POLITECNICO DI MILANO
Corso di Laurea Magistrale in Computer Science and Engeneering

Scuola di Ingegneria Industriale e dell'Informazione

Algorithm to find the equilibrium point in missile-ship simulation-based game

Relatore: Prof. Nicola Gatti
Correlatore: Francesco Trovò
Correlatore: Alberto Marchesi

Tesi di Laurea di:
Lorenzo Casalini, matricola 10690501

Anno Accademico 2019-2020

Summary

Military scenarios are, from the beginning of human history, a big area of interest where all the humans' knowledge were applied in order to defeat the enemy. Game Theory, in the last ages, offered a valid tool to analyze the behaviour of two or more players in a competitive scenario as it is war. In this work, we are study a real world military setting in which a warship is attacked by an infra-red missile. To tackle this problem, we will use the concepts of Game Theory and, in particular, since the game is known only through simulations, we will use the simulation-based games: a branch of Game Theory used when players' utilities are not known analytically. The goal of this work is that of creating and testing an algorithm which allows us to find the best strategies to follow for both the players of the game: the warship and the missile. First of all, we formulated the game, that we called boat-missile game, in order to be able to do some formal strategical reasoning on it. After that, we studied the complexity of the problem and we realized that it is too big and time consuming to be tackled as it is, so we introduced some simplifications over it. Secondly, we started to develop a new algorithm that allowed us to find the equilibrium point of the game using a limited amount of computational power. Once we have developed it, we tested it emipirically in a game simpler than the boat-missile one, showing that it can perform up to 30% better than some other state of the art approaches. Finally, we proposed an algorithm to solve the boat-missile problem, showing that, even with few simulations in comparison to the size of the problem, we can find strategies that allow the boat to escape the missile.

Sommario

Gli scenari militari sono, dall'inizio della storia umana, una grande area di interesse dove la conoscienza umana é stata applicata. La Teoria dei Giochi, recentemente, é stata uno strumento valido per analizzare il comportamento di due o piú giocatori inseriti in un contesto competitivo. In questo lavoro studiamo la situazione in cui una nave da guerra é attaccata da un missile a guida infra-rossa. Per risolvere questo problema abbiamo usato concetti derivati dalla Teoria dei Giochi e, in particolare, dato che il gioco é conosciuto solo attraverso l'uso di un simulatore, abbiamo usato la Teoria dei Giochi Simulativa, una branca della Teoria dei Giochi usata quando il gioco non é conosciuto analiticamente. L'obbiettivo di questo lavoro é stato sviluppare e testare un algortitmo che ci permetta di trovare le migliori strategie da seguire per entrambi i giocatori: la nave e il missile. Prima di tutto abbiamo formulato formalmente il gioco, che abbiamo chiamato gioco nave-missile, in modo da essere capaci di fare alcuni ragionamenti strategici su di esso. Dopo di ché abbiamo studiato la complessitá del problema e abbiamo capito che il gioco é troppo grande e troppo costoso per risolverlo cosí com'é, quindi abbiamo introdotto delle semplificazioni. In seguito, abbiamo cominciato a sviluppare un algoritmo che ci permettesse di trovare il punto di equilibrio del gioco usando una quantitá limitata di risorse computazionali. Una volta sviluppato, lo abbiamo testato in un ambiente piú semplice di quello nave-missile, mostrando che puó performare fino al 30% meglio di altri approcci che rappresentano lo stato dell'arte. Alla fine abbiamo usato l'algoritmo per risolvere il problema nave-missile mostrando che, anche con meno simulazioni rispetto alla taglia del problema, possiamo trovare strategie che permettano alla nave di evadere il missile.

Ringraziamenti

Devo ringraziare innanzitutto il Professore Gatti, Francesco e Alberto per la loro incredibile competenza, pazienza, gentilezza e disponibilitá. Hanno sempre saputo aiutarmi in maniera utilissima, nonostante il momento difficile e il fatto che credo di essere uno dei primi tesisti che non hanno mai conosciuto di persona (mi auguro di essere tra gli ultimi).
Ringrazio la mia famiglia che mi ha supportato in qualsiasi modo in questi due anni: Mamma, Papá, Angela, Nonna Carla e la mia bella Alessia.
Ringrazio i miei amici di una vita che hanno sempre saputo regalarmi un momento di allegria. Ci vediamo da Mr. Maccaroni.

Contents

Chapter 1

Introduction

1.1 General overview

Game Theory is the branch of mathematics that aims to model and study the strategic interactions among two or more players acting in a rational way. Since the last century, it tackled some important applications not only in the modeling of classical board games such as chess and Go, but also in other fields as economics [2, 28], security [1, 5], traffic [9, 29], biology and evolutionary dynamics [15, 17, 35], and political science.

Often a pure mathematical description of the model we want to analyze through Game Theory is not available, and this is usually due to the complexity of the phenomena which generate the game. In those cases it is still possible to perform some strategic reasoning on the game by building a software called simulator that is used to generate samples from the game. Such games are usually addressed as Simulation Based Games (*SBGs*).

The fact that the game is not known analytically but it is only known through simulations and the computational complexity of the simulation process itself poses all new challenges to traditional Game Theory and pushes toward the development of new algorithms that aim to find an equilibrium solution of a game using as few queries as possible.

The starting point of this work is a simulator of a specific physical system developed by Marina Militare Italiana that models a war scenario in which an infra-red guided missile is launched against a warship, which can deploy some countermeasures in order to avoid it. The problems we face when dealing with this simulator are its computational complexity and its multidimensional space of actions which makes the number of possible strategies for each player growing very fast.

The aim of this work is to develop and study a new algorithm that can find a solution to the boat-missile game in an efficient way in order to be able to find strategies that allow the warship to escape the missile.

1.2 Brief description of the work

After a study on the previous work on the SBGs we start to analyze the ship-missile game. First, we formulate it as a game, allowing us to do some strategic reasoning on it. The formulation we do is that boat-missile game is a *two-player zero-sum Simulation Based Game*, where the missile can choose its angle of attack and the boat can deploy its countermeasures, called *flares*, which aims to trick the infra-red system of the missile. Once the problem has been formulated, we analyze its complexity and we understand that its long simulation times and its big space of strategies makes it impossible to tackle it, even with a huge amount of computational resources and the most efficient algorithms, without introducing some simplifications. First, we give a structure to the curtain of flares that the boat deploys, and, then, we render the space of the parameters from continuous to discrete.

Once we make our problem tractable, we start to search for some solutions to it. The work [31] presents two algorithms designed for a setting very similar to ours, which will be the base for the development of this work. The first one is called M-GP-LUCB and it is proven that, given an *error*, finds the best strategy with probability $1 - error$ in a time depending on the error. The other one is GP-SE that, given a budget, finds the best strategy with a probability that depends on the budget. From the two, we selected GP-SE as the most promising because of the possibility of parallelizing it.

After that, we come out with a new algorithm that uses also GP-SE and it is designed specifically for the challenges posed by the ship-missile game. The basic idea is not to search the best strategy in the entire space of the strategies, which can be too big, but to search it within a smaller cloud of points in a space that moves toward the best solution during the execution of the algorithm. Our new algorithm is composed of two phases: cloud generation and cloud exploration. In the first phase, we create the cloud using a mathematical assumption over the payoffs of the simulator called *Gaussian Process* assumption, intending to generate clouds nearest to the optimal point as the number of simulation grows. In the cloud exploration phase, instead, we use GP-SE to sample the points of the cloud in an efficient way in order to update our *Gaussian Process*.

Once we develop our new algorithm we test it. First we test it in a simpler game that is called *hit-the-spitfire*, which has the advantage of having much faster simulations and a known solution. We will focus on the comparison between the new algorithm and GP-SE, which represents the state-of-the-art approach. After that, we run the algorithm in the ship-missile game to see if we are able to find strategies that allow the boat to escape the missile.

1.3 Structure of the thesis

The thesis is structured as follows:

- In Chapter 2, we give an overview of the previous studies on Simulation Based Games, focusing on the ones with military settings, and a description of the simulator.
- In Chapter 3, we give an analytical formulation of the game that is modeled by the ship-missile simulator and we introduce some simplifications in order to make the problem tractable.
- In Chapter 4, we give a description of our new algorithm, starting from its basic idea through the detailed implementation.
- In Chapter 5, we test our algorithm, first using a simpler simulator (*hit the spitfire*) and then using ship-missile simulator
- In Chapter 6, we draw the conclusions of our work and we will give ideas for some new related works.

Chapter 2

Simulation Based Games

In this section, we present an overview of the *simulation-based games* (SBGs) theory and the main literature results developed so far. Then, we define a specific SBG scenario that is the *two-player zero-sum game with infinite strategy spaces game*, which will be useful for this work. At the end we will focus on two specific algorithms, GP-SE and M-GP-LUCB, which, given some smoothness conditions on the utility function, are proven to be δ-PAC (*i.e.*, *probably approximately correct*): those will be the main block for the algorithm we have developed in this thesis.

2.1 General overview

Theoretic game modeling has received growing interest from the AI community over the last few decades as it allows us to model and solve even complex multi-agents settings. So far, Game Theory has found huge applications in different fields which represent some challenging real world problems, for instance: enforce security when a defender has to deal with multiple attackers and a limited set of resources [33, 12, 8, 7, 13, 6], designing truthful auctions for web advertising [19], solving large zero-sum recreational games as Bridge [39, 10], and studying biological evolutionary theories [3].

Definition 1 (Game)**.** A game is a tuple $G = (N,A,O,f,U)$ where:

- $N = \{1,2,3, \ldots, n\}$ is the set of players in the game.
- $A = \{A_1, A_2, \ldots, A_n\}$ is the set of possible strategies for each player. For example $A_i = \{a_1, a_2, \ldots, a_n\}$ is the set of strategies for the i-th player.
- O is the set of outcomes for the game.

- $f_i : A_1 \times A_2 \times ... \times A_N$ is a function that defines the outcome given a set of actions by the players.
- $U = \{U_1, U_2, \ldots, U_n\}$ is the set of utility functions of the players. $U_i : O \mapsto \mathbb{R}$

In practice, a game is represented by a set of players, each player can play a set of actions and those actions lead to an outcome, which gives an utility for every player.

Most of the studies in the field of Game Theory focus on applications where the analytical description of the utilities of the players is available, but this scenario often is not realistic in most cases in which it is impossible to build a model that describes completely the real world setting. This can be caused by several factors, for example in in those situations in which the game has an outcome given by complex physical models that cannot be expressed analytically, or in some other scenarios where there are some unknown parameters.

In order to address those kinds of situations, the common workaround is to build a black-box simulator, called oracle, which can be queried to obtain the utility function of the game. If in a game the players' payoff function is given by a simulator we call it *simulation-based game* (SBG) [42]:

Definition 2 (SBGs)**.** A SBG is a tuple $G = (N, R, \mathcal{O})$ where:

- $N = \{1,2,3,\ldots,n\}$ is the set of players in the game.
- $R = \{R_1, R_2, \ldots, R_n\}$is the joint strategy set with R_i the set of strategies of player i-th.
- $\mathcal{O}$ the oracle which, for every strategy profile $r \in R$, generates a sample vector of payoffs $U = \{U_1, U_2, \ldots, U_n\}$

The use of a simulator has the advantage of making it possible to analyze those complex and noisy settings that are impossible to model analytically. However, the use of a black-box simulator to address those complex scenarios poses some new issues to the solutions of those kinds of problems.

The first issue is the fact that simulators can be very complex, thus the query activity to get the values of the utility function is usually a costly operation. For that reason, the study of SBGs must use algorithms that focus on the possibility to have a result (reach an equilibrium point) using the smallest number of simulations possible.

Another important aspect that must be faced when dealing with SBGs is the noise in the observations: the algorithm used must be able to handle noisy responses by the oracle in order to find the (possibly approximate) equilibrium points.

The last observation that must be done is the fact that it is impossible to query the simulator in a real-time scenario, for instance in a military scenario where a missile is about to hit a terrestrial unit. Thus, the application of SBGs algorithms to the real world must happen in two separate phases: in the first phase, called exploration, we can use the simulator to learn as much as we can about the utility function, and, in the second phase called exploitation, we must use our knowledge learned during the previous phase in order to make instantaneous decision that allows us to maximize the utility of a player.

2.2 Game Theory in defensive and military settings

In this work we are going to apply Game Theory, and in particular Simulation-Based Games to specific military and defensive applications, so it could be interesting to make some examples of how Game Theory has been used so far in those kind of settings.

- An example of that could be found in [22], which is a work funded by the U.S Air Force where the authors develop some new methods for analyzing war strategies in *Time Critical Target* (TCT) operations using Game Theory with the goal of understanding how each side of a military conflict influences the decision making of the others and how one side can compel the other to follow a preferred course of action. In this work, some simple war scenarios are described (an aircraft against surface-air missile system) and, under some conditions regarding for example the power of the weapons involved, are able to predict the best behavior for both players (for example they predict if it is better for the terrestrial unit to try to hit the aircraft or stay hidden). Despite the method described can be used for having tactical and strategical hints of the behavior of the enemy in some situations, it does not rely on an accurate analysis of some of the important variable involved in the scenarios considered, as for example the probability for the Surface-to-Air missile to hit the aircraft in various situations.

- Another example of how GT can be applied to warfare scenario was given by [24] where the authors modeled and analyzed successfully, using agent based simulations and GT principles, the outcome of a

battle. The war scenario considered was the Battle of Biscay, which is a campaign fought during WWII between Germans U-boats and allied Air Forces. During WWII the U-boats transited the Bay of Biscay enroute to the Atlantic where they targeted the Allied convoys. In order to counter the U-boat threat, the Allies concentrated their aerial research effort within the bay.
The authors used the historical analysis made on this event to build a simulator where each U-boat and each aircraft are modeled as separate agents. The goal of the U-boats was to pass the bay without being sunk by the aircraft. The U-boat can proceed either surfaced or immersed, in the first case its speed is much higher than the latter one, and it is constrained by some bounds: it has a limited amount of fuel, food and battery. The aircrafts, on the other hand, patrol the bay searching for U-boats and if they found one, they attack it. They are also subjected to some constraints related to fuel, flying time and maintenance operations. U-boats and airforces are controlled by a super-entity which decides respectively: how the U-boats must proceed (surfaced or immerged) and the percentage of patrolling operations of the aircrafts must be made during the day. The game was modeled as a two-player zero-sum game and an algorithm that allows both players to reach their goals was designed. At the end of the simulation the authors show that the results of the percentage of effort made by both players are very similar to the ones made in reality.

- In [37], the authors describe a game-theoretic approach to evaluate air-combat simulation models. Their method consists of building a game abstraction of the AC from a set of data obtained from the simulator itself using statistical techniques, as a multivariate regression model. Then the estimated game is evaluated following some metrics as symmetry, best response of players, and equilibrium solutions. If all these measures have a sound explanation, the simulator is validated.
  This work also shows how the estimated games are good to be used for strategy analysis, as they represent a new kind of simulation metamodel.

Other examples of Game Theory applied at military scenarios can be: mission decision making in unmanned combat aerial vehicles (UCAVs) [46], anti submarine warfare techniques [32] and even information warfare [21]. Despite the power of Game Theory techniques, none of the previously described works could provide a solution to our problem. This happens because, despite their are works were SBGs are used, none of them actually aims to tackle the classical SBGs' problems (i.e. high simulation time, noise) but they aims to show that is possible to build simulators where the behaviour of players in a war scenario is reproduced in a reliable way. The work described by [37], instead, could be valid to tackle problems similar to the one we have but it uses methods as regression models that in our case would be impossible to deploy due to two reason. The first reason is that our utility function is probably too complex to be learned in this way: it depends on too much variables to be approximated in a good way. The second reason is that, since our queries of the utility function are very expensive, it would take a very long time to build a dataset large enough to allow us to deploy some supervised learning techniques.

2.3 Previous studies on SBGs

Learning approximate equilibrium points in SBGs is a task that has received growing attention from the AI community during the last years. The first work on that area was made by Vorobeychik et al. [43] which used supervised learning techniques to learn the payoff for each player and then compute the *Nash Equilibrium* (NE) of the game. This method was then evaluated in a two-player auction game with a known solution and with a five-player auction games with an unknown solution. A successive work by Gatti and Restelli [16] extended the previous paper to extensive-form games using three different optimization methods over the game tree: Simulated Annealing, Lipshitz Optimization and Crossentropy Method. Recently, another regression-based approach was presented by Wiedenbeck et al. [44]. In this work, the utility function is approximated by a Gaussian Process and was shown that using this technique leads to better results than previous works.

The last work we want to write about was presented by Garvivier et al. [14] where the authors developed two algorithms, one for fixed budget setting and one for fixed confidence setting, and they prove them to be δ-PAC under the assumptions of having a utility function that is a Bernoulli distribution and a finite set of strategies for each player.

2.3.1 GP-SE and M-GP-LUCB

Recently, Marchesi, Trovò and Gatti [31, 30] proposed a new approach to handle the case of a two-players zero-sum game with an infinite strategy space. They were also able to prove δ-PAC (*i.e., probably approximately correct*) theoretical guarantees for their algorithm under the Gaussian

Process (GP) assumption on the utility function extending the work in [14].

They designed two algorithms, M-GP-LUCB and GP-SE. The first one was developed for fixed confidence settings, which means that, given a confidence level δ, it finds a solution in a time that is guaranteed to be less than some δ-dependent number of rounds T_δ.

The second algorithm was developed for fixed budget settings, so the goal was, given a budget, find an ϵ-maxmin strategy with confidence $1 - \delta_T$ as large as possible.

Zero-sum games

The authors of the work presented in [31] take in consideration *two-player zero-sum games with infinite strategy spaces* which are a tuple $\Gamma = (\mathcal{X}, \mathcal{Y}, u)$, where $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y} \subset \mathbb{R}^d$ are the set of strategies for the first and second player, and $u : \mathcal{X} \times \mathcal{Y} \mapsto \mathbb{R}$ is the utility function of the first player.

A *two-player zero-sum games with finite strategy spaces* is defined similarly, being $\mathcal{X}$ and $\mathcal{Y}$ finite sets: $\mathcal{X} := \{x^1, \dots, x^n\}$ and $\mathcal{Y} := \{y^1, \dots, y^m\}$, with $n > 1$ and $m > 1$.

The goal of the paper is the computation of maxmin strategies, which are defined as follows: letting $\boldsymbol{\pi} := (x, y)$ with $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, we denote as $y^*(x) \in \arg\min_{y \in \mathcal{Y}} u(x, y)$ the best response of the second player of the action $x \in \mathcal{X}$ of the first player, then $x^* \in \mathcal{X}$ is a maxmin strategy for the first player if $x^* \in \arg\max_{x \in \mathcal{X}} u(x, y^*(x))$, with $\boldsymbol{\pi}^* := (x^*, y^*(x^*))$ being the maxmin strategy profile of the game.

This means that a maxmin strategy is when the first player plays the best possible move, knowing that the second player will play its best response at that move.[1]

Gaussian Processes

The only assumption that has been made on the utility function is that it has to be a Gaussian Process (GP), where a GP [38, 34] is defined as a collection of random variables, one for each action profile $\boldsymbol{\pi} \in \Pi$, every finite subset of which is multivariate Gaussian distributed.

A $\mathrm{GP}(\mu(\boldsymbol{\pi}), k(\boldsymbol{\pi}, \boldsymbol{\pi}'))$ is fully specified by its *mean* function $\mu : \Pi \mapsto \mathbb{R}$, with $\mu(\boldsymbol{\pi}) := \mathbb{E}[u(\boldsymbol{\pi})]$, and its *covariance* (or *kernel*) function $k : \Pi \times \Pi \mapsto \mathbb{R}$, with $k(\boldsymbol{\pi}, \boldsymbol{\pi}') := \mathbb{E}[(u(\boldsymbol{\pi}) - \mu(\boldsymbol{\pi}))(u(\boldsymbol{\pi}') - \mu(\boldsymbol{\pi}'))]$.

