Technische Universität München

Department of Mathematics

Implementation of Fully-Discrete Schemes for the Value Function of Pursuit-Evasion Games with State Constraints

Bachelor's Thesis by Valentin Sawadski

Supervisor:	Prof. Dr. Folkmar Bornemann
Advisor:	Prof. Dr. Folkmar Bornemann
Submission Date:	

I hereby declare that I have written the Bachelor's Thesis on my own and have used no other than the stated sources and aids.

Place, Date	Signature

Abstract

This thesis covers two player Pursuit-Evasion Games. A time and space discrete approximation scheme for the Value Function of these games has been proposed by Cristiani and Falcone. They proved the convergence of their scheme for the Fully-Discrete Value Function on the continuous Value function and showed several numeric experiments. However they did not disclose their implementation so that up to date no independent verification of their results is known to the author of this thesis. This thesis aims to provide such. It implements the scheme and recreates the original numeric experiments to prove that the scheme in fact does work if the Value function is continuous, even though the stated timings for the duration of calculation could not be verified.

Diese Thesis behandelt zwei Spieler Verfolgungsspiele. Ein zeit- und ortdiskretes Approximationschema für die Wertfunktion dieser Spiele wurde von Cristiani und Falcone vorgeschlagen. Sie bewiesen die Konvergenz von volldiskreten Wertfunktionen hinzu zur stetigen Wertfunktion bei der Anwendung ihres Schemas und führten eine Vielzahl numerischer Experimente vor. Jedoch veröffentlichten Sie ihre Implementierung nicht, so dass bis heute dem Autor der Thesis keine unabhängige Bestätigung ihrer Ergebnisse bekannt ist. Diese Thesis möchte eine solche bieten. Sie implementiert das Schema und stellt die ursprünglichen Experimente nach um zu beweisen, dass das Schema in der Tat funktioniert bei stetigen Wertefunktionen, gleichwohl wenn die behaupteten Berechungszeiten nicht bekräftigt werden konnten.

Contents

List of Figures

List of Algorithms

1 Introduction

This thesis covers two player Pursuit-Evasion Games. This game can be described in simple words with two boys playing catch in the schoolyard. These games have been initially studied by Rufus Isaacs in [3] and have since then been picked up by many e.g. [5] just to name one. And even though it can be explained quickly and in the simple words, its mathematical nature is quite challenging. The Besicovitch solution of Richard Rado's "Lion and Man" game for example showed that the Evader can avoid capture if certain conditions are met. The studies in [3, 5] evaluate continuous games and provide analytic solutions. Although lots of real world applications for the game exists that make them interesting candidates for a time and space discrete numerical solution.

Cristiani and Falcone proposed in [1] a Fully-Discrete method to calculate the Value Function of Pursuit-Evasion Games and conducted several numerical experiments to proof the effectiveness of their scheme. However they did not disclose their implementation, so that up to date no independent proof of the effectiveness of the scheme is known to the author. This thesis takes the theoretical results of [1] and aims to recreate the numerical implementation to verify the original tests and provide an open implementation that can be extended and used for further studies.

To focus on the implementation and verification, Section 2 in this paper does only contain the most important results presented in [1]. Therefore it is necessary for the reader to be familiar with the general concepts of Pursuit-Evasion Games[1] and advisable to be familiar with the results of Cristiani and Falcone. Section 3 then explains the implementation in greater detail and tests its performance in terms of CPU run time and memory consumption. Section 4 compares the results obtained by this implementation with those of Cristiani and Falcone. Finally Section 5 summarizes the results and gives recommendations for further work on the topic and numerical scheme.

[1]Chapters 1 and 2 of [3] contain a quick introduction into the topic.

2 Pursuit-Evasion Games

In this section a short introduction into the Notation and the definition of the Game and the scheme of approximation are given. The influence of the velocities of the Players on the game and approximation will be shown later.

2.1 Notation, Definitions of the Game and Scheme of Approximation

To avoid any confusion a similar notation as in [1] is used. The dynamics of an unconstrained two Player Pursuit-Evasion Game are described by the following differential equation

$$\begin{aligned} y &: \mathbb{R}_{\geq 0} \to \mathbb{R}^{2n} && (2.1) \\ \dot{y}(t) &= f(y(t), a(t), b(t)), \quad t > 0 && (2.2) \\ y(0) &= x_0 && (2.3) \end{aligned}$$

where t is the time parameter and $a(t)$ and $b(t)$ are the strategies of the Pursuer (P) and Evader (E) respectively. These strategies stem from a set of *admissible strategies* that are, for given compact sets $A, B \subset \mathbb{R}^m$, defined as

$$\begin{aligned} a(t) \in \mathcal{A} &:= \{a(\cdot) : \mathbb{R}_{\geq 0} \to A, \text{ measurable}\} && (2.4) \\ b(t) \in \mathcal{B} &:= \{b(\cdot) : \mathbb{R}_{\geq 0} \to B, \text{ measurable}\} && (2.5) \end{aligned}$$

The function $f : \mathbb{R}^{2n} \times A \times B \to \mathbb{R}^{2n}$ is assumed to be continuous with respect to all variables and Lipschitz Continuous with a Lipschitz constant $L > 0$ in the first variable for all $a \in A$ and $b \in B$.

The state of the game x can be written as $x = (x_P, x_E)$ where the coordinates of the Pursuer x_P and Evader x_E are from the $\mathbb{R}^n$. Since in this game neither player can influence the dynamics of his opponent, f can be written as a function of separated Pursuer and Evader dynamics

$$f(x, a, b) = (f_P(x_P, a), f_E(x_E, b)) \qquad (2.6)$$

with $f_P : \mathbb{R}^n \times A \to \mathbb{R}^n$ and $f_E : \mathbb{R}^n \times B \to \mathbb{R}^n$. The state of the system at time t for the initial state x_0 is the solution of (2.2) and will be denoted as y from now on. It can also be separated into a Pursuer and Evader part similar to f as

$$y(t, x_0, a, b) = (y_P(t, x_P, a), y_E(t, x_E, b)) \qquad (2.7)$$

The terminal or target set of the game is defined as $\mathcal{T} = \left\{ (x_P, x_E) \in \mathbb{R}^{2n} : |x_P - x_E| \leq \varepsilon \right\}$ for a capture radius $\varepsilon \geq 0$. As usual P wants to drive the system to $\mathcal{T}$ while E wants to drive the system away. The payoff $T(x_0)$ for a game starting in x_0 is defined as the first time of arrival of the solution y in $\mathcal{T}$. If however there exists a strategy for E so that he can not be captured, then it is $T(x_0) = +\infty$.

The state constrains are introduced with the help of two bounded sets $\Omega_1, \Omega_2 \subset \mathbb{R}^n$ and $\Omega := \Omega_1 \times \Omega_2 \subset \mathbb{R}^{2n}$. The constrained game is now played on $\overline{\Omega}$ and the two players have to satisfy $y_P \in \overline{\Omega}_1$ and $y_E \in \overline{\Omega}_2$ at all times having *constrained admissible strategies*

$$\mathcal{A}_x := \left\{ a(\cdot) \in \mathcal{A} : y_P(t, x, a(\cdot)) \in \overline{\Omega}_1, \text{ for all } t \geq 0 \right\} \tag{2.8}$$

$$\mathcal{B}_x := \left\{ b(\cdot) \in \mathcal{B} : y_E(t, x, b(\cdot)) \in \overline{\Omega}_2, \text{ for all } t \geq 0 \right\} \tag{2.9}$$

Accordingly the target set has to be adjusted to $\mathcal{T} \cap \overline{\Omega}$.

A discrete version of the dynamics in (2.2) is obtained by using the explicit Euler scheme with a positive (time-) step width of h

$$y_{n+1} = y_n + hf(y_n, a_n, b_n) \tag{2.10}$$

$$y_0 = x_0 \tag{2.11}$$

accordingly the solution of the discrete dynamics at time nh is

$$y(n, x_0, \{a_n\}, \{b_n\}) = (y_P(n, x_0 \{a_n\}), y_E(n, x_0, \{b_n\})) \tag{2.12}$$

and it must satisfy the constraints $y(n, x_0, \{a_n\}, \{b_n\}) \in \overline{\Omega}$ for all $n \in \mathbb{N}$. In order to do so the strategies $\{a_n\}$ and $\{b_n\}$ have to be chosen so that

$$\{a_n\} \in \mathcal{A}_x^h := \left\{ \{a_n\} : \forall n \in \mathbb{N} : \left(a_n \in A \wedge y_P(n, x, \{a_n\}) \in \overline{\Omega}_1 \right) \right\} \tag{2.13}$$

$$\{b_n\} \in \mathcal{B}_X^h := \left\{ \{b_n\} : \forall n \in \mathbb{N} : \left(b_n \in B \wedge y_E(n, x, \{b_n\}) \in \overline{\Omega}_2 \right) \right\} \tag{2.14}$$

For any feasible non-terminal state $y \in \overline{\Omega \backslash \mathcal{T}}$, a_n and b_n are *admissible control values* if

$$a_n \in A_h(y) := \left\{ a \in A : y_P + hf_P(y_P, a) \in \overline{\Omega}_1 \right\} \tag{2.15}$$

$$b_n \in B_h(y) := \left\{ b \in B : y_E + hf_e(y_E, b) \in \overline{\Omega}_2 \right\} \tag{2.16}$$

Also no Player can be rendered immobile if the step size is chosen sufficiently small or in mathematical terms

$$\exists h_0 > 0 : \forall\,(h,x) \in (0,h_0] \times \Omega : A_h\,(x) \neq \emptyset\ \wedge B_h\,(x) \neq \emptyset \tag{2.17}$$

Before the final approximation scheme for the Fully-Discrete Value Function is presented a few definitions regarding the capture ability and capture time as well as the discretization of $\overline{\Omega}$ are introduced. For this kind of game it is natural to assume P can not foresee E's moves or in mathematical terms a map $\alpha_x : \mathcal{B}_x^h \to \mathcal{A}_x^h$ that defines the strategy of P must satisfy

$$\exists \tilde{n} \in \mathbb{N} : \forall n \leq \tilde{n} : b_n = \tilde{b}_n \Rightarrow \forall n \leq \tilde{n} : \alpha_x\,[\{b_k\}]_n = \alpha_x\left[\left\{\tilde{b}_k\right\}\right]_n \tag{2.18}$$

α_x is then called *nonanticipating.* Γ_x^h denotes the set of all nonanticipating strategies of P.

A *reachable set* of all points that lead to capture can be defined as

$$\mathcal{R}^h := \left\{x \in \mathbb{R}^{2n} : \forall\,\{b_n\} \in \mathcal{B}_{x_E}^h \exists \alpha_x \in \Gamma_x^h, \overline{n} \in \mathbb{N} : y\,(\overline{n}, x, \alpha_x\,[\{b_n\}]\,, \{b_n\}) \in \mathcal{T}\right\} \tag{2.19}$$

This set is segmented in the following way

$$\begin{aligned} \mathcal{R}_0 &:= \mathcal{T} && (2.20)\\ \mathcal{R}_n &:= \left\{x \in \overline{\Omega} \backslash \bigcup_{j=0}^{n-1} \mathcal{R}_j : \forall b \in B_h\,(x)\,\exists \hat{a}_x\,(b) \in A_h\,(x) : x + hf\,(x, \hat{a}_x\,(b)\,, b) \in \mathcal{R}_{n-1}\right\} && (2.21) \end{aligned}$$

It is assumed that $\overline{\Omega} = \bigcup_{j=0}^{\infty} \mathcal{R}_j$.

