國立臺灣大學電機資訊學院資訊工程學研究所

碩士論文

Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master's Thesis

基於可滿足性模組理論之實數平面多智能體協同安全定位

Cooperative and Secure Multi-Agent Positioning on Real Coordinates Based on Satisfiability Modulo Theories

賈本晧

Ben-Hau Chia

指導教授：林忠緯 博士

Advisor: Chung-Wei Lin, Ph.D.

中華民國 112 年 7 月

July 2023

國立臺灣大學碩士學位論文
口試委員會審定書
MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

基於可滿足性模組理論之實數平面多智能體協同安全定位

Cooperative and Secure Multi-Agent Positioning on Real Coordinates Based on Satisfiability Modulo Theories

本論文係賈本晧君（學號 R10922060）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 112 年 7 月 18 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 18 July 2023 have examined a Master's thesis entitled above presented by BEN-HAU CHIA (student ID: R10922060) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

林忠緯

（指導教授 Advisor）

陳尚澤　　蕭旭君　　江介宏

系主任/所長 Director: 洪士灝

Acknowledgements

I would like to express my deepest appreciation to Professor Chung-Wei Lin, who has provided invaluable expertise with immense patience. I have cultivated a positive attitude and learned how to conduct research responsibly. I also could not have been part of this academic journey without Professor Wenchao Li, who generously offered professional suggestions and feedback.

My gratitude extends to all of lab members as well, for their help in proof-reading, editing, and moral support.

Lastly, I would like to mention my family and friends. Their unconditional support and faith in me have kept my motivation high throughout my pursuit for a Master's Degree.

Ben-Hau Chia

National Taiwan University
July 2023

基於可滿足性模組理論之實數平面多智能體協同安全定位

研究生：賈本晧　　指導教授：林忠緯 博士

國立臺灣大學資訊工程學研究所

摘要

多智能體系統（multi-agent system）基於各智能體之間的溝通並合力解決問題，如智慧交通系統中的各項運輸工具，軍事系統中的各種軍事設備，機器人系統中的自主機器人等，這些智能體們在各系統中藉由合作達到單一智能體無法完成的任務。其中，定位在多智能體系統中是舉足輕重的工作之一，例如自駕車需要精準定位來達到自動駕駛。本論文考慮互聯多智能體系統的協同安全定位，並解決兩項主要問題：每個智能體有各自的定位誤差，且系統中存在刻意散佈錯誤資訊的攻擊者。針對以上問題，本論文基於可滿足性模組理論設計了建構性方法及破壞性方法，兩種方法皆嘗試使各智能體達成定位上的共識，並同時確認攻擊者存在之有無。實驗結果顯示，相較於一基準方法，建構性方法能提升攻擊者辨識的準確度，而破壞性方法可以加速取得定位共識所需時間。

關鍵詞：協同定位、多智能體系統、安全定位系統、可滿足性模組理論

COOPERATIVE AND SECURE MULTI-AGENT POSITIONING ON REAL COORDINATES BASED ON SATISFIABILITY MODULO THEORIES

Student: Ben-Hau Chia Advisor: Dr. Chung-Wei Lin

Department of Computer Science and Information Engineering
National Taiwan University

Abstract

Multi-agent systems are an emerging technology with a promising future, and positioning is a fundamental task supporting applications of multi-agent systems, with vehicles needing precise positioning to perform autonomous driving as a prime example. In this thesis, we consider cooperative positioning for connected multi-agent systems. In particular, we address the two following challenges: Each agent has its own positioning errors, and malicious agents within the group intentionally provide false information. Based on Satisfiability Modulo Theories (SMT), we design two approaches, a *constructive* approach and a *destructive* approach, to reach a positioning consensus among the agents and identify the set of potential attackers. Experimental results demonstrate that the proposed approaches improve the consensus accuracy and speed up the consensus process, respectively, compared with a baseline approach.

Keywords: Cooperative Positioning, Multi-Agent System, Secure Positioning System, Satisfiability Modulo Theories

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Multi-agent systems are composed of multiple intelligent and interactive agents, providing a way for agents to communicate and collaborate. Numerous multi-agent system applications, such as autonomous driving, military defense, intrusion detection, healthcare monitoring, and emergency services have been proposed. An intelligent transport system (ITS) is a representative application of multi-agent systems as we view each transportation as an agent, having great potential to improve traffic efficiency and reduce accidents with the application of sensing, communication, analysis, and control. Vehicular ad-hoc network (VANET), based on the architecture of mobile ad-hoc networks (MANET), is one of the crucial components in ITS, utilizing various vehicles or transportation facilities as communication nodes, which can be applied to emergency event warnings, traffic flow control, etc.

Accurately locating each agent is pivotal in multi-agent systems [22]. The Global Positioning System (GPS) is one of the most common approaches for agents to position themselves, but GPS has its limitation in accuracy which is affected by the surrounding environment. Multi-agent systems can work cooperatively to perform positioning, improving the corresponding accuracy and responsiveness for all agents and even other objects in the surrounding environment. The most repre-

sentative applications include simultaneous localization and mapping (SLAM) and dynamic mapping for autonomous driving. However, malicious attacks within the group can intentionally provide false information and undermine the overall positioning systems. In this thesis, we target cooperative positioning for connected multi-agent systems with the existence of positioning errors and malicious attackers.

This thesis extends a previous study [10] that addressed the problem on a grid-based map with at most one attacker and proposed approaches based on Satisfiability Modulo Theories (SMT). However, it has its limitations, since real-world maps cannot always be divided into grids, and there is no guarantee of having only one attacker. In this thesis, we address the problem on a real coordinate map with one or multiple attackers, which is closer to reality.

1.1 Related Work

1.1.1 Global Positioning System (GPS)

Numerous approaches have been proposed to localize agents in different scenarios and environments, and GPS is one of the most common approaches. However, its provided accuracy is only to the scale of meters, which is not enough for certain cases of multi-agent systems, such as multi-agent formation. An approach that integrates GPS for agent localization has been proposed in [4]. Further improvement for accuracy has been done by fusing the observation from received signal strength (RSS) and carrier frequency offset (CFO) into GPS with the help of neural networks [17].

1.1.2 Non-Satellite Based Positioning

However, environmental factors, such as satellite-signal occlusion, multi-paths, crowded urban settings, and other non-line-of-sight factors, can severely undermine the efficacy of GPS. Therefore, many non-satellite based positioning methods, such as radio frequency based positioning and visual navigation, have been developed, including time of arrival, time difference of arrival, and angle of arrival, etc. Extensive research on ultra-wideband transmission technology has also been done [8], exploiting the low-energy and high-bandwidth communication characteristics.

1.1.3 Cooperative Positioning

In the meantime, compared with conventional approaches that require every agent to communicate with an anchor node, agents in cooperative positioning gradually help each other determine their own positions. This results in a lower number of anchor nodes and increased performance in terms of both accuracy and coverage. We point the readers to [31] for a comprehensive analysis on cooperative positioning. A Bayesian method for distributed sequential positioning of mobile networks composed of both cooperative agents and non-cooperative objects can also be found [20]. A provably coordinated attack detection algorithm at the graph level is also proposed in [30].

1.1.4 Simultaneous Localization and Mapping (SLAM)

A technique that has been receiving significant interest is SLAM [2] for autonomous driving. In this case, agents are vehicles detecting other vehicles based on camera [6], lidar [5], or radar [9]. As expected, it is ideal to have different vehicles working collaboratively to solve the localization and mapping problem. [25, 33]

are early collaborative visual SLAM frameworks supporting independently moving cameras, and [13, 27] further restrict each agent taking limited key frames. Note that all the above systems have centralized architectures, *i.e.*, a server collects the key frame images from agents to compute the local map for each agent. However, there can be adversarial attacks on visual SLAM. [3] presented a method to create universal adversarial image patches attacking on general deep learning models, and [11] showed that the ORB-SLAM [21] can be corrupted when the environment is modified by replicating a simple high textured patch.

1.1.5 Vehicular Networking and Its Security

As mentioned in [12], vehicular networking and/or connected vehicles can be used to support active road safety applications as well as other information exchanges. The goals are to decrease the probability of traffic accidents, reduce the loss of lives, and increase traffic efficiency. Object localization and positioning in dynamic maps which are provided to vehicles via vehicular networking is an essential function for the realization of autonomous driving. However, security is also a concern for vehicular networking, and [14,29] have a brief introduction on the threats that vehicular networking faces, including authentication, integrity, confidentiality, etc. Apart from authenticating the legitimacy of each vehicle, numerous ways to detect malicious vehicles have also been proposed. [15] detects malicious vehicles by evaluating the reliability of neighboring vehicles based on the observed difference, and various trust-based security algorithms for vehicular networking have also been analyzed [28].

1.1.6 Mobile Ad-hoc Networks (MANETs)

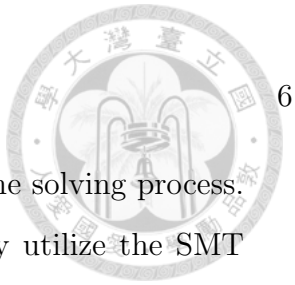
MANETs consist of nodes that form a self-figuring and self-healing network and have shown great potential in providing solutions for various situations. These networks do not rely on fixed topology or require pre-existing information and communications technology (ICT) infrastructure. We refer the readers to the references [24, 26] for details.

1.1.7 Vehicular Ad-hoc Networks (VANETs)

Established on MANETs' architectures, VANETs are an emerging technology with great potential in providing more efficient and safer traffic, but they are more challenging because of unpredictable dynamic topologies with safety concerns. [32] introduced methods for detecting Sybil attacks by analyzing signal strength. [1] has proposed an operating system for the 5G-based VANETs with software-defined network (SDN) and self-organizing map (SOM) incorporated. A trust-based framework for distributed denial-of-service attack (DDoS) detection has also been developed in [23]. For readers who are unfamiliar with VANETs, we refer to the references [16, 18].