One of the biggest advantage of working with GPs is that they admit some practical formulas to manage the posterior update after an observation:

[1]The case of general-sum games in much more involved. We point an interested reader to [11, 18].

calling $\tilde{\mathbf{u}}_t := [\tilde{u}_1, \ldots, \tilde{u}_t]^\top$ the set of utility observations given by the oracle until time t after querying it with the strategies profiles $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_t$, we can update $\mu(\boldsymbol{\pi})$, $k(\boldsymbol{\pi}, \boldsymbol{\pi}')$ and $\sigma_t^2(\boldsymbol{\pi})$ as:

$$\mu_t(\boldsymbol{\pi}) := \mathbf{k}_t(\boldsymbol{\pi})^\top \left(K_t + \lambda I\right)^{-1} \tilde{\mathbf{u}}_t, \tag{2.1}$$

$$k_t(\boldsymbol{\pi}, \boldsymbol{\pi}') := k(\boldsymbol{\pi}, \boldsymbol{\pi}') - \mathbf{k}_t(\boldsymbol{\pi})^\top (K_t + \lambda I)^{-1} \mathbf{k}_t(\boldsymbol{\pi}'), \tag{2.2}$$

$$\sigma_t^2(\boldsymbol{\pi}) := k_t(\boldsymbol{\pi}, \boldsymbol{\pi}), \tag{2.3}$$

The main issue with those formulas is that they perform some inversions of matrices, which can be a very costly operation, especially when the matrices become large. For that reason the update formulas of the GP can be also written recursively, thus avoiding the expensive matrix inversions:

$$\mu_t(\boldsymbol{\pi}) \leftarrow \mu_{t-1}(\boldsymbol{\pi}) + \frac{k_{t-1}(\boldsymbol{\pi}, \boldsymbol{\pi}_t)}{\lambda + \sigma_{t-1}^2(\boldsymbol{\pi}_t)} (\tilde{u}_t - \mu_{t-1}(\boldsymbol{\pi}_t)), \tag{2.4}$$

$$k_t(\boldsymbol{\pi}, \boldsymbol{\pi}') \leftarrow k_t(\boldsymbol{\pi}, \boldsymbol{\pi}') - \frac{k_{t-1}(\boldsymbol{\pi}, \boldsymbol{\pi}_t) k_{t-1}(\boldsymbol{\pi}_t, \boldsymbol{\pi}')}{\lambda + \sigma_{t-1}^2(\boldsymbol{\pi}_t)}, \tag{2.5}$$

$$\sigma_t^2(\boldsymbol{\pi}) \leftarrow \sigma_{t-1}^2(\boldsymbol{\pi}) - \frac{k_{t-1}^2(\boldsymbol{\pi}, \boldsymbol{\pi}_t)}{\lambda + \sigma_{t-1}^2(\boldsymbol{\pi}_t)}. \tag{2.6}$$

The GP assumption on the utility function has some other good properties which are worth to be mentioned here.

The first one is that GPs do not rely on rigid parametric assumptions as linearity, they only require a certain degree of smoothness in the utility function. That smoothness is encoded in the kernel matrix, which is in some sense the measure of how much are correlated the strategy profiles corresponding to its entries. The smoothness property makes GPs easy to be used to model a variety of different problems in several areas as: geostatistic [36], sensor placement for temperature measures [25] and visualization of high dimensional data [26].

Another important aspect of GPs that will be very useful for this work is the fact that, given its smoothness property and the kernel function which encodes it, it is possible to extract information from points in the space that are not been explored yet. This property derives from the fact that, as said before, the kernel function can be seen as the measure of the correlation between two points in the space, thus, having a sample from a given point, can give us information about all the points that are highly correlated with it.

This aspect will be very useful when we will manage a game with a large amount of strategies and a very expensive simulation phase, where either the entire scan of the strategy space will be impossible or we do not want

Algorithm 1 M-GP-LUCB(ϵ, δ)

```
1: Initialize $t \leftarrow 0$, $\mu_0(\boldsymbol{\pi}) \leftarrow \mathbf{0}$, $k_0(\boldsymbol{\pi}, \boldsymbol{\pi}') \leftarrow k(\boldsymbol{\pi}, \boldsymbol{\pi}')$
2: do
3:     Select $\boldsymbol{\pi}_{t+1}$ and $\boldsymbol{\pi}_{t+2}$ using Eqs. (2.9)–(2.10)
4:     $\tilde{u}_{t+1} \leftarrow \textsc{Sim}(\boldsymbol{\pi}_{t+1})$, $\tilde{u}_{t+2} \leftarrow \textsc{Sim}(\boldsymbol{\pi}_{t+2})$
5:     Compute $\mu_{t+2}(\boldsymbol{\pi})$ and $k_{t+2}(\boldsymbol{\pi}, \boldsymbol{\pi}')$ using
           observations $\tilde{u}_{t+1}$, $\tilde{u}_{t+2}$ and Eqs. (2.1)–(2.3)
6:     $t \leftarrow t + 2$
7: while $L_t(\boldsymbol{\pi}_{t+1}) \leq U_t(\boldsymbol{\pi}_{t+2}) - \epsilon$
8: return $\overline{\boldsymbol{\pi}} = (\bar{x}_t, \gamma_t(\bar{x}_t))$
```

to waste time querying strategies that we know being not worth it. In that case the GPs assumption will help us to extract samples from regions that we think are good, allowing us to save time from sampling strategies that are known to be bad.

M-GP-LUCB

The first algorithm described by [31] is called M-GP-LUCB. It is presented as a fixed confidence algorithm: this means that, given a confidence threshold δ, it finds a δ-PAC solution in a number of rounds $< T_\delta$.
M-GP-LUCB saves for each strategy a tuple $[L_t(\boldsymbol{\pi}), U_t(\boldsymbol{\pi})]$, which represents the lower and upper bounds of the utility value of strategy profile $\boldsymbol{\pi}$ at round t and are computed as:

$$L_t(\boldsymbol{\pi}) := \mu_t(\boldsymbol{\pi}) - \sqrt{b_t}\sigma_t(\boldsymbol{\pi}), \tag{2.7}$$

$$U_t(\boldsymbol{\pi}) := \mu_t(\boldsymbol{\pi}) + \sqrt{b_t}\sigma_t(\boldsymbol{\pi}). \tag{2.8}$$

where $\mu_t(\boldsymbol{\pi})$ and $\sigma_t(\boldsymbol{\pi})$ are the mean and the variance of the posterior distribution when using strategy $\boldsymbol{\pi}$ and b_t is a exploration term which depends by the context. The algorithm each round selects two strategy profiles which are defined as follows: letting

$$\gamma_t(x) := \operatorname*{argmin}_{y \in \mathcal{Y}} L_t(x, y)$$

the best response with the lowest lower bound of the second player for every strategy $x \in \mathcal{X}$ of the first player, and letting

$$\bar{x}_t := \operatorname*{argmax}_{x \in \mathcal{X}} \min_{y \in \mathcal{Y}} \mu_t(x, y)$$

the maxmin action for the first player computed using mean values, the strategy profiles $\boldsymbol{\pi}_t$ and $\boldsymbol{\pi}_{t-1}$ that are played are:

$$\boldsymbol{\pi}_{t+1} \coloneqq (\bar{x}_t, \gamma_t(\bar{x}_t)) \tag{2.9}$$

$$\boldsymbol{\pi}_{t+2} \coloneqq \underset{\boldsymbol{\pi} \in \{(x,\gamma_t(x))\}_{x \neq \bar{x}_t}}{\text{argmax}} \quad U_t(\boldsymbol{\pi}). \tag{2.10}$$

The Equations (2.9) and (2.10) intuitively represent the best and the second best candidate strategies, and the algorithm, after taking a sample of their utilities from the oracle, updates its representation of the GP using Equations (2.1) and (2.3).

The algorithm stops when $L_t(\boldsymbol{\pi}_{t+1}) \leq U_t(\boldsymbol{\pi}_{t+2}) - \epsilon$, which means that the lower bound of the best strategy is larger than the upper bound of the second best strategy minus ϵ.

In the paper [31], M-GP-LUCB is proven to be δ-PAC and in particular, it is proven that letting

$$b_t := 2 \log \left(\frac{n\, m\, \pi^2\, t^2}{6\delta} \right)$$

the algorithm returns a maxmin profile with confidence at least 1 - δ and

$$\begin{aligned} T_\delta \leq 64\, H^*(u)\, \lambda \Bigg(& \log \left(64\, H^*(u)\, \lambda\, \pi \sqrt{\frac{n\, m}{6\delta}} \right) \\ & + 2 \log \left(\log \left(64\, H^*(u) \lambda \pi \sqrt{\frac{n\, m}{6\delta}} \right) \right) \Bigg), \end{aligned} \tag{2.11}$$

where the authors required that $64\, \lambda\, \pi\, \sqrt{\frac{n\, m}{6\delta}} > 4.85$. Equation (2.11) shows how T_δ scales logaritmically with the square root of n and m, which are the number of possible actions for the first and second player, allowing M-GP-LUCB to be used even in settings with a very large number of possible strategies. The dominant terms of the equation are λ, which is the variance noise, and $H^*(u)$, which intuitively is a factor that characterizes the hardness of the problem instances by determining the amount of time required to indentify the maxmin solution.

Despite being proven to reach also good experimental results in [31], M-GP-LUCB has a main drawback when used in scenarios where the simulations are very expensive: its structure does not allow to extract multiple samples from utility function since its mechanism is sequential. This behavior is due to the fact that at each round only two strategy profiles $\boldsymbol{\pi}_t$ and $\boldsymbol{\pi}_{t+1}$ must be queried, and then it is necessary to wait until the end of the simulation and update phase in order to choose the other two strategies profiles to play during the next round. The only allowed parallelization in

Algorithm 2 GP-SE(T)

```
1: Initialize $\Pi_1 \leftarrow \Pi$, $\mu_0(\boldsymbol{\pi}) \leftarrow \mathbf{0}$
2: for $p = 1, 2, \dots, P-1$ do
3:     For each $\boldsymbol{\pi} \in \Pi_p$, query $\textsc{Sim}(\boldsymbol{\pi})$ for
           $T_p - T_{p-1}$ rounds
4:     Compute $\mu_p(\boldsymbol{\pi})$ using observations
5:     Select $\boldsymbol{\pi}_p$ according to Eqs. (2.12)–(2.13)
6:     $\Pi_{p+1} \leftarrow \Pi_p \setminus \{\boldsymbol{\pi}_p\}$
7: return the unique element $\overline{\boldsymbol{\pi}}$ of $\Pi_P$
```

M-GP-LUCB is to simulate in parallel $\boldsymbol{\pi}_t$ and $\boldsymbol{\pi}_{t+1}$ which can lead to a speedup of 2x in an ideal situation in which the update and selection time is reduced to zero.

This drawback is not crucial when dealing with very fast simulations in settings in which the update phase (which is harder to parallelize than the simulation phase) is more expensive than the simulation phase, or even when the hardware does not allow a massive parallelization. Conversely, it becomes very important when the simulation time is prominent and it is important to extract as much utility values as possible in a given amount of time: in this case the ideal speedup of 2x of M-GP-LUCB could not be enough to make it execute in a feasible time.

GP-SE

The second algorithm presented in the paper [31] is GP-SE. It is designed for fixed budget scenarios, which means that, given a maximum number of simulations available, called T, it finds a δ-PAC strategy profile with confidence at least 1 - δ_T.
The algorithm works in a round robin fashion, progressively eliminating the worst strategy and ending with the δ-PAC maxmin strategy.

It starts by dividing the budget T in $P-1$ rounds, letting $P := |\Pi| = n\,m$, then for each $p = 1, 2, \dots, P-1$ it queries the oracle $T_p - T_{p-1}$ times for each $\boldsymbol{\pi} \in \Pi_p$ and update its observations following (2.1) and (2.3). At the end of each round it eliminates the strategy profile that has:

$$(x_p, \cdot) := \operatorname*{argmin}_{\boldsymbol{\pi} \in \Pi_p} \mu_p(\boldsymbol{\pi}), \tag{2.12}$$

$$y_p := \operatorname*{argmax}_{y \in \mathcal{Y}:(x_p, y) \in \Pi_p} \mu_p(x_p, y). \tag{2.13}$$

Intuitively, the strategy defined by Equations (2.12) and (2.13) is the strategy which has less probability to be maxmin. In the end, GP-SE returns the only strategy profile left in Π_p.

This method made of subsequent eliminations allows having more precise estimates of the utility function in points that are probably maxmin since they are the ones who are eliminated nearest to the end, thus the oracle receives more queries for them. On the other hand the profiles which are less likely to be maxmin are eliminated earlier, in this way few simulations are used to estimate their utilities.

The authors propose a choice of T_p that, following [4], leads to the optimal convergence rate:

$$T_p := \left\lceil \frac{T - P}{\overline{\log}(P)(P + 1 - p)} \right\rceil . \tag{2.14}$$

Then, the authors provide a theorem where they demonstrated that, given a budget T, GP-SE returns the maxmin strategy profile with confidence at least $1 - \delta_T$, where δ_T is:

$$\delta_T = 2P(n + m - 2)e^{-\frac{T-P}{8\lambda \overline{\log}(P) H_2}}, \tag{2.15}$$

and $H_2 := \max_{i \in \{1,\dots,P\}} i\, \Delta_{(i)}^{-2}$.
One of the main practical differences between GP-SE and M-GP-LUCB is that the first one has the possibility to have a massive parallelization on it. At the beginning of each round $p = 1, 2, \dots, P-1$ we know which strategies we are going to query and how many times, making it possible to run the simulations in parallel. Only at the end of each round the algorithm has a coordination phase when it has to synchronize all the observations to update the GP and find the strategy to eliminate.

In an ideal situation, where the update and the choice of the worst strategies happen in negligible time, and if we had enough cores in our machine, we could run GP-SE in the time needed to do $P = |\Pi| = n\,m$ simulations.

2.4 Infra-Red simulator

A good example of the power and the capabilities of a simulator is given by *Simulatore IR*, developed by *Analisi e Valore s.r.l.* for the Italian Navy [41] and implemented in *MATLAB*. The simulator aims to model a warfare scenario in which a missile is launched against a military boat.

Modern anti-ship missiles, equipped with IR sensors, use the target IR signature, represented by IR images, the amount of energy emitted and the spectral signature of the target to self guide on it.

On the other hand, the boat can use two kinds of countermeasures to avoid being hit by the missile: active and passive. Active countermeasures consist in destroying the threat before being hit and could be represented by launching another missile against the first one. Passive countermeasures, instead, consist of adopting a series of strategies and maneuvers that allow the boat to avoid the threat without destroying it. The simulator aims only to model passive countermeasures which consist in launching a group of *flares*, false targets composed of hot burning metal with burning temperatures similar to engine exhaust [45] that are able to make the infrared-guided missile seek out their heat signature, and execute some evasive maneuvers changing speed and direction (Figure 2.1).
The effectiveness of these countermeasures may depend on various factors as the timing of their deployment, the type of IR seeker system implemented in the missile, and the type of environmental conditions (natural light, latitude and wind). At the end of the simulations, the simulator returns 1 if the ship is hit or 0 otherwise, together with some other useful measures as for example the distance between boat and missile at every time instant of the simulation, their positions and details on the seeker system of the missile. In this section, a brief description of each element of the simulator is given and, then, it will be presented how to run a simulations is presented.

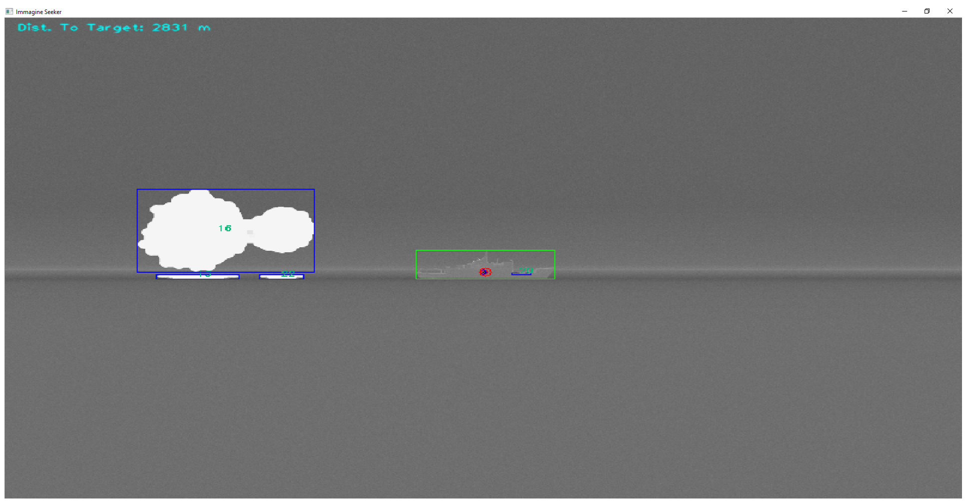


Figure 2.1: A view of the simulation from the IR system of a missile: on the center there is the boat and on the left two flares have been launched. The colored boxes are made by the seeker system.

2.4.1 Ship model

In the simulator eight kinds of different warships are modeled. Each boat is modeled by three different components: it has a dynamic behavior, a IR signature and a physical subsystem.

- The dynamic behavior models the interactions between the warship and the environment, allowing to compute the position, direction and speed of the boat at every moment. In this subsystem complex physical phenomena are taken into consideration as for example hydrodynamic forces and moments that act on the boat, the thrust generated by the propellers, and the forces on the rudder.

- The IR signature of a warship is its radiating intensity in contrast with the background and it is filtered by environmental phenomena as natural light, reflection of the sea and distance from the viewer (the missile). In order to represent the IR signature of a ship provided to the missile seeker algorithm, a database containing the IR images of the boats seen from different angles is created. Then, when the simulation starts, the angle between the axis of the boat and the missile is computed and the image corresponding to that angle is queried. Then, some filtering is made based on the environmental conditions.

- The physical model of the boat is made to determine if the boat is hit or missed. For each type of warship a three-dimensional approximate model is built, usually composed of two boxes (see Figure 2.2) one for the hull and one, superimposed for the turrets. This three-dimensional shape moves and rotates as described by the dynamic model and if the missile enters it the ship is assumed to be hit.

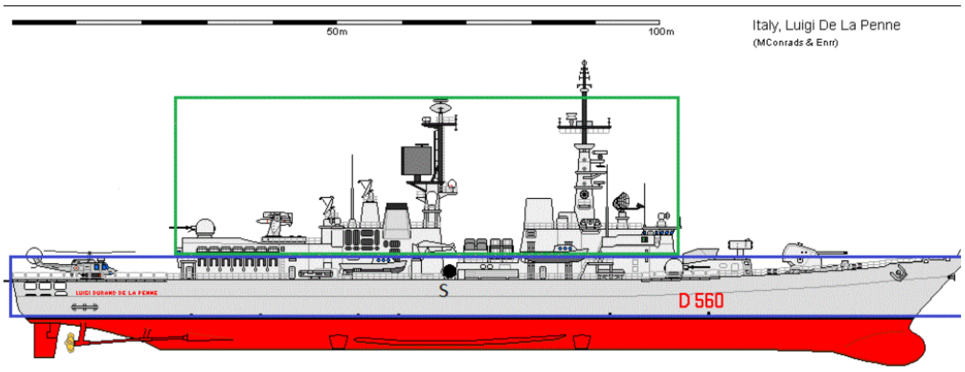


Figure 2.2: Physical model of the Durand De La Penne warship class: in blue and in green the two boxes used to approximate the real shape of the boat.