The discrete lower Value of the game is defined as

$$T_h\,(x) := \inf_{\alpha_x \in \Gamma_x^h} \sup_{\{b_n\} \in \mathcal{B}_x^h} hn_h\,(x, \alpha_x\,[\{b_n\}]\,, \{b_n\}) \tag{2.22}$$

with n_h as the minimum amount of steps necessary for the capture

$$n_h\,(x, \{a_n\}\,, \{b_N\}) := \begin{cases} \min\{n \in \mathbb{N} : y\,(n, x, \{a_n\}\,, \{b_n\}) \in \mathcal{T}\} & x \in \mathcal{R}^h \\ +\infty & x \notin \mathcal{R}^h \end{cases} \tag{2.23}$$

The approximation is then done on the *Kružkov transform* of $T_h\,(x)$

$$v_h : \overline{\Omega} \to [0,1] \quad x \mapsto 1 - e^{-T_h(x)} \tag{2.24}$$

The discretization of $\overline{\Omega}$ is achieved with a regular triangulation with a set of simplices $S := \{S_j : j \in \{1, \ldots, L\}\}$. It is assumed that S supports $\bigcup_{j=1}^{L} S_j = \overline{\Omega}$ and $\forall i \neq j : \text{int}\,(S_i) \cap \text{int}\,(S_j) = \emptyset$.

$V(S_j)$ denotes the set of vertices of a simplex S_j. The set of all vertices is $X := \bigcup_{j=1}^{L} V(S_j)$ with $N = |X|$ and $k := \max_j \{\operatorname{diam}(S_j)\}$ will indicate the resolution of the discretization. For $x \in \Omega$ its containing simplex is denoted as $S(x) = S_j$. To have a unique association for each x, j is chosen as the smallest index of a simplex that contains x and to ensure that the granularity of the time and space discretization are in the same order of magnitude

$$\forall j \in \{1, \dots L\} : x \in S_j \cap \mathcal{R}_n \Rightarrow V(S_j) \subset (\mathcal{R}_{n-1} \cup \mathcal{R}_n \cup \mathcal{R}_{n+1}) \tag{2.25}$$

shall hold true, as well as the assumption that S is *consistent* and therefore for arbitrary $j \in \{1, \dots, L\}$:

$$S_j \cap \mathcal{R}_n \neq \emptyset \Rightarrow \exists x_i \in V(s_j) \cap \mathcal{R}_n \tag{2.26}$$

The convergence is measured on the vertices $x_i \in X, i \in \{1, \dots, N\}$ with the norm

$$|v|_\infty := \max_{i \in \{1,\dots,N\}} |v(x_i)| \tag{2.27}$$

Finally, the Fully-Discrete approximation scheme for the Value Function is

$$v_h^k(x) = \begin{cases} \max_{b \in B_h(x)} \min_{a \in A_h(x)} \left\{\beta v_h^k(x + hf(x,a,b))\right\} + 1 - \beta & x \in \left(\overline{\Omega} \backslash \mathcal{T}\right) \cap X \\ 0 & x \in \mathcal{T} \cap X \\ \sum_{x_j \in S(x)} \lambda_j(x) v_h^k(x_j), \ 0 \leq \lambda_j(x) \leq 1, \ \sum_{j=1}^{|S(x)|} \lambda_j(x) = 1 & x \in \overline{\Omega} \end{cases} \tag{2.28}$$

where $\beta := e^{-h}$. The first line of the approximation defines a fixed point iteration for vertices on the grid that are not terminal. Because if they are terminal the second case applies and the value of the function will be set to 0 immediately. The third case handles the "exploration of optimal directions" for the vertices of line 1. It does so by assigning every non-vertex a value obtained by a convex combination of the values of his surrounding vertices. The fixed point iteration (2.28) has the following properties:

Theorem 1. *The Fully-Discrete approximation scheme in (2.28) has a unique solution*

$$w^\star \in W^k := \left\{w \in \mathcal{C}\left(\overline{\Omega}\right) : \forall j \in \{1, \dots, L\} \, \forall x \in \mathit{int}(S_j) \, \exists c \in \mathbb{R} : \nabla w(x) = c\right\} \tag{2.29}$$

such that $w^\star : \overline{\Omega} \to [0,1]$.

Proof. The proof is broken down into several pieces. It follows the general idea of Bardi, Falcone and Soravia presented in [2]. At first the existence and uniqueness of $w^\star$ is shown with help of the Banach Fixed Point Iteration. Recall N as the number of vertices in $X = \bigcup_{j=1}^{L} V(S_j) = \{x_1, \ldots, x_N\}$ and look at the right hand side of the first Line of (2.28) as a map $F : \mathbb{R}^N \to \mathbb{R}^N$ that for every component i performs the approximation on x_i. Let v and w be two arbitrary Value Functions and let h be sufficiently small such that (2.17) holds true. From now on, l will be the component that maximizes

$$|F(v) - F(w)|_\infty = \max_{j\in\{1,\ldots,N\}} |F_j(v) - F_j(w)| \tag{2.30}$$

Now fix $\bar{b} \in B_h(x_l)$ such that

$$\max_{b\in B_h(x_l)} \min_{a\in A_h(x_l)} v(x_l + hf(x_l, a, b)) = \min_{a\in A_h(x_l)} v\left(x_l + hf\left(x_l a, \bar{b}\right)\right) \tag{2.31}$$

and fix $\bar{a} \in A_h(x_l)$ such that

$$\min_{a\in A_h(x_l)} v\left(x_l + hf\left(x_l a, \bar{b}\right)\right) = v\left(x_l + hf\left(x_l, \bar{a}, \bar{b}\right)\right) \tag{2.32}$$

To simplify the notation for the final estimate $x_l^* := x_l^* + hf\left(x_l, \bar{a}, \bar{b}\right)$ is introduced.

$$\begin{aligned} |F(v) - F(w)|_\infty &\le |\beta v(x_l^*) - \beta w(x_l^*)| && (2.33)\\ &= \left|\beta \sum_{x_j\in S\left(x_l^*\right)} \lambda_j(x_l^*)(v(x_j) - w(x_j))\right| && (2.34)\\ &\le \beta |v - w|_\infty && (2.35)\end{aligned}$$

The proposition that $w^\star$ is in W^k follows immediately from line number 3 of (2.28). It is left to show that $w^\star : \overline{\Omega} \to [0,1]$. This is shown by starting at a arbitrary Value Function $v : \overline{\Omega} \to \mathbb{R}$ and looking at $\max_{b\in B_h(x_i)} \min_{a\in A_h(x_i)} \beta v(x_i + hf(x_i, a, b)) + 1 - \beta$ as a convex combination that moves $v_i := \max_{b\in B_h(x_i)} \min_{a\in A_h(x_i)} v(x_i + hf(x_i, a, b))$ closer to 1. Since v is a Kružkov transform we have $v_i \in [0,1]$ and therefore $\beta v_i + (1-\beta)1 \in [0,1]$. $\square$

Since the proof of Theorem 1 only showed the existence of a fixed point $w^\star$ it is not yet clear if this is in fact an approximation of the Value Function and this approximation will converge against the analytic Value Function v for $h, k \to 0$. However for some assumptions on the continuity of v it can be shown that the approximation in fact converges to v if there is a suitable *modulus* ω (a nondecreasing map $w : [0,\infty) \to [0,\infty)$ which is continuous at 0 and $\omega(0) = 0$) as can be seen in the following Theorem that resembles Corollary 3.8 in [1].

Theorem 2. *Let all of the following conditions hold true*

1. *Ω is an open bounded set and (2.17) holds true.*

2. *The triangulation is consistent and (2.25), (2.26) hold true.*

3. *$f : \mathbb{R}^{2n} \times A \times B \to \mathbb{R}^{2n}$ is continuous with respect to all variables and Lipschitz Continuous with a Lipschitz constant $L > 0$ in the first variable for all $a(\cdot) \in \mathcal{A}$ and $b(\cdot) \in \mathcal{B}$.*

4. *There exists a modulus ω such that $v(x) \leq \omega(d(x, \mathcal{T}))$ for all $x \in \overline{\Omega}$.*

5. *There exists a modulus $\omega_P(\cdot, R)$ for all $R > 0$ such that for all $w_1, w_2 \in \overline{\Omega}_1$ there are $a(\cdot) \in \mathcal{A}_{w_1}$ and a time t_{w_1,w_e} satisfying $y_P(t_{w_1,w_2}, w_1, a(\cdot)) = w_2$ and $0 \leq t_{w_1,w_2} \leq \omega_P(|w_1 - w_2|, |w_2|)$.*

6. *There exists a modulus $\omega_E(\cdot, R)$ for all $R > 0$ such that for all $z_1, z_2 \in \overline{\Omega}_2$ there are $b(\cdot) \in \mathcal{B}_{z_1}$ and a time t_{z_1,z_e} satisfying $y_E(t_{z_1,z_2}, z_1, b(\cdot)) = z_2$ and $0 \leq t_{z_1,z_2} \leq \omega_E(|z_1 - z_2|, |z_2|)$.*

7. *For all $x \in \partial\mathcal{T}$ there exist $r, \theta \in \mathbb{R}_{\geq 0}$ and $\varphi \in \mathbb{R}^{2n}$ such that $\forall x' \in B(x, r) \cap \overline{\Omega \backslash \mathcal{T}}$: $\bigcup_{0<t<r} B(x' + t\varphi, t\theta) \subset \Omega \backslash \mathcal{T}$.*

8. *k is in $\mathcal{O}(h^{1+\alpha})$ for $\alpha > 0$.*

Then the solution of the Fully-Discrete approximation scheme (2.28) converges on v uniformly in $\overline{\Omega}$ for $h \to 0$.

Proof. Theorem 2 contains all requirements for the Corollary 3.7 and Theorem 3.7 in [1]. Therefore the Proof of Corollary 3.8 in [1] can be applied here as well. Theorem 2 finally yields the result that

$$\forall x \in \overline{\Omega} : \left|v_h^k(x) - v(x)\right| \leq \mathcal{O}(h^\alpha) + |v_h(x) + v(x)|_\infty \tag{2.36}$$

The result that $|v_h(x) - v(x)|_\infty$ tends to 0 as h approaches 0 is originally shown by Theorem 4.2 in [6]. □

Please note that Condition 4 of Theorem 2 implicitly expects v to be continuous on $\partial\mathcal{T}$ as the next Corollary shows. This limits the proof of the convergence to the game where the velocity of P is greater than Es since v will be discontinuous if the velocity of P is less than or equal to the velocity of E as the next section shows.

Corollary 3. *Let v be a Value Function and ω defined like in Condition 4 of Theorem 2 . Then v is continuous on $\partial\mathcal{T}$.*

Proof. From the continuity of ω at 0 follows for $x \in \partial\mathcal{T}$ and for every $\varepsilon > 0$ exists $\delta > 0$ such that

$$\forall y \in \overline{\Omega} : |x - y| < \delta \Rightarrow \left| \underbrace{v(x)}_{=0} - v(y) \right| = v(y) \leq \omega(d(y, \mathcal{T})) < \varepsilon \tag{2.37}$$

□

2.2 The game if $V_P > V_E$ and if $V_P = V_E$

All the results so far are valid for a very general version of a two player Pursuit-Evasion game. No conditions for e.g. the velocities of the players or any other constraints on the function f describing the dynamics have been stated. However in the last paragraph of the previous section it is shown that the convergence of the Fully-Discrete Value Function v_h^k on the continuous Value Function v is only given if v_h^k is continuous on $\partial\mathcal{T}$. This section now introduces the dynamics of f used throughout the rest of the Thesis and shows that the Value Function v is discontinuous if the velocity of P is not greater than the velocity of E.

Let Ω_1 denote an open, bounded subset of the $\mathbb{R}^2$. From now on the game is played on the $\Omega_1 \times \Omega_1 =: \Omega \subset \mathbb{R}^4$ and the function f describing the dynamics of the system shall now be defined as

$$f \quad : \quad \Omega \times [0, 2\pi] \times [0, 2\pi] \to \mathbb{R}^4 \tag{2.38}$$

$$(x, a, b) \quad \mapsto \quad (V_P \cos(a), V_P \sin(a), V_E \cos(b), V_E \sin(b)) \tag{2.39}$$

with $V_P, V_E \in \mathbb{R}_{>0}$. As usual the first two components of f will be denoted as a map $f_p : \Omega_1 \times [0, 2\pi] \to \Omega_1$, the second two as $f_E : \Omega_1 \times [0, 2\pi] \to \Omega_1$ and $x = (x_P, x_E) \in \Omega$ with $x_P, x_E \in \Omega_1$. This version of f allows P and E to move freely in every direction of the $\mathbb{R}^2$ and it allows the velocities of P and E to be adjusted using the variables V_p and V_E. This matches the setup used by Cristiani & Falcone in [1] therefore the following Theorem applies here as well.