1.2 Background: Satisfiability Modulo Theories

In this thesis, we utilize the Satisfiability Modulo Theories (SMT) solver as a tool to address our problem. The Boolean satisfiability problem (SAT) is to determine if there exists an interpretation that satisfies a given Boolean formula, and SMT generalizes the SAT problem with more complex formulas (*e.g.*, lists, arrays, bit vectors, strings). To make our problem and approaches compatible with existing SMT solvers, we transform all the constraints into first-order logic formulas and solve the problem using SMT solvers. The SMT solvers also provide the flexibility



to add extra constraints and remove useless constraints during the solving process. This feature enables us to design our approaches and iteratively utilize the SMT solvers, adjusting the constraints as needed.

An SMT solver checks whether a set of first-order logic formulas can be satisfied with respect to several background theories and returns the results according to the satisfiability of those formulas. An SMT solver checks whether a set of first-order logic formulas can be satisfied with respect to several background theories and returns the results according to the satisfiability of those formulas. An SMT solver has two possible return values, satisfiable and unsatisfiable, indicating the satisfiability of the given formulas. In addition to the satisfiability, the solver can provide further information. It returns a solution if it determines that the given formulas are satisfiable. Otherwise, it returns an unsatisfiable core if no solution exists that satisfies all the formulas simultaneously.

An unsatisfiable core, also known as an unsat core, refers to any subset of formulas from a given set of formulas that demonstrates the unsatisfiability of the entire set. An unsat core is *minimal* if we can make it satisfiable after removing any formula in the given formula set, and it is *minimum* if it has the least number of formulas among all minimal unsat cores. Note that there may be numerous minimal unsat cores with respect to the same set of formulas. It is also important to note that an SMT solver can return a minimal unsat core but not a minimum one. Nevertheless, this property allows us to focus on the critical formulas responsible for the unsatisfiability, aiding in the analysis and the process of discovering the attackers.

1.3 Contributions

Our contributions are summarized as follows:

- We address the challenges on a real coordinate map that each agent has its own positioning errors, and one or multiple malicious attackers intentionally provide false information.
- Based on SMT, we design two approaches: the *constructive* approach, and the *destructive* approach. They try to reach a positioning consensus between agents and find the set of potential attackers.
- Experimental results demonstrate that the constructive approach and the destructive approach improve the consensus accuracy and speed up the consensus process, respectively, compared with a baseline approach.

1.4 Thesis Organization

The rest of the thesis is organized as follows. We introduce the formulation of the problem in Chapter 2, present the proposed approaches in Chapter 3, describe the experimental results in Chapter 4, and draw conclusions in Chapter 5.

Chapter 2

System Model and Problem Formulation

In this chapter, we scrutinize the problem. We first introduce the elements of the problem in Section 2.1. Section 2.2 and Section 2.3 describe positions and observations made by agents in detail, respectively. Section 2.4 discusses the attacking strategies an attacker can adopt. Last, the formulation for the SMT solver is introduced in Section 2.5. The notations throughout this thesis are listed in Table 2.1.

Table 2.1: Notations throughout this thesis.

Coordinates, Indices, Direction	
x, y	the coordinates
i, i', i''	the index of an agent
a	the index of an attacker
t	the index of a broadcast position
d	the direction (north, east, south, or west)
Agents	
α_i	the i-th agent
α_a	the attacker (also an agent)
Given Parameters	
M	the side length of the map
N	the number of agents
S	the size of each agent
$R_i^t = (X_i^t, Y_i^t)$	the t-th broadcast position of α_i
E	the error shift
$F_{i,d}$	the observation of α_i along direction d
O_i	the observation of α_i
T	maximum round
Given Sets	
$\mathcal{R}_i$	the set of all broadcast positions of α_i
$\mathcal{R}^t$	the set of the t-th broadcast positions of all agents
$\mathcal{R}$	the set of all broadcast positions of all agents
$\mathcal{O}$	the set of all observations
Decision Variables	
(X_i, Y_i)	the position variables for α_i
$\mathcal{A}$	the set of potential attackers
Others	
(x_i, y_i)	the real position for α_i

2.1 Elements

The system model has the following elements:

- **Map**. We consider the problem on an $M \times M$ real coordinate map, which can be viewed as a coordinate system. The coordinates of each point are defined as (x, y), with $(0, 0)$ in the bottom-left corner.

- **Agent**. We assume that an agent is a square with a side length of S, and there is an aggregate of N agents on the map. Each agent α_i owns a unique integer ID i, where $0 \leq i < N$. α_i is located on (x_i, y_i), where both x_i and y_i are static. In the system, all agents are capable of broadcasting their positions and observations to others. We will further introduce the positioning and observations in Section 2.2 and Section 2.3.

- **Attacker**. An attacker is also an agent in the system. We assume that there are one or multiple attackers in the system. The objective of the attackers is to confuse the other agents by broadcasting wrong information, including false positions and observations. The behavior and attacking strategies of the attackers will be detailed in Section 2.4.

- **Solution**. A solution includes a feasible position for each agent on the map. We utilize the SMT solver to derive solutions.

- **Goal**. The goal is to reach a consensus among all agents. The consensus is either one of the following outcomes: there is no solution, or there exists at least one solution. If there is no solution, it implies that there are attackers in the system, and the goal also includes deriving the set of potential (possible) attackers $\mathcal{A}$.

2.2 Positioning

Each agent uses its sensors such as GPS to position itself. The t-th sensed (and then broadcast) position of α_i is denoted as $R_i^t = (X_i^t, Y_i^t)$. The set of all broadcast positions of α_i is denoted as $\mathcal{R}_i$, and the set of all agents' t-th broadcast positions is denoted as $\mathcal{R}^t$. Furthermore, the set of all broadcast positions of all agents is denoted as $\mathcal{R}$. Due to errors and limitations of GPS, there exist discrepancies between α_i's real position (x_i, y_i) and sensed positions (X_i^t, Y_i^t). We assume that there is a bounded value of the error between the real position and every sensed position along each axis, which we refer to as error shift, denoted as E.

It should be mentioned that, when there is an error shift, it is possible that an agent always broadcasts its position with an error, so the goal in this thesis is to find possible positions (solutions) for each agent, not to find the real position for each agent.

2.3 Observation

Each agent utilizes its sensors (*e.g.*, cameras, radars, lidars) to observe the environment on the map. Numerous observations can be designed and implemented by an agent, for example, distance measurement, object detection, and object identification. The more observations an agent makes, the more information (as constraints) we can consider during the solving process. In this thesis, we consider a "line-of-sight" observation, where each agent checks four directions, including north, east, south, and west. An agent is able to observe all agents along each direction. We assume that there is no error with the observations of an agent, but an agent does not measure the distance between itself and the observed agents. We denote the observation of α_i along direction d as $F_{i,d}$, where d is north, east, south or west:

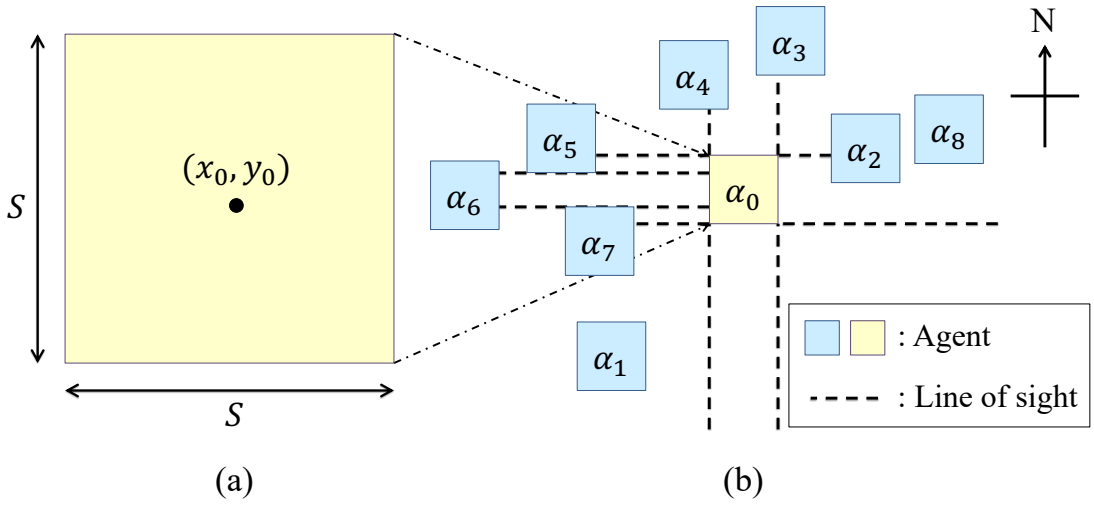


Figure 2.1: (a) An agent on the map is a square with a side length equal to S, and the position of an agent is the coordinate of its central point. (b) An agent can observe the other agents with a 4-directional line-of-sight observation. In this example, α_0's real position is (x_0, y_0). α_0 can observe α_2 along the east side, and therefore $F_{0,east} = [2]$. Similarly, $F_{0,north} = [4, 3]$, $F_{0,west} = [7, 6, 5]$, and $F_{0,south} = \emptyset$.

- $F_{i,d} = \emptyset$: α_i observes no agents along the direction d.
- $F_{i,d} = [i_{d,0}, i_{d,1}, \ldots, i_{d,n-1}]$: α_i observes n agents along the direction d, and they are $\alpha_{i_{d,0}}$ to $\alpha_{i_{d,n-1}}$. Moreover, α_i broadcasts the observed agents from its left-hand side to its right-hand side. For example, when observing along the north side, α_i broadcasts the observed agents from the westernmost (left-hand side) one to the easternmost (right-hand side) one, which are $\alpha_{i_{north,0}}$ and $\alpha_{i_{north,n-1}}$, respectively.

