Multi-Robot Adversarial Patrolling: 
Facing Coordinated Attacks

Efrat Sless  
Department of Computer Science  
Bar Ilan University, Israel  
efrat.slas@live.biu.ac.il

Noa Agmon  
Department of Computer Science  
Bar Ilan University, Israel  
agmon@cs.biu.ac.il

Sarit Kraus  
Department of Computer Science  
Bar Ilan University, Israel  
sarit@cs.biu.ac.il

ABSTRACT

The use of robot teams is common for performing patrol tasks, in which the robots are required to repeatedly visit a target area (perimeter, in our case) controlled by an adversary, in order to detect penetrations. Previous work has focused on determining the optimal patrol algorithm when facing a general adversary that tries to penetrate once through the patrol path. There, the robots’ goal is to detect penetrations, i.e., the robots do not change their behavior once a penetration is detected. Requiring the robots to physically inspect penetration attempts can have far reaching consequences on the performance of the patrol algorithm. Specifically, it creates vulnerability points along the patrol path that a knowledgeable adversary can take advantage of. In this work we investigate the problem of coordinated attacks, in which the adversary initiates two attacks in order to maximize its chances of successful penetration, assuming a robot from the team will be sent to examine a penetration attempt. We suggest an algorithm that computes the optimal robot strategy for handling such coordinated attacks, and show that despite its exponential time complexity, practical run time of the algorithm can be significantly reduced without harming the optimality of the strategy.

Categories and Subject Descriptors

I.2.11 [Distributed Artificial Intelligence]: Multiagent systems

General Terms

Algorithms

Keywords

Agent Cooperation::Multi-robot systems; Agent-based system development::Security aspects of agent systems;

1. INTRODUCTION

The problem of multi-robot patrol is well known in the robotic and agent community. Given an area and a team of robots, the robots should repeatedly visit the area in order to either maximize point-visit frequency criterion (e.g., [4, 5]) — frequency based patrol — or maximize the robots’ chance of detecting penetrations through the patrol path (e.g., [1, 3]) — adversarial patrol. In this paper we concentrate on the latter, i.e., the robots’ goal is to detect penetrations that are controlled by an adversary, in our case around a perimeter.

Previous work on multi-robot patrol has concentrated mainly on the detection problem [1, 3, 13]. There, the robots’ task is to detect penetrations, i.e., once a penetration is detected, the robots report it and move on. The robots’ world model does not change, thus their behaviors do not change. Therefore, the efficiency of the patrol strategy remains the same for any number of penetration attempts — coordinated or uncoordinated. However, in many realistic settings, the robot that detects the penetration has to handle it as well, for example by confining the penetrator or examining the penetration using additional sensors. Thus the penetration will necessarily influence the remaining team’s ability to handle them, which makes the patrolling team more vulnerable to coordinated attacks.

When the adversary initiates a coordinated attack, it will try to penetrate through one location, and then (once a robot is removed from the team in order to handle that penetration attempt) it will try to penetrate again, this time through vulnerability points generated by the need to handle the first penetration. We distinguish between the behavior of the robots in three different phases: (Phase I) the steady-state prior to the extraction of the robot for handling the penetration, i.e., patrolling by $k$ robots (Phase II) The reorganization phase: the team transfers from having $k$ patrolling robots to having $k-1$, and (Phase III) The steady-state of patrolling by $k-1$ robots. Agmon et al. [1] established the optimal patrol algorithm for phases I and III. However they did not consider handling the penetration attempts and thus did not need to reorganize the robots. In this paper we examine the optimal behavior during the reorganization phase (Phase II), in which the robots are most vulnerable to additional attacks. One of the difficulties in finding the robots’ strategies in this phase is that the robots are not symmetric; one needs to compute specific strategies for each agent. This is in contrast to the steady phase where all agents follow the same strategy.

As the vulnerability points of the patrolling robots exist during the reorganization phase, a natural solution would be one that minimizes the duration of this phase. Surprisingly, we prove that this is not true. On the contrary, the shortest reorganization phase is inevitably deterministic and thus most vulnerable. Moreover, we show that as the reorganization time grows, the minimal probability of penetration detection along the perimeter grows until it levels up with probability of penetration detection of the steady state. Therefore, this allows us to focus our attention on computing an optimal strategy and limit the computation time.

In this paper we propose an algorithm that determines the optimal patrol strategy for the robots during the reorganization phase. The strategy is based on providing a probability distribution over all possible paths for each robot, leading it from its initial position to its final position (preparing for the next steady state). Unfortunately, the number of such paths is exponential in the size of the perimeter. However, we have shown that it is unnecessary to examine all possible paths, but only a small fraction of them, leading
to another significant drop in the computation time of an optimal solution.