Theorem 4. *Let Ω_1 be open and bounded. Moreover let the target be*

$$\mathcal{T} := \left\{ (x_P, x_E) \in \overline{\Omega} : |x_p - x_E| \leq \varepsilon \right\} \quad \varepsilon \geq 0 \tag{2.40}$$

Then,

1. *If $V_P > V_E$, then the capture time $t_c = T(x_p, x_e) = -\ln(1 - v(x_P, x_E))$ is finite and bounded by $t_c \leq \frac{|x_P - x_E|}{V_P - V_E}$.*

2. *If $V_P = V_E$, $\varepsilon \neq 0$ and Ω_1 is convex then the capture time t_c is finite.*

Proof. The interested reader can find the proof in [1]. Only the general idea will be stated here

1. Let E use an arbitrary strategy. The strategy of P is to first cover the distance $|x_P - x_E|$ to the point x_E. Then he follows the trajectory of E. Since P is faster than E he will capture E for $\varepsilon = 0$ in $t_c = \frac{|x_P - x_E|}{V_P - V_E}$. Hence a better strategy and a greater capture radius ε can further reduce t_c.

2. Again, let E use an arbitrary strategy. Since in this case Ω is convex P can choose the strategy to always follow the line directly connecting P and E. In this case E can temporarily stop P on decreasing the distance between P and E by running directly away from P. However since Ω is bounded he can only chose this direction for a finite time. Then he has to chose a different direction which is no longer optimal and thus allowing P to close in on him and ultimately capture him.

□

With the help of Theorem 4 the following important result can be proved that links the applicability of Theorem 2 to the velocities of P and E

Theorem 5. *Let Ω_1 be open and bounded. Let $V_P > V_E$ and let the target be*

$$\mathcal{T} := \left\{(x_p, x_E) \in \overline{\Omega} : |x_P - x_E| \leq \varepsilon\right\} \quad \varepsilon \geq 0 \tag{2.41}$$

Then v will be continuous on $\partial\mathcal{T}$.

Proof. The proof is done by finding a modulus that satisfies the Condition 4 of Theorem 2 and therefore allowing Corollary 3 to be used. First recall $v(x) = 1 - e^{-T(x)}$ with $T(x)$ as the first time of capture for the game starting at $x \in \overline{\Omega}$. Now denote

$$\tilde{\omega} : [0, \infty) \to [0, \infty) \quad d \mapsto 1 - e^{-d} \tag{2.42}$$

and observe that $\tilde{\omega}$ is a modulus. With the use of $T(x) \leq \frac{|x_P - x_E|}{V_P - V_E}$ from Theorem 4 and since $\frac{|x_P - x_E|}{V_P - V_E} \leq \frac{\mathrm{diag}(\Omega_1)}{V_P - V_E} \leq \frac{\mathrm{diag}(\Omega_1) + d(x, \mathcal{T})}{V_P - V_E}$, the modulus $\tilde{\omega}$ can be tweaked to

$$\omega : [0, \infty) \to [0, \infty) \quad d \mapsto 1 - e^{-\frac{\mathrm{diag}(\Omega_1) + d}{V_P - V_E}} \tag{2.43}$$

Hence $v(x) \leq \omega(d(x, \mathcal{T}))$ and finally satisfying the Condition 4 of Theorem 2 so that Corollary 3 can be applied to prove that v is continuous on $\partial\mathcal{T}$. □

Although no proof is known, a simple example shows that v is discontinuous on $\partial\mathcal{T}$ if $V_P = V_E$.

Example 6. Let $V_P = V_E$. Assume Ω_1 to be bounded and assume that $\{(r,0) : r \in [0,1]\} \subset \Omega_1$ with $(1,0) \in \partial\Omega_1$. The capture shall occur if $|x_P - x_E| \leq \varepsilon$ for $1 > \varepsilon \geq 0$ and set $x_P = (0,0)$ and $x_E = (k,0)$ having $\varepsilon < k < 1$. Now fix a strategy for E that makes him run to $(1,0)$ until he reaches the point and wait there for his capture. If E follows that strategy, the time of capture will be greater than or equal to $\frac{1-\varepsilon}{V_P}$ as this will be the time for P to reach the closest terminal state $(1-\varepsilon,0)$ on a direct trajectory from his starting point $(0,0)$. However E might be able to delay or possible even avoid capture if there exists a better strategy for him. In summary this leads to

$$T(x_P, x_E) = 0 \qquad k \leq \varepsilon \tag{2.44}$$

$$T(x_P, x_E) \geq \frac{1-\varepsilon}{V_P} > 0 \quad k > \varepsilon \tag{2.45}$$

which shows that the time of capture and therefore the Value Function is not continuous on $((0,0),(\varepsilon,0)) \in \partial\mathcal{T}$.

Therefore this shows the crucial part the velocities of the two players play in the Value Function of the game and the convergence of (2.28) respectively.

3 Implementation of the Scheme

This section starts with an overview over the implementation before certain aspects are reviewed in more detail. At last a runtime analysis of the code as well as profiling measurements are provided.

The numerical scheme is implemented in ANSI/ISO C++ and is intended to work across platforms, hence no libraries outside of the C++ Standard Library are being used. The computation is done without using multithreaded libraries and techniques, although the codebase can be very easily paralleled with parallel programming libraries like OpenMP[2]. For a easy visualization of the computed functions and trajectories all the output of the experiments is done so that the data can be directly used with the powerful Gnuplot[3] tool, which has been used to generate all the plots of this thesis. Looking at the size of the implementation it seems rather large with 20 files of code counting more than 5.500 lines in total. However approximately 2.000 lines of code are dedicated to implementing most of the experiments of [1] and internal test routines. The rest of the code is very well documented, as the attached PDF document generated from the source code with the help of Doxygen[4] has more than 150 pages of in depth documentation.

3.1 The extensible Class Model

Most properties of the implementation can be tweaked at run time, however to fasten and simplify the implementation some compile time variables found in `Definitions.h` have to adjusted to the correct values. At least the dimension of $\overline{\Omega}$ has to be defined with the compile time declaration of `STATE_SPACE_DIMENSION` which will be 4 for the rest of this thesis. Also, some parts of the implementation won't work if the dimension is not 4, as it seamed unnecessary difficult for the study of two-player Pursuit-Evasion Games in the plane to provide a dimension independent implementation.

The implementation utilizes the Object Oriented Programming paradigm to separate the Game, its Domain and the approximation Scheme into classes. They are all contained in a single namespace called `bsc` and their inheritance and associations are depicted in the Figure 3.1 using the notation of the Unified Modeling Language (UML).

The class `GameStateDomain` represents an abstract class[5] where the game can take place where. As an abstract class, it does not depend on any dimension or geometric shape of the domain, as all of this information has to be provided by its derived classes. It merely defines a set of functions

[2] `http://www.openmp.org/`
[3] `http://www.gnuplot.info/`
[4] `http://www.doxygen.org/`
[5] See [4], Chapter 8 "Class Inheritance" and especially Section 8.2 "Abstract Classes" for an introduction into Class Inheritance in C++

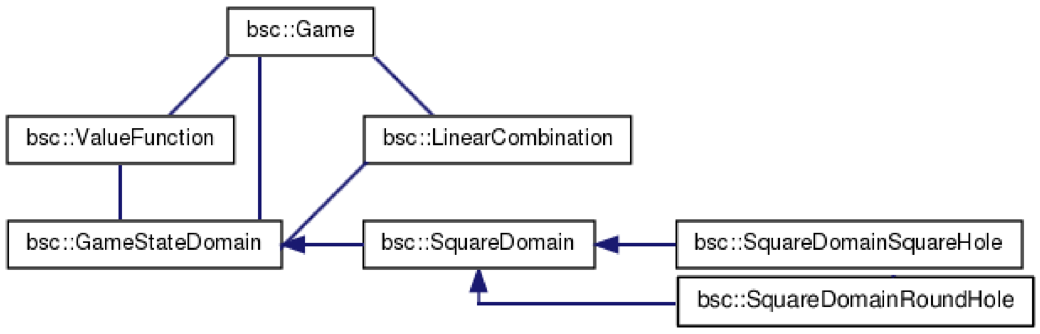

Figure 3.1: Inheritance and association UML diagram.

to be implemented by derived custom domains in order to be used by the approximation scheme. Most importantly the `isFeasible` function that tells if a State or index of a vertex is feasible for the given domain and other functions for mapping of indexes to vertices and vice versa. In the code, to avoid any confusion, the index of a vertex will always be referred to as a `GridPointIndex`, which is implemented as an Integer array. Three specific Domains are already implemented

1. the `SquareDomain` which represents the cube $\overline{\Omega} = [a,b]^4 \subset \mathbb{R}^4$ for $a, b \in \mathbb{R}$.
2. the `SquareDomainRoundHole` which sets $\overline{\Omega}$ to $[a,b]^4 \setminus B_r^4(x,y,x,y) \subset \mathbb{R}^4$ for $a,b,r,x,y \in \mathbb{R}$ with $B_r^4(x,y,x,y)$ being the 4-dimensional ball with radius r surrounding $(x,y,x,y) \in \mathbb{R}^4$.
3. the `SquareDomainSquareHole` which removes the cube $[c,d]^4$ from the SquareDomain, leading to $\overline{\Omega} = [a,b]^4 \setminus [c,d]^4 \subset \mathbb{R}^4$ for $a < c < d < b \in \mathbb{R}$.

The abstract `GameStateDomain` class makes it easy to perform experiments on other Domains like e.g. $B_2^4(0,0,0,0)$ as only a new child class of the `GameStateDomain` has to implemented, leaving the rest of the code completely in tact.

The `LinearCombination` class stores the $\lambda_j(x) \in [0,1]$ and $x_j \in X$ to perform the convex combination of $x \in \overline{\Omega}$ such that $x = \sum_{x_j \in S(x)} \lambda_j(x) x_j,\ 0 \leq \lambda_j(x) \leq 1,\ \sum_{j=1}^{|S(x)|} \lambda_j(x) = 1$. This is necessary to determine the value of $x \in \overline{\Omega} \backslash X$ during the approximation as specified in (2.28).

The `ValueFunction` class is the Fully-Discrete approximation v_h^k of v. It stores the value $v_h^k(x_i)$ for each vertex $x_i \in X$ and contains a method to determine the value $v_h^k(x)$ for any $x \in \overline{\Omega}$ by first performing a convex decomposition to obtain a `LinearCombination` for x out of the vertices in $S(x)$ and then computing the value of x as specified in line number 3 of the Fully-Discrete approximation scheme for the Value Function (2.28).

Finally the `Game` class provides the implementation of (2.28) as well as the dynamics defined by (2.39) and the computation of the optimal trajectories for P and E. Creating a new child class of the Game class would allow to define new dynamics by overriding the function `evalKE`.

3.2 Implementation details of the domain and control values

This section describes the mathematical concepts and their numerical implementation that have not or just very briefly been covered by Cristiani and Falcone in [1]. Nevertheless a correct and fast implementation of these details are crucial to use the Fully-Discrete approximation scheme for the Value-Function (2.28).

3.2.1 Linear Combination and Convex decomposition in $\overline{\Omega}$

On Page 191 of [1] the authors introduce a method to quickly compute the value of $v_h^k(x)$ for $x \in \overline{\Omega}$ by iteratively projecting x alongside each dimension to find both bounds of its containing cell until "at [the last] stage a tree structure containing all points $P_i^j,\ i = 1, \dots, 2^n, j = 1, \dots, n$ is computed from top to bottom"[6]. With n being the dimension of the State space and P_i^j being a vertex in the cell with $P_i^j = x_k$ for some $k \in \{1, \dots, N\}$ at the last step. Figure 3.2 shows the principle of this projection and the resulting tree structure in a 2 dimensional showcase.