We then denote the observation of α_i as O_i, which is an array of length 4, including the observation of α_i along the north, east, south, and west, respectively. The set of all observations is denoted as $\mathcal{O}$.

Figure 2.1(b) shows an example of the observation of α_0. In the example, α_0 observes only α_2 along the east direction, so $F_{0,east} = [2]$. Note that α_0 cannot

observe α_8 since the line of sight for α_0 to see it is blocked by α_2. α_0 also observes both α_3 and α_4 along the north side, so $F_{0,north} = [4, 3]$. Note the order matters, since we assume that the observed agents are broadcast from the observer's left-hand side to its right-hand side. Similarly, we have $F_{0,west} = [7, 6, 5]$. Last, α_0 cannot observe α_1, and therefore $F_{0,south} = \emptyset$.

Although the 4-directional line-of-sight observation is applied in this thesis, there are many various kinds of observations that can be applied. An agent can have more directions (angles) for line-of-sight observation. An agent can also measure the distance between itself and an observed agent, using radars for example. It is also possible that the observations have bounded ranges or errors due to the limitation of sensors. We can deal with various kinds of observations as long as the observations can be transformed into first-order logic formulas.

2.4 Attacking Strategies

As mentioned in Section 2.1, we assume that there are one or multiple attackers in the system, and the attackers aim to confuse the other agents by broadcasting incorrect positions and observations. Attackers may have different motivations for broadcasting incorrect positions. Here are some of the motivations:

- **Evasion of Detection**. By claiming a wrong position, an attacker can hide its real position, which is beneficial if the attacker does not want to be tracked or identified.

- **Misdirection or Diversion**. An attacker may intentionally offer wrong position information to mislead or divert the attention of the system or the other agents, causing confusion or communication disruption among the agents. For

example, man-in-the-middle attacks are one of the attacks that mislead the agents by broadcasting incorrect messages.

- **System Manipulation**. An attacker can deceive and further manipulate the system by providing wrong positions. For example, in a vehicular system, an attacker can create congestion or occupy the traffic resources (*e.g.*, intersections) by pretending to be in a wrong location. It can lead to a waste of traffic resources and even chaos on the road.

- **Unauthorized Access**. An attacker may gain unauthorized access to enter restricted areas by broadcasting incorrect positions. For example, it is possible to deceive a location-based access control system by claiming a different position.

We have two assumptions for the attackers, and both assumptions make it easier for the attackers to perform attacks:

- **Complete Information**. We assume that the attackers have all information needed to perform attacks, including all agents' real positions on the map and the details about system flow, making it easier for the attackers to perform tricky attacks.

- **Outstanding Computational Power**. We assume that each attacker owns powerful computational resources, which allows them to compute the needed information and derive the best fake position to attack within a short time period.

The objective of the attackers is to confuse the other agents by broadcasting wrong positions and observations, in order to obtain personal benefits or cause

safety concerns. However, randomly choosing a fake position to broadcast can make it easier for those good agents to detect the threat and further discover the attackers. Therefore, good attacking strategies are important for the attackers.

Based on the broadcast positions and observations made by all agents (including the attackers), an approach solves the positions for all agents. There are three kinds of attacking strategies: *No Solution* attack, *One Wrong Solution* attack, and *Multiple Solutions* attack. All three attacks try to confuse the other agents by broadcasting false positions. From the perspective of the approach, it knows neither the number of attackers on the map nor the attacking strategies they choose. We shall elaborate on these three attack strategies:

- **Strategy 1: No Solution**. The attacker computes and derives a fake position violating some observations made by other agents, causing the approach unable to find a solution. It should be emphasized that from the approach's perspective, it just discovers an inconsistency among the agents. However, it does not know which agents are the attacker, and thus it cannot solve the possible positions of all agents at the same time.

- **Strategy 2: One Wrong Solution**. The attacker computes and derives a fake position following all observations made by other agents. From the approach's perspective, all broadcast positions and observations are consistent. However, the solved positions for the attackers are different from their real ones, and thus the real positions of the attackers cannot be solved.

- **Strategy 3: Multiple Solutions**. The attacker computes and derives a fake position following all observations made by other agents. From the approach's perspective, all broadcast positions and observations are consistent. However,

for some agents, there is more than one possible position for each of them, and thus the real positions of the attackers cannot be solved.

We have three more assumptions here, and these assumptions make the attackers act more like good agents, thus making it more difficult for the approach to discover them:

- The attackers never broadcast the positions that are outside the map or overlap with the other agents.
- For each attacker, the fake position it tries to attack is fixed.
- To simulate the errors caused by the sensor such as GPS, the attackers also broadcast their positions with error shift E involved.

To observe and underline the power of the SMT solver, we assume that all attackers in the system choose *No Solution* attack as their attacking strategies.

2.5 SMT Formulation

As mentioned in Section 2.1, the goal is to reach a consensus among all agents. The consensus is either one of the following outcomes: there is no solution, or there exists at least one solution. If there is no solution, it implies that there are attackers in the system, and the goal also includes deriving the set of potential (possible) attackers $\mathcal{A}$.

We use the SMT solver to derive solutions based on all broadcast positions and observations. The SMT solvers can be executed in a centralized or distributed way. If it is executed in a centralized way, there is a centralized unit that collects all agents' broadcast positions and observations, executes the SMT solver, solves

the problem, and then broadcasts the solution back to all agents. For example, a centralized unit can be an edge server or a roadside infrastructure for vehicles. If it is executed in a distributed way, each agent receives all positions and observations broadcast by the others, executes the SMT solver, and solves the problem.

Given M, N, S, E, $\mathcal{R}$, $\mathcal{O}$, the SMT solver decides the possible position (X_i, Y_i) for each agent α_i, while following all the positions and observations broadcast by all agents. The constraints put into the SMT solver are detailed as follows:

- **Validity of Real Positions**. The real position of each position is inside the map, and therefore α_i's real position coordinates are between 0 and M:

$$\forall\, i, \quad (0 \leq X_i \leq M) \wedge (0 \leq Y_i \leq M)\,. \tag{2.1}$$

- **No Overlap**. There is no spatial overlap between agents, otherwise a collision occurs, and we do not consider the collision cases in this thesis. Therefore, the following constraint stands:

$$\forall\, i, i', i \neq i', \quad (|X_i - X_{i'}| \geq S) \vee (|Y_i - Y_{i'}| \geq S)\,. \tag{2.2}$$

- **Consistency of Observations**. As mentioned in Section 2.3, we assume that there is no error with all agents' observations, and therefore the observations are consistent with the agents' real positions. For example, when d is north:

  - If $i' \in F_{i,north}$, α_i observes $\alpha_{i'}$ along the north side:

$$\forall\, i' \in F_{i,north}, \quad (|X_i - X_{i'}| < S) \wedge (Y_i < Y_{i'}). \tag{2.3}$$

We can add further constraints between those agents in $F_{i,north}$. Assume there are n agents in $F_{i,north}$, which are $i_{north,0}, i_{north,1}, \ldots, i_{north,n-1}$:

$$\forall\ k, 0 \leq k < n-1, \quad X_{i_{north,k}} < X_{i_{north,k+1}}. \tag{2.4}$$

We have similar constraints when d is east, south, or west. They are omitted here.

– If $i' \notin F_{i,north} \cup F_{i,east} \cup F_{i,south} \cup F_{i,west}$, $\alpha_{i'}$ is not observed by α_i. There are two possible reasons:

  * The first possible reason is $\alpha_{i'}$ is out of α_i's observable range (*e.g.*, in Figure 2.1(b), α_0 cannot observe α_1):

$$(|X_i - X_{i'}| \geq S) \wedge (|Y_i - Y_{i'}| \geq S)\,. \tag{2.5}$$

  * The second possible reason is that there exists another agent $\alpha_{i''}$ which blocks α_i to observe $\alpha_{i'}$(*e.g.*, in Figure 2.1(b), α_0 cannot observe α_8 because of α_2):

$$\exists i'' \in F_{i,north}, \quad Y_{i'} - Y_{i''} \geq S. \tag{2.6}$$

  We have similar constraints when d is east, south, or west. They are omitted here.

- **Bounded Errors**. As mentioned in Section 2.2, we assume that there is a bounded value of the error E between the real position and every sensed position along each axis:

$$\forall\ i, t, \quad \left(\left|X_i^t - X_i\right| \leq E\right) \wedge \left(\left|Y_i^t - Y_i\right| \leq E\right). \tag{2.7}$$

For the first three types of constraints, we call them "basic constraints", since the approach adds them to the assertion list of the SMT solver just once during the initialization. For the fourth type of constraints (bounded errors), they are added to the assertion list whenever the approach receives a sensed position from the sensors or a broadcast position from another agent. Note that the usage of the fourth type of constraints (bounded errors) also depends on whether the error shift E is given (known) or not. If the error shift E is given, the constraints are added to the assertion list as mentioned above; if the error shift E is not given, we can iteratively add the constraint with different values of E in Equation 2.7 into the assertion list of the SMT solver and analyze the satisfiability outcome. We assume that E is given for the remaining part of the thesis.

Chapter 3

Proposed Approaches

In this chapter, we introduce the proposed approaches. Section 3.1 defines the broadcast protocols first, and Section 3.2 describes the system flow. Subsequently, we present two proposed approaches, the constructive approach and the destructive approach, in Section 3.3 and Section 3.4, respectively.