2. RELATED WORK

The problem of multi-robot patrolling has received considerable attention in the literature during the past decade. The problem has been investigated in different areas, for example perimeter patrol [1], patrol along an open fence [6], a 2D continuous environments [5] and graphs [4, 9, 10]. The problem has also been examined with respect to two different perspectives of the patrolling robots: adversarial patrol [1, 3, 11], where the robots' goal is to detect penetrations controlled by an adversary, and frequency-based patrol [4, 6, 9, 10], where the robots' goal is to optimize some point-visit frequency criterion, for example by maximizing the average frequency along the area. Another perspective is the game-theoretic domain [3, 13], which refers to a single robot in the patrol, but they also formulated their problems as non-linear optimization problems. Another approach is modeling the problem as a Stackelberg game, as in [14] which confronted a similar problem. They introduced a method of representing the exponential number of paths. However, in our work we assume that the penetration is not instantaneous, therefore we cannot use this representation. Moreover, they assume that the contribution of multiple patrol units covering the same edge is additive, thus the problem can be formulated as a linear programming problem, whereas in our settings multiple robots covering the same segment contribute the same as a single robot.

In this paper we concentrate on the problem of multi-robot adversarial perimeter patrol. In these types of problems, the adversarial model commonly depends on the adversarial knowledge. An adversary with full knowledge was considered in [1, 3], and an adversary with partial information was considered for example in [12].

In [2] it was shown that when the adversary has no knowledge about the patrolling robots and acts randomly, an algorithm maximizing the probability of penetration detection is the one that maximizes the point-visit frequency. Yet, to the best of our knowledge, the problem of coordinated attacks, i.e., handling more than one attack, has not been considered in previous.

The problem of handling events in frequency based patrol was considered, among others, by Elmaliach et al. [5] and by Fazli and Mackworth [7]. Jensen et al. [8] examined the problem of removing robots for re-charging. These events are predictable, and can be taken into consideration in advance as opposed to adversaries' attacks. Elmaliach et al. [5] considered the removal of robots due to failure, where they focused on minimizing the reorganization time. In all these cases, the robots' goal is to adjust to the new state with one less robot as quickly and efficiently as possible, by adopting a deterministic behavior. This behavior is necessarily vulnerable to additional penetration attempts by a knowledgeable adversary, thus cannot be used in our case.

3. PROBLEM SETTINGS

We are given $k$ homogeneous robots $R = \{r_0, \ldots, r_{k-1}\}$ that are required to patrol along a perimeter, i.e., a cyclic path around a closed polygon. The fence is partitioned into $N$ segments $S = \{s_0, \ldots, s_{N-1}\}$, with endpoints $ep_0, \ldots, ep_{N-1}$, where each robot travels through one segment per time unit. $s_i$ starts at $ep_i$ and ends at $ep_i + 1$. The length of the segments and the robots' speed can differ from one segment to another, as long as the motion time required to travel through the segments is equal. Each robot can either move forward, backwards or stay put, at every step, where turning around has an associated cost, modeled in time. We denote the number of time units it takes the robots to turn around by $\tau$. When an attempt to penetrate is detected, either by a robot or other sources, a robot must inspect it until the threat is removed, thus the robot is removed from the team. During that time it is still able to observe part of the perimeter (this value is quantified in Section 4).

3.1 Adversarial Model

The system consists of two coordinated adversaries $A_0, A_1$ (equivalent to one adversary initiating two attacks). Their joint goal is for at least one of them to successfully penetrate the perimeter, that is, without being detected by the robots. Both adversaries earn the same utility if they manage to achieve their joint goal. We assume the time it takes for each one of the adversaries to penetrate lasts $t$ time units, during which it may be detected by a robot passing through the segment in which it resides in. In order to maximize their chances of successful penetration, the adversaries initiate a second penetration attempt if the first one is detected. We consider two cases for the second penetration with respect to the detection of the first attack:

1. **Known penetration period.** Second penetration initiated at some known time, or bounded time $[x_1, x_2]$ after the first penetration is detected.

2. **Unknown penetration time.** Second penetration is initiated at an unknown time.

We consider a strong adversarial model, in which the adversaries have full knowledge of the patrol strategy and the robots' positions and a-priory decide the segments to penetrate and the timing of both attacks.

