Game Theoretic Modeling of Patrolling Stackelberg Games for MAS

Emma Chabane and Vinh Le
16.412/6.834 Grand Challenge - Spring 2022

Abstract

With Bayesian Stackelberg games we can frame a patrolling scenario as a solvable problem. Normally, those who are patrolling are at a direct disadvantage compared to those who are trying to attack, since they have to make their move first and they can be observed. However, this exact scenario can be framed as a Stackelberg game where the leader is the security doing patrol and the follower is an attacker. While accounting for possible attacker types, the optimal policy for the leader is solved for using a Decomposed Optimized Bayesian Stackelberg Solver. For games where number of valuables, length of patrol, or number of attacker types remains under 5, the game is solved on average under .10 seconds.

Motivation

Security technological innovations have recently been a prevalent area of interest for many researchers. In the wake of new various threats in the 21st century, developing best responses to security attacks has been a critical concern in the United States and the world. For our team, security problems were quite an interesting topic to look at. Indeed, we were hoping to develop our knowledge of game theoretic modeling, while also studying an area that could truly be helpful to society. Because of the diverse amount of security and safety scenarios that exist nowadays, from defending infrastructure to protecting flora and fauna, this domain is truly interesting. Often, security scenarios involve various adversarial actors with different goals in mind, and complex variables that need to be taken into account. As such, security scenarios can be modeled well using Stackelberg security games, and game theory is well-suited to represent these kinds of situations. Various research projects have been started using Stackelberg game modeling for security applications, and we have been seeing very promising results from our literature review.

Related Work

We were inspired by research projects that have been studied and implemented in real-life scenarios. Many applications use Stackelberg games and a solving algorithm to better allocate resources, optimize efficiency, and schedule security checkpoints for instance.

Patrol and resource allocation

The first project we read about was a resource and patrol allocation project using the A.R.M.O.R. system (Assistant for Randomized Monitoring over Routes), used by Los Angeles International Airport (Pita et al., 2008). In a huge airport such as LAX, thousands of passengers come by every day and travel through different kinds of roads and terminals. Various variables come into play, such as traffic flow, size, flight type, etc… thus making this a complex security problem.

A key problem that arises in many security scenarios is the fact that our "defenders" have limited resources (i.e. only so many patrollers, agents,...). This approach therefore needs to better allocate the airport's resources to optimize security in all the parts of the airport. The goal of this game theoretic model is to therefore represent the scenario as a Bayesian Stackelberg game. The airport has different leaders: the airport patrollers and K9 dogs, and can be under different kind of threats (e.g. bombs). A.R.M.O.R. uses the DOBSS algorithm, which we will delve into in the Implementation section.

A significant amount of research papers have also focused on resource allocation for security Stackelberg games, see Shieh et al. for an overview of this game theoretic modeling in the context of the US Coast Guard.

Utilizing complex behavioral data to better allocate resources

A second, and perhaps the one that was initially most interesting to us was the P.A.W.S. algorithm for environmental and animal protection. The research proposed by Yang et al. aims to protect endangered species from illegal poaching using an improved and optimized Stackelberg game formulation. The researchers improved their model using behavioral data based on the poacher's heterogenous decision process. Based on this extension, they were able to improve their model to a PAWS-Learn framework that betters predicts where and how poachers will act. By leveraging game theory and machine learning, PAWS was able to best predict where attackers will be and help forest rangers protect their environment. Their method was trialed in Uganda's Queen Elizabeth National Park and proved to be quite successful! Environmental challenges were quite interesting to us as it is a topic that is quite relevant nowadays and we toyed around with the idea of creating an environmental protection bot.

Problem statement

Our project was inspired by these different security and protection concerns. Based on our work from the advanced lecture and our literature review, we wanted to see how game theoretic modeling could be a solution to these challenges and explore a novel side of game theory for MAS that we had not studied before. Previous research we studied often focused on threats to national and federal infrastructure or on environmental protection. However, there exists much frequent security threats everywhere in the world: burglaries. In the United States, there are almost three burglaries every minute. Therefore, for our Grand Challenge, we ultimately decided to focus on a robot that can be used by every single one of us, in an environment familiar to all of us: our own homes. In this light, our bot approaches security challenges from a very different angle, and we are excited about applying our game theory model to scenarios that have not been studied in liaison with Stackelberg games. As such, in this paper, we propose Inspecti-bot, a home robot-patroller that uses Bayesian Stackelberg game modeling to correctly and efficiently deploy patrol teams to counteract potential adversarial agents.