3.1 Broadcast Protocols

A broadcasting protocol describes the broadcasting behavior of all agents, including the feasible time for an agent to broadcast its sensed position and the solving time for the SMT solver [10]. There are two reasonable settings:

- **Protocol 1**. As shown in Figure 3.1(a), the SMT solver starts solving the SMT formulation as long as all agents broadcast their positions at least once.
- **Protocol 2**. As shown in Figure 3.1(b), there is a time interval, and all agents continuously broadcast their positions until the end of the time interval. The SMT solver starts solving after the end of the time interval.

From the perspective of the SMT solver, there is not too much difference as the two protocols only create different numbers of constraints of bounded errors. However, from the perspective of the attacker, Protocol 1 has more limitations as the system

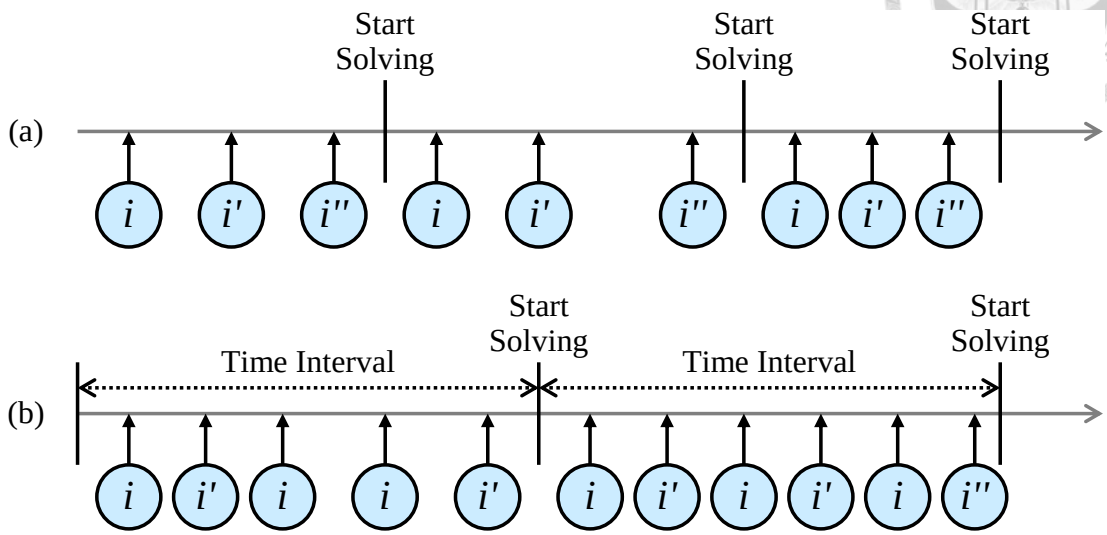


Figure 3.1: Two broadcasting protocols. (a) Protocol 1 and (b) Protocol 2.

can ask each agent to broadcast its position and observation once all the other agents do so. With Protocol 2, the attackers can keep silent, continuously collect information from all the other agents, decide their best strategies, and broadcast their information near the end of the time interval. In the following thesis, we use Protocol 1 as a better defending strategy, although Protocol 2 can still be applied to the SMT formulation and the SMT solver.

3.2 System Flow

Figure 3.2 demonstrates the system flows of two proposed approaches. Our approaches utilize the SMT solver as the solving tool, which holds an assertion list as a container and adds various types of constraints, presented as logical formulas, into the assertion list. The basic constraints, defined in Section 2.5, are added to the assertion list during initialization. The fourth constraint (bounded errors), also defined in Section 2.5, is added to the assertion list after the broadcast positions are received. Two approaches assume all agents to be good agents initially. The

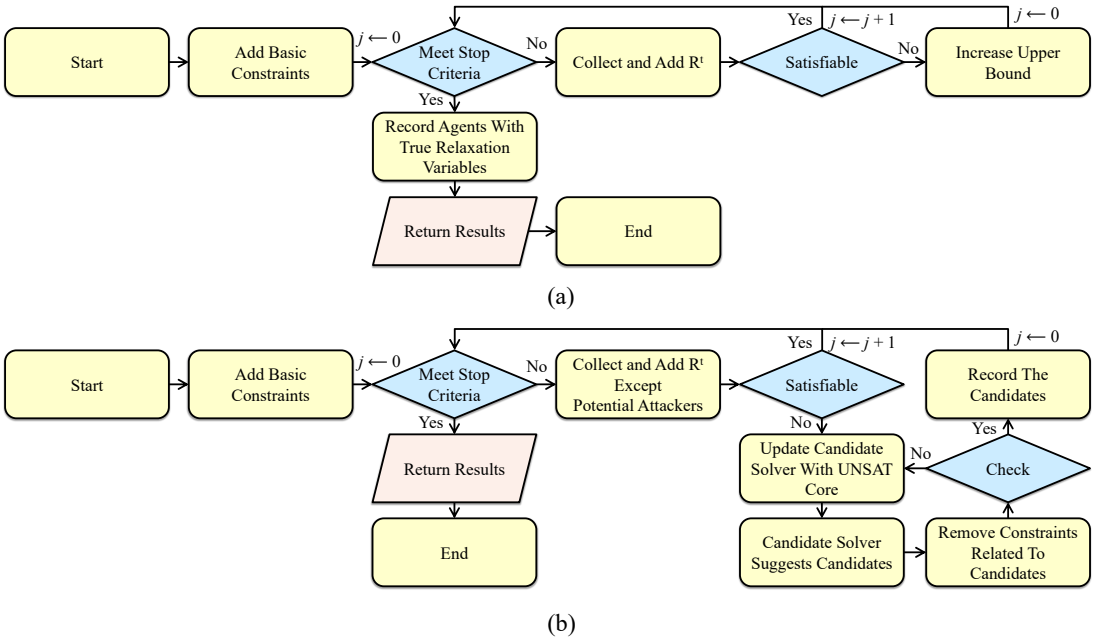


Figure 3.2: Two approaches. (a) Constructive approach and (b) Destructive approach.

system will return the results when the SMT solver continuously returns satisfiable for T times. There are two possible outputs for the system when meeting the stop criteria. If there are no possible attackers found, it outputs all agents' positions. Otherwise, the system outputs the set of potential attackers $\mathcal{A}$.

3.3 Constructive Approach

See Algorithm 1. The notations and the descriptions in Algorithm 1 follow [19]. We call it "constructive" since the number of "assumed" potential attackers increases during the system flow as if it is constructed. In the constructive approach, we add relaxation variables (Line 5) to all constraints. A relaxation variable r_i is added to a given constraint if the constraint is related to α_i. For example, to restrict α_0 to be inside the map (Equation 2.1), the constraint we put into the

Algorithm 1: Constructive Approach

Data: $M, N, S, E, \mathcal{R}, O, T$
Result: Solved Positions $\mathcal{P}$ or Potential Attackers $\mathcal{A}$

1 **begin**
2 $(\mathcal{C}, \mathcal{A}, j, t, ub) \leftarrow (\emptyset, \emptyset, 0, 0, 0)$
3 *add basic constraints to* $\mathcal{C}$
4 $\nabla \leftarrow \{r_i | r_i$ is the relaxation variable for $\alpha_i\}$
5 $\mathcal{C}^{\nabla} \leftarrow$ addRelaxation $(\mathcal{C}, \nabla)$
6 $\sqcap \leftarrow CNF(\sum_{i=1}^{N} r_i \leq ub)$
7 **while** $j < T$ **do**
8 $\mathcal{C}^{\nabla} \leftarrow \mathcal{C}^{\nabla} \cup$ addRelaxation $(\mathcal{R}^t, \nabla)$
9 $(sat, \mathcal{P}, \mathcal{U}) \leftarrow \text{SMT}(\mathcal{C}^{\nabla} \cup \sqcap)$
10 **if** $sat \neq true$ **then**
11 $(j, ub) \leftarrow (0, ub + 1)$
12 $\sqcap \leftarrow CNF(\sum_{i=1}^{N} r_i \leq ub)$
13 **end**
14 $(j, t) \leftarrow (j + 1, t + 1)$
15 **end**
16 **while** $sat = True$ **do**
17 $\mathcal{A}' \leftarrow \text{getTrue}(P, \nabla)$
18 $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{A}'$
19 $\mathcal{C}^{\nabla} \leftarrow \mathcal{C}^{\nabla} \cup \text{avoid}(\mathcal{A}')$
20 $(sat, \mathcal{P}, \mathcal{U}) \leftarrow \text{SMT}(\mathcal{C}^{\nabla} \cup \sqcap)$
21 **end**
22 **return** $\mathcal{P}, \mathcal{A}$
23 **end**

SMT solver is $((0 \leq x_0 \leq M) \wedge (0 \leq y_0 \leq M)) \vee (r_0 = 1)$. The constraint suggests that either α_0 is indeed inside the map, or r_0 is true, indicating α_0 is an attacker. In the beginning, we assume that there are no attackers in the system. An *AtMost* constraint, denoted as $\sqcap$ in Algorithm 1 (Line 6), is added to the assertion list, asserting no relaxation variables can be assigned true value. The SMT solver in the constructive approach (Line 9) should find the satisfiability between the constraint set (c^{∇}) and the AtMost constraint ($\sqcap$) at the same time. If the SMT solver returns unsatisfiability, the system increases the upper bound, denoted as ub in Algorithm 1, of *AtMost* constraint (Line 11), since there exist more than ub attackers in the sys-

tem, which violates the original *AtMost* constraint's assumption. After the SMT solver continuously returns satisfiable for T times, the approach finds all combinations of the potential attackers (Line 16–21). In the end, the approach returns the solved positions of all agents $\mathcal{P}$ if no potential attacker is found. Otherwise, the set of potential attackers $\mathcal{A}$ is returned (Line 22). Algorithm 1 calls the following key functions:

- SMT($\mathcal{C}$) tests the satisfisfiability of a constraint set $\mathcal{C}$. *sat* is a Boolean variable assigned *true* if its constraint set is satisfiable, in which case $\mathcal{P}$ contains a solution (*i.e.*, solved positions of all agents) to $\mathcal{C}$, or assigned value *false*, in which case $\mathcal{U} \subseteq \mathcal{C}$ is an unsatisfiable core.
- addRelaxation($\mathcal{C}$, ∇) adds relaxation variables after all constraints. For each constraint $c \in \mathcal{C}$, if c is related to an agent α_i, we add the corresponding relaxation variable r_i into c, and the new constraint is $c \vee r_i$.
- avoid($\mathcal{A}$) returns a constraint for avoiding finding the same combination of potential attackers as $\mathcal{A}$.