2.4.2 Missile model

The simulator comprises 10 IR guided tactical SSM (surface-to-surface) or ASM (air-to-surface) missiles. Each missile model is composed by seven subsystems (see Figure 2.3):

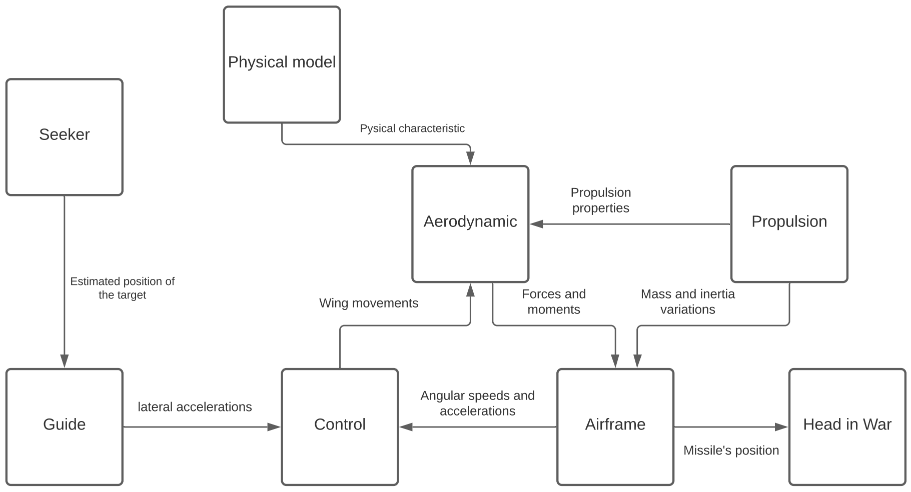


Figure 2.3: Interactions between simulator's subsystems.

- The aerodynamic subsystem determines the forces and moments which the missile has to face during its fly. It is a complex physical model and it is composed of five elements: computation of aerodynamic coefficients, propulsive forces, aerodynamic forces, the force of gravity and aerodynamic moments. All the forces and moments computed by this subsystem will be used by the airframe component to determine the speed and the position of the missile.

- The control subsystem is in charge of determining the coefficients of aerodynamic deflection of the missile control wings. It is composed of three autopilots: one for each lateral direction (*pitch autopilot* and *yaw autopilot*) and one (*roll autopilot*) for the roll of the missile. The control subsystem receives the desired direction to obtain from the guide subsystem, then calculates the deflection coefficients necessary for the wings and moves them through an actuator.

- The airframe model is made of a six degree of freedom dynamic body. To this model are applied all the forces and moments generated by control, aerodynamic and propulsive subsystem to compute the speed and the position of the missile.

- The propulsion subsystem is the one who creates thrust by burning fuel. It computes the quantity of fuel to burn to reach the desired speed and, since the airframe is a dynamical model, it also computes the movements of the center of mass and the moment of inertia of the missile, which changes because of the smaller fuel weight during the simulations.

- The most complex subsystem is the Seeker one (Figure 2.1). It is in charge to detect the target, tracking it and process the IR signal. It is composed by three different elements:

  1. Imaging System: it receives the IR signal of the missile field of view and transforms it into a digital signal.
  2. The gimbal is a component that allows the rotation of the imaging system in order to keep the target in its field of view (FOV). It has two rotation angles θ and ϕ and it has two different possible goals during a simulation: if the target is in the FOV of the tracker it has to keep it in the center of the FOV, if the target is not in the FOV it has to rotate the FOV in order to get the target.
  3. The processor unit is activated only when the target is detected, it has to apply the tracking algorithm in order to allow the missile to follow the correct path in order to hit the target.

  Therefore, the goal of the seeker subsystem is to drive the missile on the target through the use of tracking algorithms. In the simulator there are implemented two kinds of tracking algorithms: Gated Video Tracker algorithms and Correlation Tracker. Gated Video Tracker are Binary Centroid, Intensity Centroid and Centroid with Intensity Threshold:

  1. The binary centroid of a segmented image (in other words at each pixel is associated with a value which is either 1 or 0) can be seen as the center of mass a surface. Thus, Binary Centroid algorithm first filters the image, putting to value 1 the pixels with an intensity beyond a certain threshold, and then puts the target in the center of mass of the segmented image.
  2. Intensity Centroid method is similar to Binary Centroid, with the difference that does not apply the segmentation phase, it only uses the intensity of each pixel in the image to calculate the center of mass.
  3. The Centroid with Intensity Threshold instead, puts to zero all the pixels with an intensity lower than a threshold and then uses the intensities of all the other pixels to compute the center of mass of the image.

Correlation Tracker algorithms are:

1. A Correlation Tracking Algorithm with Fixed Reference is an iterative procedure that aims to track the trajectory of the target. It is based on the calculation of the likelihood between a portion of the image $I(x, y)$ and a reference model (template) $T(x, y)$. Typically the dimension of the template $T(x, y)$ is smaller than the one of the image $I(x, y)$. Localizing a model within an image consists of finding the translations of the model such that the likelihood between them is maximum. The drawback of this algorithm is that, in order to implement the correlation algorithm with fixed reference, it is necessary to have a ship template for every possible angle of observation. A template is like a snapshot taken from the IR image of the ship.
2. The Correlation Tracking Algorithm with Adaptive Reference does not require previous knowledge of the template like the one described before. In the case the target signature is not known a priori, it is possible to use alternative correlation algorithms. The adaptation of the reference template is made by estimating the predicted intensity distribution of the target. This is achieved using a temporal filter that processes the previously acquired images and extracts the template that can be used to compute the correlation in the following instants.

- The guide subsystem is the one in charge of making the missile follow the desired trajectory. It has two ways of working: if the target is already acquired, the missile follows the target trying to minimize the gimbal angle putting the target in the center of the FOV of the seeker system. If the target is lost the missile has to follow a straight trajectory pointing at the place where the last target was observed. It receives from the seeker system the estimated position of the target and it is in communication with the control system in order to give it the right instructions to direct the trajectory of the missile.

- The Head in War subsystem models the fuze of the missile. It can be an impact fuze or a proximity fuze and it has the job of making the missile explode at the right time.

2.4.3 Environment model

In the environment subsystem are encoded all the environmental factors that affect the simulation.

- The atmospheric mitigation is the main environmental source of noise: while traveling in the air, the IR waves are partially absorbed and deviated by the gasses.
- The wave motion of the sea is able to partially hid the target to the seeker system for some periods of time, so it is modeled in the simulator.
- Reflection of IR waves on the sea surface is another source of noise encoded in the simulator and it is generated both by the target and by the countermeasures.
- The gravitational acceleration is a function that varies with the position on the globe, so it is possible to set some war scenarios in different locations where the gravity force is different.
- Wind, although not simulated for the warship behavior, can have an impact on the countermeasures deployment.

2.4.4 Countermeasures Model

The deployment of countermeasures is the way a warship can neutralize an IR threat by disguising its seeker system. Every countermeasure (flare) is composed of N sub-munitions of mass M and it is modeled in two ways.

The first aspect taken into consideration is the dynamic behavior of the countermeasure which is characterized by the launch angle, position and speed, the wind, the force of gravity and the air conditions. After the launch, the flare has an explosion phase that changes its dynamical attribute.

The other way to model the flares behavior is the radiant intensity way: the flare starts with low intensity and, then, after the explosion it grows for a few seconds and then decreases asymptotically.

2.4.5 Simulations

The process of running custom simulations is composed of four phases. In the first phase, the user can choose the different models to adopt, respectively for warship, missile and countermeasure. The simulator offers 10 different models of ship, each one with its own size, weight and maneuver capability.

The missile instead differs for their tracking algorithms, their speed, size, and also for their general behavior: *Sea Skimmer* missiles travel around 18 meters above sea level, then, in the proximity of the target rise until 60

meters and at the end they do an abrupt descent while *Rapid Descendant* missiles travel more regularly.
The flares instead differ from each other for their radiation capabilities. After choosing missile, flares and ship models the user has to define the environmental variables which characterize the simulation. Different condition of climate, refraction of the air, time of the day and geographical position in the globe must be set in order to build an accurate environmental model.
After that the user has to choose the initial conditions of the scenario. It has to set the initial boat and ship position and speed, together with the number of flares available on the boat. Then, the last thing to be set is the countermeasures deployment that the boat could do in order to neutralize the threat. The countermeasures are composed of two parts: the flares launch and the evasive maneuver.
For the flares launch the user can specify, for each countermeasure, the moment of launch and the 3-D spatial coordinates in which the flares will be launched while the boat can change its direction and speed at a given time. While several flares can be launched in different moments, it is important to notice that the boat can change its speed and direction only once during a simulation.
At the end of the simulation the program will return, beside of the boolean variable *boom*, which is 1 if the boat is hit, some other interesting data as for example the log of the position of boat and missile (figure 2.4) or the estimated and real LOS of the missile's seeker.
It is very important to notice, for the sake of this work that, given the complexity of this entire system, especially the tracking algorithms of the missiles, the simulation takes considerable time to execute. Approximately the first 5 missiles (which are the ones with the less sophisticated tracking system) take about 1 minute to complete a simulation, the missiles from 5 to 10 about 2 minutes and the missile 11, which is the most complex one, up to 6 minutes.

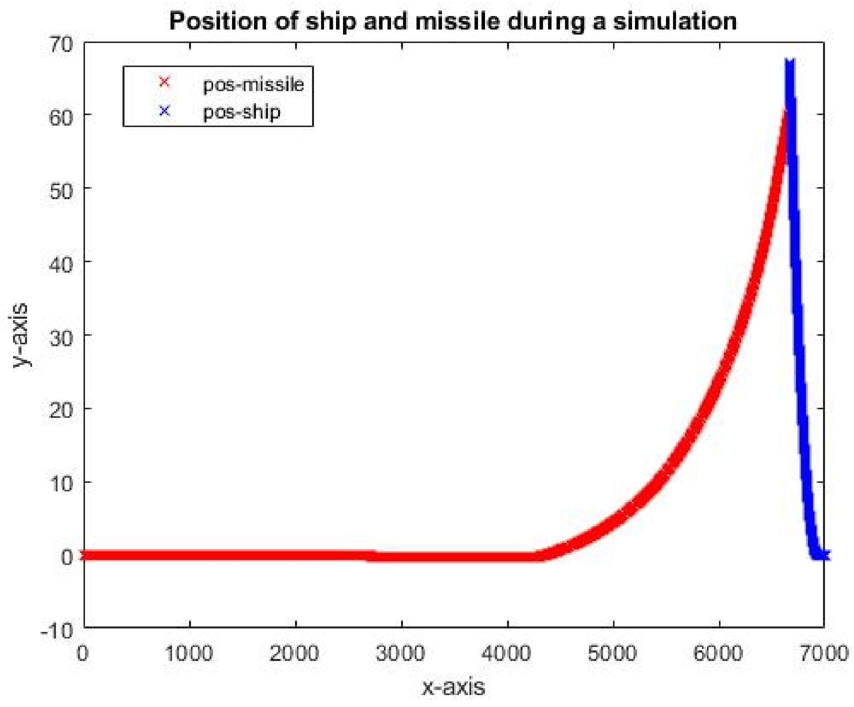


Figure 2.4: Plot of the position of boat and missile during a simulation

2.5 Previous work on Infra-Red simulator

This is not the first work made to study a solution to the boat-missile problem: the first one was made by Longobardi [27].

The first effort of that work, that has been useful also for this thesis is a preliminary study of the problem. The author studied in detail how to formulate the problem, the complexity of it and made some useful simplifications that we are going to adopt, as for example the deployment of flares using a given shape or the idea to launch flares and starting the evasive maneuver at the same time.

In that work the author started from a different formalization of the game than the one we did: he described the game as a three players game, where the first player is the warship, the second is the missile, and the third is an agent called Nature. In his formulation, the first player's strategy represents the evasive maneuver, the second player's strategy is the choice of the *Id* of the missile, and the third player's strategy is the initial conditions.

After this first formalization of the game, the author proceeded to the search of optimal strategies within the search space but, since the second player (i.e. the missile) in his formulation does not have any possible choice for its strategy, the search is reduced in finding the best set of actions for the boat. The fact that it is searching the strategies only for a player is a dramatic change in comparison to our case: finding the optimal strategy only for the boat allows to search just for a minimum in the utility function while, since we are dealing with a function that is defined by the two players' strategies, we have to search for a maxmin point of equilibrium. This important aspect of the work will allow us to use more Game Theory related concepts and algorithms than the ones the author of [27] used.

The algorithm used for the search of optimal boat's actions are in *Matlab Global optimization toolbox* and are divided in two methods: Pattern Search and Surrogate Optimization. Pattern Search algorithms work initializing one point as optimal, then searching in the nearest point and, if a better point is found, this becomes the new optimal and the search continues. Surrogate Optimization instead, aims to find an approximation of the utility function (i.e. a surrogate) and finds the minimum within this approximation.

After the search phase, the author proposes another phase of the algorithm that is the interpolation phase, which aims to use the previously acquired knowledge in the first phase in order to find good solutions even when the initial conditions change. For the interpolation phase the author proposes three different approaches: Nearest Neighbour, Cubic Spline and Linear. In this work we will not work on the interpolation phase since we assumed that the methods found by Longobardi were good enough for this work.

This work aims to expand the results found by [27] especially in the strategy search phase of the algorithm. The main difference is that our formulation of the game as a two-player zero-sum game does not allow us to use standard minimization techniques over the utility function since we are not searching for a minimum point, but a maxmin. This forced us to adopt and implement a new algorithm different from the others already present in the literature.

Another important difference is that we had the opportunity to test the algorithm we developed in another scenario different from ship-missile one. *Hit the spitfire game* allow us to assess the performance of the algorithm in a very precise way, running multiple experiments in a toy environment with a known solution. This allows us also to compare our algorithm with the actual state of the art ones, understanding if it is comparable, worst or even better. Only after this first experimental analysis we will use the algorithm to try to find a solution in the ship-missile scenario. On the other hand, Longobardi did not have the opportunity to test the algorithm in *hit the spitfire game*, both because he did not have it and also because his algorithms are not applicable to the game. So, he was only able to test its pipeline in the boat-missile game, that for sure is a valid test scenario, but not as complete as *hit the spitfire game*.

Despite the differences between this work and Longobardi's one, his work for very useful for us, especially the preliminary one which allows us to spare time in studying the basis of the problem. Unfortunately, due to the differences of our algorithms, was not possible to have a complete performance confrontation of our works.

Chapter 3

Ship-missile game

In this section, we formulate the settings of our problem. First,we provide a physical description of the scenario that we are facing, describing the initial conditions of the simulations and the defensive maneuvers that can be made. Then, we formulate a Game Theory model of the scenario. At the end of the chapter, we discuss the problem that this game poses us and we introduce some simplifications which make the problem more tractable.

3.1 Initial conditions

The simulator represents a war scenario in which an IR missile is targeting a warship. In order to have a complete representation we need to set those parameters:

- $Id_{ship} \in [1, \ldots, 10]$: this is the entry that we use to model the kind of warship that will be used during the simulations. Different warships have different size and moving capabilities.
- $Id_{missile} \in [1, \ldots, 11]$: this is the entry that we use to model the kind of missile that will be used during the simulations. Different missiles have different dynamic characteristics, different tracking systems, different way to explode and a different behaviour when they are near the target.
- $Id_{flares} \in [1, \ldots, 5]$: this is the entry that we use to identify the kind of countermeasures that will be used during the simulations. Different flares have different radiation areas and radiation intensities.
- $n_{flares} \in \mathbb{N}$ is the number of flares that can be deployed by the ship. The fact that are known in advance and are not part of the game is a sound assumption since we can imagine that the captain of the

warship knows how many flares it has and it wants to use all of them to escape the threat.

- $d_{boat-missile} \in \mathbb{R}$ is the initial distance between the missile and the warship when the simulation starts (Figure 3.1).
- $z_{missile} \in \mathbb{R}$ is the height of the missile.
- $v_{missile} \in \mathbb{R}$ is the starting speed of the missile.
- $v_{ship} \in \mathbb{R}$ is the starting speed of the ship.
- Other environmental variables like wind, sea condition, natural light and geographical position.

Note that the simulation starts every time with the head of the missile targeting the center of gravity of the boat, with the missile at its cruise altitude and speed. We do not need to specify the cartesian coordinates of the boat and the missile because, since there are only two entities involved in the simulation, their reciprocal position is defined by a one-dimensional variable: $d_{boat-missile}$. Once $d_{boat-missile}$ is set, we assume the position of the missile in the space will be $(0, 0, z_{missile})$ and the boat will be positioned in $(d_{boat-missile}, 0, 0)$.

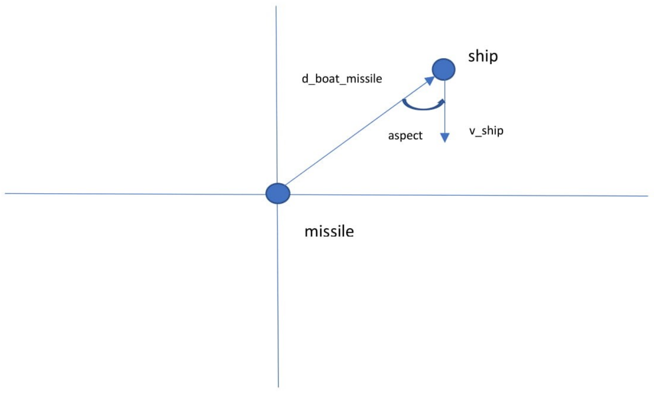


Figure 3.1: A representation of the simulation space.

3.2 Game strategies

During the game, the goal of the missile is to hit the warship while obviously the boat aims to escape the threat of the missile. In this section we discuss the possible strategies that missile and boat can implement to reach their goal.

3.2.1 Action of the missile

The only choice that can make the missile is to set the α_{aspect} of the boat (see Figure 3.1), that is the angle between the vector defined by the movement of the missile and the one of the ship. This means that the missile has the possibility to choose from which angle to start the attack of the warship. The assumption that the missile chooses the angle of attack seems fine because it is logical than it is launched from the position where it has more possibility to hit the target. We can also imagine that, if the missile is launched from a fixed position, for example a SSM (surface-to-surface missile) launched from land, it can go around the warship at a distance where it will not be detected before approaching it.

After the missile approaches the ship following the desired direction, we can assume that the simulator will model the perfect behavior of the missile: so all the following actions will be the best in order to hit the boat.

The decision of letting the missile decide its angle of attack, instead of assuming it as an initial condition, is necessary to model the simulation as a game. It also has a main advantage: the algorithms that we are going to use will spend the most part of the simulations using attack angles that are more effective for the missile, in such a way we will be able to have more precise estimates of the countermeasure efficacy in case of good approach angles.