3.2 Patrol Task

We define the Probability of Penetration Detection in a segment $s_i$, $ppd_i$, as the probability that an adversary trying to penetrate through $s_i$ during $t$ time units will be detected by some robot passing through $s_i$ during that time period. We would like to find an optimal patrol algorithm for the robot team when confronting a strong adversary that will necessarily penetrate through the segment with minimal $ppd$, i.e., an algorithm that maximizes the minimal probability of penetration detection throughout the perimeter. Agmon et al. [11], proved that the optimal patrol algorithm that maximizes the minimal $ppd$ along the perimeter for a team of $k$ robots is one that requires the robots to be spread out uniformly (in time) along the perimeter, i.e., with a distance of $d_t$ between every two consecutive robots. In their work, they define a nondeterministic patrol framework in which, at each time step, the robots either continue straight with a probability of $p$ or turn around with a probability of $1-p$. Based on this framework, it is possible to find, in polynomial time, the optimal patrol strategy (characterized by the value $p$). We therefore distinguish between the behavior of the robots in three different phases: Phase I- the steady-state prior to the extraction of the robot for handling the penetration, i.e., patrolling by $k$ robots; Phase II- the reorganization phase, where the team’s composition changes from $k$ patrolling robots to $k-1$ patrolling robots; and Phase III- the steady-state of patrolling by $k-1$ robots. Since the optimality of the patrol strategy is established in [11] for phases I and III, in this paper we focus our attention on the reorganization phase. We therefore handle the following problem:

Given $k$ robots patrolling around the perimeter, where one robot is extracted from the team in order to handle a penetration at location $s$, determine the optimal behavior for the robots that will maximize the minimal probability of penetration detection along the perimeter.
4. THE REORGANIZATION PHASE

When a robot is extracted from the team to handle the first penetration, we wish to reorganize the remaining \( k-1 \) robots such that they will again be organized uniformly around the perimeter, to achieve optimal behavior [1], while the extracted robot remains close to the location of the initial penetration. The set of final positions of the robots is the set of positions in which all robots are placed uniformly along the perimeter. First, we are interested in determining the amount of time allocated for the reorganization phase, i.e., the time it will take the robots to reach their final positions. The duration of the reorganization phase is denoted by \( \rho \).

Note that the length of the path between a robot’s current position and its final position varies from one robot to the other, and depends on its location relative to the penetration point (see Figure 1). A naive approach would be for each robot to go straight to its final position and wait until the reorganization phase ends, thus minimizing the time of the reorganization. However such an approach will create vulnerability points as there are segments that will not be visited during \( t \) time units, thus a knowledgeable adversary will manage to penetrate through those weak points. In the example in Figure 1, all points between the final positions of the robots and the current positions of the adjacent robots will not be visited during the reorganization phase using this naive algorithm. Thus these points can be chosen for penetration by the adversary. We are therefore interested in a non-deterministic algorithm that will guarantee that for each segment \( s_i \) around the perimeter, \( \text{ppd}_i > 0 \) during the reorganization phase.

**Determining final positions.**

It is easy to see that as \( \rho \) grows, each robot has a greater range of segments it can visit during the reorganization phase, thus \( \text{ppd}_i \) potentially increases. Hence, theoretically, we would like \( \rho \) to have large values. However, this conflicts with the fact that we wish to minimize the reorganization phase length since an optimal patrol is achieved when all robots are organized uniformly. In order to find the \( \rho \) value that balances this trade-off, we start by finding the final positions for all the robots such that the reorganization time is minimized, i.e., the maximal distance traveled by some robot is minimized. We assume, without loss of generality, that \( r_0 \) is the extracted robot. The uniform distance between \( m \) robots is denoted by \( d_m = \frac{\min}{m} \).

We also use \( D_i \) to denote the path length for robot \( i \), \( 1 \leq i \leq k-1 \) from its current position to its final position (which provides the minimal path length). The sign of \( D_i \) indicates the direction of the path: + for counter clockwise movement, - for clockwise. We begin by proving the following supporting lemma in order to find the final positions (by calculating the values of \( D_i \)’s). Let \( D_{\text{max}} := \max_{1 \leq i \leq k-1} |D_i| \).

**Lemma 1.** For \( k \) robots that are spread uniformly around the perimeter with a distance of \( d = N/k \) segments between every two consecutive robots, once robot \( r_0 \) is removed from the system, \( D_j = D_1 - \frac{k-1}{k} d_k \) for \( 2 \leq j \leq k-1 \), and \( D_{\text{max}} = D_1 = \frac{1}{2} \frac{k-1}{k} d_k \).

**Proof.** After reorganizing the distance between robot \( r_j \) and \( r_{j-1} \), \( j=2,k-1 \) is \( d_k + D_{j-1} - D_j \). The robots are distributed uniformly along their final positions, hence they are within a distance of \( \frac{k-1}{k} d_k \) from each other, therefore \( d_k + D_{j-1} - D_j = \frac{k-1}{k} d_k \), and we find that \( D_j = D_{j-1} - \frac{k-1}{k} d_k \). It is easy to verify that \( D_j = D_1 - \frac{k-1}{k} d_k \).