3.4 Destructive Approach

See Algorithm 2. This approach is "destructive" since the potential attackers are found by removing some agents' constraints. Inspired by [30], to avoid solving the combinatorial problem by applying brute force search, the destructive approach requires another SMT solver, also known as *candidate solver* (Line 4). When the main solver returns unsatisfiable (which means the system discovers that there exist attackers on the map), the system analyzes the unsatisfiable core, updates the candidate solver with a constraint according to the unsatisfiable core (Line 11), and

Algorithm 2: Destructive Approach

Data: $M, N, S, E, \mathcal{R}, O, T$
Result: Solved Positions $\mathcal{P}$ or Potential Attackers $\mathcal{A}$

begin
  $(\mathcal{C}, \mathcal{A}, j, t) \leftarrow (\emptyset, \emptyset, 0, 0)$
  add basic constraints to $\mathcal{C}$
  $\mathcal{S} = candidateSolver()$
  while $j < T$ **do**
    $\mathcal{C} \leftarrow$ rmRelated$(\mathcal{C} \cup \mathcal{R}^t, \mathcal{A})$
    $(sat, \mathcal{P}, \mathcal{U}) \leftarrow$ SMT$(\mathcal{C})$
    if $sat \neq true$ **then**
      $(j, ub, \mathcal{A}') \leftarrow (0, 1, \emptyset)$
      while $\mathcal{A}' = \emptyset$ **do**
        update$(\mathcal{S}, \mathcal{U})$
        $\mathcal{B} \leftarrow$ getCandidates$(\mathcal{S})$
        if $\mathcal{B} \neq \emptyset$ **then**
          while $\mathcal{B} \neq \emptyset$ **do**
            $\mathcal{C}' \leftarrow$ rmRelated$(\mathcal{C}, \mathcal{B})$
            $(sat, \mathcal{P}, \mathcal{U}) \leftarrow$ SMT$(\mathcal{C}')$
            if $sat = true$ **then**
              $\mathcal{A}' \leftarrow \mathcal{A}' \cup \mathcal{B}$
              update$(\mathcal{S}, \mathcal{B})$
            else
              update$(\mathcal{S}, \mathcal{U})$
            end
            $\mathcal{B} \leftarrow$ getCandidates$(\mathcal{S})$
          end
        else
          $ub \leftarrow ub + 1$
          update$(\mathcal{S}, ub)$
        end
      end
      $\mathcal{A} =$ combine $(\mathcal{A}, \mathcal{A}')$
    end
    $(j, t) \leftarrow (j + 1, t + 1)$
  end
  return $\mathcal{P}, \mathcal{A}$
end

asks the candidate solver to return a list of possible attacker candidates (Line 12). We are certain that at least one attacker exists when unsatisfiability occurs, and there should be some constraints related to the attacker inside the unsatisfiable core. Given an unsatisfiable core, the system analyzes and derives all agents related to the core, and adds a corresponding constraint, claiming that at least one attacker exists among those agents, into the candidate solver. If the candidate solver is unable to provide a list of candidates, it means that the upper bound for the attackers is too small. Therefore, we increase the upper bound (Lines 26–27) and ask the candidate solver to return a list once again. If the candidate solver successfully returns a list of candidates, the system then removes all the constraints related to those candidates and makes the main SMT solver check whether the remaining constraints are satisfiable (Lines 15–16). If the remaining constraints are satisfiable, it implies that all contradictory constraints (because of the attackers) are removed, and the system records the candidates as possible attackers (Lines 18–19). On the other hand, if the remaining constraints are still unsatisfiable, we can infer that there are some attacker(s) that caused the unsatisfiability but are not included in the candidate list. The system can further add a corresponding constraint to the candidate solver in the light of the unsatisfiable core (Line 21), and ask the candidate solver to generate the next candidates (Line 23). For the "check" part in Figure 3.2, the system will only record candidates and start the next round if it encounters unsatisfiability after removing constraints and has already found some potential attackers. Otherwise, it will continuously update the candidate solver to find additional potential attackers. In the end, the approach returns the solved positions of all agents $\mathcal{P}$ if no potential attacker is found. Otherwise, the set of potential attackers $\mathcal{A}$ is returned (Line 34). Algorithm 2 calls the following key functions:

- SMT($\mathcal{C}$) is identical to the one in Algorithm 1.

- rmRelated($\mathcal{C}$, $\mathcal{A}$) is a function to remove several constraints from the constraint set $\mathcal{C}$ regarding the agents set $\mathcal{A}$. For each constraint $c \in \mathcal{C}$, if there exists an agent α in $\mathcal{A}$ such that α is related to c, then c will be removed from $\mathcal{C}$. This function eventually returns the constraint set after all removals.

- getCandidates($\mathcal{S}$) asks the candidate solver $\mathcal{S}$ to generate an available candidate list. It returns an empty list $\emptyset$ when there are no available candidates.

- update($\mathcal{S}$, _) updates the candidate solver in three ways based on the second argument's type. If the second argument is an unsatisfiable core ($update(\mathcal{S}, \mathcal{U})$), it indicates that there is at least one attacker which caused the unsatisfiability, and we can add a constraint asserting at least one agent in the core is an attacker. If the second argument is a set of agents ($update(\mathcal{S}, \mathcal{A}')$), it implies that $\mathcal{A}'$ is a set of potential attackers, and we add a constraint into the candidate solver to avoid generating an identical list again. If the second argument is an upper bound value ($update(\mathcal{S}, ub)$), it means that the upper bound is so low that the candidate solver cannot generate an available integer, and therefore we increase the upper bound by one.

- combine($\mathcal{A}, \mathcal{B}$) returns all possible combinations of two lists. For example, if $\mathcal{A} = [[1]]$ and $\mathcal{B} = [[2], [3]]$, the output results of combine($\mathcal{A}, \mathcal{B}$) is $[[1, 2], [1, 3]]$.

Chapter 4

Experimental Results

In this chapter, we demonstrate the experiments in detail. We first introduce the experimental setting in Section 4.1. The results for experiments with no attackers are presented in Section 4.2, while those with attackers are in Section 4.3. We discuss and conclude the results in Section 4.4.

4.1 Experimental Setting

We elucidate basic setting, broadcast positions sampling, attacker generation, and methods to evaluate the performance of the two proposed approaches as follows.

4.1.1 Basic Setting

All experiments are run on a MacBook Pro with Apple M1 Chip and 8 GB LPDDR4 memory. All computations are performed with PYTHON, and we use Z3 in *pySMT* [7] package as the SMT solver in both approaches. For each setting, we randomly generate 100 test cases. The real positions for all agents are randomly generated during every test case, and observations O of all agents are further computed based on their real positions. The attackers perform *No Solution* attacks throughout all experiments so that we can compare the solving times for $\mathcal{A}$. The size of each agent S is 1.0, and the error shift E is 1.0 in each experiment.

4.1.2 Broadcast Positions Sampling

To simulate the error shift in the experiments, we model the error as a Gaussian distribution along both axes, with the mean and the standard deviation equal to 0 and $\frac{E}{3}$, respectively. Precisely, let $N(\mu, \sigma)$ be the notation of a Gaussian distribution, where μ is the mean and σ is the standard deviation. The x-coordinate of α_i's broadcast position in round t, X_i^t, is sampled at the beginning of the round from the distribution $N(x_i, \frac{E}{3})$, and samely, the y-coordinate, Y_i^t, is sampled from $N(y_i, \frac{E}{3})$. The system samples an agent's broadcast position iteratively until it differs from α_i's actual position by at most E along each axis, as described in Section 2.5.

4.1.3 Attacker Generation

As mentioned in Section 2.4, the attackers own thorough information and outstanding computational power to derive needed information, including which fake coordinate to attack. To generate the attack in the following experiments, we derive the fake position for each attacker trying to attack before the system flow starts. Since all attackers perform *No Solution* attacks, for each attacker, we repeatedly choose a random fake coordinate until it violates other constraints by using another SMT solver to check. During the experiment, the attacker cheats its position according to the fake coordinate. Note that the solving times of each approach do not include finding the fake positions for attackers.

4.1.4 Baseline Approach

We design a straightforward approach for comparing the performance of two proposed approaches, and we call it the *baseline approach*. The baseline approach uses brute force search, assuming that an agent is an attacker one by one and removing all the constraints related to the agent when the system SMT solver returns

unsatisfiability. If the remaining constraints are satisfiable, the system has removed all the attackers causing the contradiction. The removed agent is considered a possible attacker, and the system appends it to the set of potential attackers $\mathcal{A}$. If no possible attackers are found based on the current assumption, the system will increase the assumed number of potential attackers. For instance, if the system initially assumes that there is only one attacker but cannot identify any possible attacker, it will then proceed to assume that there are two attackers.