3.2.2 Actions of the warship

Conversely, what it concerns the warship is different: it has two kinds of countermeasures to deploy in order to escape the threat. The first countermeasure the warship can adopt is to launch flares, in order to launch flares. Each flare needs to be set (see Figure 3.2):

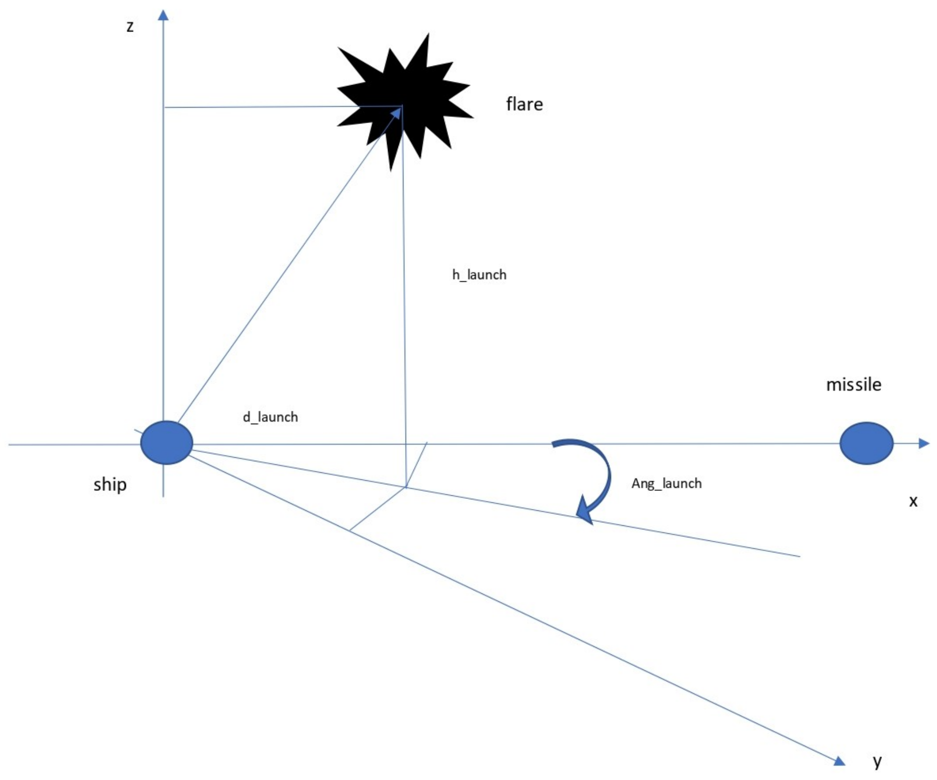


Figure 3.2: Flare launching scheme. There are the three spatial parameters that characterize the launch of a countermeasure.

- h_f^i determines the height of the i-th flares launch. In other words specify its position in the z axis.
- d_f^i is measure of the distance between the center of gravity of the warship and the i-th flare.
- α_f^i Is the angle defined by the line that passes from the center of gravity of the ship and the center of gravity of the missile, and the xy projection of the position of the i-th flare.
- t_f^i Is the instant in which the i-th flair is launched.

Note that the simulator does not accept those kinds of relative coordinates as d_f and α_f to determine the position of the flare in the space, but it needs three dimensional coordinates in the same reference system of the ship and the missile, so it is necessary compute (x_{flare}, y_{flare}) using the position of the ship at time t_{launch}. Together with flares' launch the boat can escape the threat using a maneuver which is defined by three measures:

- t_{man} is the time when the escaping maneuver begins.
- α_{man} is the new angle of the ship with respect to the missile's trajectory.
- v_{man} is the speed that the boat reaches during the maneuver.

3.3 Formulation of the game

We are now ready to give a game-theoretic formulation for the ship-missile game. We can describe it as a *two-player zero-sum simulation based games.*

- The set of players N is defined as $N = [1, 2]$ where 1 represents the missile, which is the first player to choose a strategy, and 2 represents the ship which, after looking at the strategy of the first player, deploys its countermeasures.
- $\mathcal{X} = [\alpha_{aspect}]$, where $\alpha_{aspect} \in \mathbb{R}$ and $\alpha_{aspect} \in [0, 360]$, is the set of strategies for the player 1, as said in Section 3.2.1 this set contains only one variable.
- $\mathcal{Y} = [t_{man}, \alpha_{man}, v_{man}, t_f^1, h_f^1, d_f^1, \alpha_f^1, \ldots, t_f^n, h_f^n, d_f^n, \alpha_f^n]$, where $n \in \mathbb{N}$ is the number of flares available, is the set of strategies for the player 2.
- The utility u given by the Oracle can be defined in two way. The first method returns a utility $u \in [0, 1]$ and $u \in \mathbb{N}$, and corresponds to the variable *boom* of the simulator, which is 1 if the missile hits the ship, 0 otherwise. The other possibility to define the utility function u is to take the minimum distance during the simulation. In this case the problem is a minmax because the player 1 tries to minimise u, while in the previous case the game is a maxmin. The distinction between minmax and maxmin is purely formal, since it is sufficient to take the opposite of the utility function $(-u)$ to trasform the problem into a minmax one and vice versa.

3.4 Complexity issues

The formulation of the ship-missile game gives some problems: some of them are the classical ones of the simulation-based games theory, others are more specific for this scenario.
The most typical issue of simulation-based games is sampling the utility function due to the cost of running the oracle. As discussed before in

Section 2.4.5, our simulator is extremely computationally expensive due to environment complexity: the time of a simulation can vary from two to six minutes based on the complexity of the tracking algorithm of the missile. Another characteristic of this simulator is noise as can be seen in Figure 3.3: it is easy to find situations where two identical queries bring different results. This will force us to run multiple simulations in order to estimate with a given accuracy the utility function in a point of the strategy space.
The second issue is the complexity of the strategy space: in every simulation there is one variable for the first player and $3 + 4 * n_{flares}$ possible different actions for the second player, this brings to a utility function which lays in a high dimensional space. In this settings, it is nearly impossible to define a set of points that is small enough to be analyzed with actual algorithms and hardware: taking for example $n_{flares} = 8$, which is a typical number of flares, and taking only 10 different values for each possible strategy of the first and second player, the number of multivariate points that we will need to take into consideration for our algorithms would be in the order of 10^{35}, which is a number way far from what computers allow to manage, especially when we are talking about complex physical simulations as in this case.
The high dimensionality problem becomes even more challenging when we think at the nature of the variables that model the possible strategies: they are all represented by a rational number, thus we can say that, even assuming a single flare is deloyed in the simulation, our utility function lays in $\mathbb{R}^7$.
Another aspect that needs to be pointed out is the difference of the number of possible strategies between the two players: the possible strategy of the first player is made by only one variable while the second player can choose from a set of very high dimensional strategies. This means that usually there are far less possible strategies for the first player than for the second one.
The high dimensionality of the game and the cost of a single simulation make the problem probably impossible to tackle using standard techniques for SBGs, even with the best technologies and algorithms present nowadays.

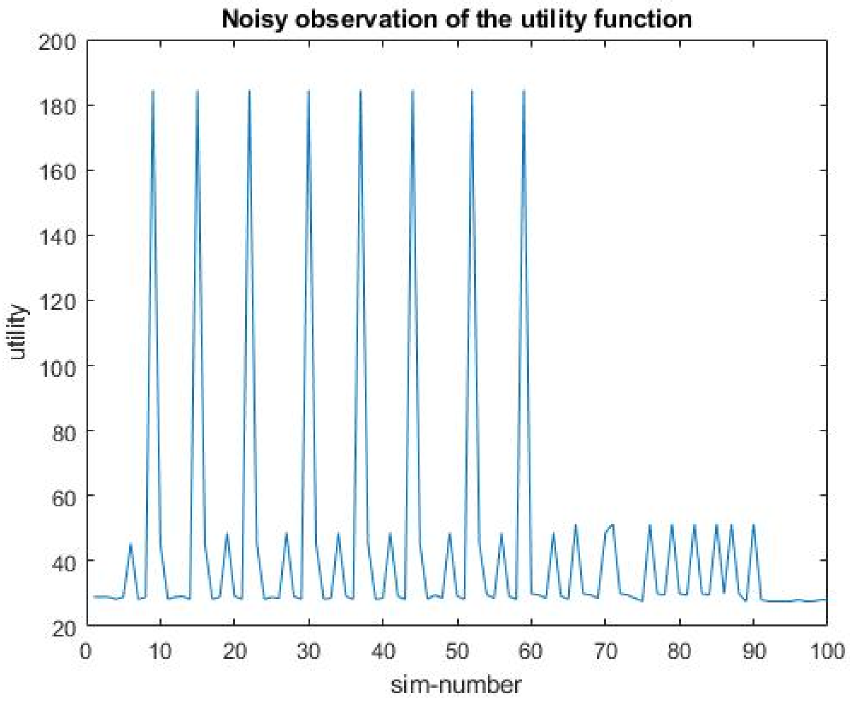


Figure 3.3: Those are 100 different observation of the utility function taken in the same point where we can see the effect of noise. In this case the utility function is the assumed to be the minimum distance between the ship and the missile during the simulation

3.5 Complexity Reduction

In the previous section we explained why the problem is nearly unfeasible to be faced as it is. Now, we describe the set of simplifications that we make in order to allow an empirical tractation of it. First, we decrease the search space by factorizing it, then we describe how we deal with infinite spaces.

3.5.1 Factorization of the search space

The assumption that was used for the complexity analysis of the search space in Section 3.4 and that makes it so high-dimensional, is that every flare is launched in an independent fashion from all the others. In that way we are going to add four new dimension $(t_f, d_f, h_f, \alpha_f)$ to our problem every time a flare is added to the simulation.

Actually this assumption is not sound in a real-world scenario: we can think that the flares will be used in a more effective way if they are launched following some patterns: for example creating a curtain in front of the ship or making the missile target them until is too far from the ship to hit it. In this way we can model complex patterns of flares in a simpler way than launching them independently, reducing drastically the dimensionality of the search space.

Following [27] the most effective way to deploy flares is creating a horizontally aligned curtain (Figure 3.4) in front of the boat which hides it.

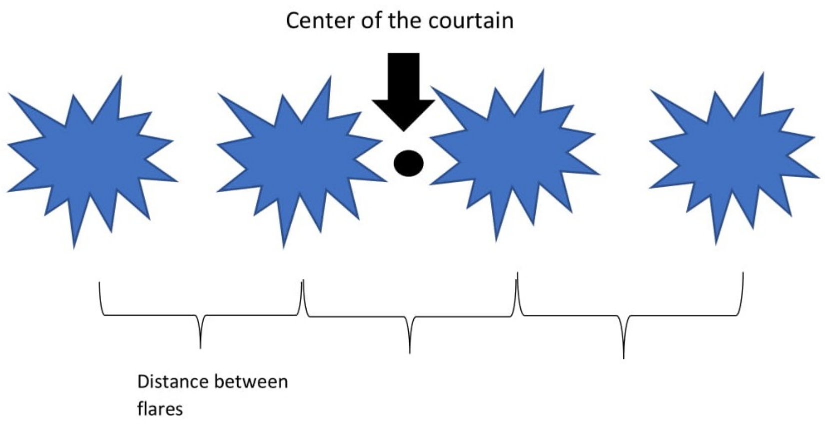


Figure 3.4: The scheme of the deployment of a curtain of flares horizontally aligned.

In this case, instead of setting all the parameters necessary to deploy all flares in the desired shape, it is enough to set only the three-dimensional position of the center point of the curtain and the time where the deployment starts. So, in the example depicted before where we are launching 8 flares, now we are working in a space of 7 dimensions instead of a space of 35 dimensions. The seven dimensions we are considering now are:

- $aspect$: the angle between the ship and the missile;
- t_{start}: the time when the evasive maneuver starts and the flares are launched;
- h_{flares}: the height of launch of the center of the courtain;
- d_{flares}: the distance from the boat of the center of the courtain;
- α_{flares}: the angle in which the center of the courtain is positioned;
- v_{man}: the speed of the evasive maneuver;

- ang_{man}: the angle of the evasive maneuver.

There is also another parameter that needs to be set which is the inter-flare distance, which is a distance between a flare and another and the time in which each flare is launched. For the sake of simplicity in this work the inter-flare distance is set to a constant value of $20.5m$ as suggested by [27] while the flares are launched with a constant interval of time within each other starting from the leftmost.

The choice of not launching them at the same time was taken because most missiles are able to detect such abrupt variation on IR conditions and keep following the place they were pointing before the launch and eventually hit the boat.

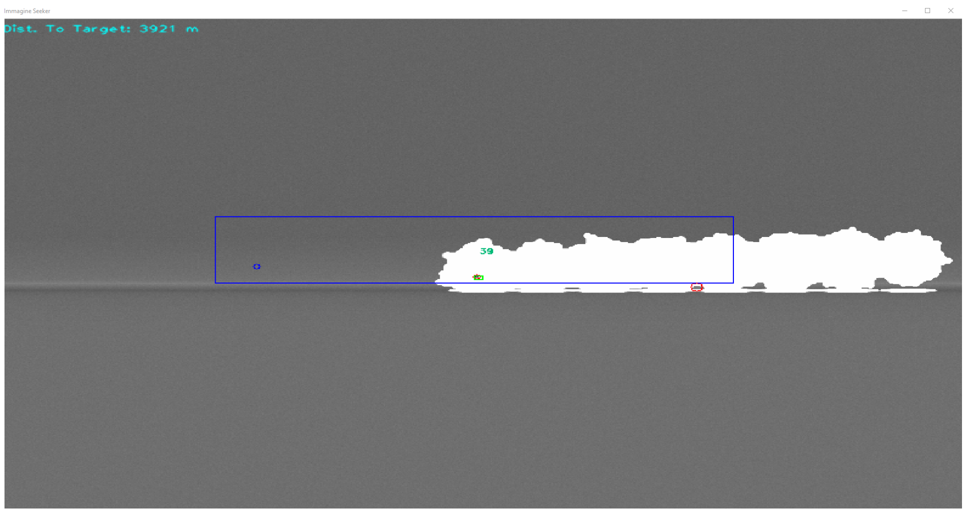


Figure 3.5: The field of view of the missile when a courtain of 8 flares is deployed. The ship is in the center of the courtain and it cannot be seen by the missile, which is trying to track its position.

Other examples of fixed structures that can be used are a rectangular one, where the flares are also vertically aligned, a set of seductor flares that makes the missile point on it or a double curtain of flares, with the idea that the boat hides behind the second curtain and, once the missile pass the first one, finds another curtain in front of it. Note that in all the rest of this work we will take into consideration only the horizontally aligned flares scenario, leaving the analysis of the other factorizations for successive works.
Once we have defined the structures we want to analyze we will run the algorithms depicted in the next chapter in order to find the best setting for each one and eventually the best structure overall. We can imagine this procedure of using predetermined patterns to model the shape of the flares

as a factorization of the input space. This means that, instead of searching for a solution in the global space, which is way too big, we are going to divide it into smaller regions and we will find a good solution for every sub-space. Then, we can put all we have learned from the various subspaces together to deploy the best countermeasure in a real-world scenario.

3.5.2 Inifite spaces and further simplifications

The last problematic aspect we have to deal with is the infinite nature of the strategy space which lays in $\mathbb{R}^7$. This would not be an issue if our utility function was known a priori but, since we have to query a simulator in order to sample it, the use of an infinite space is not feasible because we cannot sample infinite points. Thus we are going to use a multidimensional grid of discrete points in order to approximate the continuous space.
Ideally, the granularity of the grid should approximate the continuous situation. Note that the Gaussian Process assumption that we made on the utility function will allow us to estimate the value of the strategies even in points that are not part of the grid, even if with less accuracy than the ones we have actually sampled that belong to the grid.
Another simplification we did, and this was for code simplicity reason as described by [27], is to assume that the ship maneuver and the launch of the flares happen at the same time, called t_{start}. Actually in launching simulation we do not specify directly t_{start} but we just specify the distance the missile has to reach when the ship starts the escaping maneuver.

Chapter 4

Strategies Generation Algorithm

In this chapter, we present the algorithm we developed in order to face the issues described in the previous chapter. First of all we explain why the methods that are already present are not able to scale in this scenario and why we need to develop a new method. Then, we explain the general idea behind the algorithm and, then, we give a detailed description of it.

4.1 Preliminaries

Since the ship-missile game is a *two-player zero-sum infinite space simulation-based game*, we evaluate the possible use of the applications of the two algorithms described in Section 2.3.1: GP-SE and M-GP-LUCB. Those algorithms were developed for the same type of game we have and have proven to be δ-PAC under some smoothness assumptions for the utility function. So the first thing we do is trying to visualize the shape of our utility function in order to understand if it has the smoothness properties that would help us to apply those algorithms to our problem in a more efficient way.

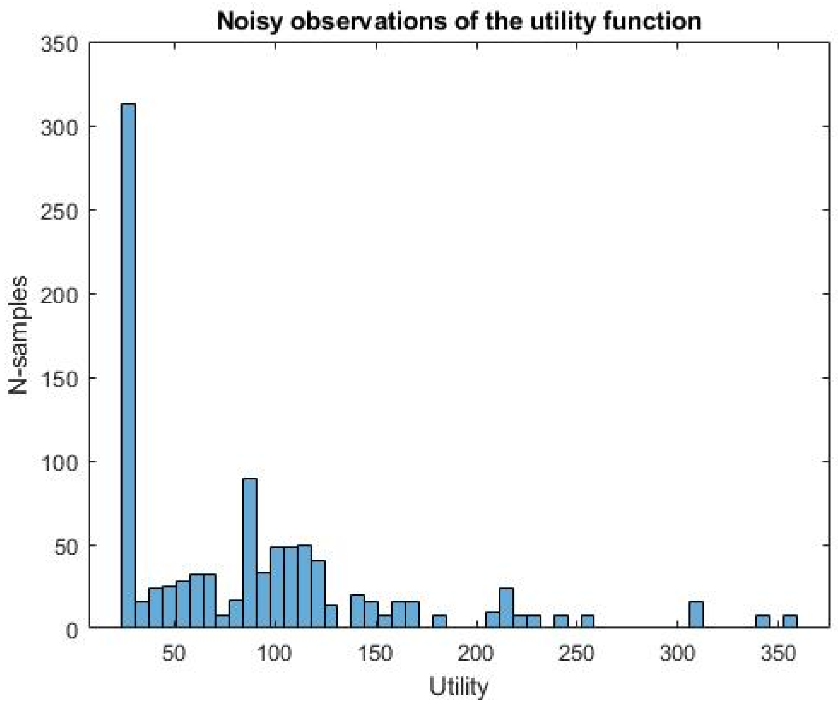


Figure 4.1: This plot represent one thousand samples of the utility (min distance ship-missile) taken on a single point.

In Figure 4.1, we plotted the histogram of the noise in order to understand if a single point measurement can be modeled as a known random variable (e.g., Gaussian or Normal), but unfortunately the distribution of samples does not seem to follow any characteristic pattern.
We also plotted how the utility function varies when we do small changes to the input parameters. We tried to go from a point where the boat avoids the threat to a point where the missile hits the boat changing one parameter at time, to see if the changes are abrupt or happen in a smooth way. The results of this experiment are depicted in Figure 4.2. Note that, in this plot, the utility function is expressed as the mean value of the variable *boom* using 20 simulations for each point. From those plots we can see that the utility function seems to have gradual changes and usually near points have near utilities. This aspect could be useful when using our algorithm to solve the problem.