In order to show that the maximal value of \( D_j (D_{\text{max}}) \), we notice that there are two robots \( r_1, r_{k-1} \), within a distance of \( 2d_k \), and the rest of the robots remain within a distance of \( d_k \). Since \( 2d_k \geq \frac{k-1}{k} d_k \) for all \( k \geq 2 \), the minimal duration of reorganizing is achieved when those two robots will head in opposite directions in order to reduce the distance between them.

Thus \( D_1 + D_{k-1} = 2d_k - \frac{k-1}{k} d_k = \frac{k-1}{k} d_k \) holds. The minimum path length is achieved when \( D_j = D_{k-1} = \frac{k-1}{k} d_k \). Hence the minimal of the longest path length for all robots, denoted by \( \min \), satisfies that \( \min \geq \frac{k-1}{2} d_k \). The maximum of \( |D_j| \), is achieved at the boundary points \( j = 2, k-1 \) and \( |D_2| < |D_{k-1}| \) holds, therefore \( D_{\text{max}} = \max|D_j| = D_1 = D_{k-1} = \frac{k-1}{k} d_k \). Since \( D_{\text{max}} \) is the longest path length it holds that \( \min \leq D_{\text{max}} \), hence \( \min = D_{\text{max}} \). \( \square \)

![Figure 1: Illustrating of the deterministic reorganization phase.](image)

**Monitoring requirement from the extracted robot.**

We are interested in visiting all segments during all periods of \( t \) time units. We wish to examine the range of segments close to the penetration location that cannot be visited during that time by the remaining robots, but can be observed by the extracted robot, or any other sensing element. The number of segments that cannot be visited during a period of \( t \) time units starting from \( t_1 \) equals \( 2(d_k - (t_1 + t)) \). This is due to the fact that the number of segments between \( r_1, r_{k-1} \) is \( 2d_k \). During \( t \) time units, \( r_1 \) can visit at most \( t \) of the segments between \( r_1, r_{k-1} \). The same holds for \( r_{k-1} \). Therefore the number of remaining unvisited segments is \( 2(d_k - (t_1 + t)) \). These are the segments that need to be monitored, either by the extracted robot, or other sensing elements (security cameras for instance).

5. PRELIMINARIES

Even after determining the final positions, there are still many ways each robot can travel from its current position to its final position during \( \rho \) time units. A deterministic approach is no good against a full-knowledge adversary. Unlike the steady state, where the world is symmetric, enabling each robot to execute the same
patrol strategy, in our problem there is no symmetry thus it is necessary to calculate a strategy for each of the robots. Randomizing at each endpoint does not guarantee the robot will arrive at its final position at the end of the reorganization phase. Therefore we randomize over the possible paths that start at the current location of the robots, and end at the final position after ρ time units.

**Modeling the problem by a graph.**

Similar to [1], we model the problem as a directed graph in order to capture the directionality of the robots’ movement (facing forward is different than facing backward, considering the time τ it takes the robots to turn around). The graph G is constructed as follows (see the illustration in Figure 2):

For each pair in the original problem of end point and direction we follow (see the illustration in Figure 2):

- We construct chains of length τ between vertices with different direction.
- If v = (ep, forward), then v’ = (ep, backward).
- We define the endpoints of the same segment as: ep = (v, v), backward = (u, u).
- We define the cost of weights between endpoints: W is the cost of weights between u, v.
- We define the adjacency matrix of G, ep and find all paths leading to v, u.

**Algorithm 1 NumberOfPaths(u, v, L)**

1: Construct G
2: A ← Adjacency matrix of G
3: Return A^L(u, v)

![Figure 2: Example of a weighted graph that takes into account the cost of turning, τ.](image)

5.1 Algorithm For Finding All Paths

The behavior of the robots during the reorganization phase has to be random in order to handle a knowledgeable adversary. In this case, we randomize over the possible choices of paths leading the robots from their current position to their final position during ρ time units. First, we need to find the possible paths for each robot before further evaluation. Algorithm FindPaths receives the current position v and the final position goal of a given robot, and finds all paths with length τ. The algorithm considers the modeled graph G, the current position and direction, encompassed by v, and the final position and direction encompassed by goal. path and cost are the current path and its cost (initially empty and 0). The results are stored in paths (initially empty).

**Algorithm 2 FindPaths(G, v, goal, ρ, path, cost, paths)**

1: add v to path
2: if cost ≥ ρ then
3: add path to paths
4: return
5: if cost > ρ then
6: return
7: for each v’ in neighbors(v) do
8: if numberOfPaths(v’, goal, ρ – (cost + W(v, v’))) > 0 then
9: FindPaths(G, v’, goal, ρ, path, cost + W(v, v’), paths)
10: return paths

The algorithm recursively (DFS based) generates the paths, each step continues only to branches that can yield a valid path.