4.1.5 Evaluation Metrics

When the system finds possible attackers on the map, it returns $\mathcal{A}$ in the end. To evaluate the performance, besides the average and the standard deviation of solving times for deriving $\mathcal{A}$, we design four additional metrics to assess these approaches:

- **Precision**. The precision score of an output is the percentage of agents in $\mathcal{A}$ that are real attackers.
- **Recall**. The recall score of an output is the percentage of attackers that appear in $\mathcal{A}$.
- **Success**. An output is counted as a success only if the real attackers are one of the combinations in $\mathcal{A}$.
- **Ambiguity**. If an output is counted as a success, its ambiguity score is the reciprocal of $\mathcal{A}$'s length.

Table 4.1: Average of Euclidean distances between each agent's solved solution and real position.

System Flow	(M, N)				
	(10,5)	(10,10)	(20,10)	(10,20)	(20,20)
Baseline	0.51	0.50	0.51	0.46	0.49
Constructive	0.51	0.48	0.51	0.41	0.48
Destructive	0.51	0.50	0.51	0.46	0.49

For example, if the real attackers are α_0 and α_1, and $\mathcal{A} = \{[0,1],[0,2]\}$, which means the system states that either "α_0 and α_1 are the attackers" or "α_0 and α_2 are the attackers". The precision score of this $\mathcal{A}$ is $\frac{2}{3}$, because there are two real attackers (α_0, α_1) in $\mathcal{A}$ (α_0, α_1, α_2). The recall score of this $\mathcal{A}$ is 1, since all real attackers (α_0, α_1) are included in $\mathcal{A}$. This $\mathcal{A}$ is counted as a success, as the real attacker combination($[0,1]$) is one of those in $\mathcal{A}$, and the ambiguity score is $\frac{1}{2}$.

4.2 Experiments without Attackers

In this section, $T = 25$ is fixed, and there are no attackers on the map yet. As mentioned in Section 3.2, the system outputs a solution (*i.e.*, a possible coordinate for each agent) when there are no potential attackers found. Table 4.1 shows the average of the Euclidean distances between each agent's real position and solved position. The results show that there is little difference between the three approaches and five different map settings, and it is reasonable since the main factor causing the error between the two positions is the error shift E, which is fixed among all experiments.

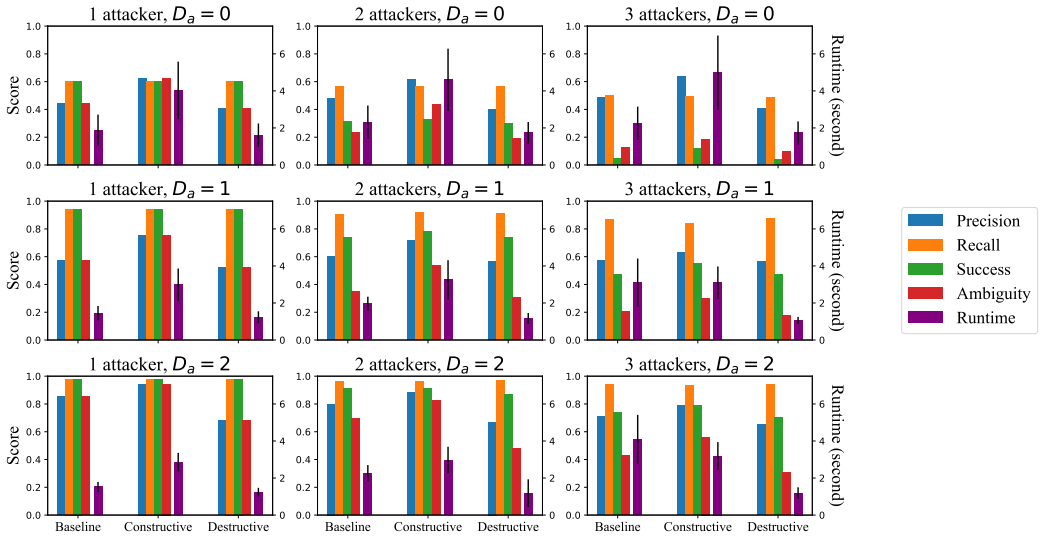


Figure 4.1: Experimental results with different numbers of attackers and D_a.

4.3 Experiments with Attackers

4.3.1 Different Number of Attackers and Different D_a

Let $D_a = |\{d : F(a, d) \neq \emptyset\}|$ be the number of directions that an attacker α_a is observed by others. In this section, $(M, N, T) = (20.0, 10, 25)$ are fixed. Attackers are randomly chosen among all agents α_i, where D_i is fixed at a specific number. Figure 4.1 compares the results of the two proposed approaches and the baseline. We can observe that the more directions that the attackers are observed by the other agents, the shorter the solving times and the higher the scores of the results, and thus it can be seen that having more observations between agents helps the system find the attackers more efficiently and more accurately. We can also discern that the average solving times of the constructive approach are longer than those of the destructive one. Further explanations will be covered in Section 4.4.

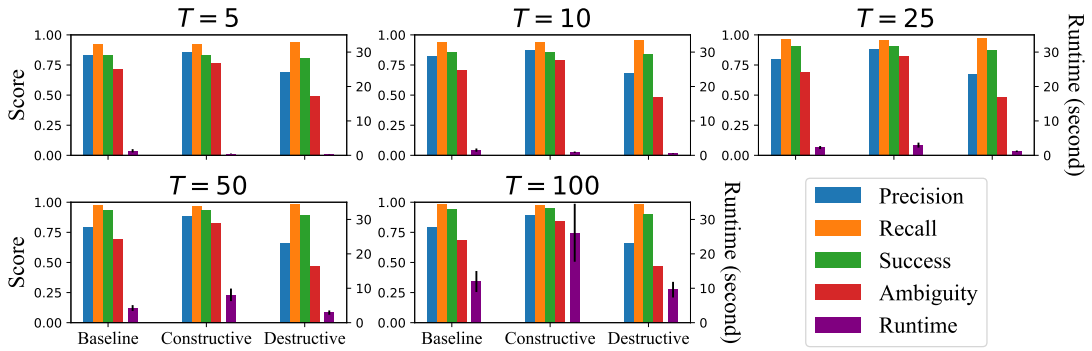


Figure 4.2: Experimental results with different T.

4.3.2 Different T

In this section, $(M, N) = (20.0, 10)$ are fixed, and two attackers are randomly chosen among all agents α_i, where $D_i = 2$. We set the number of maximum round $T = \{5, 10, 25, 50, 100\}$. Figure 4.2 compares the results of the two approaches and the baseline with different T. We can observe that in all three approaches, the more rounds the system runs, the higher the recall scores and success rates are. However, there is a trade-off between solving times and the score of results. The destructive approach requires the least solving times among all settings. The baseline approach consumes the longest average solving times when $T = 5$ and $T = 10$, and a possible reason is that doing a brute force search in the baseline approach is more time-consuming than solving a complicated SMT problem T times in the constructive approach when T is relatively small. Differences between the two proposed approaches' results are the trend of precision score and ambiguity score. On one hand, the constructive approach receives higher precision and ambiguity scores when T is greater. On the other hand, those two scores drop when raising T in the destructive one. We will also give explanations pertaining to this scenario in Section 4.4.

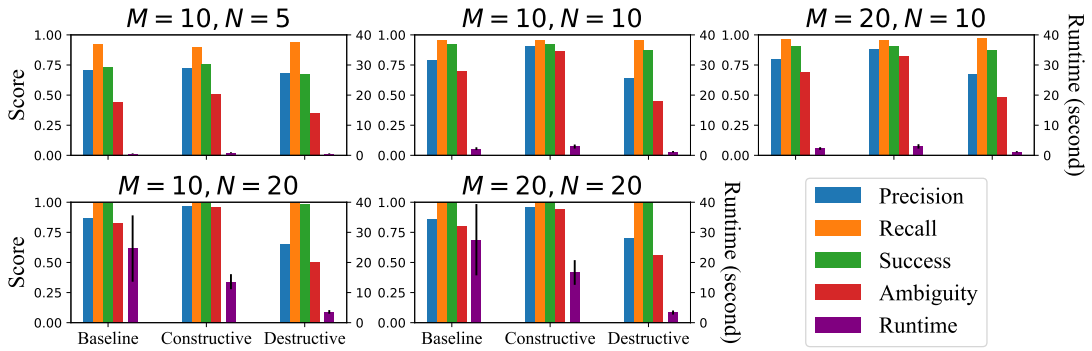


Figure 4.3: Experimental results with different (M, N).

4.3.3 Different M and N

In this section, $T = 25$ are fixed, and two attackers are randomly chosen among all agents α_i, where $D_i = 2$. We set $(M, N) = \{(20, 10), (10, 5), (20, 20), (10, 10), (10, 20)\}$, with the aim of finding the relation between map size M, agent count N, and the performance. Figure 4.3 shows the results of the two approaches and the baseline with different (M, N). One can notice that the main factor which affects the solving time and scores is N. When fixing M, *i.e.*, fixing the map size, the more agents on the map, the higher scores it gets. A plausible reason is that having more agents on the map implies having more constraints to be put into the SMT solver. The more constraints to be satisfied at the same time, the higher opportunity to reveal the attackers, but it will also cost the SMT solver more time to determine satisfiability. Conversely, when fixing N, the change in solving times and performance scores are relatively small. The baseline approach suffers relatively long solving times when $N = 20$, since it uses brute force search when meeting unsatisfiability during the system flow. In contrast, the destructive approach's solving times don't rise much when N goes greater, and the reason is that it removes constraints from the SMT solver after finding potential attackers, which can ease the burden of the solver when N is large.