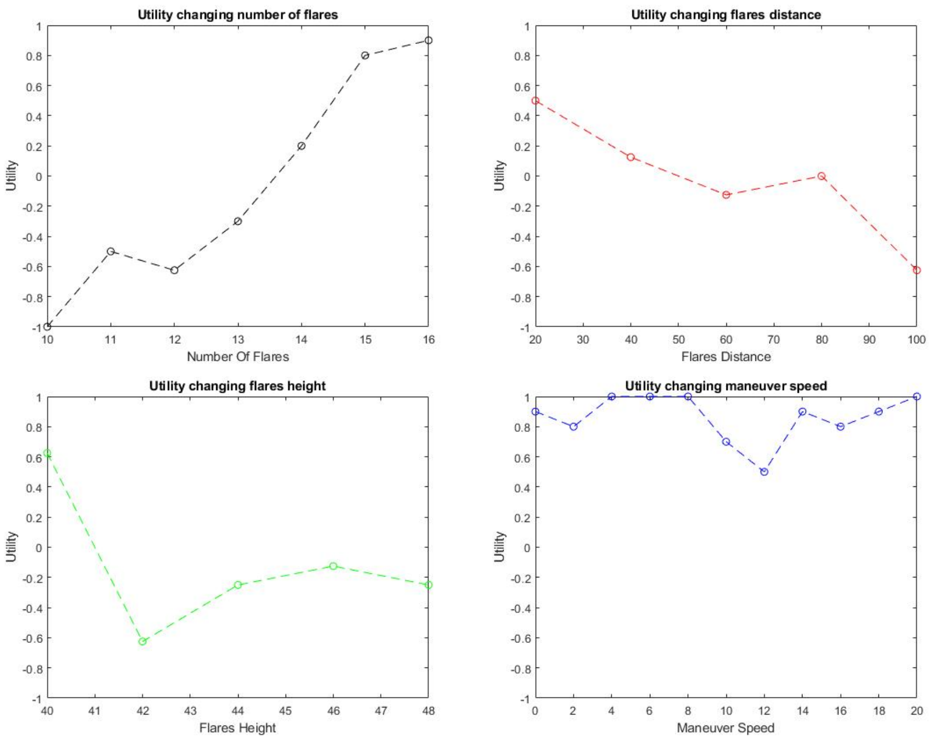


Figure 4.2: How the utility changes by varying in a gradual way some input parameters. We changed respectively: number of flares, distance of deployment, flares height and maneuver speed

Even if we are not able to prove that our utility function is generated by a Gaussian Process, we still model it as a GP in order to exploit some of its properties.
The most interesting property of the GPs for this work is that the kernel function is able to estimate the value of the utility in one point based on the utilities of the nearest points without sampling it, allowing to concentrate our sampling procedure only in zones of the space that are more promising for us. In this way, we will have some regions in the entire search space where the probability of having a maxmin strategy is estimated to be low, and in that regions we will use few simulations to obtain utilities, while the regions of the space where there is the equilibrium strategy with high probability will be sampled more. This property is crucial for this work because there is no possibility to scan the entire strategy space for its dimensions, the cost of a simulation and the noise, so we need to direct our query in a smart way.
Another nice property of GPs in our setting is that this not only gives estimates of the value of the utility function in each point, but also gives us a measure of uncertainty of the estimation, which is useful to build upper

or lower confidence intervals of the estimations.
The only drawback of the GPs is the space complexity when the number of strategies grows: the kernel function is a matrix of shape ($n_{strategies}$, $n_{strategies}$), so it grows quadratically with the number of strategies and becomes quickly impossible to invert. Even if we use the recursive formulation of kernel update Equations (2.5) and (2.6) the number of computations to be made grows quadratically with the number of strategies in the grid.
In the ship-missile game the complexity of GPs does not affect the running time of our algorithm since the simulation phase is much more expensive than the update phase even with a high number of strategies (4500 for example), but this is an aspect to be taken in consideration when dealing with games where the simulation phase is less expensive.
Despite all the good properties of GPs, where GP-SE and M-GP-LUCB are based, we found that those algorithms are still not suitable for the ship-missile problem. GP-SE should be called with a budget bigger than the number of points in the grid, but we want to develop an algorithm that even with a budget similar or smaller to the grid size, will be able to find a good solution.

4.2 General idea

The basic idea of the new algorithm is to reduce the strategy space size by sampling iteratively some strategies to create a new subspaspace. In this subspace we can GP-SE or M-GP-LUCB to find the optimal solution in the reduced problem. We want that, as the learning process proceeds, the reduced problem is more likely to contain the solution (the equilibrium point), so we want to be able to use the information acquired in the previous epochs to sample with high probability strategies that are near to the maxmin one. In practice, this procedure produces a cloud of points that moves towards the optimal solution.

This algorithm can be divided into two steps: the strategy generation phase and the search phase: the strategy generation phase is the one where we sample a subset from the large strategy set while in the search phase we are exploring the subspace given by the previous step. Now we are going to describe more in details those two steps.

4.2.1 Strategy generation phase

During the strategy generation phase, we want to sample strategies that are likely to be maxmin with high probability, so the most important part of this phase is the assignment of probabilities at each strategy profile of the game. We will explain it better with a toy example in Figure 4.3.

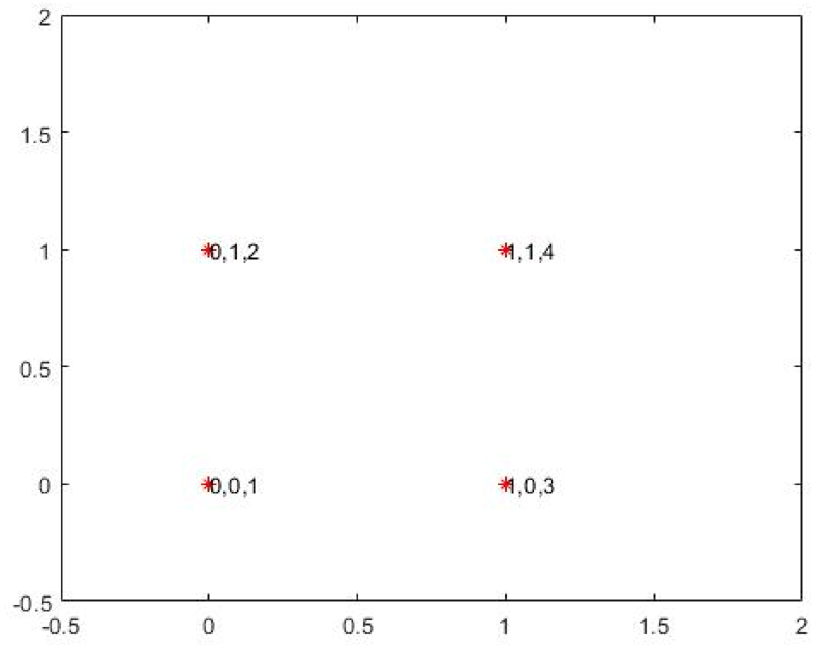


Figure 4.3: Toy example of a game. Each red point is a strategy profile and near each point are indicated in order the index of the strategy of the first player (either 0 or 1), the index of the strategy of the second player (0 or 1) and the value of the utility function in that point.

First, we want to sample the strategies of the first player that in this case can be either zero or one. So, since we are searching for a maxmin strategy profile, we are assigning probabilities to the first player actions which are directly proportional to the minimum utility of all the strategies profiles that contain the action. In a mathematical notation, being $x_i \in \mathcal{X}$ the i-th possible strategy for player 1 and p_{xi} the probability of sampling x_i, calling $\pi(x, y)$ the strategy profile where the first player plays a generic action x and the second player plays y, we will assign to p_{xi} a value that is:

$$p_{xi} \propto u(\pi(x_i, y_i^*(x_i)), \tag{4.1}$$

where y_i* is the best response of the second player to action x_i :

$$y_i^*(x_i) = \operatorname*{argmin}_{y \in \mathcal{Y}} u(\pi(x_i, y)). \tag{4.2}$$

The exact values that p_i assumes will be discussed in the next section when explaining all the possible probability attribution methods.
So in our example we have that

$$\min_{y \in \mathcal{Y}} u(\pi(x_i = 0, y)) = 1, \tag{4.3}$$

$$\min_{y \in \mathcal{Y}} u(\pi(x_i = 1, y)) = 3. \tag{4.4}$$

If we assume a linear attribution of probabilities the formula to assign the probability to action x_i is:

$$p_{x_i} = \frac{\min_{y \in \mathcal{Y}} u(\pi(x = x_i, y))}{\sum_n \min_{y \in \mathcal{Y}} u(\pi(x = x_n, y))}, \tag{4.5}$$

$$\tag{4.6}$$

thus, the probabilities of sampling strategies 0 or 1 will be:

$$p_{x_0} = \frac{1}{1+3} = \frac{1}{4}, \tag{4.7}$$

$$p_{x_1} = \frac{3}{1+3} = \frac{3}{4}. \tag{4.8}$$

Once we have generated this distribution, we take n samples without replacement from it, in this case $n = 1$. After sampling n strategies for the first player we want to sample the joint strategies: we sample m strategies of the second player for each one of the n of the first player. Calling $y_i \in \mathcal{Y}$ the i-th strategy of the second player, p_{y_i} its probability and x_s the already sampled strategy of the first player, we have that:

$$p_{y_i} \propto \frac{1}{(u(\pi(x_s, y_i)))}. \tag{4.9}$$

For the probability assignment of a generic strategy y_i, once we have selected x_s:

$$p_{y_i} = \frac{u(\pi(x = x_s, y_i))}{\sum_n u(\pi(x = x_s, y_n))}. \tag{4.10}$$

In our toy example, assuming that $x_i = 1$, we have:

$$p_{y_0} = \frac{3}{3+4} = \frac{3}{7} \tag{4.11}$$

$$p_{y_1} = \frac{4}{3+4} = \frac{4}{7} \tag{4.12}$$

From the above distributions we created we sample m strategies for each one of the n strategies of the first player, creating a cloud of $n * m$ strategy profiles.

Figures 4.4 represent the true position in the space of the cloud of strategies generated in a 2D game after n epochs, the blue dots are the strategies selected in the given epoch, while the red cross is the maxmin strategy of the space. It is easy to see how the density of the points after some epochs is getting closer and even including the optimal point.

4.2.2 Strategy search phase

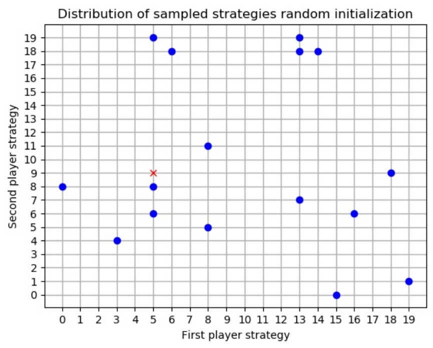
In the search phase, GP-SE is used to find the maxmin strategy in the reduced strategy space that generation phase creates. We are not interested in finding the local maxmin strategy but we want only to explore the subspace. We will still use GP-SE for this task because it allows us to do this exploration in an efficient way: using more simulations in the parts of the subregion where is more likely to find the maxmin strategy.

The choice of GP-SE for this phase, instead of M-GP-LUCB is due to the possibility of parallelizing the sampling process: the round-robin and subsequent eliminations fashion that uses GP-SE gives the opportunity to execute all the simulations needed in a round in parallel. This aspect is crucial to speeding up the process in the case of ship-missile game because in this way it is possible to lower the mean time of a simulation depending on how many cores has the machine where the algorithm is running. When the simulation time is lower than the update time, and this can happen with a less complex simulator than boat-missile one as we can see in the next chapter, the importance of the parallel nature of GP-SE decreases.

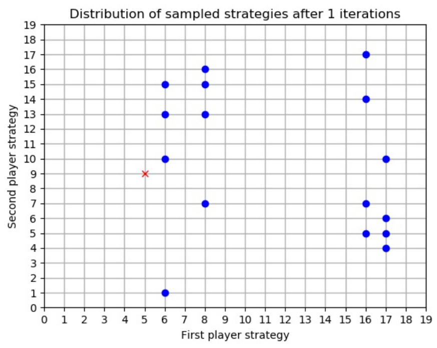
Another important aspect of GP-SE that makes us prefer it over M-GP-LUCB is the fixed budget setting: the number of simulations that are done to finish its execution it is a priori known and it can be tuned in the following chapter to obtain better results. It is not possible to tune the number of simulations to be done until termination in a direct way using M-GP-LUCB.

4.3 The proposed algorithm

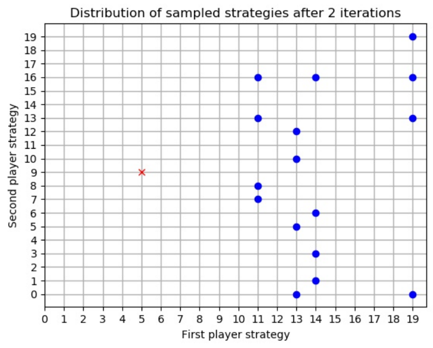
In this section, we are going to describe how we design the new strategy generation algorithm. First, we will provide the pseudocode for it and then we describe in detail every step.



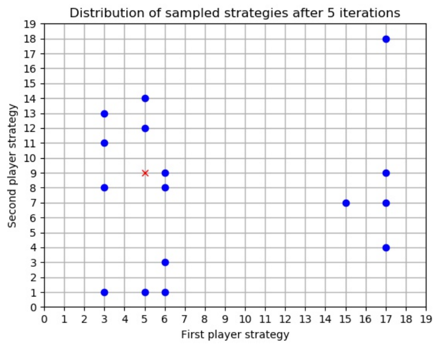
(a) Random Initialization



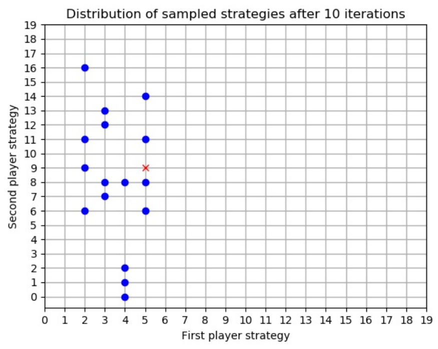
(b) 1 epoch



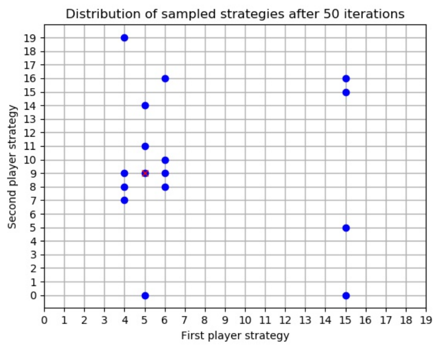
(c) 2 epochs



(d) 5 epochs



(e) 10 epochs



(f) 50 epochs

Figure 4.4: Movements of the cloud of points with the growing of number of epochs done.

4.3.1 Pseudocode

Algorithm 3 STRATEGY-GENERATION-ALGORITHM(t_{epoch}, n_{epochs},n,m)

```
1: Initialize $\mu_0(\boldsymbol{\pi}), k_0(\boldsymbol{\pi}, \boldsymbol{\pi}')$
2: Initialize $\textit{strategies-round} = \text{random}(n * m)$
3: $i = 0$
4: do
5:     $[k(\boldsymbol{\pi}, \boldsymbol{\pi}'),\ \mu(\boldsymbol{\pi})] = \text{GP-SE}(\textit{strategies-round},\ t_{epochs}, k(\boldsymbol{\pi}, \boldsymbol{\pi}'),\ \mu(\boldsymbol{\pi})\ )$
6:     $\textit{strategies-round} \leftarrow \textit{select-strategies}(\mu(\boldsymbol{\pi}), n, m)$
7:     $i \leftarrow i + 1$
8: while $i < n_{epochs}$
9: return $\textit{find-max-min}(\mu(\boldsymbol{\pi}))$
```

Algorithm 4 GP-SE(*strategies-round*, t_{epoch}, $k(\boldsymbol{\pi}, \boldsymbol{\pi}'), \mu(\boldsymbol{\pi})$)

```
1: Initialize $\mu_s(\boldsymbol{\pi}) \leftarrow \mathbf{0}$, $k_s(\boldsymbol{\pi}, \boldsymbol{\pi}')$
2: Initialize $\textit{n-remaining} \leftarrow \textit{size}(\textit{strategies-round})$
3: Initialize $T_p \leftarrow 0$, $i \leftarrow 0$
4: while $\textit{n-remaining} > 1$ do
5:     $T_{p-1} \leftarrow T_p$
6:     Calculate $T_p$ using Equations (2.14) using $t_{epoch}$
7:     for $i \leftarrow T_p - T_{p-1}$ do
8:         Query all the remaining strategies
9:         Update $k(\boldsymbol{\pi}, \boldsymbol{\pi}'), \mu(\boldsymbol{\pi})$ using (2.1) and (2.3)
10:        Update $k_s(\boldsymbol{\pi}, \boldsymbol{\pi}'), \mu_s(\boldsymbol{\pi})$ using (2.1) and (2.3)
11:    Eliminate worst strategy using (2.12) and (2.13)
12:    $\textit{n-remaining} \leftarrow \textit{n-remaining} - 1$
13: return $k(\boldsymbol{\pi}, \boldsymbol{\pi}'), \mu(\boldsymbol{\pi})$
```

Algorithm 5 select-strategies($\mu(\boldsymbol{\pi}), n, m$)

```
1: selected-strategies = [ ]
2: Initialize min-util-per-action
3: for $i \in$ number of possible first player strategy do
4:     min-util-per-action[i] = min(utilities of $\boldsymbol{\pi}(i, y)$)
5: probabilities = normalize(min-util-per-action)
6: actions-one = sample(probabilities,$n$)
7: for $j \in actions - one$ do
8:     $mean_2$ = all the strategies with first action $j$
9:     probabilities-two = normalize($mean_2$)
10:    actions-two = sample(probabilities-two, $m$)
11:    append to selected-strategies new strategies generated
12: return selected-strategies
```

Algorithm 6 find-min-max($\mu(\boldsymbol{\pi})$)

```
1: minima = zeros(size(possible strategies player 1))
2: for i in size(minima) do
3:     minima[i] = min(strategies whit $x = i$)
4: return index-of-strategy(argmin(minima))
```

4.3.2 Detailed Description

Algorithm 3 starts initializing the means vector to 0 and the kernel matrix. There are several functions that can be used as kernel, in this work we choose the squared exponential kernel, so we initialize it like this:

$$k(\boldsymbol{\pi}, \boldsymbol{\pi}') := e^{-\frac{1}{2l^2}||\boldsymbol{\pi}-\boldsymbol{\pi}'||^2}, \tag{4.13}$$

where l is a length-scale parameter.

Once we have initialized the GP, we need to choose the first set of strategies to explore and, since we still have no information about the utility function, we choose it at random. We use GP-SE to explore the set of strategies that we generated, but we need to do some modification to the algorithm described in Algorithm 2. The modifications rely on the fact that the objective of each run of GP-SE is not to find the maxmin strategy in the small set of strategies we give him, but to explore them in an efficient way, so we need two kernel matrices and two means vectors. The first kernel and mean vector are for the global strategy set which represents the whole game, while the second kernel and mean vector are for the local strategy set which has just been sampled. Every time we do a simulation we need to update both the *global* kernel and μ and the *local* ones. We decided to initialize from scratch *local* kernel and means, but a valid solution could also be to use *global* kernel and μ as prior to initialize them. This choice is made because we want GP-SE not to use the previous information about the Gaussian Process, that in some cases can be wrong and can change the result of the exploration. After its execution, GP-SE returns the updated matrix *global* kernel and the vector *global* μ.

Once the execution of GP-SE is finished, the strategy generation phase described in Algorithm 5 starts. First, since we are searching for a maxmin equilibrium, for each strategy of the first player we take the minimum possible value in $\mu(\boldsymbol{\pi})$ and we save it in a list that we normalize to obtain a probability. We have designed three different ways of normalizing this vector:

- Linear: this is the simplest way to normalize a vector in order to obtain a probability. Given a vector $\mathbf{x} = [x_1, x_2, \dots, x_n]$, $x_i \in \mathbb{R}$, it transform it in a vector $\mathbf{z} = [z_1, z_2, \dots, z_n], z_i \in \mathbb{R}$ such that:

$$z_i = \frac{x_i + 2|\min(\mathbf{x})| + \epsilon}{\sum_{k=1}^{n}(x_k + 2|\min(\mathbf{x})| + \epsilon)} \tag{4.14}$$

  , where $\epsilon > 0$. This simple method has the drawback that, for high values of n, the probability of each strategy vanishes.

- Softmax normalization: it is a nonlinear normalization which should allow to a better choice of strategies. After a linear normalization following Equation (4.14), the vector $\mathbf{x}$ becomes $\mathbf{z}$, from vector $\mathbf{z}$ we extract the probabilities vector $\mathbf{p} = [p_1, p_2, \dots, p_n]$ with:

$$p_i = \frac{\exp(\frac{z_i}{\tau})}{\sum_{k=1}^{n} \exp\left(\frac{z_k}{\tau}\right)} \tag{4.15}$$

  where the τ parameter is called temperature and it is an exploration factor. If $0 \leftarrow \tau$ we will have pure exploitation (only one possible action with probability 1), while with larger values of τ we will have more exploration.