The complexity of NumberOfPaths is polynomial in N, i.e., O(N^ε) for some fixed ε. The time complexity of FindPaths is linear in output size, and is O(N^ε+1), since the number of paths as calculated by numberOfPaths is A^ρ(v, goal), where A is the adjacency matrix of graph G. We can bound this value by defining matrix B_{N×N} such that B_{ij} = 1, i = 1...N, j = 1...N. Since 0 ≤ A_{ij} ≤ 1, i = 1...N, j = 1...N. It holds that 0 ≤ A_{ij} ≤ B_{ij}, i = 1...N, j = 1...N and thus A^ρ ≤ B^ρ, and B^ρ = N^ρ.

5.2 ppd Computation

The values of the ppd depend on the probability of each path to be chosen. Suppose that for each robot we are given a set of paths and a distribution over this set. Algorithm calcPPD computes the values of ppd’s, at a given time and illustrates the influence of the distribution on the ppd values.

**Input for calcPPD:**

- problem settings: S := segments set, R := robots set
- paths := set of all paths for all robots: {path_r}, where path_r is the set of possible paths for robot r
- path[t1:t2] := sub-path of path between t_1, t_2
- t_a := inspected time of penetration initiation
- P := The sets of distribution over paths for all robots, {P_r} where P_r is the distribution of robot r over path_r.
- prob_r(s) := the probability for robot r to visit segment s.

**Result: ppd[t_a]**

**Algorithm 3 CalcPPD(paths, t, S, R, t_a, P)**

1: for each s in S do
2: ppd(s) ← 0
3: for each r in R do
4: prob_r(s) ← 0
5: for each path in paths do
6: I(s, r, path)[t_a] ← \begin{cases} 1 & \text{path}[t_a : t_a + τ] \text{ visits } s \\ 0 & \text{otherwise} \end{cases}
7: prob_r(s) ← prob_r(s) + P_r(path) · I(s, r, path)[t_a]
8: ppd(s) ← ppd(s) · (1 − prob_r(s)) + prob_r(s)
9: return ppd

1Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.
\[ \text{prob}(s) = \sum_{\text{path}k} P_r(\text{path}) \cdot I(s, r, \text{path}[t_a], \text{t}) \]  
where \( t_a \) is the time at which the robot visits segment \( s \). The probability that \( r \) will visit segment \( s \) between \( [t_a, t_a + t] \). Next we prove a lemma that will assist us in iteratively calculating the \( \text{ppd} \) for each segment, iterating the contribution of each robot.

**Lemma 2.** Assume that \( F_1(s) = \text{prob}_{r_1}(s) \), and that \( F_{j+1}(s) = F_j(s) \cdot (1 - \text{prob}_{r_{j+1}}(s)) + \text{prob}_{r_{j+1}}(s) \), \( 1 \leq j \leq k - 2 \) then \( \text{ppd}(s) = F_{k-1}(s) \).

**Proof.** We prove by induction that \( F_1(s) = 1 - \prod_{i=1}^{j}(1 - \text{prob}_{r_i}(s)) \). This statement holds for \( F_1(s) \). We will now prove that if it holds for \( F_j(s) \) it holds for \( F_{j+1}(s) = F_j(s) \cdot (1 - \text{prob}_{r_{j+1}}(s)) + \text{prob}_{r_{j+1}}(s) \), as required.

It holds that \( \text{ppd}(s) = 1 - \prod_{i=1}^{k-1}(1 - \text{prob}_{r_i}(s)) \), which is the complement of the probability that none of the robots will visit segment \( s \). Therefore \( \text{ppd}(s) = F_{k-1}(s) \). □

6. **Bounded Penetration Period**

In this section we consider the case where the time of the second penetration, denoted by \( t_a \), is bounded, and formally that \( x_1 \leq t_a \leq x_2 \) for some known \( x_1, x_2 \).

As stated in the preceding sections, our goal is to find the minimal \( \rho \) that satisfies the trade-off between increasing the reorganization time that will increase the \( \text{ppd} \), and minimizing of the reorganization time (since the optimal patrol algorithm for \( k - 1 \) robots is known to have the robots placed uniformly along the perimeter).

We therefore want to find the minimal \( \rho \) value that provides the \( \max \min \text{ppd} \) between \( [x_1, x_2] \).

We provide a series of constraints regarding the value of \( \rho \). First, it is necessary that \( \rho > t \), otherwise there are segments that will not be visited during a period of \( t \) time units from the beginning of the reorganization phase.

The existence of a path from a current position to the final position that visits a segment guarantees that there is a probability greater than zero of choosing it, then the probability of penetration detection at that segment is greater than zero.

As stated in previous sections, using a greater \( \rho \) improves the results. Since at some point increasing \( \rho \) yields the same sub-paths of a fixed length, the results will remain the same. We proceed to bound this point in time, and use \( b[x_2] \) to denote the bound. That means, using \( \rho = b[x_2] \) yields the \( \max \min \text{ppd} \), and any greater \( \rho \) yields the same. For that we denote by \( \text{path} \) the length of \( \text{path} \).