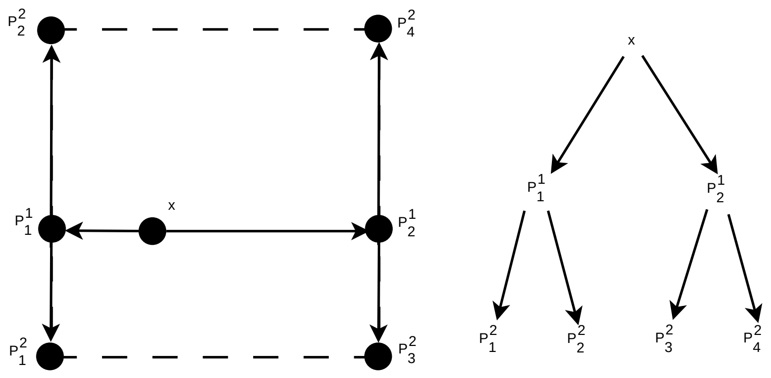

Figure 3.2: Convex decomposition and the resulting tree structure.

since v_h^k is known at the vertices they then "evaluate by unidimensional linear interpolations the values of f [v_h^k in the notation of this thesis] at the points $P_i^j, i = 1, \dots, 2^n, j = 1, \dots, n$ in the reverse order [...]. This procedure leads to an approximate value of f [$=v_h^k(x)$] obtained by $2^n - 1$ unidimensional linear interpolations"[7]. To put this into an equation, they use

$$v_h^k(x) \approx \lambda_1^1 \left(\lambda_1^2 v_h^k\left(P_1^2\right) + \left(1 - \lambda_1^2\right) v_h^k\left(P_2^2\right)\right) + \left(1 - \lambda_1^1\right)\left(\lambda_2^2 v_h^k\left(P_2^3\right) + \left(1 - \lambda_2^2\right) v_h^k\left(P_2^4\right)\right) \tag{3.1}$$

and the implementation of that method would be highly recursive. Written in Pseudocde[8], Algorithm 1 shows how to evaluate a tree structure t generated for a point x

[6][1], page 191

[7][1], page 191

[8]Pseudocode is used through out this Thesis to illustrate the essence of algorithms and ideas in way that is easy to parse. If suitable a syntax close the C++ will be used.

Algorithm 1 Pseudocode recursive evaluation of $v_h^k(x)$

```
function eval_v(Tree t) {
    if(T is not a leaf) {

        /* t is not a leaf. Therefore go one step down in the recursion
         * for the lower and higher projection, but remember to
         * multiply the values with lambda (and (1 - lambda)
         * respectively) to get correct value for this tree. */
        return t.lambda * eval_v(t.lowerProjection) +
            (1 - t.lambda) * eval_v(t.higherProjection)

    } else {

        /* If this is a leaf, the recursion has reached a vertex and
         * the value of v is known at the vertex therefore obtain the
         * value and pass it up to the caller. */
        return v(t.vertex)

  }
}
```

This recursive implementation might look very elegant but "recursive functions may impose a lot of run-time overhead compared to nonrecursive functions. [...] The overhead of function calls can significantly slow down a program when there are a large number of them".[9] In this case it would generate $2^n - 1$ recursion for every evaluation of $v_h^k(x)$. In total this can be quite costly, depending on the amount of feasible control values for a vertex x_i. Since line 1 of (2.28) requires $|A_h(x)| \cdot |B_h(x)|$ convex decompositions and recursive evaluations for each vertex x_i in each iteration of the approximation.

Therefore it is preferred to eliminate the recursion by opening the braces of the right hand side of (3.1) to obtain an expression that can be evaluated without recursion since it is now a simple linear combination

$$\underbrace{\lambda_1^1\lambda_1^2}_{=:\tilde{\lambda}_1} v_h^k\left(P_1^2\right) + \underbrace{\lambda_1^1\left(1-\lambda_1^2\right)}_{=:\tilde{\lambda}_2} v_h^k\left(P_2^2\right) + \underbrace{\left(1-\lambda_1^1\right)\lambda_2^2}_{=:\tilde{\lambda}_3} v_h^k\left(P_2^3\right) + \underbrace{\left(1-\lambda_1^1\right)\left(1-\lambda_2^2\right)}_{=:\tilde{\lambda}_4} v_h^k\left(P_2^4\right) \tag{3.2}$$

The `convexDecomposition` method of the `SquareDomain` class follows that idea. Algorithm 2 shows the essence of the implementation in Pseudocode notation. It is followed by a short example to illustrate the changes in the `lambda` and `p` arrays of the algorithm.

[9][4], page 81

Algorithm 2 Pseudocode linear decomposition of x

```
function convexDecomposition(State x) {
    /* The two arrays will contain the points and factors of the linear
     * combination. Since x is contained in a cube in the R^n, the
     * arrays need to be large enough to hold the 2^n verticies of
     * the cube. */
    Initialize arrays: lambda[i] = 1 and p[i] = x for i = 1 : 2^n.

    /* This first loop determines the direction of the projection.
     * Since for each change in direction the amount of points double
     * it also functions as a "virtual recursion" counter. */
    for d = 1 : n {

        /* This loop ensures that every point is being projected. */
        while i <= 2^n {

            /* This marks that a new point has been reached.
             * Therefore the new floor and ceil values, as well as the
             * new factor lambda have to be calculated. */
            calculate the floor and ceil values of
            point p[i] for the dimension d

            calculate the unidimensional lambda for the
            floor and ceil values of p[i]

            /* Now propagate the floor values to the 2^(n-d) children
             * of p[i]. */
            for j = 1 : 2^(n-d) {
                set lambda[i] = lambda[i] * lambda

                set component d of p[j] to the lower
                value calculated previously.

                i++
            }

            /* Now propagate the ceil values to the
             * 2^(n-d) children of p[i]. */
            for j = 1 : 2^(n-d) {
                set lambda[i] = lambda[i] * (1 - lambda)

                set component d of p[i] to the higher
                value calculated previously.

                i++
            }
        }
    }
}
```

Example 7. Let $x = (0.4, 0.8) \in \mathbb{R}^2$ and $S(x) = [0,1]^2$. The following table shows the changes of the arrays `lambda` and `p` of Algorithm 2 as the loop counters `d` and `i` increase

d	i	λ[1]	λ[2]	λ[3]	λ[4]	p[1]	p[2]	p[3]	p[4]
0	0	1	1	1	1	$(0.4, 0.8)$	$(0.4, 0.8)$	$(0.4, 0.8)$	$(0.4, 0.8)$
1	1	0.6	1	1	1	$(0, 0.8)$	$(0.4, 0.8)$	$(0.4, 0.8)$	$(0.4, 0.8)$
1	2	0.6	0.6	1	1	$(0, 0.8)$	$(0, 0.8)$	$(0.4, 0.8)$	$(0.4, 0.8)$
1	3	0.6	0.6	0.4	1	$(0, 0.8)$	$(0, 0.8)$	$(1, 0.8)$	$(0.4, 0.8)$
1	4	0.6	0.6	0.4	0.4	$(0, 0.8)$	$(0, 0.8)$	$(1, 0.8)$	$(1, 0.8)$
2	1	0.12	0.6	0.4	0.4	$(0, 0)$	$(0, 0.8)$	$(1, 0.8)$	$(1, 0.8)$
2	2	0.12	0.48	0.4	0.4	$(0, 0)$	$(0, 1)$	$(1, 0.8)$	$(1, 0.8)$
2	3	0.12	0.48	0.08	0.4	$(0, 0)$	$(0, 1)$	$(1, 0)$	$(1, 0.8)$
2	4	0.12	0.48	0.08	0.32	$(0, 0)$	$(0, 1)$	$(1, 0)$	$(1, 1)$

Table 1: Changes of the variables of Algorithm 2

3.2.2 The domains, their holes and their numbering scheme

For $\overline{\Omega} \subset \mathbb{R}^{2n}$ and a set of vertices X a discrete domain $\overline{\Omega}_X$ is defined as

$$\overline{\Omega}_X = \left\{x \in \overline{\Omega} : S(x) \subset \overline{\Omega}\right\} \tag{3.3}$$

the numerics are then done on $\overline{\Omega}_X$ to ensure that the boundaries of $\overline{\Omega}$ are carefully aligned on the Grid of X. Otherwise there might be the case that for $x \in \overline{\Omega} \backslash X$ there is a $x_i \in S(x)$ such that $x_i \notin \overline{\Omega}$. If now an attempt is started to compute the value of v_h^k at x, v_h^k will also be evaluated at x_i which is mathematically incorrect and leads to numerically unpredictable results as v_h^k is not defined there. Figure 3.3 shows an example for the differences between $\overline{\Omega}$ and $\overline{\Omega}_X$.

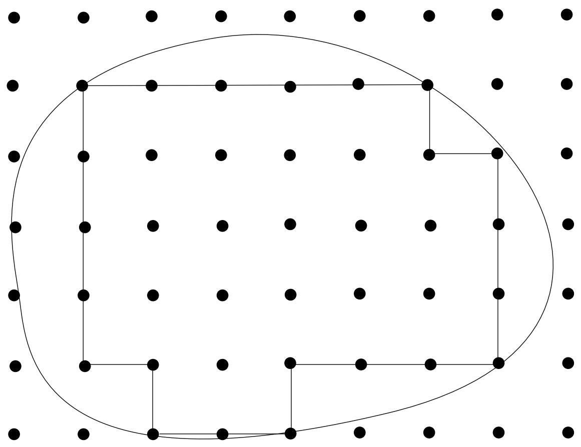

Figure 3.3: $\overline{\Omega}$ and $\overline{\Omega}_X$

However as the previous section showed, computing the vertices of the cell $S(x)$ can be quite costly therefore quicker means to determine if a point x is inside the numerically valid bounds need to be developed. For the `SquareDomainSquareHole` which defines $\overline{\Omega} = [a, b]^4 \setminus [c, d]^4 \subset \mathbb{R}^4$ this is simply done by rounding the parameters c, d to ensure that they align with the grid. This can be done with a very simple calculation

$$roundToCell(x) = a + \frac{b-a}{l}\left\lfloor \frac{x + \frac{b-a}{2l} - a}{\frac{a-b}{l}} \right\rfloor \tag{3.4}$$

if the domain is discretizised with l steps in each dimension. Then finally yield as the numerical domain

$$\overline{\Omega}_X = [a,b]^4 \setminus [roundToCell\,(c)\,, roundToCell\,(d)]^4 \tag{3.5}$$

Unfortunately because of the more difficult geometry of the round hole in the `SquareDomainRoundHole` a more sophisticated method is needed. Let $(x,y) \in \mathbb{R}^2$ be the coordinates of either player and $(x_h, y_h) \in \mathbb{R}^2$ the center for the hole with radius $r > 0$. To ensure that the point is feasible only the distance $d\,(S\,(x,y)\,, (x_h, y_h))$ needs to be checked. Therefore it would suffice to determine the vertex that is closest to the center of the hole with the following algorithm

$$closestVertex\,((x,y)\,,(x_h,y_h)) = \begin{pmatrix} \begin{cases} roundToCell\left(x + \frac{b-a}{2l}\right) & x < x_h \\ roundToCell\left(x - \frac{b-a}{2l}\right) & x \geq x_h \end{cases} \\ \begin{cases} roundToCell\left(y + \frac{b-a}{2l}\right) & y < y_h \\ roundToCell\left(y - \frac{b-a}{2l}\right) & y \geq y_h \end{cases} \end{pmatrix} \in \mathbb{R}^2 \tag{3.6}$$

as before l denotes the discretization steps in each dimension on the interval $[a,b]$. If then the distance of the closest vertex is greater than r, the position (x,y) of the player is feasible.