Background

Up to now, we have mentioned Stackelberg games quite a bit. What are Stackelberg games, and why are they so well-suited for security challenges? Generally, a Stackelberg game is composed of two players: a leader and a follower. Although this makes it seem like these kind of games can only represent situations between two people, they can actually do quite a bit more. Indeed, the leader and the follower need not represent two individuals, rather, they can represent adversarial teams, the leader team often being the canonical "good guys" and defenders, while the follower team are seen as the threats. Each player has a set of different actions they can take. For security Stackelberg games, often, these different actions are which places to go to so that a leader can best intercept a follower (see PAWS and ARMOR), which is also the case for our application. The leader must first commit to a strategy before a follower can choose a strategy. As you can imagine, this is quite reflective of real-life scenarios: before a real life attack, followers will survey a leader's strategy and actions to best augment their chances of winning. Yet, leaders usually only have one type: police forces, rangers, and patrollers follow a rigorous set of rules that are not often interchangeable. This makes Stackelberg games an excellent tool to illustrate security challenges, and also means researchers have studied how to optimize leader strategy at length, to give them the best chances of intercepting an attacker.

There are various algorithms that have been used to solve Stackelberg games: simpler approaches, like the multiple-LPs algorithms, helps us find the best strategy for normal Stackelberg games by playing a set of optimization problems. This method is however not as well-suited when we include randomness in the environment. Frameworks like PAWS and FORTIFY provide promising results and have been introduced by research papers, though there unfortunately exists much less resources and algorithmic explanations for those systems.

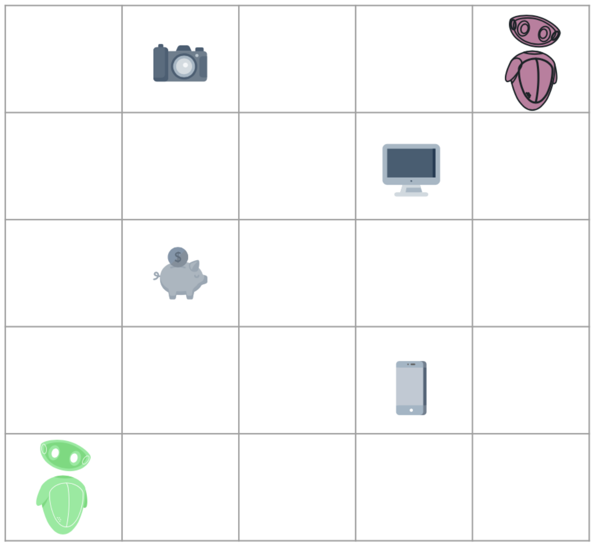
Figure 1: A grid world representation of our problem

Method

Scenario modeling

We modeled our scenario as follows (Figure 1): The Inspecti-bot must continuously protect a house by monitoring its valuables. Valuables are represented by a set of items of different values, such as a television, a phone, a safe with money, and a camera. As per the Stackelberg game framework, there are two teams of players: a leader, who is here the Inspecti-bot, and a follower, the potential attackers or burglars.

In our advanced lecture, we choose to approximate POSGs to a smaller set of solvable Bayesian games. Thus, in our challenge, we chose to extend Stackelberg games to a Bayesian environment by including randomness of the player types. Indeed, not all attackers will follow the same kind of strategy or think in similar ways: there could be risky attackers, who would rather heighten risk by going for higher-value items, or attackers that play it safe and go for lesser-value items which have less risk of being monitored.

Our Inspecti-bot does not know the attacker's type, but we must optimize our solution set to find the best strategy in order to counteract the attacker.