**Lemma 3.** \( b[x_2] := 2x_2 + 2t + D_{\max} + \tau \) is a bound for the minimal \( \rho \) such that \( \max \min \text{ppd} \) remains the same for every \( \rho > \rho \).

**Proof.** In order to prove that \( b[x_2] \) is a bound, it is sufficient to prove that: \( \{\text{path}[x_1 : x_2] : \text{path} = \rho\} = \{\text{path}[x_1 : x_2] : \text{path} = b[x_2]\} \) for \( \rho \geq b[x_2] \), which means that sub-paths of length \( x_2 \) from all paths of length \( \rho \) are the same as sub-paths of paths of length \( b[x_2] \), hence they have the same \( \text{ppd} \). A path of length \( x_2 + t \) can be distant at most \( x_2 + t + D_{\max} \) from its final position. In order for a path of length \( \rho \) to be of that distant, \( \rho \) needs to be greater than \( 2x_2 + 2t + D_{\max} + \tau \). □

**Optimization problem.**

After setting all preliminaries, we can formulate the problem as a QCP (quadratic constraint problem). The formulation considers the contribution of each \( \text{path}_j \) of robot \( j \) to segment \( s \) and maximizes the minimal value of \( \text{ppd} \)’s of all segments. The decision variables are the probabilities of choosing each path \( P_r(\text{path}) \), and extra variables to formulate the problem as a QCP. The paths for each robot are computed by: \( \text{path}_j = \text{FindPaths}(G, \text{currentPos}, \text{finalPosition}(\text{currentPos}), \rho, [[]], 0) \).

The \( \text{ppd} \) values are computed iteratively using Lemma 2, to generate quadratic constraints, instead of higher order.

\[
\text{max } m
\]
\[
s \in S, t_a \in [x_1, x_2] \quad \text{ppd}_r[t_a] = \text{calcPPP}(\text{paths}, t, \text{S}, \text{R}, t_a, \text{PPD})
\]
\[
r \in R, \sum_{\text{path}} P_r(\text{path}) = 1
\]
\[
\forall r, path, r \in R \quad P_r(\text{path}) \geq 0
\]
\[
s \in S, t_a \in [x_1, x_2] \quad \text{ppd}_r[t_a] \geq m
\]

**Patrol algorithm.**

The final algorithm is \text{PatrolRearrange} which computes the optimal distribution and randomizes the path for each robot.

**Algorithm 4** \text{PatrolRearrange}(R, S, \{\text{init}_r\}, \tau, x_1, x_2)

1: construct \( G \)
2: for each \( r \) in \( R \) do
3: \( \text{goal} \leftarrow \text{CalcFinalPosition}(\text{init}_r) \)
4: \( \rho \leftarrow \max\{2t + 2x_2 + D_{\max} + \tau, \frac{k-1}{k}d + \tau\} \)
5: \( \text{paths}_r \leftarrow \text{FindPaths}(G, \text{init}_r, \text{goal}, \rho, [[]], 0) \)
6: \( P_r \leftarrow \text{solve}(\text{paths}, k, n, x_1, x_2) \)
7: return random path from \( \text{paths}_r \), with distribution \( P_r \)

The time complexity of the problem is the complexity required to solve a non-linear optimization problem with \( O(k \cdot N^{2t}, (x_2 - x_1)) \) constraints and variables. This differs between solvers. The call to \text{solve} uses a solver to solve the non-linear optimization problem, which takes into consideration the paths of all robots, the number of robots, the number of segments, and the inspected area.

6.1 **Lower bound for ppd values**

In the following lemma we prove a lower bound on the minimal probability of penetration detection guaranteed by Algorithm \text{PatrolRearrange} during the reorganization phase. This lower bound is relatively high, and the implied “cost” to the system is discussed in the next section.

**Lemma 4.** If \( t_a = x \), during the reorganization phase for some known \( x \) then the \( \text{ppd} \) values during the first \( t \) time units in the reorganization phase guaranteed by performing \text{PatrolRearrange} satisfies that \( \text{ppd}_r = 0.5 \) for \( t = d_u(k) - t \ldots n - (d_u(k) - t) \).