- Optisofmax normalization: since the tuning of τ parameter is not a trivial task, we implemented a solution presented by [23]. The works aims to find a method to minimize:

$$L(\tau) = (H(\mathbf{z}) - H(\mathbf{p}))^2 - \lambda H(\mathbf{p})^2 \tag{4.16}$$

where $H(\mathbf{x}) = \sum_{k=1}^{n} x_k \log(x_k)$ for a generic $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ and $\mathbf{z}$ is a vector such that $\sum_{k=1}^{n} z_k = 1$ and $z_i \geq 0$, thus making $H(\mathbf{z})$ an entropy. In the same way, being $\mathbf{p} = [p_1, p_2, \ldots, p_n]$ such that $p_i = \frac{\exp(\frac{z_i}{\tau})}{\sum_{k=1}^{n} \exp(\frac{z_k}{\tau})}$, we can say that $H(\mathbf{p})$ is the entropy of $\mathbf{p}$. So the Equation (4.16) means that we want to minimize the difference between the information contained in $\mathbf{z}$ and $\mathbf{p}$, using a regularization term which is proportional to the information in $\mathbf{p}$.
After defining the function to be minimized, the authors propose a gradient descent method where τ is updated until convergence with:

$$\tau = \frac{\sum_{k=1}^{n} (z_k \exp(\frac{z_k}{\tau_{-1}}))}{[\sum_{k=1}^{n} \exp(\frac{z_k}{\tau_{-1}})][\log \sum_{k=1}^{n} \exp(\frac{z_k}{\tau_{-1}})] - \frac{H(\mathbf{z})}{1+\lambda}} \tag{4.17}$$

where τ_{-1} is the value of τ in the previous iteration. This approach automatically tunes the τ parameter but adds some computations to our algorithm.

Once we have obtained a vector of probabilities $\mathbf{p_1}$ for the strategies of the first player, we take n samples from it, then, for each selected strategies of the first player, we select m strategies for the second player in a very similar fashion: we take all the utilities of the strategies which have the first move equal to the one already selected and we normalize that utilities, then we will sample m joint strategies from them.

After sampling nm joint strategies we repeat the search and generation phase until the algorithm ends. The termination of the algorithm is given by a budget constraint: the total number of simulations we perform must not exceed it. It is very easy to know how many epochs will run the algorithm knowing the budget of each GP-SE run. Once the algorithm finishes his budget it uses $\mu(\boldsymbol{\pi})$ to estimate the maxmin strategy and returns it. This termination condition is given by the fact that in every experiment we did we had some budget constraints given by the time of the execution of the experiment. For a more accurate analysis of the game could be interesting using also some others termination conditions. In one other terminating condition that we have tried we used the upper confidence bounds: the upper confidence bound of a strategy $\boldsymbol{\pi}$ is given by $\mu(\boldsymbol{\pi}) + \sigma(\boldsymbol{\pi})$, thus, every time an epoch of the algorithm terminates we compute the upper bound of the estimated second maxmin strategy (the estimated maxmin of the strategy set without the best one) and we compute its upper confidence bound. If the estimated mean of the best strategy is higher than the upper confidence bound of the second best strategy, the algorithm stops.

Chapter 5

Experimental Analysis

In this chapter, we prove the performances of our algorithm in different scenarios. First, we apply it to a smaller game than ship-missile one: *hit-the-spitfire*. This allows us to assess the performance of the algorithm, a thing that would not be possible in the boat-missile game. Using *hit-the-spitfire*, we also change some parameters of the algorithm in order to understand its behavior. Then, we move our analysis to the boat-missile game in order to see if it can find some good strategies even in more difficult cases.

5.1 Analysis on hit-the-spitfire game

In this section, we describe hit the spitfire game, highlighting the difference and the analogies to boat-missile one. Then we will explain what are the strategies to assess the performances of the algorithm and at then we present our results.

5.1.1 Hit the spitfire game

Hit-the-spitfire game is, analogously to the ship-missile game, a two-player-zero-sum game with infinite strategy spaces which models the scenario of a missile launched against an aircraft that can deploy a flare in order to make the missile miss. That was also used by [31] to do some experiments. As can be seen in Figure 5.1 the initial conditions to be set are:

- l: the length of the plane;
- v_d: the speed of the plane;
- v_a: the speed of the missile;
- $h_\perp$: the distance between the aircraft and the missile;

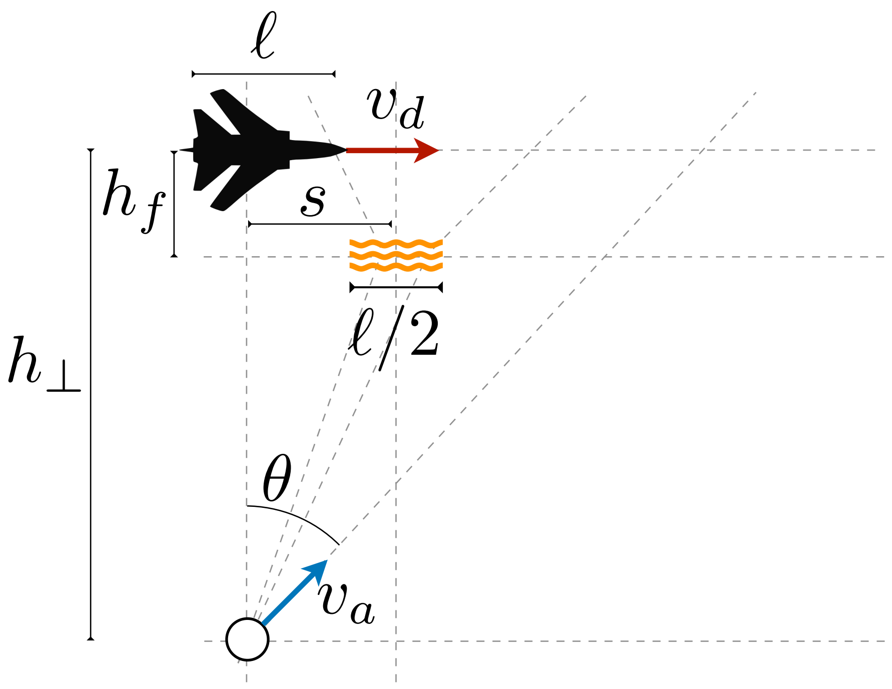


Figure 5.1: Scheme of hit-the-spitfire game.

- h_f: the distance between the plane and the flare deployed.

The missile, represented by the first player, can determine the angle $\theta \in [0, 1]$ of attack while the aircraft chooses the position $s \in [0, s_{max}]$ where to deploy its flare. If the missile hits the plane it causes a damage $d \in \mathbb{R}^+$, based on where it hits the aircraft, while if it hits the flare it has a probability of being deflected away. This probability is large when h_f is larger.
The most important difference between the ship-missile game and *hit-the-spitfire* is the time used to a simulation to finish: being *hit-the-spitfire* a much simpler game, without complex physical phenomena involved, each simulation takes some fraction of second to complete. Another important difference is the symmetry of the set of possible strategies in *hit-the-spitfire*: both first player and second player strategies are composed of only one possible action. In the ship-missile game the first player can choose only one action and the second player, after all the simplification we have done, seven. The last important aspect is that the optimal strategy can be apriori known since the simulations are so fast and the grid not so large. This property allows us to know the maxmin strategy and it will be useful when applying the developed algorithms.
Other two differences between the two games, which are purely formal,

are the implementation platform (ship-missile is in MATLAB while *hit-the-spitfire* is implemented in Python) and the fact that boat-missile is a minmax problem while *hit-the-spitfire* is maxmin.
The above mentioned differences make *hit-the-spitfire* a good way to evaluate our algorithm in easier settings, without all the complications described in the previous chapters, in order to tune it before running it in the real scenario represented by the ship-missile game.

5.1.2 Experimental results

First, we describe how we are set up our experimental environment. We run all our simulations using the same initial conditions (l = 15m, v_d = 120 m/s, v_a=500m/s, $h_\perp$=100m, h_f=10m) in order to have consistency with the measurement of the utility function and we repeat every experiment one hundred times, taking the means of the results at the end.
We start trying to obtain a qualitative description of the behavior of the algorithm visualizing the various utility values obtained at the end of each epoch, we do this by comparing the various probability attributions we have defined with all the other parameters of the algorithm fixed. After that, we vary one parameter each time to understand how the metrics of the algorithm change.
Every time we compare our algorithm with two of its degenerate cases. The first case is the one in which the algorithm chooses completely at random the strategies to sample without following any of the previously described functions, using the cloud analogy we can think at that as a cloud that moves randomly in the space. The second comparison we do is with the version of the algorithm in which the cloud does not move at all, which is exactly GP-SE algorithm described in [31].

Qualitative behaviour

In Figure 5.2 we plotted a run of GP-SE with a budget of 7000 simulations and a grid of 20 $\times$ 20. In order to build this plot we stopped the algorithm every time it finishes a round and we plotted in orange the estimated utility of the estimated maxmin strategy at that point taken from vector $\mu(\boldsymbol{\pi})$, then we plotted also the true utility of the estimated maxmin strategy in blue. The dotted red line is the utility of the actual maxmin strategy.
The first thing we notice looking at the plot is that GP-SE, after an initial exploration phase where its solution is far from the optimal, converges at the optimal solution or some solution really near the optimal. Another thing that is interesting to notice is the fact that in the early rounds of

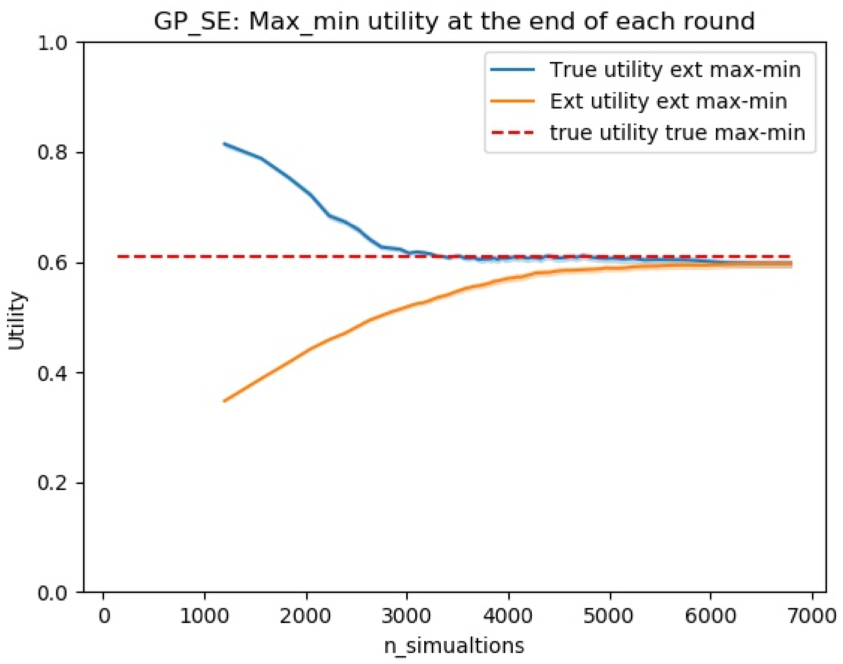


Figure 5.2: Execution of GP-SE with 7000 *budget over a* 20×20 *grid.*

the execution there is a big difference between the estimated values of the strategies and their respective true values, this is due by the fact that the GP does not have enough samples to update itself enough in order to estimate accurately the utilities of the strategies.
Note the fact that GP-SE is designed to find the optimal solution at the end of its budget, so the intermediate results we have plotted here are not significant, but we still use them as an indicator of its behavior. Another thing that has to be said is the fact that, even the true utility of a strategy is equal to the utility of the true maxmin strategy is not a guarantee that it is the maxmin strategy itself, it could easily be another suboptimal strategy with the same utility.
We have done the same plot with the same settings using the new strategy generator algorithm with optisoftmax normalization and a cloud of 16 strategies in Figure 5.3, where, instead of taking the utility values at the end of each round of GP-SE, we take them at the termination of the exploration phase, so at the end of the execution of GP-SE over the *small* set of strategies.
Here, we can see that the orange line is a good estimator of the true value of the estimated maxmin strategy and becomes better as the learning process

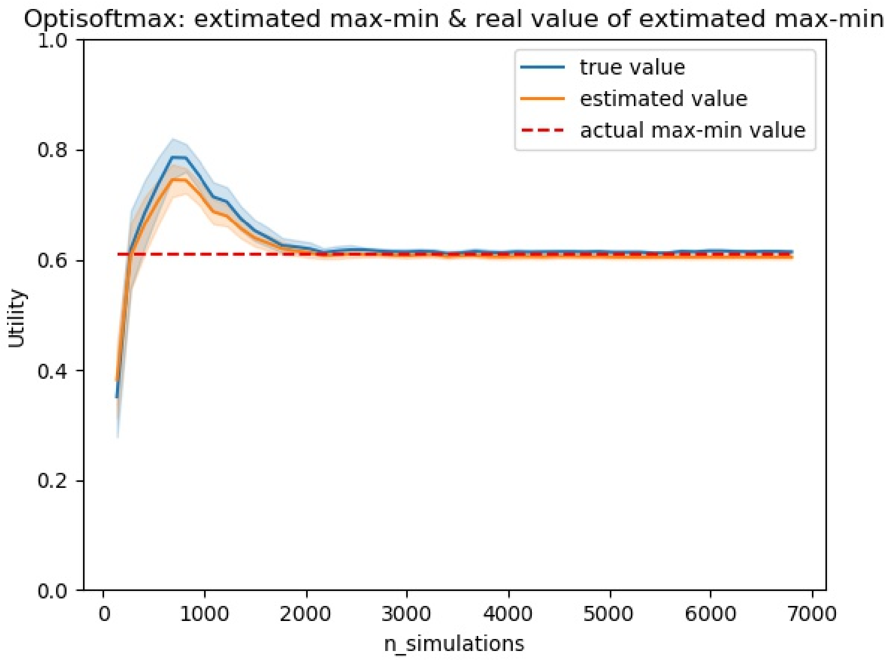


Figure 5.3: Strategy generator algorithm with optisoftmax normalization.

progresses. This is due to the fact that, being the set of strategies to be explored very small, it can estimate them better. We can note also that the variance related to the estimations at the beginning of the execution of the algorithm is bigger than the one observed in the original GP-SE algorithm. Intuitively, this could be caused by the fact that the only source of noise in the GP-SE is the one in the samples of the utility function, while in strategy generator algorithm we add an additional source of noise which is the random choice of the strategies to be sampled.

In Figure 5.4, we plotted in orange the estimated utility of the best strategy belonging to the *small* strategy set at the end of each epoch of GP-SE while the blue line is its true utility. The difference with the plot before is that this figure represents the maxmin strategies that belong to the sampled set, while before the maxmin equilibrium was searched within the set of all the possible strategies.
As the number of simulations grows, the two lines become closer and they get near to the utility of the true maxmin strategy. At the end of the 7000 simulations the values plotted does not seem to converge in the optimal value, but they keep fluctuating. This justifies our choice described in

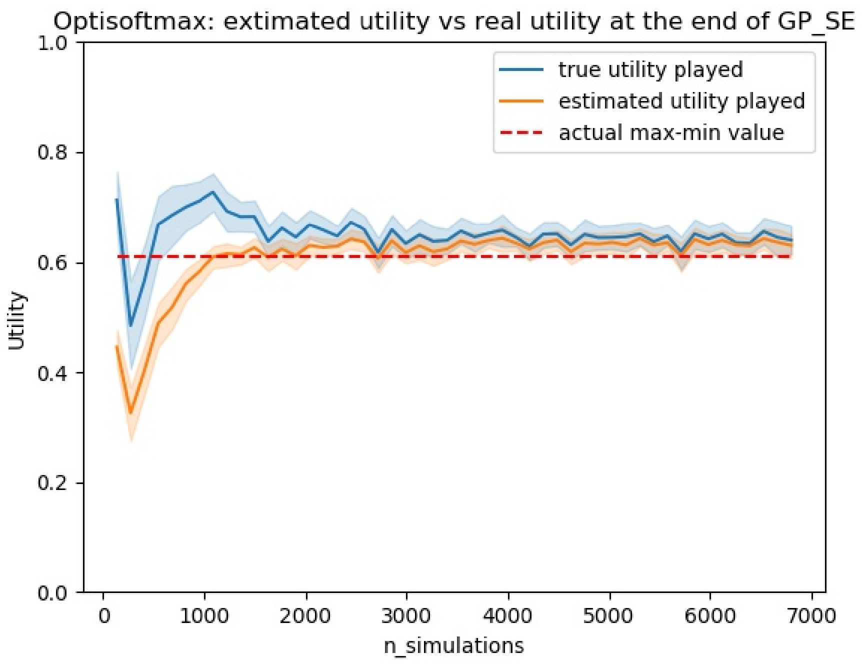


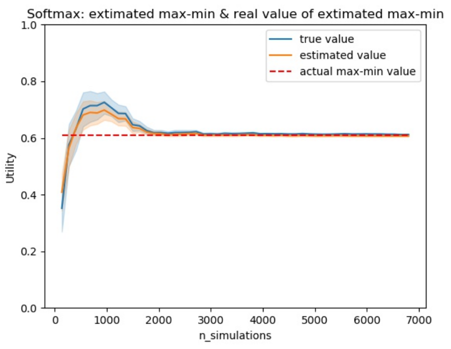
Figure 5.4: Utilities of the best strategy played from strategy generator algorithm with optisoftmax normalization.

Algorithm 3 to take the best strategy over the set of all strategies, not only the ones we have sampled in that epoch with GP-SE.

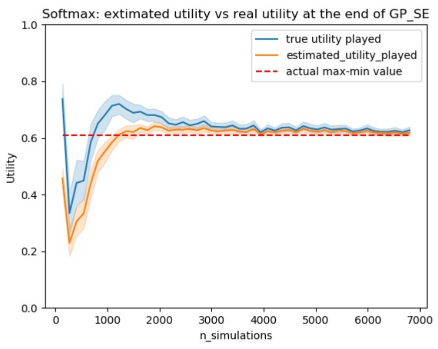
In Figures 5.5a, 5.5b, 5.5c and 5.5d we repeated the same plots as the ones described before using the softmax normalization with temperature parameter $\tau = 0.1$ and the linear one. Figures 5.5e, 5.5f are made using softmax normalization but, instead of the standard one, here we have normalized not the estimated means $\mu(\boldsymbol{\pi})$, but the lower bounds that are defined as $l(\boldsymbol{\pi}) = \mu(\boldsymbol{\pi}) - \sigma(\boldsymbol{\pi})$: in this way we are looking to have more exploration in the algorithm.

The considerations made before for the optisofmax normalization still hold and, looking at the plots, we imagine that a non-linear normalization can bring to better results than the linear one. Indeed, we can see how the algorithm converges faster to optimal or near-optimal solutions than its linear counterpart and, having with a large number of samples in that area, it estimates better the true utility of its strategy. This happens because, being more precise in selecting strategies near the optimal one, it spends more simulation in sampling that region.