Algorithm choice

Our method of choice was a Decomposed Optimized Bayesian Stackelberg Solver (DOBSS). DOBSS is a way to get an exact solution for a Stackelberg game quickly and

$$
\begin{array}{ll}
\max_{q,z,a} & \displaystyle\sum_{i\in X}\sum_{l\in L}\sum_{j\in Q} p^l R^l_{ij} z^l_{ij} \\
\text{s.t.} & \sum_{i\in X}\sum_{j\in Q} z^l_{ij} = 1 \\
& \sum_{j\in Q} z^l_{ij} \leq 1 \\
& q^l_j \leq \sum_{i\in X} z^l_{ij} \leq 1 \\
& \sum_{j\in Q} q^l_j = 1 \\
& 0 \leq (a^l - \sum_{i\in X} C^l_{ij}(\sum_{h\in Q} z^l_{ih})) \leq (1-q^l_j)M \\
& \sum_{j\in Q} z^l_{ij} = \sum_{j\in Q} z^1_{ij} \\
& z^l_{ij} \in [0\ldots 1] \\
& q^l_j \in \{0,1\} \\
& a^l \in \Re
\end{array}
$$

Figure 2: DOBSS Algorithm

efficiently. DOBSS is essentially a mixed-integer linear program, and we are optimizing for the leader strategy rather than searching for a Nash Equilibrium. Also, for this implementation we consider only the pure strategies of adversaries rather than mixed strategies. The combination of all these traits leads to a efficient algorithm. X and Q respectively represent sets of leader and follower's pure strategies. The leader's policy is denoted by x. The set of follower types is denoted by $l \in L$, q^l denotes a vector of strategies. R^l and C^l respectively represent payoff matrices for leader and follower. M is also just a large positive number. The given *a priori* probabilities are given as p^l. We do the following change of variables to help linearize the quadratic programming problem $z^l_{ij} = x_i q^l_j$. With all these variables the leader solves this following linear problem (Figure 2). To make sense of the algorithm above, everything after the "s.t." are the constraints to solving our problem. Constraints 1-4 define the set of feasible solutions. Constraints 2 and 5 limit the actions to be a pure distribution over the set Q.

Python implementation

Game Framework

This implementation was done in Python making use of the PuLP package to solve linear programs[1]. We had a script games.py to create the PatrolGame instance which will contain all the important variables for solving with DOBSS: m(number of targets), d(length of the patrol/policy), num_attacker_types(number of types for follower), items(list of the valuables), item_prob(an a priori list of probabilities corresponding to each valuable for how likely an attacker will appear there), attacker_types(the types of attackers), attacker_type_prob(probabilities for attacker to be each type). With these variables we can generate the necessary details for the DOBSS. We generate possible attacker and defender strategies. And also, we are able to get attacker and defender payoff matrices.

DOBSS

We first use the PuLP package to set up the linear problem to add variables to[2]. We then set up our objective function and our constraints. We then define a solver function to solve the linear problem, and output optimal defender and attacker strategies.

[1]https://github.com/vinhle169/Grand-Challenge-16.412/blob/main/games.py

Figure 3: AWS WorldForge house simulation

Demo/Visualizer

Python Visualization For the visualizer, we iterate through each step in the patrol length. And extract the next best move from both the attacker and the defender. We then check for any conflict resolution. In this demo, we don't reset the game after the attacker has been caught. However, if an attacker successfully takes an item the Defender must then recalculate its patrol[3].

AWS RoboMaker/WorldForge Visualization Using AWS WorldForge, we were able to create templates for what potential environments would look like for our problem (Figure 3). We toyed around with different kind of environments and made houses with various number of rooms and objects. We aimed to use those environments, implement them with our Python implementation, and simulate them on Gazebo to show how an Inspecti-bot would move around a house to protect it from potential attackers, though this turned out more complicated than expected due to unfamiliarity with the system. With some help from the Safeti-bot team, we got set up on AWS RoboMaker and were able to learn a lot about ROS and simulation/visualization environments, as well as use the AWS navigation sample files. Figure 4 shows an example of what WorldForge environments could have looked like on Gazebo.

