and 4b it can be understood that the velocity of the robots is more critical for the assignment algorithms than the blocking methods. When the robots try to chase the agents instead of passing and blocking them, they have a longer way to travel that requires higher speed. This is another advantage of blocking algorithms, which provide a good solution even for relatively slow robots. Even the *StaticLine* algorithm on a relatively limited number of agents outperforms the baseline algorithms in the damage manner for a low robot-agent speed ratio.

B. *Partial Blockage Evaluation*

We also empirically tested the algorithms for the case where we cannot fully cover the horizontal range of agents. To complete the picture about the effectiveness of the blocking lines, we present the performance of the algorithms for different sizes of swarms and for different numbers of robots.



(a) Accumulated damage (b) Task completion time

Fig. 6: Partial blockage as a function of the number of robots.

1) Agent Number Analysis: As for the case of full blockage lines, figure 5a shows that the blocking lines outperform the assignment algorithms regarding the damage avoidance starting from a certain size of swarm. In addition, mission completion times are shorter and stable, despite the superior performance in terms of avoiding damage. As seen in figure 5d, the blocking algorithms give up on more agents, relatively, as the swarm grows, and yet the performance versus the assignment algorithms gets better.

Among the blocking lines, *StaticLineLack* and *SeparateStaticLack* perform very well, and the *AdditiveLineLack* versions (which differ in the number of agents in each vertical cell) are slightly behind. However, the runtime of *StaticLineLack* is relatively high, and the runtime of *SeparateStaticLack* is also significant as figure 3c shows. In contrast, the runtime of the *AdditiveLineLack* versions is almost negligible. The task completion times of *AdditiveLineLack* are higher than those of other blocking algorithms but do not increase as those of the assignment algorithms.

2) Robot Number Analysis: Figure 6 illustrates well the dependence of assignment-based approaches on the number of robots available for assignment, compared to the blocking-line methods that perform well even for a relatively small number of robots. For the assignment methods, the accumulated damage decreases and the completion time increases significantly as more robots are added to the system, while the gaps are much more limited for the blocking methods.

VII. Conclusions and Future Work

In this work we have introduced the *Multi-robot Dynamic Swarm Disablement* problem and showed its hardness. We have presented a variety of blocking-line solutions, based on analytical calculations, classic optimization problems and flow networks, and showed their optimality relative to other options of blocking lines. Empirical comparisons with other current approaches substantiate the effectiveness of the solutions, especially for limited robots versus a large swarm.

We would like to continue the research to find better solutions with proven optimality, especially for the case where the number of robots is not enough for a full blockage. In particular, we would like to generalize the use of solving the TRP problem in one dimension for two dimensions as well, with the additional dimension divided into buckets, by adjusting the TSP problem solution for parallel lines [27]. Reinforcement-learning methods are another direction of progress, as well. In addition, we would like to examine the suitability of the solutions or the development of additional solutions for more complex agents' movement models involving additional noise and uncertainty.

References

[1] Foto Afrati, Stavros Cosmadakis, Christos H Papadimitriou, George Papageorgiou, and Nadia Papakostantinou. The complexity of the travelling repairman problem. *RAIRO*, 20(1):79–87, 1986.

[2] Noa Agmon, Sarit Kraus, and Gal A Kaminka. Multi-robot perimeter patrol in adversarial settings. In *ICRA*, pages 2339–2345. IEEE, 2008.

[3] Nuzhet Atay and Burchan Bayazit. Mixed-integer lp solution to mrta problem. Technical report, St. Louis, Tech. Rep, 2006.

[4] Maxim A Batalin and Gaurav S Sukhatme. Spreading out: A local approach to multi-robot coverage. In *DARS*. Springer, 2002.

[5] Avrim Blum, Prasad Chalasani, Don Coppersmith, Bill Pulleyblank, Prabhakar Raghavan, and Madhu Sudan. The minimum latency problem. In *ACM*, pages 163–171, 1994.

[6] Austin K Chen, Douglas G Macharet, Daigo Shishika, George J Pappas, and Vijay Kumar. Optimal multi-robot perimeter defense using flow networks. In *DARS*, pages 282–293, 2021.

[7] Yehuda Elmaliach, Noa Agmon, and Gal A Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Mathematics and Artificial Intelligence*, 57(3):293–320, 2009.

[8] Jan Faigl, Miroslav Kulich, and Libor Přeučil. Goal assignment using distance cost in multi-robot exploration. In *IROS*. IEEE, 2012.

[9] Ori Fogler and Noa Agmon. Multi-robot dsd (full version with proofs). `https://u.cs.biu.ac.il/~agmon/DSDFull2022.pdf`.

[10] Pablo Gonzalez-de Santos et al. Fleets of robots for pest control in agriculture. *Precision Agriculture*, 18(4):574–614, 2017.

[11] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network using networkx. Technical report, LANL, 2008.

[12] Mikael Hammar and Bengt J Nilsson. Approximation results for kinetic variants of tsp. In *ICALP*, pages 392–401. Springer, 1999.

[13] Karla Hoffman et al. Tsp. *Encyclopedia of operations research and management science*, 1:1573–1578, 2013.

[14] Seohyun Jeon et al. Multi-robot task allocation for real-time hospital logistics. In *SMC*, pages 2465–2470. IEEE, 2017.

[15] Roy Jonker and Ton Volgenant. Improving the hungarian assignment algorithm. *Operations Research Letters*, 5(4):171–175, 1986.

[16] Nare Karapetyan, Kelly Benson, Chris McKinney, Perouz Taslakian, and Ioannis Rekleitis. Efficient multi-robot coverage of a known environment. In *IROS*, pages 1846–1852. IEEE, 2017.

[17] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. Multi-robot task allocation: A review of the state-of-the-art. *Cooperative Robots and Sensor Networks 2015*, pages 31–51, 2015.

[18] Andreas Kolling and Stefano Carpin. Multi-robot pursuit-evasion without maps. In *ICRA*, pages 3045–3051. IEEE, 2010.

[19] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[20] Yi-Lin Liao and Kuo-Lan Su. Multi-robot-based intelligent security system. *Artificial Life and Robotics*, 16(2):137–141, 2011.

[21] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[22] Yuval Maymon and Noa Agmon. Multi-robot containment and disablement. In *IROS*, pages 11724–11731. IEEE, 2020.

[23] Alejandro R Mosteo and Luis Montano. Simulated annealing for multi-robot hierarchical task allocation with flexible constraints and objective functions. In *IROS*. Citeseer, 2006.

[24] Agris Nikitenko, Janis Grundspenkis, Aleksis Liekna, Martins Ekmanis, Guntis Kulikovskis, and Ilze Andersone. Multi-robot system for vacuum cleaning domain. In *PAAMS*, pages 363–366. Springer, 2014.

[25] Trevor Olsen, Nicholas M Stiffler, and Jason M O'Kane. Robust-by-design plans for multi-robot pursuit-evasion. *arXiv:2109.08715*, 2021.

[26] Yaniv Oshart, Noa Agmon, and Sarit Kraus. Non-uniform policies for multi-robot asymmetric perimeter patrol in adversarial domains. In *MRS*, pages 136–138. IEEE, 2019.

[27] Günter Rote. The n-line tsp. *Networks*, 22(1):91–108, 1992.

[28] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *JACM*, 23(3):555–565, 1976.

[29] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial intelligence*, 101(1-2):165–200, 1998.

[30] Stephen L Smith, Shaunak D Bopardikar, and Francesco Bullo. A dynamic boundary guarding problem with translating targets. In *CDC*, pages 8543–8548. IEEE, 2009.