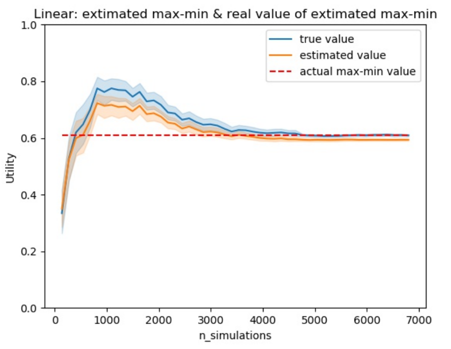
We repeated the same plots also in the case in which our cloud of



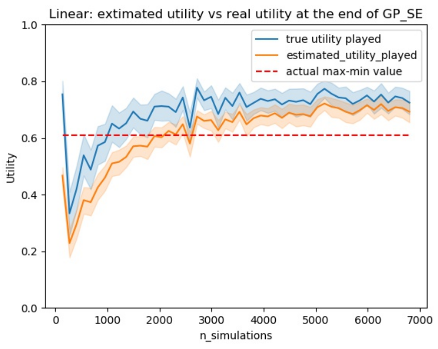
(a) Softmax overall maximin



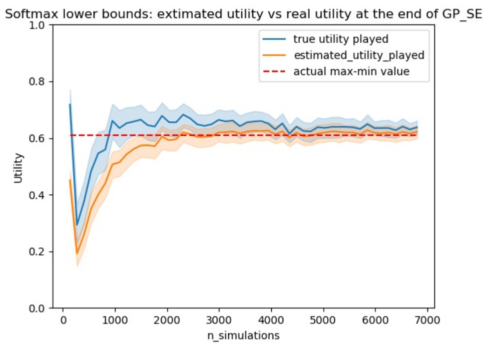
(b) Softmax played maxmin



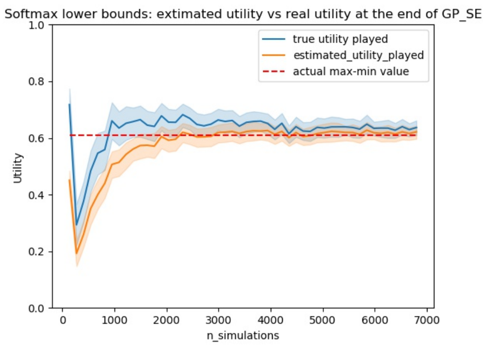
(c) Linear overall maxmi



(d) Linear played maxmin



(e) Softmax lower bounds overall maxmin



(f) Softmax lower bounds played maxmin

Figure 5.5: Comparison of different exploration strategies.

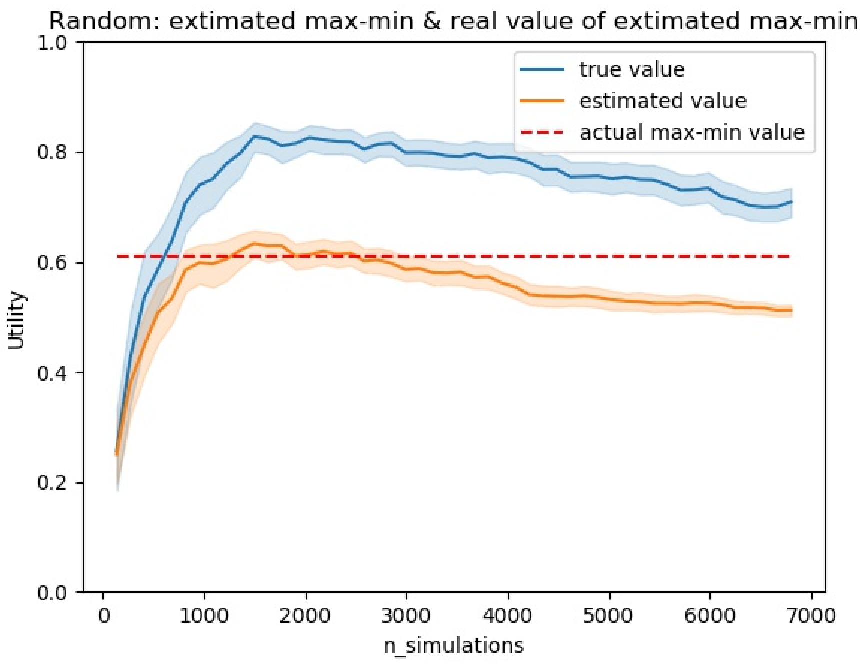


Figure 5.6: Utilities of the estimated maxmin equilibrium from strategy generator algorithm with random choice.

points moves in a random way to have a comparison with the original implementation of our algorithm. Those plots are in Figures 5.6 and 5.7. From the first one of those figures we can see how, using the same amount of samples that allows the other possible implementations to converge, the random sample approach performs worst compared to the others.

The first thing to notice is that it does not converge: even after 7000 simulations it is still far from the optimal solution. Even if it seems to get near to it with the growth of the number of simulations, it is still slower than the other approaches described before. Then, we can notice how the estimated value of the estimated maxmin strategy is far from its real value. This probably happens because in this case the simulations are no more concentrated in the regions of the search space near to the maxmin, but are distributed homogeneously. The last difference we have is the fact that the confidence interval of the utilities of the best strategies does not become narrower with the simulation as happened in the previous case.

From all the plots we showed in this section we can have an idea of how our algorithms work: in the first phase the algorithms do not have still enough information about the game so they perform an exploration phase in which

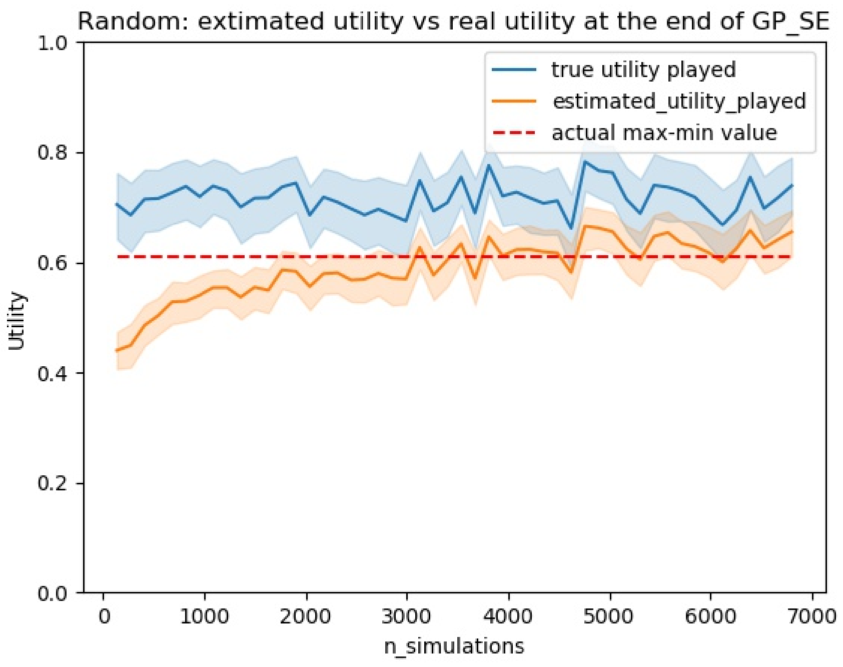


Figure 5.7: Utilities of the best strategy played by strategy generation algorithm with random cloud move.

the utilities usually start below the optimal one, then they quickly grow above and then they start to descend to converge in the optimal solution. We will discuss in the next section how fast they converge to the optimal solution.

Performance measurement

We are now assessing how our algorithm performs in various situations with different parameters. As before, we run the experiments one hundred times and we measure the percentage of optimal solutions found after a given number of simulations. We use GP-SE as a baseline to evaluate the algorithm but, since GP-SE distributes the simulations in a different way depending on its budget, and we run it several times with different budgets (e.g., 1000, 2000).
In the first plot we are measuring the percentage of optimal strategies found. We are using a grid of 20×20 strategies, a cloud of 16 strategies and a total budget per-epoch of 136 simulations. We run GP-SE so that it draws a single sample for each strategy during each round before the elimination of a strategy occurs. The only thing that we vary here is how we assign the probabilities to the strategies when we build the cloud. The results are depicted in Figure 5.8. GP-SE using a small number of samples starts to perform better than all the implementation of the strategy generator algorithm, but it seems to reach a plateau before the strategy generation algorithm: after only 2000 samples, softmax and optisoftmax start to perform better. At the end of the 7000 simulations the algorithm using the softmax normalization seems to be the best one, and it provides better performances than GP-SE. The other two algorithms with nonlinear normalization have similar performances, but still perform a little worse than the softmax version. Conversely, the linear normalization does not seem to work well enough in comparison to pure GP-SE and also how the algorithm with the random distribution of samples provides results far worse than the others.
Table 5.1 reports the same results depicted in Figure 5.8 in a tabular form. Looking at the table, we notice that the difference between GP-SE and strategy generation algorithm with softmax normalization after 7000 simulations is 14%, while the moment when optisoftmax normalization performs better than GP-SE is at 4000 simulations with an improvement over the GP-SE algorithm of 17%. Optisoftmax and softmax-lower-bound normalization at the end of 7000 simulations provide similar performance to the softmax normalization with a less effective discovert of the optimum

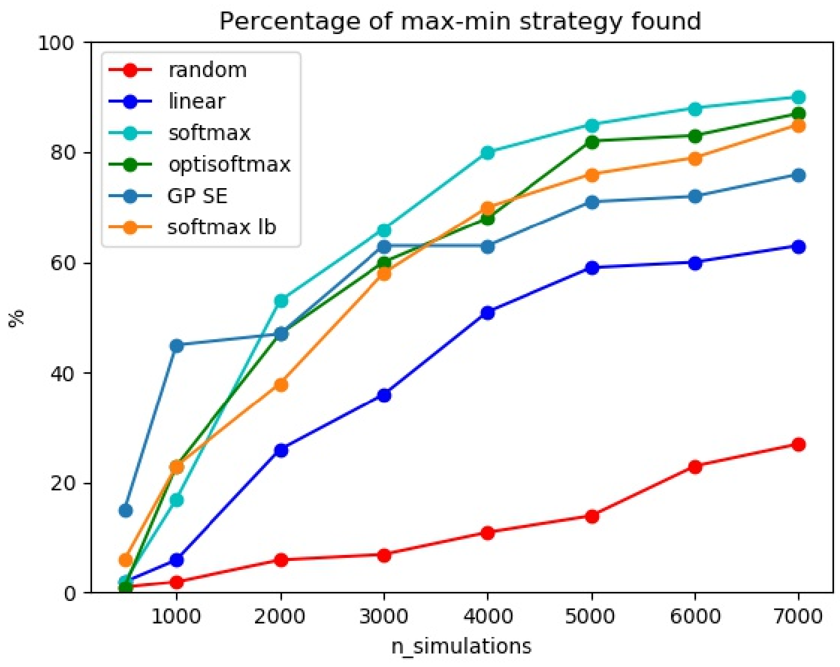


Figure 5.8: Percentage of maxmin strategy found by strategy generator algorithm varying the normalization.

Table 5.1: Percentage of maxmin strategy after n simulations

Algo	500	1000	2000	3000	4000	5000	6000	7000
GP-SE	15	45	47	63	63	71	72	76
Optisoftmax	1	23	47	60	68	82	83	87
Softmax	2	17	53	66	80	85	88	90
Linear	2	6	26	36	51	59	60	63
Random	1	2	6	7	11	14	23	27
Softmax lb	6	23	38	58	70	76	79	85

Table 5.2: Percentage of maxmin strategy after n simulations with different budgets

Budget	500	1000	2000	3000	4000	5000	6000	7000
budget 25	11	28	59	58	68	73	83	84
budget 50	6	37	58	73	79	83	90	90
budget 75	6	37	58	72	79	83	86	90
budget 100	1	28	78	58	86	72	100	100
budget 125	1	14	56	58	72	100	100	100
budget 150	2	4	51	72	83	87	92	94

of 3% and 5%, respectively.

After showing that our algorithm can perform better than GP-SE in the analysed settings, we analyse the impact of changing some parameters of the algorithm. We start from the softmax normalization with $\tau = 0.1$, that is the best algorithm we have found in the previous setting (20×20 strategies, a cloud of 4×4 and a budget of 136 simulations) and we compare our results with pure GP-SE algorithm. First of all, we try different budgets for the GP-SE part of the strategy generation algorithm and we use the Equation 2.14 in order to distribute better the simulations in the various round of GP-SE.
The results of this experiments are reported in Figure 5.9 and in Table 5.2.

From the plot and the table we can see some interesting aspects of the application: first of all we reach a 100% of exact results in two cases, obtaining a 24% better performance than pure GP-SE at the end of 7000 simulations. Moreover, we notice that, observing the first 1000 simulations, seems that could be better to run the algorithm using a small budget per epoch but, when the number of the simulations grows, it seems that is better

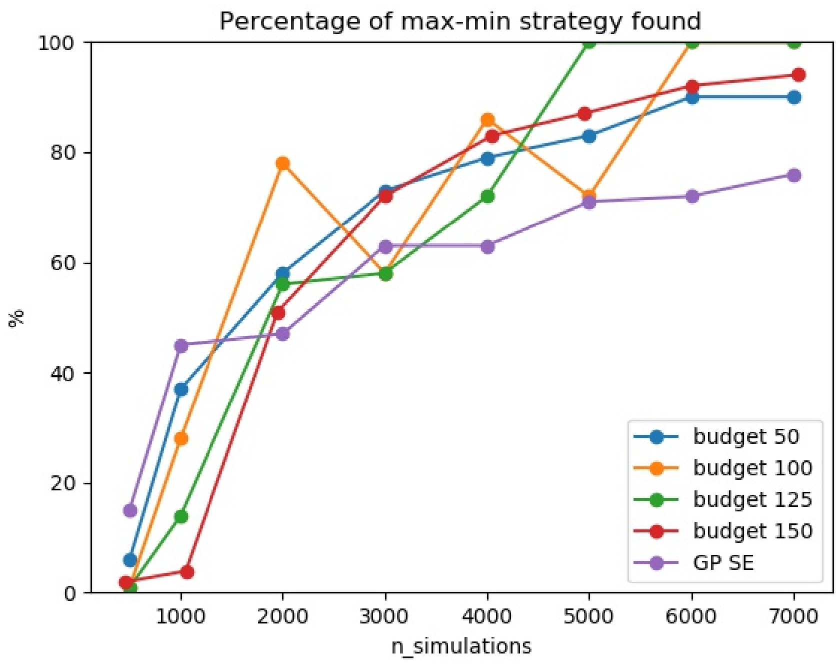


Figure 5.9: Percentage of maxmin strategy found by strategy generator algorithm with softmax normalization varying the budget.

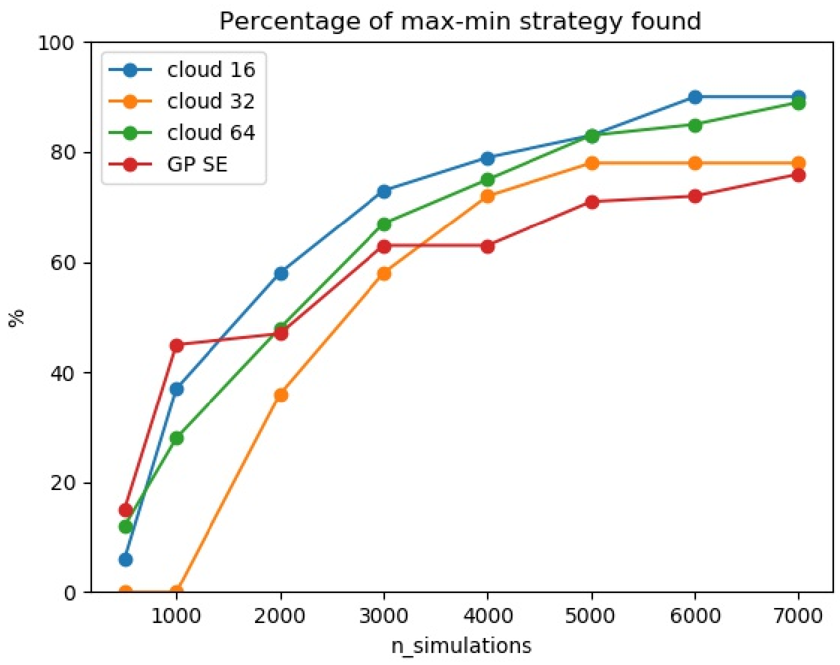


Figure 5.10: Percentage of maxmin strategy found by strategy generator algorithm with softmax normalization varying the size of the cloud.

to have an higher budget. We are not able to explain why the run with a budget of 100 simulations sometimes decreases the percentage of maxmin strategies found with a higher number of simulations, but it still reaches 100% accuracy at the end of the experiment.
The last thing we want to tune is the shape of the cloud. We keep the same normalization (softmax with $\tau = 0.1$) and we set a budget which is the nearest number divisible by 50 of three times the number of strategies in the cloud. So, since we use clouds of size 16, 36, and 64, we have a budget of 50, 100, 200, respectively. We use again GP-SE as a benchmark in our experiments. The results of the experiments are depicted in Figure 5.10 and Table 5.3.
From the plot and the table we can see that probably in this settings the choice of a small cloud is the best one, even if there are no significant differences in terms of performances from each other.

After seeing how the performance of our algorithm changes when we change its parameters, we conclude that in these settings our algorithm performs better than the GP-SE one, which is the current state-of-the-art algorithm,

Table 5.3: Percentage of maxmin strategies after n simulations with cloud size.

Cloud size	500	1000	2000	3000	4000	5000	6000	7000
cloud 16	6	37	58	73	79	83	90	90
cloud 36	0	0	36	58	72	78	78	78
cloud 64	12	28	48	67	75	83	85	89

in most cases. Generally, GP-SE has a faster growth in the first 1000 simulations but after that point in the majority of tests performed the new algorithm does better. We were also able to show that in some cases the algorithm finds the exact solution 100% of the times, improving by 24% the performances of GP-SE. The last aspect that is worth noticing is that our algorithm seems quite robust to the changing of parameters: it is able to find the best solution with high confidence with an extended set of different parameters.

5.2 Ship-missile game

After analyzing our algorithm using the game *Hit-the-spitfire* we are now ready to use it in the ship-missile game to try to find the best strategies that allow the boat to escape the missile. We use as utility function the minimum distance between the ship and the missile during the simulations. In this case, we are not able to run an accurate analysis on the experiments we do for two reasons: the first one is that is not possible to run the same experiment 100 time as we did before because it would take too much time, and the second reason is that we do not know what is the best strategy of the game so we do not know if our algorithm reaches the maxmin strategy. We used a grid of 4500 strategies and we run our algorithm for a fixed period of time. Note that in the previous paragraph we used 7000 simulations to explore a grid of 400 strategies, so this time we would need a lot more simulations since the grid is bigger. For time and hardware reasons we were not able to make more than a small number of simulations compared with the size of the grid, so our experiment will have only a demonstrative purpose. We expect this algorithm to be run on a more powerful machine than the one we have for more time than we do and, considering that Marina Militare Italiana provided us the simulator, this for us is a fair assumption. In Figure 5.11 we used our algorithm with a linear normalization and a budget of 136 strategies for each round of GP-SE (we make GP-SE doing a round and then an elimination) and a cloud of 16 strategies. We plotted the

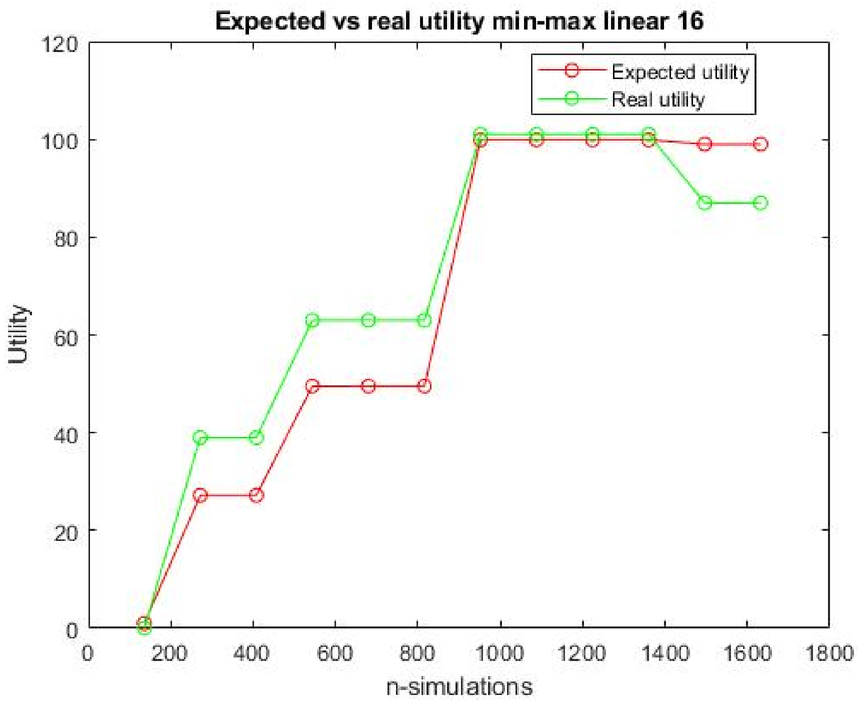


Figure 5.11: Expected utility of estimated best strategy and real utility of the estimated best strategy.

estimated utility and the real utility of the estimated equilibrium strategy. In this case, since we do not have the real utility of every strategy as we had in the *hit-the-spitfire* game, it was necessary to make other 20 simulations for each point it the plot in order to estimate the true utility of the strategy.