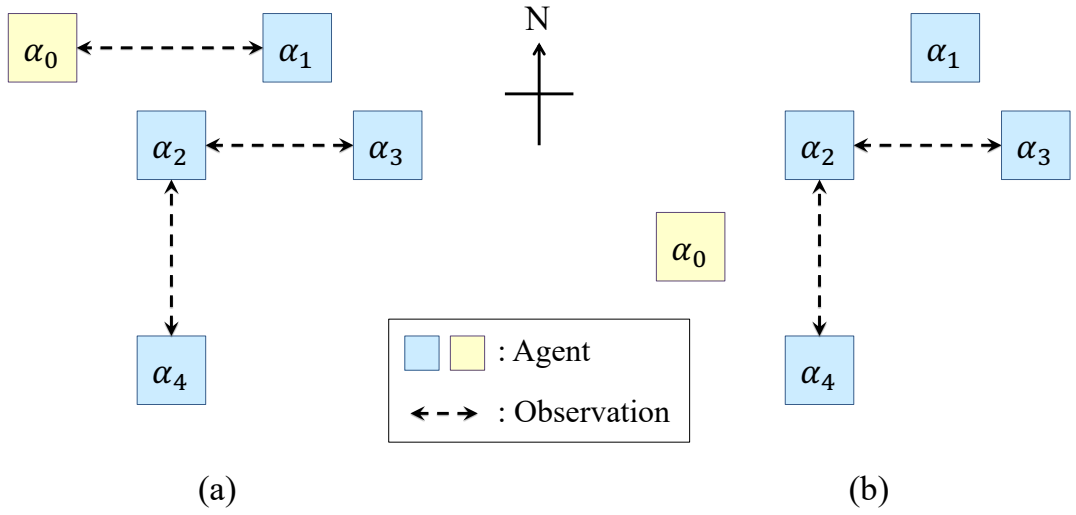
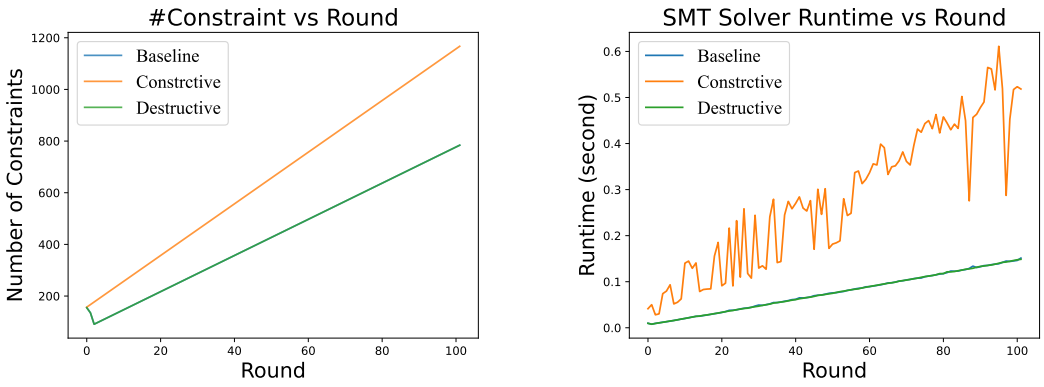


Figure 4.4: An example illustrates a map with 5 agents, where α_0 is the only attacker. An arrow between two agents indicates that those agents observe each other. (a) The real positions of all agents. (b) The broadcast positions of all agents.

4.4 Discussion

We first illustrate why the precision and ambiguity scores are mainly between 60% and 80%, rather than the deep-learning-based methods whose scores usually exceed 90%. Figure 4.4 is an example with five agents on the map, and an arrow indicates an observation between those two agents. Figure 4.4(a) and (b) show the real and the broadcast positions of all agents, respectively. In this example, α_0 is the only attacker that broadcasts incorrect position. Three observations are made between (α_0, α_1), (α_2, α_3), and (α_2, α_4). To be more precise, we have the observation functions $F_{0,east} = [1]$ and $F_{1,west} = [0]$. The SMT solver finds the constraints unsatisfiable. The reason is that based on the position broadcast by each agent, there should be no observation between α_0 and α_1, which violates the observations made by these two agents. Whatever proposed approach we apply, after removing (ignoring) all constraints related to either α_0 or α_1, the remaining constraints will be satisfiable, and the result set of potential attackers $\mathcal{A} = \{[0], [1]\}$.



(a) (b)

Figure 4.5: A result when two attackers exist and $(M, N, T, o_a) = (20, 10, 100, 2)$. (a) Number of constraints in the SMT solver in each round. (b) Solving time in each round. The lines for the baseline and destructive approaches overlap in (a) and (b). The difference between these approaches lies in identifying potential attackers after the SMT solver returns unsatisfiability.

According to the evaluation metrics introduced in Section 4.1.5, the recall score in the example is 100%, while the precision and ambiguity scores are both 50%. We can see that the potential attackers usually lie in one of two (or even one of three) agents, and we cannot further narrow it down since the observations are not plenty.

We can notice that the solving times of the destructive approach are less than those of the constructive one. The reason for this is the number of constraints in the SMT solver (Figure 4.5(a)). When the constructive approach meets unsatisfiability, it adjusts the upper bound of the number of true relaxation variables, and no constraints are removed. In contrast, when the destructive one meets unsatisfiability, it attempts to identify the attackers causing the error and ignores all the constraints related to those potential attackers in the following rounds. Therefore, the removal of the constraints in the destructive approach eases the burden on the SMT solver.

Next, we can see that the standard deviations of the solving times in the constructive approach are greater than those in the destructive one. One potential

cause for this is the solving time in each round (Figure 4.5(b)). Compared to the destructive one, the solving times in the constructive approach throughout each round is more fluctuate, resulting in a larger amount of variability and diversity among the total solving times. This is because the constraints put into the SMT solver in the constructive approach are more complicated, as mentioned in Section 3.2. In contrast, the solving times in the destructive approach throughout each round are almost linear, and thus the variability of the solving times is relatively low.

Last, the recall scores and success rates are similar in both proposed approaches across all experiments. However, the constructive approach outperforms the destructive one in terms of precision and ambiguity scores. This results from the "cautiousness" of the constructive approach. When encountering unsatisfiability, unlike the destructive approach that removes all constraints related to the agents which can cause mistakes, the constructive approach only adjusts the upper bound of the attackers. As a result, the constructive one is able to preserve all constraints until the final round and generate the potential attacker list $\mathcal{A}$. Because of taking all constraints into consideration at the same time, the constructive approach is capable of outputting $\mathcal{A}$ more precisely, as well as enhancing ambiguity scores. On the other hand, the destructive approach may add a certain number of agents, including innocent good agents typically, into $\mathcal{A}$. Those agents lose their opportunity to prove their innocence in the following rounds, hence reducing precision and ambiguity scores. This also explains the relationship between the number of maximum round T and the performance. With a greater T, the constructive approach is allowed to collect more constraints and make the result more precise. On the flip side, in the destructive approach, a higher value of T may increase the likelihood of finding additional attackers, but it can also result in increased combinations in $\mathcal{A}$, leading to lower precision and ambiguity scores.

Chapter 5

Conclusions

In this thesis, we targeted cooperative positioning for connected multi-agent systems. We considered positioning errors and that attackers intentionally provide false information. We designed SMT-based approaches, constructive and destructive, aiming to reach a positioning between agents and find potential attackers. The experimental results showed that by leveraging the SMT solver, the accuracies and runtimes can be improved by using the constructive and destructive approaches, respectively, compared to the baseline approach.

There are a few directions for future work. First, we can achieve better performance by adopting more constraints, but redundant constraints can also burden the SMT solver. For this reason, the minimization of additional information is crucial. Second, after the system notices the existence of the attackers, it is important to discover their real positions on the map immediately. The uncertainty of the attackers' positions can harm the other agents' safety, and therefore designing efficient methods to localize them is critical. Furthermore, we would like to consider the problem with moving agents involved. The real positions of all agents, including attackers, change over time, and we will need to reformulate the constraints regarding the relations between broadcast positions in different timestamps.

Bibliography

[1] F. G. Abdulkadhim, Z. Yi, C. Tang, A. N. Onaizah, and B. Ahmed, "Design and development of a hybrid (SDN + SOM) approach for enhancing security in VANET," *Applied Nanoscience*, vol. 13, no. 1, pp. 799–810, 2023.

[2] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): Part II," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

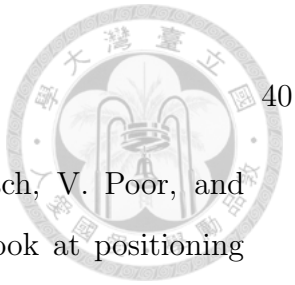
[3] T. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," *arXiv preprint arXiv:1712.09665*, 2017.

[4] Y. Cai and Y. Shen, "An integrated localization and control framework for multi-agent formation," *IEEE Transactions on Signal Processing*, vol. 67, no. 7, pp. 1941–1956, 2019.

[5] D. Droeschel and S. Behnke, "Efficient continuous-time SLAM for 3D lidar-based online mapping," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5000–5007. IEEE, 2018.

[6] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.

[7] M. Gario and A. Micheli, "PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms," in *SMT Workshop 2015*, 2015.



[8] S. Gezici, Z. Tian, G. Biannakis, H. Kobayashi, A. Molisch, V. Poor, and Z. Sahinoglu, "Localization via ultra-wideband radios: a look at positioning aspects for future sensor networks," *IEEE Signal Processing Magazine*, vol. 22, no. 4, pp. 70–84, 2005.

[9] Z. Hong, Y. Petillot, and S. Wang, "Radarslam: Radar based large-scale slam in all weathers," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5164–5170. IEEE, 2020.

[10] S.-T. Hou, "Cooperative and secure multi-agent positioning based on satisfiability modulo theories," Master's thesis, National Taiwan University, 2021.

[11] M. H. Ikram, S. Khaliq, M. L. Anjum, and W. Hussain, "Perceptual aliasing++: Adversarial attack for visual SLAM front-end and back-end," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4670–4677, 2022.