**Proof.** Each robot has a path where during \( t \) time units, it visits segments that are at a distance of \( t \) forward and a path that visits backwards. Since \( 2t \geq d + \tau \), a pair of adjacent robots has paths that cover all segments between them during \( t \) time units. Assume we determine two possible paths for each robot: one in which the robot visits the segments forward to its position, and one that it visits segments backward from its position. There exist such paths for each robot such that any two robots ensures that all segments at a distance of \( d \) between them are covered. If each of these two paths is chosen with a probability of 0.5 for each robot, then the values of \( \text{ppd} \)’s in this example are at least 0.5, since all segments that are at a distance of \( t \) are covered. This implies that the resulting \( \text{ppd} \)’s from solving the optimization problem necessarily satisfy that \( \text{ppd}_r \geq 0.5 \) for \( t = (d_u(k) - t) \ldots n - (d_u(k) - t) \). □
6.2 Unknown Penetration Time

In this section we consider the time of the second penetration attempt to be unbounded. In the previous case we maximized the minimal ppd during a bounded period. Which enabled the selection of a ρ that results the optimal solution, even higher than the steady state. This is due to the fact that we overlooked the segments afterwards.

As an example, consider the case in which N = 84, k = 7 and t = 8, tₐ = 0. In this case, ρ₀ = 15, which means that although we handle the first t time units, the remaining ρ₀ − t = 7 time units are overlooked. This allows the robots to be very well prepared against a coordinated attack, but not against general attacks that may occur at any time. Following this example, the minimal ppd in the steady state for k = 7 robots is 0.15, the value for k = 1 = 6 robots is 0.05, and during the (first t time units of the) reorganization phase is 0.5. However, if a penetration is initiated after 12 time units (and not at time 0), the minimal ppd drops to 0.

We start by proving that there are points of vulnerability during the reorganization phase, in every possible choice of a single ρ for reorganization. In the bounded case we could increase ρ in order to avoid them, because the penetration time was bounded. The problem arises when the penetration can occur at any time during the reorganization phase, even during those vulnerable points. Therefore, increasing ρ in this case would not solve the problem.

**Lemma 5.** For ρ − t ≤ tₐ ≤ ρ, there exists a segment i such that ppdᵢ(tₐ) = 0 for ρ − (tₐ − t) ≤ tₐ ≤ ρ + t = (tₐ + t).

**Proof.** At time tₐ each of rₖ is at a distance of ρ − tₐ from its final position. During the remaining t − (ρ − tₐ) time units, the robots are in the steady phase, and each rₖ can be at a distance of at most t − (ρ − tₐ) from its final position. Thus during t time units each robot is at distance of at most max(ρ − tₐ, t − (ρ − tₐ)). This is illustrated in Fig. 3. If none of the segments is overlooked then each pair of robots cover all of the segments between their final positions, at a distance of dₖ₋₁. Thus it is essential that 2max(ρ − tₐ, t − (ρ − tₐ)) ≥ dₖ₋₁, which implies either tₐ ≤ ρ − t + dₖ₋₁ or tₐ ≥ ρ − t + dₖ₋₁, meaning that when ρ − t + dₖ₋₁ ≤ tₐ ≤ ρ − t + dₖ₋₁ segments with ppdᵢ(tₐ) = 0 exist.

The above lemma implies that if the opponent has full knowledge of the patrol scheme, and of ρ in particular, it can choose to penetrate during that time and it is guaranteed to succeed. As a corollary ρ must be randomized in order to ensure ppdᵢ(tₐ) ≠ 0 for all tₐ.

Since all robots must end the reorganization phase at the same time, all robots must draw paths for the same ρ. This means that each robot can draw a path only after a ρ is drawn.

**Optimization Problem.**

Similar to the first case of bounded time, we wish to formulate the optimization problem in this case. As illustrated by the following example in Fig. 4 calculating the optimal paths probabilities for each ρ and combining them together might yield a solution that is worse than computing all probabilities together. Therefore all paths of different ρ’s must be considered together.

**Figure 3:** (a) represents the most distant positions of rₖ after tₐ ≤ ρ time units (during reorganization). (b) represents the most distant positions of rₖ during the remaining period of t (after the reorganization phase).

**Figure 4:** Assuming there are four paths with ppd’s as described above, for some two segments. Each of path₃, path₄ has a smaller maxₚₙₚₚd than path₁, path₂. Yet, when maximizing the combination of each pair maxₚₙₚₚd of path₃ and path₄ is greater.

The optimal patrol algorithm is an extension of PatrolRearrange. After solving the optimization problem, first a ρ is drawn with probability of qₐ, and then a path with length ρ is drawn for each robot.

6.3 Bounding the reorganization time

In the previous section we proved a bound for the value of ρ that yields the maximum minimal ppd. In this section we prove that there exists a ρ that bounds the improvement towards the maxminppd.

**Lemma 6.** When maxΓ − inf, the solution of the optimization problem converges to the maxminppd of the reorganization phase.

**Proof Sketch.** The solution is monotonic with respect to ρ, since if a larger ρ does not improve the results it will have a probability of 0. In addition the solution is bounded (it represents probability). Therefore the process of adding more ρ’s converges. Due to the fact that every possible path is contained in {paths}ₐ, inf, the optimal paths for the reorganization are also contained in this set. Consequently a solution to an optimization problem containing these paths will yield the maxminppd of the reorganization phase.
The following corollary states the existence of a bound for adding larger ρ's.