Even with a very low number of simulations compared with the size of the grid we can still note some interesting features of the plot. First of all we note how the true value and the expected value of the utility seem correlated: when the expected value grows the true value behaves similarly. This means that from the early stages of the algorithm it has a good estimation of the values of the utility in the regions that it explores. Another important aspect is that the algorithm, even with a low number of simulations, finds strategies that are actually good for our work, this means strategies that allows the warship to evade the threat of the missile. So, even if the strategies indicated by the algorithm probably are still not minmax profiles, the algorithm finds a strategy for each angle of attack that allows the warship to escape.

We have repeated the exact same experiment with the same grid, the same budget, but a softmax normalization and the results are presented in Figure 5.12.

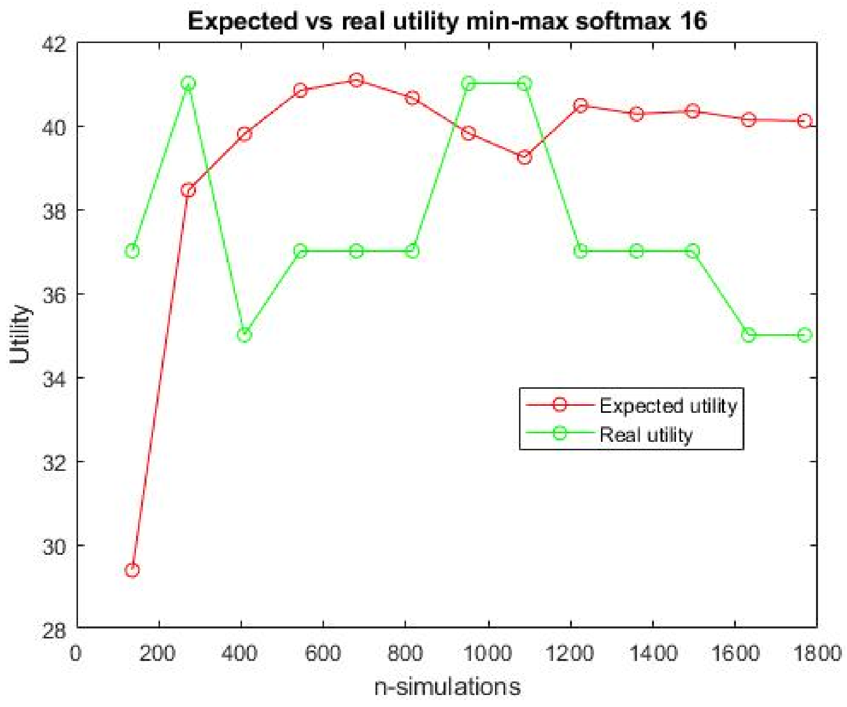


Figure 5.12: Expected utility of estimated best strategy and real utility of the estimated best strategy with softmax normalization.

The first thing we notice is that we are still finding strategies that allow the boat to escape the missile, but this time the minimum distance between the boat and the missile is lower than the previous case with linear normalization. We can also observe that the curve of real and estimated utility, despite being nearer than the previous case, does not seem to be correlated anymore. This is due to the fact that with so little simulations in comparison to the size of the grid, the algorithm at this phase is still exploring the space.
The last plot we show is Figure 5.13, in which we plotted the real utility of the estimated maxmin strategies.
Observing the plot we can say that we are able to find strategies that allows the boat to escape the missile in every settings of the algorithm, but obviously, with such a low number of simulations, the algorithms are not able to converge near the maxmin point.
Note that, as explained before, we have not done enough run of the algorithm in every single setting, so the results we have had for the boat-missile game have not any statistical validity, we have just to take them as an example of the behaviour of our algorithm in such difficult settings.

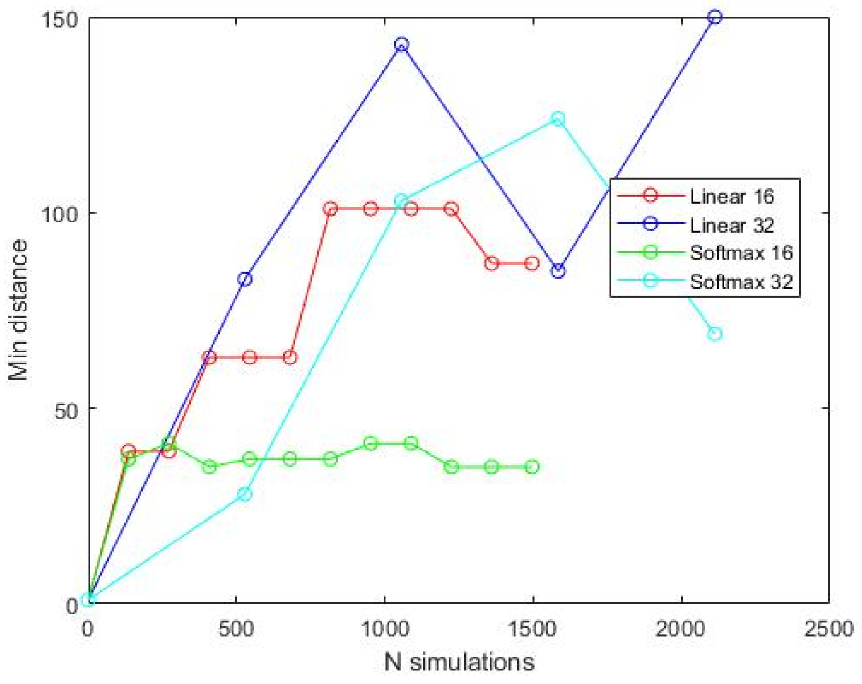


Figure 5.13: The true utility of the estimated best strategy with different normalizations and different shapes of the cloud.

Chapter 6

Conclusions and future works

In this chapter we discuss the results obtained in the experimental analysis we did in the chapter before. Then, we give some suggestion of further works that could be done to improve this method.

6.1 Conclusions

This work starts with a software that simulates a situation in which a warship is target by a infra-red guided missile. First of all, we spent some efforts in formulating the problem as a two-player zero-sum game, then, analyzing the problem we understood the complications related to the size of the game and the long simulation time, so we made the problem easier to tackle using some simplifications, such as giving a structure to the courtain of flares emitted by the ship and making start the flares launch and the evasive maneuver at the same time.
Once we have made such simplifications, we studied in literature what could help us to solve the problem and we found that the approach presented by Marchesi et al. in [31] was well suited for our problem. The algorithm presented in that paper uses the Gaussian Process assumption over the utility function to find the minmax solution using few queries to the simulator as possible. We to build on top to this solution to design a new algorithm suited for the boat-missile game.
The basic idea of our new algorithm is to do the research in smaller subspaces of the strategy space and it is based on the fact we can still use the Gaussian Process assumption to create those subspaces in a probabilistic way, sampling strategies from regions in the space that GP estimates to be good. This can be seen as a cloud of strategies that at the beginning is sparse in the entire space, but as the number of simulations grows it moves

and concentrates in the part of the space with higher probability to contain the maxmin strategy.
We combined this strategy generation method with a phase in which we are explore that strategies to extract information and update our Gaussian Process. We decided to use GP-SE in this step because it has the possibility to be parallelized.
Once we have designed our algorithm we have tested it in a simplified scenario: *hit-the-spitfire* game. Using that game we were able to show how the algorithm, after a first phase of exploration where it is far from the maxmin strategy, at the end converges toward it, and we were also able to show how, increasing the number of simulations, increases also the accuracy of the estimation of the utility of the strategies.
After that we measured experimentally the performance varying some parameters of the algorithm, such as the normalization method, the size of the cloud, and the budget per epoch. We were able to show that the our algorithm performs better that a state-of-the-art algorithm (GP-SE) in the majority of the cases, reaching an improvement of about 30% of more maxmin strategies found in a specific setting. We were also able to show that our algorithm in some settings is able to reach a 100% accuracy in found maxmin strategy after only 5000 simulations.
After testing our algorithm, we applied it to ship-missile game. We were not able to do a complete experimental analysis as we did before but we were able to show that, even with a small number of samples that does not allow them to converge, the algorithm is able to find some strategies that allows the ship to evade the missile.

6.2 Further works

The new strategy generation algorithm, despite having good performances, can be tested and extended in various way. The first and more obvious way to do so are changing the implementation of the algorithm and its testing: new non linear normalization techniques could be implemented and tested beyond the ones which are presented here and would be interesting to see how the algorithm can improve through them. Some others sampling techniques that could be used are importance-sampling [20] and cross-entropy [40]. Another interesting study would be changing the factorizations of the search space, for example changing the shape of the flares' courtain in order to see how the utility changes.
The algorithm we presented is meant to be part of a pipeline that allows the warship to escape the missile in a real word situation. With the

new algorithm we developed we are able to find the best response to a threat given some initial conditions, but what happens if that conditions changes? Obviously in a real world situation we cannot expect that the initial conditions are identical to the ones simulated and we cannot do any simulations since the missile is pointing to the boat. For this reason it is needed a software that uses some interpolation techniques in order to use the information gained in the simulation phase to find a good response in a small amount of time. Some work about that was made by [27] but this methods need to be adapted to our new algorithm.
Ideally the best way to solve boat-missile game is to find a function that, given the inputs as initial conditions, strategy of the missile, and strategy of the boat, is able to approximate the utility function with good precision. If we have that, it would be possible to perform just a scan on the function to find a way to evade the missile. Probably it is not possible to approximate this function because it is too complex but it would be nice to understand if it is so or not.
Another open question, that has received big attention by the expert of Marina Militare is what to do when more than one missile is targeting the boat. In the simulator there is not the possibility to add a missile to the simulations so how to model the scenario and how to find the best strategy to this situation are questions of big interest.
A more theoretical question, not related to ship-missile game is what happens when we have so much possible strategies that is impossible even to enumerate them? How we can generate the probability from which we sample the strategies?

Bibliography

[1] Francesco Amigoni et al. “Moving game theoretical patrolling strategies from theory to practice: An USARSim simulation”. In: (2010), pp. 426–431.

[2] Bo An, Nicola Gatti, and Victor Lesser. “Bilateral bargaining with one-sided uncertain reserve prices”. In: *Autonomous agents and multi-agent systems* 26.3 (2013), pp. 420–455.

[3] “An Introduction to Evolutionary Games”. In: *Game Theory: A Multi-Leveled Approach.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 111–120.

[4] Jean-Yves Audibert, Sebastien Bubeck, and Remi Munos. “Best Arm Identification in Multi-Armed Bandits”. In: Nov. 2010, pp. 41–53.

[5] Nicola Basilico, Giuseppe De Nittis, and Nicola Gatti. “A security game combining patrolling and alarm-triggered responses under spatial and detection uncertainties.” In: *30th AAAI Conference on Artificial Intelligence, AAAI 2016;* AAAI press. 2016, pp. 397–403.

[6] Lorenzo Bisi et al. “Online Follower’s Behaviour Identification in Leadership Games”. In: *16th European Conference on Multi-Agent Systems.* 2018, pp. 1–15.

[7] Lorenzo Bisi et al. “Regret Minimization Algorithms for the Followers Behaviour Identification in Leadership Games.” In: *UAI.* 2017, pp. 1–10.

[8] Matteo Castiglioni, Alberto Marchesi, and Nicola Gatti. “Be a Leader or Become a Follower: The Strategy to Commit to with Multiple Leaders.” In: *IJCAI.* 2019, pp. 123–129.

[9] Matteo Castiglioni et al. “Leadership in singleton congestion games: What is hard and what is easy”. In: *Artificial Intelligence* 277 (2019), p. 103177.

[10] Andrea Celli et al. "Learning to correlate in multi-player general-sum sequential games". In: *Advances in Neural Information Processing Systems*. 2019, pp. 13076–13086.

[11] Sofia Ceppi et al. "Local search techniques for computing equilibria in two-player general-sum strategic-form games." In: *AAMAS*. 2010, pp. 1469–1470.

[12] Giuseppe De Nittis, Alberto Marchesi, and Nicola Gatti. "Computing the strategy to commit to in polymatrix games". In: *AAAI*. 2018.

[13] Giuseppe De Nittis and Francesco Trovò. "Machine learning techniques for stackelberg security games: a survey". In: *arXiv preprint arXiv:1609.09341* (2016).

[14] Aurelien Garivier, Emilie Kaufmann, and Wouter Koolen. *Maximin Action Identification: A New Bandit Framework for Games*. 2016. arXiv: 1602.04676.

[15] Nicola Gatti, Fabio Panozzo, and Marcello Restelli. "Efficient evolutionary dynamics with extensive-form games". In: *arXiv preprint arXiv:1304.1456* (2013).

[16] Nicola Gatti and Marcello Restelli. "Equilibrium approximation in simulation-based extensive-form games". In: *10th International Conference on Autonomous Agents and Multiagent Systems 2011, AAMAS 2011*. Vol. 1. Jan. 2011, pp. 199–206.

[17] Nicola Gatti and Marcello Restelli. "Sequence–form and evolutionary dynamics: realization equivalence to agent form and logit dynamics". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. 2016, pp. 509–515.

[18] Nicola Gatti et al. "Combining local search techniques and path following for bimatrix games". In: *arXiv preprint arXiv:1210.4858* (2012).

[19] Nicola Gatti et al. "Truthful learning mechanisms for multi-slot sponsored search auctions with externalities". In: *Artificial Intelligence* 227 (2015), pp. 93–139.

[20] Sebastian Geyer, Iason Papaioannou, and Daniel Straub. "Cross entropy-based importance sampling using Gaussian densities revisited". In: *Structural Safety* 76 (2019), pp. 15–27.

[21] Samuel Hamilton et al. "The Role of Game Theory in Information Warfare". In: (Jan. 2002).

[22] Thomas Hamilton and Richard Mesic. *A Simple Game-Theoretic Approach to Suppression of Enemy Defenses and Other Time Critical Target Analyses*. Santa Monica, CA: RAND Corporation, 2004.

[23] Yu-Lin He et al. "Determining the optimal temperature parameter for Softmax function in reinforcement learning". In: *Applied Soft Computing* 70 (2018), pp. 80–85.

[24] Raymond R. Hill, Lance E. Champagne, and Joseph C. Price. "Using Agent-based Simulation and Game Theory to Examine the WWII Bay of Biscay U-boat Campaign". In: *The Journal of Defense Modeling and Simulation* 1.2 (2004), pp. 99–109.

[25] Andreas Krause, Ajit Singh, and Carlos Guestrin. "Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies". In: *Journal of Machine Learning Research* 9 (Feb. 2008), pp. 235–284.

[26] Neil Lawrence. "Gaussian Process Latent Variable Models for Visualisation of High Dimensional Data". In: *Advances in neural information processing systems* 16 (Mar. 2004).

[27] Andrea Longobardi. "Algorithm for searching ship countermeasures to missiles attack". MA thesis. Politecnico di Milano, 2019.

[28] Fernando Lopes and Helder Coelho. *Negotiation and argumentation in multi-agent systems: Fundamentals, theories, systems and applications*. Bentham Science Publishers, 2014.

[29] Alberto Marchesi, Stefano Coniglio, and Nicola Gatti. "Leadership in singleton congestion games". In: International Joint Conferences on Artificial Intelligence. 2018.

[30] Alberto Marchesi, Francesco Trovò, and Nicola Gatti. "Learning Maximin Strategies in Simulation-Based Games with Infinite Strategy Spaces". In: *Smooth Games Optimization and Machine Learning Workshop (SGOandML)*. 2019.

[31] Alberto Marchesi, Francesco Trovò, and Nicola Gatti. "Learning Probably Approximately Correct Maximin Strategies in Simulation-Based Games with Infinite Strategy Spaces". In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 2020, pp. 834–842.

[32] D. Neveu et al. “Pursuit Games with Costly Information: Application to the ASW Helicopter Versus Submarine Game”. In: *New Trends in Dynamic Games and Applications.* Ed. by Geert Jan Olsder. Boston, MA: Birkhäuser Boston, 1995, pp. 247–257. ISBN: 978-1-4612-4274-1.

[33] Giuseppe De Nittis and Nicola Gatti. “Facing Multiple Attacks in Adversarial Patrolling Games with Alarmed Targets”. In: *CoRR* abs/1806.07111 (2018).

[34] Alessandro Nuara et al. “Driving exploration by maximum distribution in gaussian process bandits”. In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems.* 2020, pp. 948–956.

[35] Fabio Panozzo, Nicola Gatti, and Marcello Restelli. “Evolutionary dynamics of Q-learning over the sequence form”. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence.* 2014, pp. 2034–2040.

[36] Chiwoo Park and Daniel Apley. *Patchwork Kriging for Large-scale Gaussian Process Regression.* 2018. eprint: `1701.06655`.

[37] Jirka Poropudas and Kai Virtanen. “Game-Theoretic Validation and Analysis of Air Combat Simulation Models”. In: *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 40 (Oct. 2010), pp. 1057–1070.

[38] Carl Edward Rasmussen. “Gaussian Processes in Machine Learning”. In: *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures.* Ed. by Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 63–71.

[39] Jiang Rong, Tao Qin, and Bo An. “Competitive Bridge Bidding with Deep Neural Networks”. In: *CoRR* abs/1903.00900 (2019). arXiv: `1903.00900`. URL: `http://arxiv.org/abs/1903.00900`.

[40] Reuven Rubinstein and William Davidson. *The Cross-Entropy Method for Combinatorial and Continuous Optimization.* 1999.

[41] Servizi e Sistemi s.r.l. *Rapporto Tecnico-Attività di modellazione e progettazione del Dimostratore Tecnologico.* 2010.

[42] Yevgeniy Vorobeychik and Michael Wellman. “Strategic Analysis with Simulation-based Games.” In: Dec. 2009, pp. 359–372. DOI: `10.1109/WSC.2009.5429347`.

[43] Yevgeniy Vorobeychik, Michael P. Wellman, and Satinder Singh. "Learning Payoff Functions in Infinite Games". In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence*. IJCAI'05. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005, pp. 977–982.

[44] B. Wiedenbeck, Fengjun Yang, and Michael P. Wellman. "A Regression Approach for Modeling Games With Many Symmetric Players". In: *AAAI*. 2018.

[45] Wikipedia. *Flare (countermeasure) — Wikipedia, The Free Encyclopedia*. `http://en.wikipedia.org/w/index.php?title=Flare\%20(countermeasure)&oldid=962268121`. 2020.

[46] Zongxin Yao et al. "Mission decision-making method of multi-aircraft cooperatively attacking multi-target based on game theoretic framework". In: *Chinese Journal of Aeronautics* 29.6 (2016), pp. 1685–1694.