The discretization of $\overline{\Omega}$ on page 5 defines that the index i of a vertex $x_i \in X$ must be from the set $\{1, \ldots, N\}$. In the implementation, the index is denoted as a `GridPointIndex` which is an integer array of length STATE_SPACE_DIMENSION, or formally a speaking, it is a point from the $G := \{1, \ldots, N_1\} \times ... \times \{1, \ldots, N_d\} \subset \mathbb{N}^d$ with N_l being the amount of discretization steps for dimension l. All three domains use the same numbering scheme throughout the whole domain. For the domains with a hole there might be a set of `GridPointIndexes` $G_{illegal} \subset G$ so that for $g \in G_{illegal}$ the corresponding point x_g is not in $\overline{\Omega}$. The advantage of this implementation is that it leverages the need to come up with an complex indexing scheme that is exhaustive for all the vertices $x_i \in X$ and yet of minimal size. Instead a simple feasibility check is used to determine if a `GridPointIndex` g is in $G \backslash G_{illegal}$.

3.2.3 Implementation of the Control Values

For the dynamics introduced in Formula (2.39) the numeric implementation is achieved by adjusting the control variables $a, b \in [0, 2\pi]$. In the implementation they are now picked out of a set $a_h, b_h \in$

$\{0, 1, \ldots, n_c\}$ and changing the discrete dynamics for each player to

$$f \quad : \quad \overline{\Omega} \times \{0, 1, \ldots, n_c\} \to \mathbb{R}^4 \tag{3.7}$$

$$(x, c) \quad \to \quad \begin{cases} 0 & c = 0 \\ V \cdot \left(\cos\left(\frac{2\pi c}{n_c}\right), \sin\left(\frac{2\pi c}{n_c}\right)\right)^T & c \neq 0 \end{cases} \tag{3.8}$$

with n_c being the resolution of controls. According to the speed of the player, V has to be adjusted to either V_P or V_E. If it is not wanted to for the players to remain still, $a_h, b_h \neq 0$ must be enforced at all times.

3.3 Runtime analysis and profiling

The following results where obtained by a `gprof`[10] analysis of the computation for a game on $\overline{\Omega} = [-2, 2]^4$. The discretization of the domain has 14 steps in each dimension and the discretization of the control parameters was 8 since it is allowed for the players to stand still in this Game. The experiment has been conducted on an Intel(R) Core(TM)2 Duo CPU P9400 @ 2.40GHz. The command to start the experiment is "`BscThesis 0 0 1`".

Without any optimization on the code or on the symmetry of the problem, `gprof` shows the following results.

% time	# calls	name
46.74	49 351 374	`bsc::SquareDomain::convexDecomposition`
10.27	49 351 374	`bsc::ValueFunction::eval`
9.80	1 423 468 126	`bsc::LinearCombination::setFactor`
7.29	1 772 708 870	`bsc::LinearCombination::getFactor`
3.59	49 351 375	`bsc::SquareDomain::getGridIndex`

Table 2: gprof output for the unoptimized code: Top 5 functions with regards to runtime

As expected the convex decomposition takes up most of the computational time. This method is called $|X| \cdot |A_h(x)| \cdot |B_h(x)|$ times for each iteration of the approximation and therefore grows exponentially with the discretization of controls and domain in each dimension. The methods of the `LinearCombination` are called even more often, but since they are very cheap they don't take up to much time. This motivates two approaches for optimization. First reduce the amount of vertices to be evaluated by using the symmetry of the Value Function. Second, use the features of C++ to minimize the overhead on the CPU.

Cristiani and Falcone showed in [1] on pages 194 and 195 that the Value Function of the game

[10] `http://www.gnu.org/software/binutils/`

has the following symmetry:

$$v(x_P, y_P, x_E, y_E) = v(-x_P, -y_P, -x_E, -y_E) \tag{3.9}$$

$$v(x_P, y_P, x_E, y_E) = v(-x_P, y_P-, x_E, y_E) \tag{3.10}$$

$$v(x_P, y_P, x_E, y_E) = v(x_P, -y_P, x_E, -y_E) \tag{3.11}$$

This symmetry can be used to reduce the computational effort as shown in [1] on page 192. If all the proposed optimization has been implemented the final computation algorithm is

Algorithm 3 Symmetric Optimization in the computation of v_h^k

```
while( v has changed && iterations < maxIterations ) {
    for i=1 : n/2 {
        for j=1 : n/2 {
            for k=1 : n {
                for l=1 : n {
                    if(not isFeasible(i, j, k ,l)
                        skip this loop

                    v(i, j, k, l) = computeValueOnGrid(...)
                    v(i, n-j+1, k, n-l+1)=v(i, j, k, l)
                }
            }
        }
    }

    /*
    for j=1 : n {
        for k=1 : n {
            for l=1 : n {
                v(n/2 + 1, n-j+1, n-k+1, n-l+1) = v(n/2, j, k, l)
            }
        }
    }
}

for i=1 : n/2 {
    for j=1 : n {
        for k=1 : n {
            for l=1 : n {
                v(n-i+1, n-j+1, n-k+1, n-l+1) = v(i, j, k, l)
            }
        }
    }
}
```

Algorithm 3 resembles the basic structure of the `computeValueFunction` function of the `Game` class. This implementation can switch the symmetry optimization on at compile time by setting the `OPTIMIZE_SYMMETRY` definition to 1.

The rest of the optimizations can be turned on the same by the setting the corresponding `OPTIMIZE_` definition to 1. They follow a more technical nature by removing recursion from the convexDecomposition as stated before in Algorithm 2. Additionally very defensive validation checks are disabled and the internal structure of the data storage of the `ValueFunction` class is tweaked for faster read access. At last the vast number of 1 423 468 126 function calls of the `setFactor` and `getFactor` methods of the `LinearCombination` class bear optimization potential, because when "a function is called, its arguments are copied and the program branches out to a new location [...] and comes back after the instructions contained in the function are executed. To avoid such function calling overhead, inline functions can be defined [...] for small functions"[11]. Since those methods are only small read or write operations on internal variables of the `LinearCombination`, inlining proved to be effective.

The graph measures the run time in seconds using the unix `time` command[12]. Again the experiment was conducted on an Intel(R) Core(TM)2 Duo CPU P9400 @ 2.40GHz. The command that started the experiments was "`BscThesis 0 0 1`" so the parameters did not change. It shows that optimization of the second kind saves a few milliseconds of each convex decomposition (bar labeled "Symmetry disabled"), however the clever use of the symmetry reduces the computational time to a fraction of the original. All optimizations together achieve a reduction to 53 seconds, a sixth of the original 334 seconds.

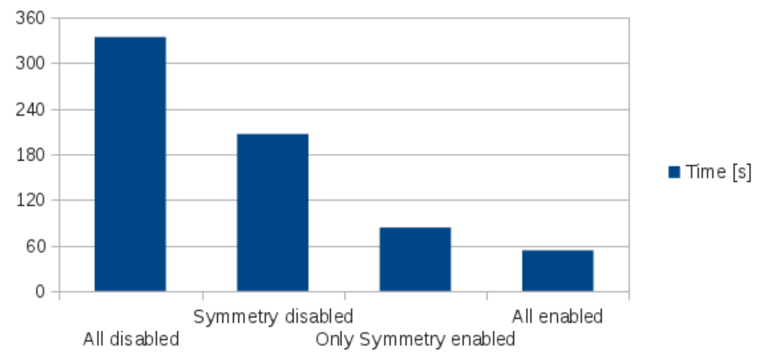

Figure 3.4: Computational time for different optimizations performed.

Despite all this optimization success, the run time of this implementation still is quite slow compared to the speed of [1]. On page 196 of [1] the cumulated CPU time of Test 1 was approximately 17.5 hours on a IBM Power5 CPU @ 1.9 GHz. However, the same experiment took on an Intel(R) Xeon(R) CPU X5680 @ 3.33GHz almost 160 hours. On page 198 of [1] a single dual core PC @ 2.8 GHz performed Test 2 in 68 minutes. On the Intel Xeon(R) CPU it took 174 minutes.

[11] [4], pages 81 and 82

[12] `http://www.kernel.org/doc/man-pages/online/pages/man1/time.1.html`

A second `gprof` analysis on the fully optimized code revealed the following usage It shows that

% time	# calls	name
80.40	11 479 884	`bsc::SquareDomain::convexDecomposition`
7.07	11 479 884	`bsc::ValueFunction::eval`
3.27	15 257 006	`bsc::Game::evalKE`
2.37	11 479 884	`bsc::LinearCombination::LinearCombination`
2.97	11 479 884	`bsc::LinearCombination::~LinearCombination`

Table 3: `gprof` output for the optimized code: Top 5 functions with regards to runtime

the CPU is busy with the convexDecomposition 80% of the time. This is expected as most of the small functions that have been inlined, were called by the convex decomposition algorithm. They are now embedded into the `convexDecomposition` function, and `gprof` can no longer track their runtime and adds it to the `convexDecomposition` runtime. However further optimization attempts on the `convexDecomposition` showed no positive results. Also changing the precision of the numeric calculation from double to single floats did not show an improvement, hence the cause of this slowdown is unknown to the author of the thesis.

In terms of memory consumption the implementation is very moderate. Most memory is used by the `ValueFunction` class as it stores the value of v_h^k at each vertex in X. Therefore it stores $|X|$ decimal with double precision . In case where the domain is discretized with n points in each dimension the memory consumption grows like n^d with d being the dimension of the domain. For a game in the plane with a resolution of $n = 50$ and under the assumption that a double precision decimal takes up 8 bytes of memory, this leads to a total of $50^4 \cdot 8$ bytes $=$ 50 000 000 bytes $=$ 47.68 Mbytes of memory necessary to store a `ValueFunction`. Since this implementation uses two `ValueFunctions`, in total of little less then 100 MBytes are used.

4 Numerical Experiments

This section finally shows the results of the numeric experiments. At first a comparison of trajectory plots and level sets of the Value Function is made for the case $V_P > V_E$. Later, differences and parameters that influence the numeric outcome are being highlighted.

All the tests have been performed on an Intel(R) Xeon(R) CPU X5680 @ 3.33GHz at the Chair for Scientific Computing at the Technical University in Munich[13]. As in [1] the following notation is used

- n denotes the amount of nodes in each dimension.
- n_C denotes the mount of directions for each player, as specified in 3.2.3. In compliance with Cristiani and Falcone, for arbitrary $i > 0$, the notation $n_c = i$ does not permit the players to remain still (enforcing $a_h, b_h \neq 0$ at all times). If standing still is wanted the notation is changed to $n_c = i + 1$.
- $\varepsilon > 0$ denotes the minimal difference between two iterations of the fixed point iteration F. Therefore if

  $$\left|v_h^k - F\left(v_h^k\right)\right|_\infty \leq \varepsilon \tag{4.1}$$

  the iteration stops.
- V_P is the velocity of the Pursuer, V_E the velocity of the Evader.

All experiments take place in $\overline{\Omega} = [-2, 2]^4$. And capture occurs if the P and E are in neighboring cells

$$\mathcal{T} = \left\{(x_P, y_P, x_E, y_E) \in \overline{\Omega} : |x_P - x_E| \leq \triangle x \ \wedge \ |y_P - y_E| \leq \triangle x\right\} \tag{4.2}$$

$\triangle x$ is the minimum distance between two nodes on the grid. It is set to $\frac{4}{n-1}$ throughout the experiments. The timestep is always set to $\frac{\triangle x}{2}$. The plots of the Value Function show the time of capture $T\left(x_{P,} y_P, x_E y_E\right) = -\ln\left(1 - v_h^k\left(x_p, y_p, x_E, y_E\right)\right)$.

4.1 Comparison with the original results

Tests 1, 3 and 4 of [1] in section 6.1 were performed and are now being compared in greater detail. The left hand side always shows the original results of Cristiani and Falcone, the right side, the genuine experiments of this independent implementation.