[12] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil, "Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 4, pp. 584–616, 2011.

[13] M. Karrer, P. Schmuck, and M. Chli, "CVI-SLAM—collaborative visual-inertial SLAM," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2762–2769, 2018.

[14] R. Kaur, T. P. Singh, and V. Khajuria, "Security issues in vehicular ad-hoc network (VANET)," in *International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 884–889, 2018.

[15] S.-H. Kong and S.-Y. Jun, "Cooperative positioning technique with decentralized malicious vehicle detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 3, pp. 826–838, 2018.

[16] M. Lee and T. Atkison, "Vanet applications: Past, present, and future," *Vehicular Communications*, vol. 28, p. 100310, 2021.

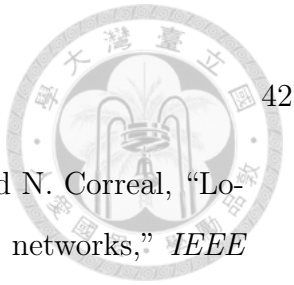
[17] S. Liu, D. He, Y. Xu, C. Zhang, S. Sun, and D. Ru, "Adaptive vehicle cooperative positioning system with uncertain GPS visibility and neural network-based improved approach," in *IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, pp. 303–308, 2018.

[18] M. J. N. Mahi, S. Chaki, S. Ahmed, M. Biswas, S. Kaiser, M. S. Islam, M. Sookhak, A. Barros, and M. Whaiduzzaman, "A review on VANET research: Perspective of recent emerging technologies," *IEEE Access*, 2022.

[19] J. Marques-Silva and I. Lynce, "On improving MUS extraction algorithms," in *Theory and Applications of Satisfiability Testing-SAT 2011: 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings 14*, pp. 159–173. Springer, 2011.

[20] F. Meyer, O. Hlinka, H. Wymeersch, E. Riegler, and F. Hlawatsch, "Distributed localization and tracking of mobile networks including noncooperative objects," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 2, no. 1, pp. 57–71, 2016.

[21] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.



[22] N. Patwari, J. Ash, S. Kyperountas, A. H. III, R. Moses, and N. Correal, "Locating the nodes: cooperative localization in wireless sensor networks," *IEEE Signal Processing Magazine*, vol. 22, no. 4, pp. 54–69, 2005.

[23] M. Poongodi, M. Hamdi, A. Sharma, M. Ma, and P. K. Singh, "DDoS detection mechanism using trust-based evaluation system in VANET," *IEEE Access*, vol. 7, pp. 183 532–183 544, 2019.

[24] D. Ramphull, A. Mungur, S. Armoogum, and S. Pudaruth, "A review of mobile ad hoc network (MANET) protocols and their applications," in *2021 5th international conference on intelligent computing and control systems (ICICCS)*, pp. 204–211. IEEE, 2021.

[25] L. Riazuelo, J. Civera, and J. M. Montiel, "C^2TAM: A cloud framework for cooperative tracking and mapping," *Robotics and Autonomous Systems*, vol. 62, no. 4, pp. 401–413, 2014.

[26] F. Safari, S. Izabela, H. Kunze, and D. Gillis, "'the diverse technology of MANETs: A survey of applications and challenges," *International Journal of Future Computer and Communication*, vol. 12, no. 2, 2023.

[27] P. Schmuck and M. Chli, "CCM-SLAM: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams," *Journal of Field Robotics*, vol. 36, no. 4, pp. 763–781, 2019.

[28] K. N. Tripathi, S. C. Sharma, and A. M. Yadav, "Analysis of various trust based security algorithm for the vehicular AD-HOC network," in *International Conference on Recent Innovations in Electrical, Electronics Communication Engineering (ICRIEECE)*, pp. 1546–1551, 2018.

[29] P. Tyagi and D. Dembla, "Investigating the security threats in vehicular ad hoc networks (VANETs): Towards security engineering for safer on-road transportation," in *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 2084–2090, 2014.

[30] M. Weber, B. Jin, G. Lederman, Y. Shoukry, E. A. Lee, S. Seshia, and A. Sangiovanni-Vincentelli, "Gordian: Formal reasoning-based outlier detection for secure localization," *ACM Transactions on Cyber-Physical Systems*, vol. 4, no. 4, pp. 1–27, 2020.

[31] H. Wymeersch, J. Lien, and M. Z. Win, "Cooperative localization in wireless networks," *Proceedings of the IEEE*, vol. 97, no. 2, pp. 427–450, 2009.

[32] B. Yu, C.-Z. Xu, and B. Xiao, "Detecting sybil attacks in VANETs," *Journal of Parallel and Distributed Computing*, vol. 73, no. 6, pp. 746–756, 2013.

[33] D. Zou and P. Tan, "CoSLAM: Collaborative visual SLAM in dynamic environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 2, pp. 354–366, 2012.

Appendix

The results (Table 5.1, Table 5.2, and Table 5.3) are presented with the mean solving time (Mean) and the standard deviation (STD) in seconds, while the precision score (PRE), the recall score (REC), the success rate (SUC), and the ambiguity score (AMB) are presented using percentages.

Table 5.1: Experimental results with different numbers of attackers and different numbers of observers D_a (Mean & STD in seconds; the others in percentages).

Number of Attackers		1			2			3		
Number of Observers		0	1	2	0	1	2	0	1	2
Baseline	Mean	1.88	1.45	1.52	2.31	1.95	2.26	2.25	3.10	4.09
	STD	0.84	0.39	0.27	0.90	0.39	0.44	0.90	1.30	1.32
	PRE	44.4	57.3	85.4	47.8	59.9	80.0	48.8	57.6	70.8
	REC	60.0	94.0	98.0	56.5	90.5	96.5	50.3	87.0	93.7
	SUC	60.0	94.0	98.0	31.0	74.0	91.0	5.0	47.0	74.0
	AMB	44.4	57.3	85.4	23.4	35.2	69.4	12.7	20.2	42.7
Constructive	Mean	4.03	2.98	2.86	4.60	3.25	2.97	4.98	3.10	3.19
	STD	1.55	0.88	0.50	1.68	1.06	0.72	2.01	0.87	0.75
	PRE	62.5	75.4	94.4	61.4	71.5	88.4	63.4	63.1	78.8
	REC	60.0	94.0	98.0	56.5	92.0	96.0	49.0	84.0	93.0
	SUC	60.0	94.0	98.0	33.0	78.0	91.0	12.0	55.0	79.0
	AMB	62.1	75.4	94.4	43.4	53.7	82.6	18.2	29.8	56.2
Destructive	Mean	1.61	1.23	1.24	1.73	1.16	1.18	1.73	1.06	1.19
	STD	0.63	0.32	0.23	0.59	0.30	0.22	0.62	0.19	0.31
	PRE	40.5	52.5	67.9	40.0	56.3	67.0	40.4	56.2	65.0
	REC	60.0	94.0	98.0	56.5	91.0	97.0	48.7	87.7	94.0
	SUC	60.0	94.0	98.0	30.0	74.0	87.0	4.0	47.0	70.0
	AMB	40.5	52.5	67.9	19.3	30.6	48.2	9.9	17.6	30.4

Table 5.2: Experimental results with different T
(Mean & STD in seconds; the others in percentages).

System Flow		Maximum Round (T)				
		5	10	25	50	100
Baseline	Mean	1.32	1.52	2.26	4.26	11.96
	STD	0.48	0.47	0.44	0.84	3.05
	PRE	83.1	82.1	80.0	79.4	79.4
	REC	92.5	94.0	96.5	97.5	98.0
	SUC	83.0	86.0	91.0	93.0	94.0
	AMB	71.8	71.1	69.4	69.2	68.8
Constructive	Mean	0.45	0.91	2.97	8.09	26.13
	STD	0.07	0.19	0.72	1.82	8.42
	PRE	85.9	86.9	88.4	88.1	89.6
	REC	92.5	94.0	96.0	97.0	97.5
	SUC	83.0	86.0	91.0	93.0	95.0
	AMB	76.6	78.7	82.6	82.7	84.6
Destructive	Mean	0.34	0.50	1.18	2.93	9.59
	STD	0.05	0.08	0.22	0.63	2.27
	PRE	69.0	68.2	67.0	66.1	66.3
	REC	94.0	95.5	97.0	98.0	98.5
	SUC	81.0	84.0	87.0	89.0	90.0
	AMB	48.8	48.0	48.2	47.1	46.9

Table 5.3: Experimental results with different M and N
(Mean & STD in seconds; the others in percentages).

System Flow		(M, N)				
		(10,5)	(10,10)	(20,10)	(10,20)	(20,20)
Baseline	Mean	0.55	2.21	2.26	24.62	27.56
	STD	0.10	0.47	0.44	11.02	11.84
	PRE	71.1	78.9	80.0	86.7	86.0
	REC	92.5	96.0	96.5	99.5	100.0
	SUC	73.0	92.0	91.0	99.0	100.0
	AMB	44.4	69.5	69.4	82.4	80.2
Constructive	Mean	0.87	2.92	2.97	13.61	16.67
	STD	0.16	0.67	0.72	2.50	4.10
	PRE	72.6	90.7	88.4	96.9	96.3
	REC	89.5	96.0	96.0	99.5	100.0
	SUC	76.0	92.0	91.0	99.0	100.0
	AMB	50.7	86.1	82.6	95.7	94.3
Destructive	Mean	0.53	1.18	1.18	3.58	3.37
	STD	0.12	0.26	0.22	0.60	0.68
	PRE	68.6	63.7	67.0	64.8	70.1
	REC	94.0	96.0	97.0	99.5	100.0
	SUC	67.0	87.0	87.0	98.0	100.0
	AMB	35.3	44.7	48.2	50.2	56.4