[2]https://github.com/vinhle169/Grand-Challenge-16.412/blob/main/dobbs.py

[3]https://github.com/vinhle169/Grand-Challenge-16.412/blob/main/play$_{stackelberg.py}$

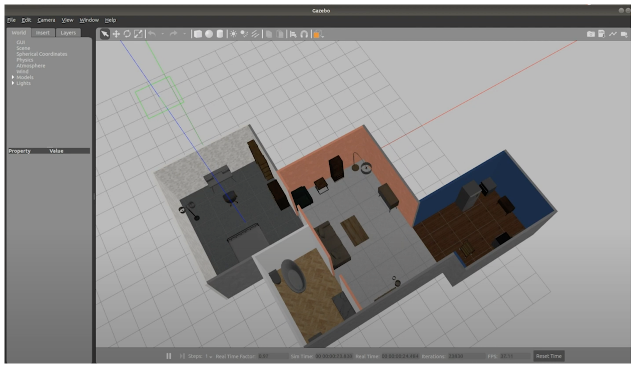

Figure 4: WorldForge house on Gazebo

Evaluation

We wanted to optimize inspecti-bot to perform well in different kinds of security scenarios, as well as to see how well we can scale our problem. As such, we ran various simulations and changed the different variables of our problem to learn more about our robot's efficiency:

	(3, 3, 2)	(3, 3, 3)	(4, 3, 2)	(4, 3, 3)	(5, 4, 3)
Acc	0.571%	0.7936	0.619%	0.746	0.766%
Time	4.327	4.95	7.183	9.318	120

Table 1: Results *Column Names are formatted as (number of targets, patrol length, and number of attacker types), Acc is short for accuracy and Time is in seconds, both stats are averaged over 20 game playthroughs*

The results from table 1 are based off of 20 game averages. We see remarkable performance across different variations of variables. Any success rate that's higher than 1/m(random chance) is great, because it is one agent defending multiple objects. However, even with the 20 games average per statistic, we think there is still too much randomness to take the results completely at face value. This is because the patrol length is only length 3 or 4 and encounter probability and attacker type probability is also chosen at random. We think that's why we see so much variation between the different variations even though we hypothesized (3,3,2) to be the best performing(because there is less to defend and less attacker types to worry about). Another note is that we see that as soon as number of items reaches 5 and patrol length reaches 4, the search space explodes and the time it takes to run this algorithm becomes extremely long. But as a final note, we do not think this randomness completely detracts away from the value of this algorithm since it overall still has a much higher than random success rate.

Discussion

Reflections on algorithm research

When we first began our research, we were quite interested in the topic of Stackelberg games, as they truly seemed like the best game theoretic framework to approach security and safety problems. We spent a good amount of time reviewing literature to see how academics solved these games. We were able to find a couple research papers that explained solution concepts and efficient algorithms to solve Stackelberg games. However, most of these explanations gave more abstract mathematical descriptions and not enough details to go about coding up the algorithm ourselves. This was the case for algorithms such as HBGS, DOBSS, and multiple-LPs. Newer research frameworks, like PAWS, were promising, but they used machine learning classification models, and we lacked data to train it. We are curious as to what kind of results we could have gotten if we had been able to use behavioral data to better optimize, which was the case for PAWS. We ended up choosing choosing the DOBSS method because it was well suited for introducing randomness in the environment. For example, the multiple LPs approach requires changing the method by using a Harsanyi transformation, which transforms uncertainty over players strategy sets into uncertainty over their payoffs.

Reflections on Python implementation

It was thus hard for us to find any pseudocode or algorithmic implementations to go off of. The first code implementation we found was quite outdated and used packages that had since been modified. We had trouble modifying that code, but were thankfully able to find some new open-source code for DOBSS. This new code provided great help in better understanding the nuts and bolts of the algorithm, and we were able to use a new linear optimization package (PuLP). However, optimization packages are at times buggy and we had to figure out new ways to assign utility. The Python implementation we went off of randomly assigned utility, but for our scenario, the leader knows beforehand what the value of each item is and should take this into account when making his choice of where to go. We had to do a lot of brainstorming to think about how to best modify the DOBSS code implementation to insert the variables and settings we wanted for our scenario.

Reflections on simulation modeling

Learning about robotic simulation was an exciting topic of this project, but also proved to be one of the most complicated. These new kind of software and systems were ones that we were completely unfamiliar with, and navigating ”foreign territory” proved to be much harder than expected. We first worked on our own to learn how to use RoboMaker and the Cloud9 IDE, and followed tutorials on how to use AWS and ROS navigation packages. We ultimately got some help from our companion subteam to get set up in the collaborative AWS environment, which was quite helpful. When we ran into a lot of troubleshooting and error, we made a backup demo using Python, which illustrates the main idea of our work. It would have been nice to fully show that on Gazebo, but that was a bit complicated for our team and that was disappointing.