[13] `http://www-m3.ma.tum.de/Allgemeines/Home`

4.1.1 Comparison with Test 1

The parameters are: $\varepsilon = 10^{-3}$, $V_P = 2$, $V_E = 1$, $n = 50$, $n_c = 48 + 1$. The CPU time of Cristiani and Falcone was 17h 36m 16s until they reached convergence in 85 iterations[14]. The TUM computation took 6d 15h 5m 22s to converge in 87 iterations.

The results provide a very good fit to the original calculations of Cristiani and Falcone. Little difference can be found for example in Figure 4.5 if the trajectory of P is examined very closely. The curve of P reaches his peak at approximately $(1, -1.2)$ on the left and at $(1.2, -1.2)$ the right side. This is most likely a effect of the inaccuracy of the numeric calculations.

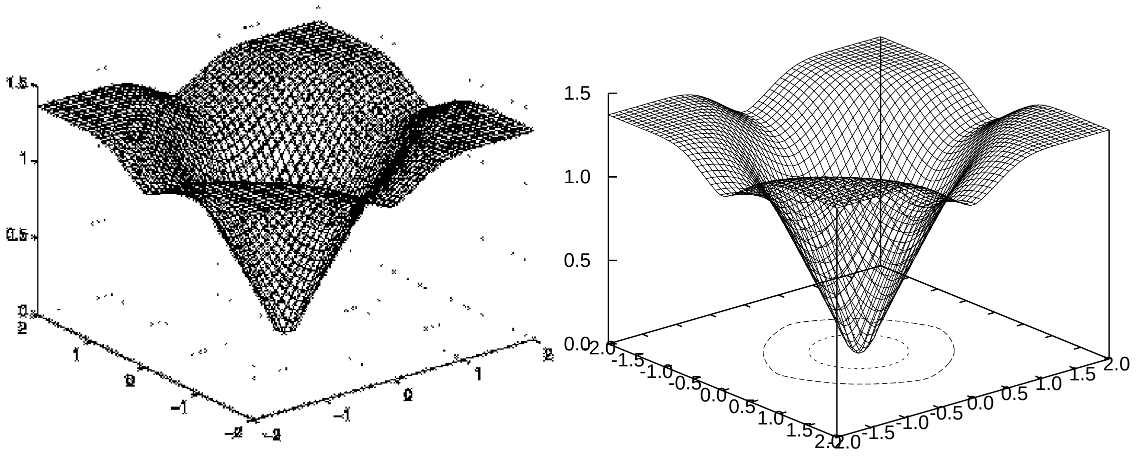

Figure 4.1: Value function $T\,(0, 0, x_E, y_E)$

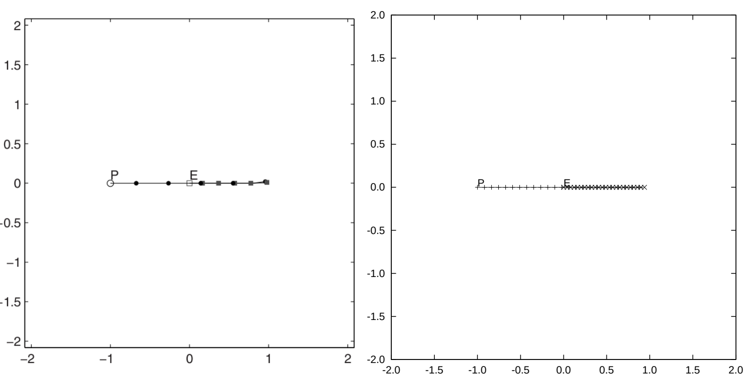

Figure 4.2: Trajectories for $P = (-1, 0)\,,\ E = (0, 0)$

[14][1], page 196

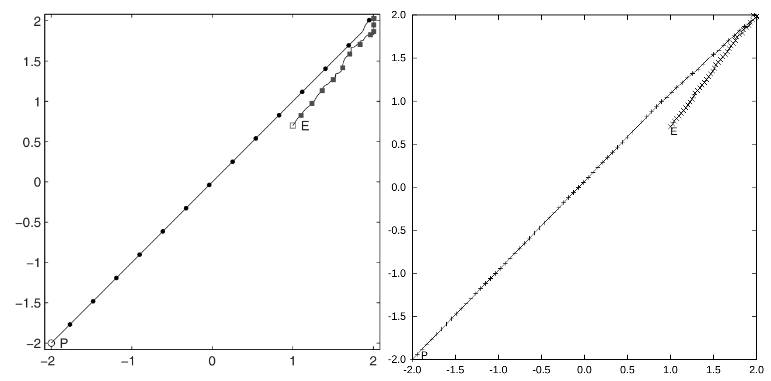

Figure 4.3: Trajectories for $P = (-2, -2)$, $E = (1, 0.7)$

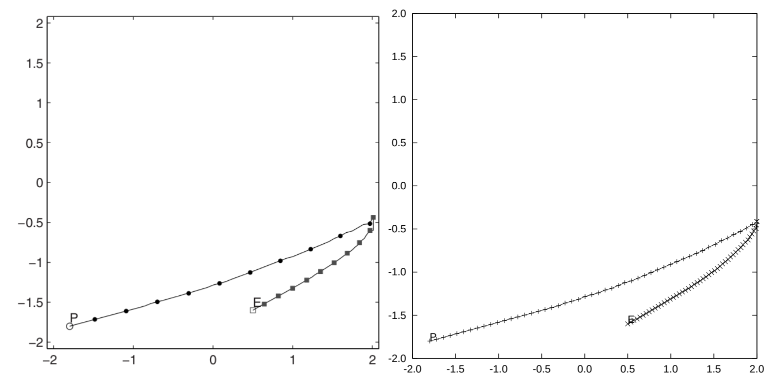

Figure 4.4: Trajectories for $P = (-1.8, -1.8)$, $E = (0.5, -1.6)$

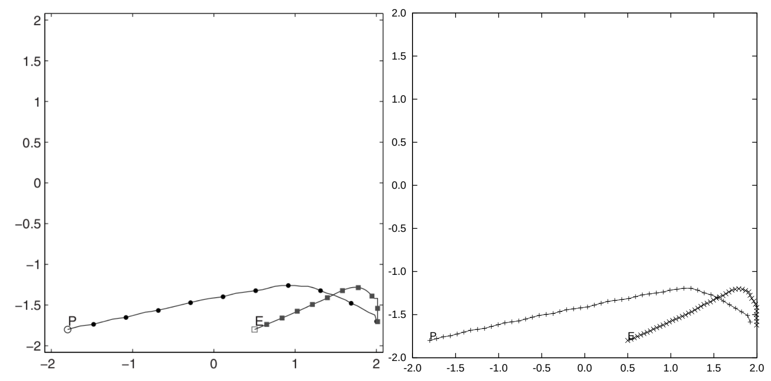

Figure 4.5: Trajectories for $P = (-1.8, -1.8)$, $E = (0.5, -1.8)$

4.1.2 Comparison with Test 3

This time $\overline{\Omega} = [-2, 2]^4 \setminus [-0.53, 0.53]^4$ and $\varepsilon = 10^{-4},\ V_P = 2,\ V_E = 1,\ n = 50,\ n_c = 48 + 1$. The CPU time of Cristiani and Falcone was 1d 0h 34m 18s until they reached convergence in 109 iterations[15]. The TUM computation took 6d 13h 25m 11s to converge in 100 iterations.

While the Value function depicted in figure 4.6 and the trajectories in figure 4.7 match the original results. The last trajectory in figure 4.8 shows interesting behavior. It shows the trajectories for a game starting at $P = (-1.9, 0, 0)$ and $E = (1, 0)$. In the simulation of Cristiani and Falcone E runs towards the obstacle and waits there until P decides on the top or bottom road around the obstacle. E then chooses the different side and both circle the obstacle until capture occurs. The time of capture is in the range of 2.57 to 2.63 as the trajectory of P is 5.14 - 5.26 long. In the independent calculation the time of capture is 2.69. So the trajectory turning away from the obstacle is favorable for the evader. It is hard to tell which is the correct solution or if both are optimal and the difference in the time of capture comes from numeric inaccuracy since the starting points of P and E are located on, what Isaacs called, a dispersal surface[16]. These are set where the players are left with the multiple equally good choices. Unfortunately in his study of dispersal surfaces, Isaacs did not further investigate the obstacle tag chase game. In his limited study of the unconstrained obstacle tag chase game[17] he first proposes for P to randomly choose a direct path to one tangent to the obstacle. P then follows the obstacle until he reaches the tangent to the initial point of E. In the meantime E runs away from the obstacle on a trajectory connecting this initial point to the tangent to the obstacle. However, he quickly discovers that in some case the "tangent strategy" of P and E can not be applied if this leads to a situation where there is a direct line connecting P and E at some time $t > t_0$. Because then P can avoid the obstacle and directly chase after E. No results for the constrained game are known but the trajectory suggests that P and E follow modified "tangent strategy" where E hides behind the obstacle until P finally decides for a tangent. E then tries to follow the tangent, but since it is not possible because of the state constraints he chooses to run to the corner.

[15][1], page 198
[16][3], pages 134 - 135
[17][3], pages 134 - 135 and 152 - 153

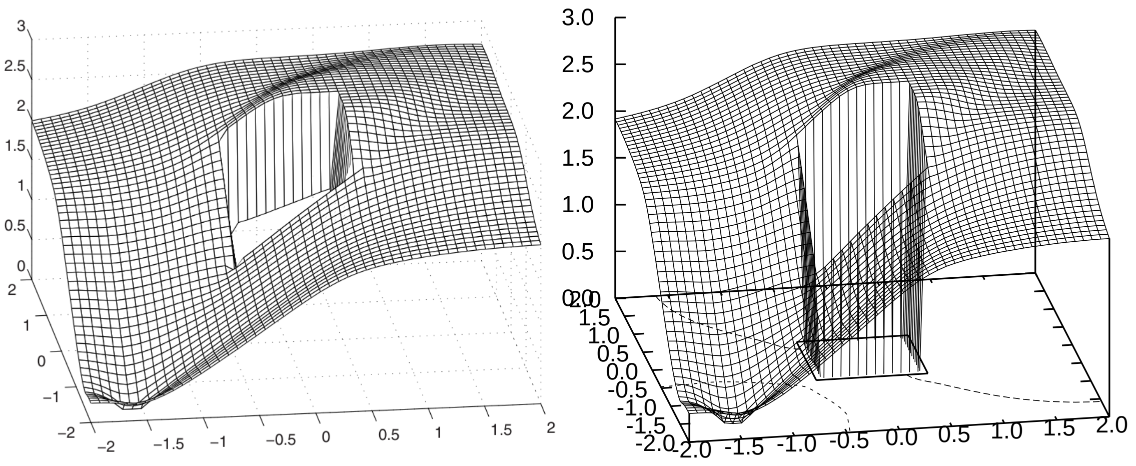
Figure 4.6: Value function $T\left(-1.5,-1.5,x_E,y_E\right)$

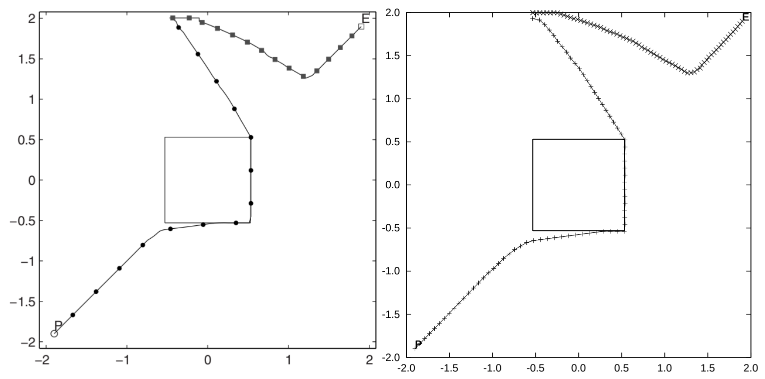
Figure 4.7: Trajectories for $P=\left(-1.9,-1.9\right),\ E=\left(1.9,1.9\right)$

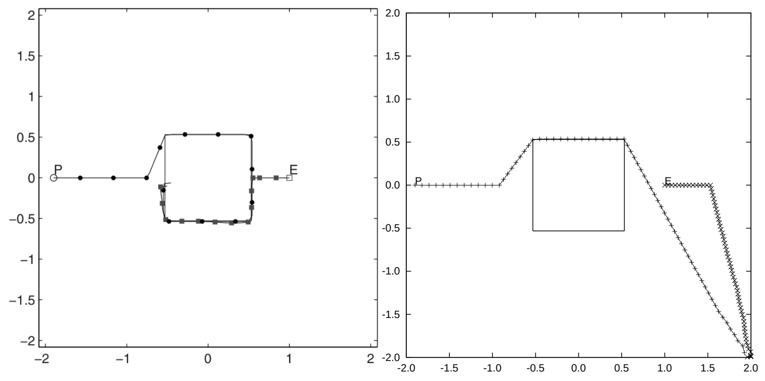
Figure 4.8: Trajectories for $P=\left(-1.9,0\right),\ E=\left(1,0\right)$

4.1.3 Comparison with Test 4

This time $\overline{\Omega} = [-2,2]^4 \backslash B^4_{7\triangle x}(0,0,0,0)$ and $\varepsilon = 10^{-4},\ V_P = 2,\ V_E = 1,\ n = 50,\ n_c = 48+1$. The CPU time of Cristiani and Falcone was 1d 17h 27m 43s until they reached convergence in 108 iterations[18]. The TUM computation took 5d 7h 37m 20s to converge in 91 iterations. Since the original implementation has not been disclosed, no explanation can be given why Cristiani and Falcone have seen their CPU time more than doubled. The TUM server managed to do Test 4 quicker than Test 1, most likely because there are less nodes to be evaluated since some have been rendered illegal because of the obstacle.