**Corollary 1.** For every ε > 0 there exists a ρ₀ such that adding any ρ > ρ₀ would not change more than ε of the solution.

**Proof.** This is proven by using Cauchy’s criterion on previous lemma.

This is demonstrated in Fig. 5, which shows the finite bound for achieving the optimal ppd for a certain ε, and that lower ρ values have a higher probability.

### 6.4 Reducing the Problem Size

So far, our suggested algorithm considers every possible path for the optimization problem. Not only does this mean a very large input is considered, it is also unnecessary. Some of these paths clearly have less contribution than others, and there is no need to consider them in the optimization problem.

**Definition 1.** A path \(path₁\) dominates \(path₂\) if \(I(s, r, path₁)[tₐ] \geq I(s, r, path₂)[tₐ]\), for every \(s \in S\), \(0 \leq tₐ \leq ρ\).

Let \(Pₙ\) be a distribution function over the paths for robot \(r\). We wish to examine the effect of transferring the probability from \(path₂\) to \(path₁\) that dominates it. We denote the resulting distribution function by \(Pₙ\), and the probability of robot \(r\) to visit segment \(s\) according \(Pₙ\) by \(probₙ(s)\).

**Lemma 7.** If \(path₁\) dominates \(path₂\) then \(\sum_{s} Pₙ(s)[tₐ] \geq \sum_{s} Pₙ(s)[tₐ]\) for every \(s \in S\), \(0 \leq tₐ \leq ρ\).

**Proof.** \(\sum_{s} Pₙ(s)[tₐ] \geq \sum_{s} Pₙ(s)[tₐ]\) if \(I(s, r, path₁)[tₐ] \geq I(s, r, path₂)[tₐ]\) for every \(s \in S\), \(0 \leq tₐ \leq ρ\).

Let \(Pₙ\) be a distribution function over the paths for robot \(r\). We wish to examine the effect of transferring the probability from \(path₂\) to \(path₁\) that dominates it. We denote the resulting distribution function by \(Pₙ\), and the probability of robot \(r\) to visit segment \(s\) according \(Pₙ\) by \(probₙ(s)\).

**Lemma 8.** The number of Pareto-paths in the above case is \(Ω(4^{ρ})\).

**Proof.** Let us consider 4 sub-paths with a length of 12:
1. \(s₂, 0, s₁, 0, s₁, s₂, 0, s₂, 0, s₁, 0, s₁\)
2. \(s₂, 0, s₁, 0, s₁, s₂, 0, s₂, 0\)
3. \(s₂, s₂, 0, s₁, s₂, s₁, 0, s₁, s₂, 0\)
4. \(s₂, s₂, 0, s₁, s₂, s₁, 0, s₁, s₂, 0\)

In this case the contribution to segment \(s₁\) at time \(t₁\) is 1 if \(path₁[t₁] = s₁\). Therefore, these paths are not dominated by each other. Moreover, there cannot be a valid path that has the same contribution except for at least one 0 that is replaced.

Starting the ending position of all paths is \(s₁\), facing forward. Therefore we can construct paths by concatenating these sub-paths. Every combination of these sub-paths cannot dominate another combination since their sub-paths are non-dominating. Thus, we receive a sub-set of the Pareto-paths. The number of such combinations for paths of length \(ρ\) is \(4^{(ρ/12)}\).

### 7. EXPERIMENTAL RESULTS

In this section we analyze the results of the experiments we conducted with more than 11,000 settings for the problem of reducing input size.

In Section 6.4 we have shown that, theoretically, the number of pareto paths can be exponential in the input size. However, in this section we show that practically, in most cases the number of pareto paths is significantly lower than this theoretical upper bound.

The number of Pareto-paths depends on \(t, ρ, dist, N\), where \(dist\) is the actual distance between a robot’s current and final position.

We have examined over 11,000 different settings, where the examined parameters were: \(k = 2..10, d = 3..6, t = 1..d - 1, ρ = 2..12, dist = 0..ρ\).

Fig. 6 illustrates that as \(t\) grows the number of Pareto-paths drops exponentially. Fig. 7 and Fig. 8 show that even when the number of Pareto-paths grows exponentially as \(ρ\) grows, the number of Pareto-paths is much smaller than the number of paths.
higher ratio. In this case Pareto-paths and all paths, with respect to $\rho$

Figure 7: Number of paths and Pareto-paths as a function of $\rho$.

... further. Another venue would be to examine the influence of the time of inspection of a robot, which might lead to a more robust patrol against multiple attacks.

9. REFERENCES