We think working with new systems has taught us a lot about teamwork and leveraging the resources we have around us. We think a key insight from this is that we should not underestimate the amount of time it takes to learn and get used to a new environment, but we have definitely learned a lot about robotics and simulation environments. We are hoping to learn more about those in the future, and hopefully get some simulations running, as it is fun work!

Reflections on game theoretic modeling

We have now spent almost a whole semester studying and learning about game theoretic modeling. From our advanced lecture given in March to this challenge, we have amassed a lot of knowledge about how to model MAS scenarios with games. This challenge was incredibly interesting. Since we are focusing on security applications, the kinds of scenarion that we will encounter are immensely complex. There can be life threatening situations and usually involve unpredictable situations

Game theory has been an insightful tool, and makes it easy to represent complex situations by using different players, actions, and variables. However, it does fall short in terms of scalability as runtime tends to grow quite fast when the number of players, actions, and variables grows. Yet, this is a key aspect of security scenarios. We cannot assume anything as these kind of situations hold a lot of surprises. In that light, game theoretic solvers and modeling may not be the way to approach this problem. We wonder if integrating machine learning algorithms might be able to help with optimizing the solution space, as it seems that some new frameworks have had quite successful results by integrating the both of them.

Further work

We are quite excited and enthusiastic about the work we did and the new things we learned with this project. However, we are still curious as to the kinds of improvements we could make upon our algorithm, or more generally, our scenario:

1. We made various simplifications in our game, such as the fact that the defender could only protect one object at a time. In a real-life scenario, inside of a house, a defender can protect objects in a specific radius, which could for instance be a whole room in the house. It would be insightful to see how we could modify this algorithm to take that into account.

2. Though this challenge was done at the scale of a household, we would be curious to see how this could apply to other kind of burglaries. Interesting examples would be illegal art traffic for instance, and we could model a new scenario using a museum environment, which is much larger in terms of area, set of items that could be stolen, etc... From the research we reviewed and the evaluation we did, it seems there are still difficulties to be able to fully scale Stackelberg games, and though we are not expert in the field, it would be quite intriguing to see what kind of modifications or simplifications to be made to be able to computer solutions efficiently.

Conclusion

In conclusion, this Grand Challenge was incredibly resourceful and provided a lot of insights about new ways to model multi agent systems using game theory. We think our focus, security applications of games, was a great choice as it taught us a lot about how to take into account item value/utility, randomness, and player types, especially in the context of potential infrastructure-threatening and life-threatening situations. The research that has been done in that area has been interesting to read and we enjoyed reviewing the different methods that have been used so far and for what applications they were done. Our implementation did a good job of optimizing strategy and finding solutions to our Bayesian Stackelberg game scenario for household environments with under 5 items and attacker types. With more items and types, the game does however become harder to solve, and runtime grows quite significantly. We still find good solutions but the algorithm takes much longer, which is not ideal. For real-life scenarios where Inspecti-bot would need to solve and optimize as fast as possible, this is not a great result. This has lead us to reflect on whether or not game theoretic modeling is the best way to represent an online security MAS scenario. We are however excited about the potential applications of game theoretic model, and will be monitoring this research area with great interest!

Thank you!

Thank you for an awesome semester. It was a pleasure learning from you and we hope you enjoyed this report!

References

An, B., Tambem M. (2017). Game Theory for Security: An Important Challenge for Multiagent Systems. European Workshop on Multi-Agent Systems. (LNAI,volume 7541).

Kiekintveld, Christopher and Jain, Manish. (2017). Basic Solution Concepts and Algorithms for Stackelberg Security Games.(pp. 508-537).

Paruchuri, Praveen et al. (2008). Efficient Algorithms to Solve Bayesian Stackelberg Games for Security Applications. 1559-1562.

Author contribution

Our team split did the initial research work as well as final report reflections together, and we split up to each focus on Python implementation and RoboMaker demonstration.

Emma: literature review, algorithm research, AWS RoboMaker/WorldForge demo, final report (motivation, related work, problement statement, discussion)

Vinh: literature review, algorithm research, Python implementation, code explanation, final report (abstract, method, evaluation)