Figure 4.9 shows another example of a dispersal surface where a different exploration of optimal controls leads to choosing the opposite direction as in the experiments of Cristiani and Falcone. Other than that the results match those presented in [1] quite well.

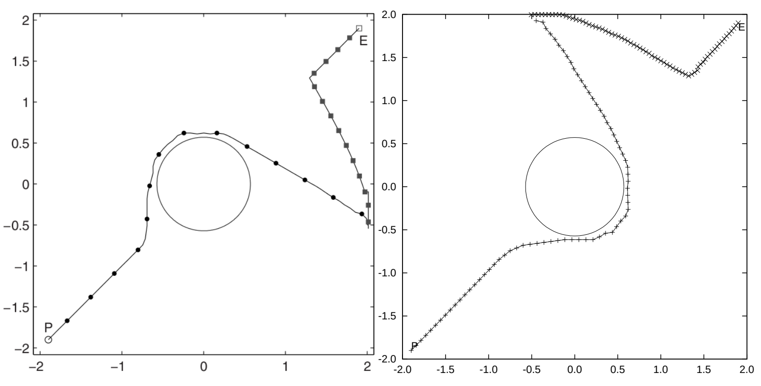

Figure 4.9: Trajectories for $P = (-1.9,-1.9)\,,\ E = (1.9,1.9)$

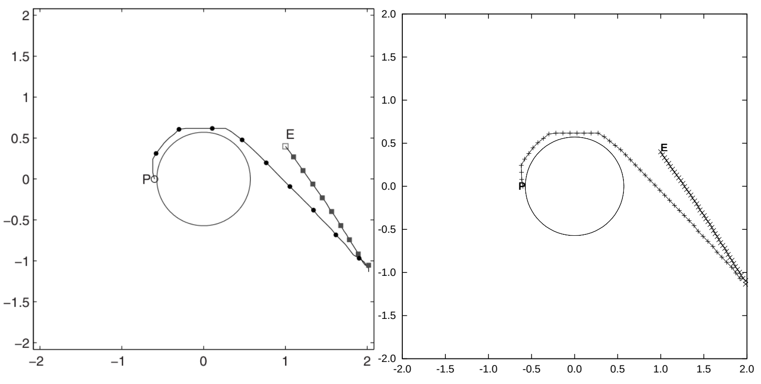

Figure 4.10: Trajectories for $P = (-0.6,0)\,,\ E = (1,0.4)$

[18][1], page 199

4.2 Experiments on discontinuous Value functions

Now the numeric approximation scheme is taken out of the comfort zone and applied on games where $V_p \geq V_E$, thus on Value Functions that are discontinuous on $\partial\mathcal{T}$. It is shown that in this case certain parameters influence the results quite much.

The initial observation has been made by comparing the results of Test 5 in [1]. The parameters are $\varepsilon = 10^{-3}$, $V_P = 1$, $V_E = 1$, $n = 50$, $n_c = 36$ and it is played on the $\overline{\Omega} = [-2,2]^4$. Since P and E will not be placed at the border of the domain, this setup matches Example 6 showing that v_h^k is discontinuous. A first observation on the Value functions depicted in Figures 4.11 and 4.12 show that both do not reach the height of the computation by Cristiani and Falcone. Especially the second graph lacks height as its plateau is located at 4 where the original value is around 8.

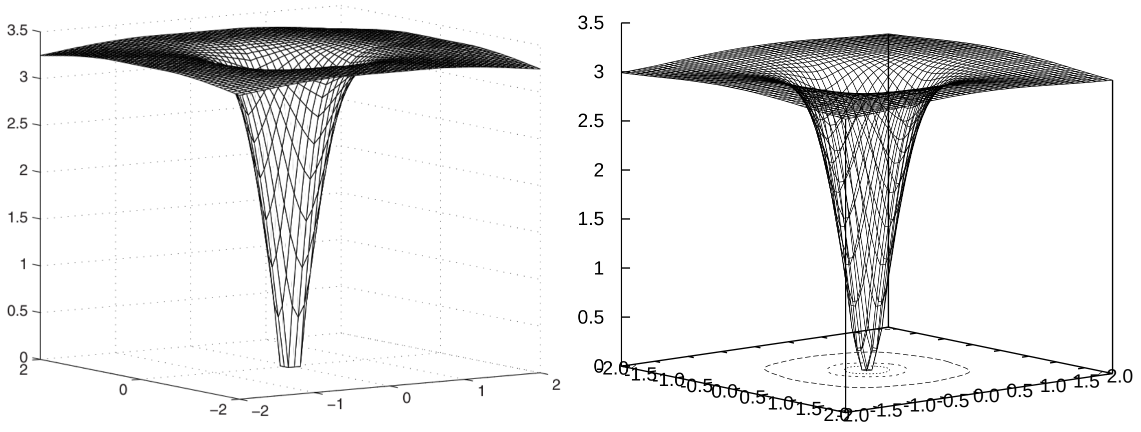

Figure 4.11: Value function $T\,(0, 0, x_E, y_E)$

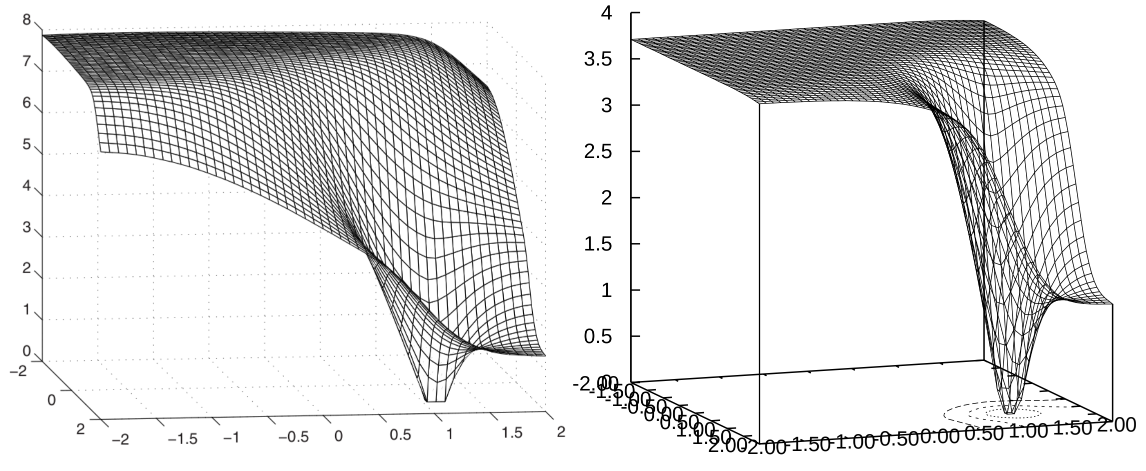

Figure 4.12: Value function $T\,(1.15, 1.15, x_E, y_E)$

These results were obtained by setting $v_h^k = 0$ as the starting point for the fixed point iteration. Cristiani and Falcone did not state their initial values for the fixed point iteration and from a purely theoretical point of view they did not have to. In Theorem 1 it is shown that the Fully-Discrete approximation scheme for the Value function is a fixed point iteration that will converge on the same fixed point regardless of the starting point. From a numeric perspective however it can be seen that the stop condition for the fixed point iteration (4.1) can kick in quite early thus leaving v_h^k still far away from the fixed point. A experiment with setting $v_h^k = 1$ before the fixed point iteration shows that this in fact raises the value of v_h^k at the end of the calculation. Figure 4.13 shows the Value functions for Test 5 with $v_h^k = 1$ as the starting value for the fixed point iteration. Now $T(0, 0, x_E, y_E)$ matches the calculation of Cristiani and Falcone where $T(1.15, 1.15, x_E, y_E)$ lies clearly above the original results.

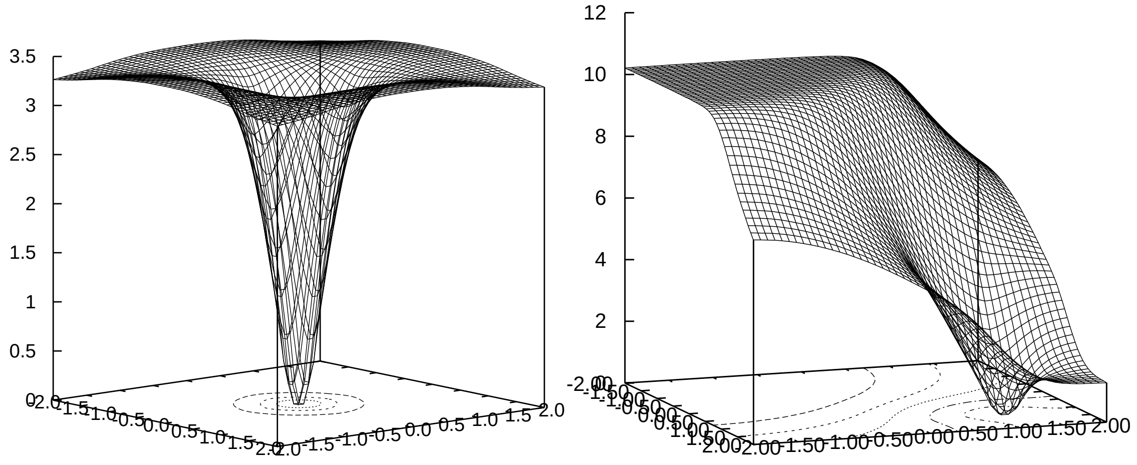

Figure 4.13: Value function $T(0, 0, x_E, y_E)$ and $T(1.15, 1.15, x_E, y_E)$ for the initial value $v_h^k = 1$

The trajectories the for Value functions starting at $v_h^k = 0$ or 1 show also interesting behavior. While Figure 4.14 shows no effect on trajectories starting at $P = (-1.9, -1.9)$ and $E = (-1.7, -1.9)$, they are heavily influenced if starting at $P = (1.3, 1.8)$ and $E = (0, 0)$ as can be seen in Figure 4.15.

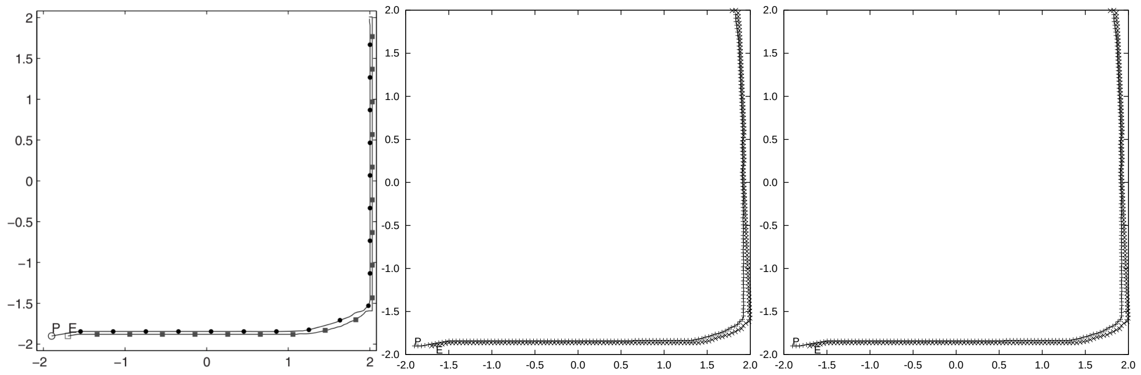

Figure 4.14: Trajectories for $P = (-1.9, -1.9)$ and $E = (-1.7, -1.9)$ from left to right by Cristiani and Falcone, for initial $v_h^k = 0$ and 1

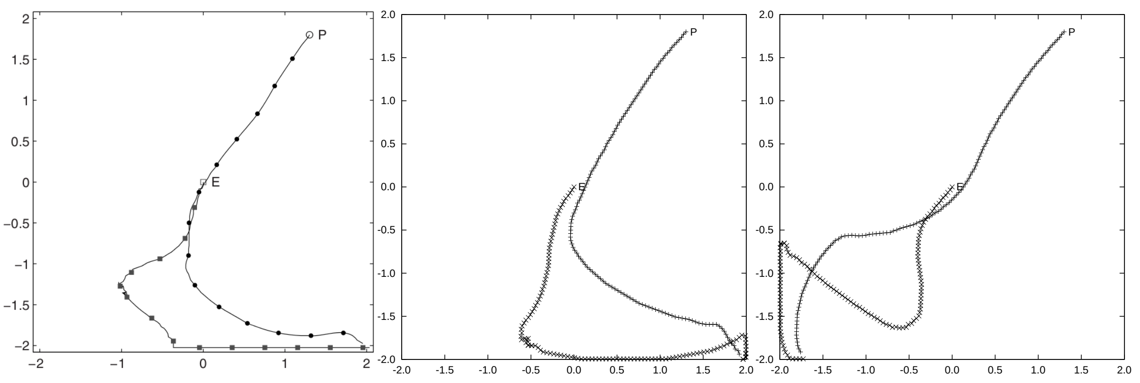

Figure 4.15: Trajectories for $P = (1.3, 1.8)$ and $E = (0, 0)$ from left to right by Cristiani and Falcone, for initial $v_h^k = 0$ and 1

The effect gets even worse as can be seen by comparing the results of Test 7 with Value function starting at $v_h^k = 0$ or 1 respectively. In this test $V_P < V_E$ therefore capture can no longer be guaranteed, the other parameters are $\varepsilon = 10^{-3}$, $V_P = 1$, $V_E = 1.25$, $n = 50$, $n_c = 48 + 1$. Figure 4.16 shows that especially the Value function obtained by having $v_h^k = 0$ at the start shows irregular behavior according to its result, capture could always occur. The Value function on the right for $v_h^k = 1$ at the beginning resembles the one by Cristiani and Falcone better but it still does not match it. Moreover it also does not tend to $+\infty$ on this domain[19]. The trajectories in Figure 4.17 show that Cristiani and Falcone can capture E as expected from $T(-1, -1, -1, 1) < \infty$ while E can evade capture in the TUM experiment on the right.

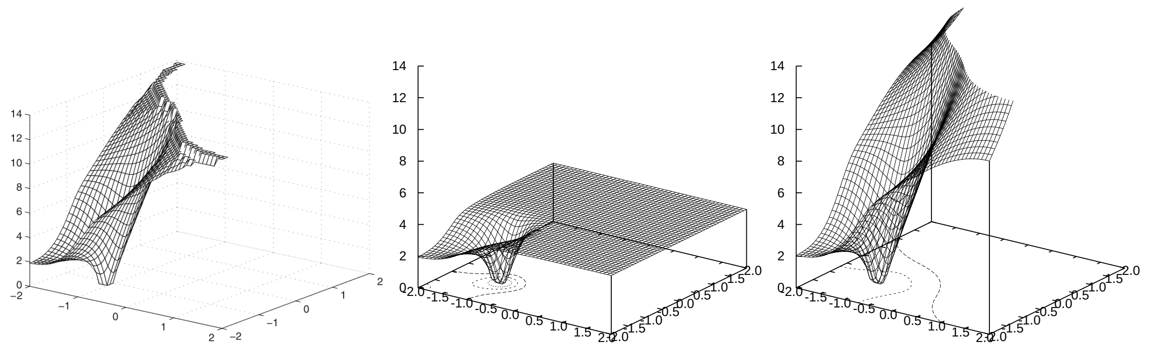

Figure 4.16: Value function for $T(-1, -1, x_E, y_E)$ from left to right by Cristiani and Falcone, for initial $v_h^k = 0$ and 1

In summary this leads to the conclusion that at least this implementation of the Fully-Discrete approximation scheme does not work very well outside of the valid domains. Further investigation would be necessary to determine if this is because of the numerical instability of the algorithm or implementation or because the problem is not well conditioned and therefore very sensitive to changes in the initial values and parameters.

[19]For a better comparability the plot has been cut off at 14 like the original. In the far corner its highest value is around 20.

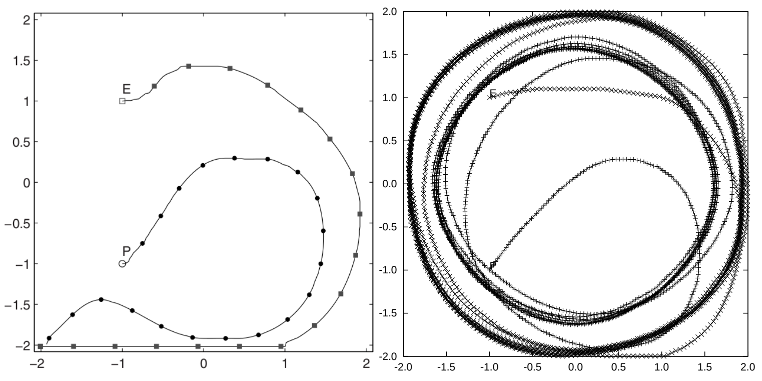

Figure 4.17: Trajectories for $T(-1,-1,-1,1)$ by Cristiani and Falcone (left) and TUM (right)

4.3 Influence of the Resolution on the Value function

At last, the influence of the resolution has been studied by performing a simulation for three different resolutions. The game is played on $\overline{\Omega} = [-2,2]^4$ and $V_P = 2$, $V_E = 1$ the parameters sets that changed are 1) $\varepsilon = 10^{-3}$, $n = 14$, $n_c = 8+1$, 2) $\varepsilon = 10^{-5}$, $n = 26$, $n_c = 36+1$ and 3) $\varepsilon = 10^{-3}$, $n = 50$, $n_c = 48+1$. The results of the tests are shown in Figure 4.18.

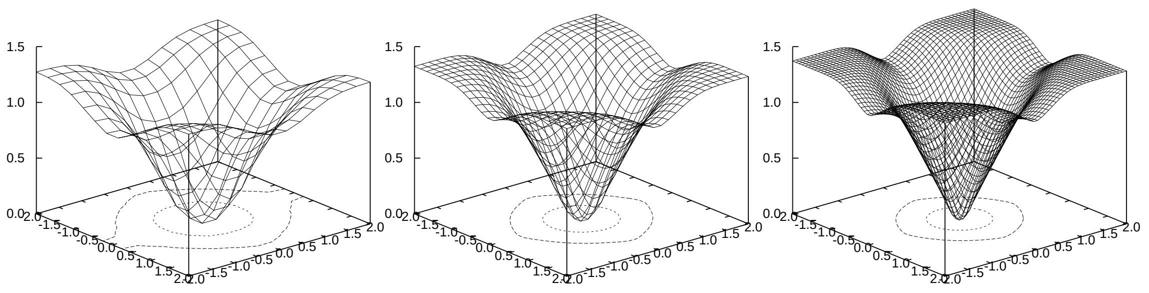

Figure 4.18: Value function $T(0,0,x_E,y_E)$ from left to right for parameter sets 1), 2) and 3)

E seems to be profiting more from the increase in resolution as the plateau in the corners of the domain grows in each plot. Also the radius of the "funnel" near the Position of P at $(0,0)$ seems to shrink, especially in the step from set 1) to set 2). But also the transition from 2) to 3) decreases the funnel as can be seen in the level sets in the x,y-plane. They mark the values 0.5 and 1.

5 Summary and Outlook

Looking back at the theoretical concepts from Theorems 1 and 2 it seems like they work quite well when executed on a machine if $V_P > V_E$. All the results from Cristiane and Falcone could be reproduced although the CPU time needed was up to 7 days. To much compared to the time of Cristiane and Falcone. Since the code spends most of his time performing a convex decomposition of points in the game domain, a clever optimization of that might be a good lever to drastically reduce runtime. The clever use of symmetry also is very important to save a lot of time as Section 3.3 shows. If $V_P \leq V_E$ the situation is quite different. In this case the theoretical results hold no longer true and the numeric experiments show to be very sensitive about the starting point of the fixed point iteration as was shown in Section 4.2.

This already leads to the first proposed research that can be done based on this results. It is not clear what causes this sensitivity, is it the numerical instability of the algorithm or implementation or may it be a ill conditioned problem. Also the exact order of convergence is still unknown[20]. With regards to the implementation, the convex decomposition algorithm should be further improved to reduce runtime. Also the software engineering could be improved by decoupling the `Game` class from the domain. Currently it expects a certain kind of symmetric domain. This assumption should be removed. It should not rely on this symmetry and not on the particular order in which the nodes are traversed. Instead it should only be fed nodes, perform the approximation on that node and return the new value. The domain than knows it's symmetry and how to handle that result. This refactoring would allow a greater variety of domains to be used with this implementation of the approximation scheme and ultimately even games that are not two player Pursuit-Evasion Games on the plane.

6 Acknowledgments

I would like to thank my parents for supporting me throughout my studies and always encouraging me to do my best and my colleagues at tado° for watching my back so I could take my time to write this thesis.

Special thanks also goes to my Thesis examiner Prof. Dr. Folkmar Bornemann for introducing me into Pursuit-Evasion games, providing me this interesting topic and his support and great patience. At last I want to thank Dr. Christian Ludwig for granting me access to the university server.

[20][1], page 178

References

[1] Cristiani E., Falcone M., Fully-Discrete Schemes for the Value Function of Pursuit-Evasion Games with State Constraints, in Bernhard P. et al (eds.), “Advances in Dynamic Games and Their Applications”, Annals of the International Society of Dynamic Games 10, 177-206, Birkhäuser, Boston, 2009

[2] Bardi M., Falcone M., Soravia P., Fully discrete schemes for the value function of pursuit-evasion games, in T. Başar and A. Haurie (eds.), “Advances in Dynamic Games and Application”, Annals of the International Society of Dynamic Games 1, 89-105, Birkhäuser, Boston, 1994

[3] Isaacs R., Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization, in R. F. Drenick, H. Hochstad, D. Gillette “The SIAM Series in Applied Mathematics”, John Wiley and Sons Inc., New York, London, Sydney, 1965

[4] Yang D., C++ and object-oriented numeric computing for scientists and engineers, Springer-Verlag, New York, 2001

[5] Başar T., Olsder G. J., Dynamic Noncooperative Game Theory, Academic Press, London, 1982

[6] Bardi M., Koike S., Soravia P., Pursuit-evasion games with state constraints: dynamic programming and discrete-time approximations, in Discrete and Continuous Dynamical Systems 6 (2000), 361-380