Solving High Dimensional Pursuit-Evasion Games with Dynamic Programming

by

Christopher Lee Grimm Jr

Submitted to the Department of Aeronautical and Astronautical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautical and Astronautical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

Author .....
Department of Aeronautical and Astronautical Engineering
May 18, 2016

Certified by .....
Sertac Karaman
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Paulo Lozano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Solving High Dimensional Pursuit-Evasion Games with Dynamic Programming

by

Christopher Lee Grimm Jr

Submitted to the Department of Aeronautical and Astronautical Engineering
on May 18, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautical and Astronautical Engineering

Abstract

The research presented in this paper details a new algorithm for solving high dimensional pursuit-evasion problems, called Best-Response Tensor-Train-decomposition-based Value Iteration(BR-TT-VI). This algorithm builds on concepts from game theory, dynamic programming(DP), and tensor-train (TT) decomposition. Various other methods used to solve pursuit-evasion games are often plagued with unreasonable computational times when the problem involves a state space that is defined by an excessive number of variables. This problem, called the curse of dimensionality, can be mitigated under certain circumstances by utilizing tensor-train decomposition. This work culminates in the development of the BR-TT-VI algorithm as well as its application to two different pursuit-evasion problems. First, a four-dimensional problem capable of being solved by traditional value iteration(VI) is tackled by the BR-TT-VI algorithm. This problem allows a direct comparison between VI and BR-TT-VI to demonstrate the reduced computational time of the new algorithm. Finally, BR-TT-VI is used to solve a six-dimensional problem that is impractical to solve with VI. The results of these examples demonstrate that BR-TT-VI can greatly reduce the computational time of solving problems while only permitting a modest reduction in solution accuracy.

Thesis Supervisor: Sertac Karaman
Title: Associate Professor of Aeronautics and Astronautics

Acknowledgments

I would like to thank my MIT advisor, Sertac Karaman, and my Draper advisor, Peter Chin, for all the support and guidance throughout my research as well as the opportunity to come to MIT to conduct this research. I would also like to thank Draper Laboratory for the financial support. My colleagues Ezra Tal,Alex Gorodetsky, and John Alora have provided numerous help when I have become stuck in solving a problem for which I am especially grateful. To the numerous teachers who have helped me reach here including Col Jeffery Butler, Col Anne Clark, Col Ed Kaplan, and Professor Marsh, thank you for challenging me when I was on top and helping me when I was not. To my siblings Matthew, Tyler, Gracie, Michael, and Kaitlyn, thank you for being the best friends I had growing up. I would like to thank my Dad, Chris Grimm Sr., for being my inspiration and guidance and I would like to thank my Mom, Mary Grimm, for the constant support and love. Finally, I would like to thank my beautiful wife Nicole for always being there; for loving me when times were trying and being the first to share in my enjoyment during success.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Control problems involving more than one vehicle often require the vehicles to either follow or avoid each other. When the target has worst-case or adversarial behavior, the vehicle can be described as pursuing or evading the target. These sorts of problems are called pursuit-evasion games and provide quiet the challenge to solve. The difficulty in solving these problems is inherent in how they involve two or more players attempting to meet their objectives while working against each other. This paper will discuss a new method for solving pursuit-evasion games called Best-Response Tensor-Train-decomposition-based Value Iteration(BR-TT-VI).

In order to discuss this method, this paper will provide information on the techniques that build into BR-TT-VI. In Chapter 1, a basic understanding of pursuit-evasion games and dynamic programming will be explored. Chapter 2 will explain two different methods that have been used to solve pursuit-evasion games. Tensors and tensor-train(TT) decomposition are pivotal to the success of BR-TT-VI. These things will be described in Chapter 3. In Chapter 4, it will be shown how tensor-based value iteration(TVI) was used to solve stochastic optimal control problems. TVI will finally be applied to pursuit-evasion games in Chapter 5. In this chapter, a basic format for modeling pursuit-evasion games as dynamic programming problems will be provided. The BR-TT-VI algorithm will also be presented in this chapter. In Chapter 6, the BR-TT-VI algorithm will be implemented on two different pursuit-evasion problems, a simple four-dimensional problem and a more advanced six-dimensional

problem. Finally, a summary of results, ideas for future research, and remarks on the importance of the work presented in this paper will be in Chapter 7.

1.1 Motivation for Solving Pursuit-Evasion Games

Pursuit-evasion games contain any number of problems in which one player, called the pursuer, attempts to capture some other player, called the evader. These problems have many variations including the number of pursuers or evaders, the dimensions in which the game is played in, or the manner in which a capture occurs. Pursuit-evasion games can even be used to model worst case scenarios of problems that typically are not pursuit-evasion games. For example given an autonomous car navigating a busy street, pursuit-evasion games can potentially model a safe path under the assumption that all the other cars are actively trying to run into the autonomous car. Due to this flexibility, pursuit-evasion games can be used to solve a variety of problems from an autonomous robot navigating a busy sidewalk full of pedestrians to smart munitions attempting to collide with an enemy unit.

Much like the subsequent example, pursuit-evasion games are especially prevalent in military problems due to the adversarial nature of the pursuer and evader. In his book on applications of differential games to warfare, Rufus Isaacs notes that the "various guises pursuit games may assume in warfare are legion" [7]. Surface-to-air missiles attempting to collide with aircraft, autonomous jets in a dogfight, and smart torpedoes tracking target ships are just a few of the warfare applications of pursuit-evasion games. As computers become better equipped to solve many of these pursuit-evasion games, autonomous weapons may provide the decisive edge in the future of warfare.

1.2 Description of Pursuit-Evasion Games

Pursuit-evasion problems involve two players controlling a dynamic system of the form:

$$x^{k+1} = f(x^k, u_1, u_2). \tag{1.1}$$

In these types of problems, x^k is the current state, u_1 is the control of the first player (the pursuer), u_2 is the control of the second player (the evader), and x^{k+1} is the new state based on the current state and the control of both players. Furthermore, each of the players have competing interests where they wish to move the state x into mutually exclusive target regions $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively. Because of this the pursuer and evader choose their strategies with accordance to some value function $V(x, u_1, u_2)$:

$$u_1^* = \underset{u_1}{\text{argmin}} V(x, u_1, u_2) \tag{1.2}$$

$$u_2^* = \underset{u_2}{\text{argmax}} V(x, u_1, u_2). \tag{1.3}$$

Often in pursuit-evasion games it is common to either set the state as the distance between the two players or to place some cost on the distance between the players. The pursuer then tries to reach the origin or point of lowest cost in the shortest amount of time. Meanwhile the evader either prolongs the state reaching this point for the longest amount of time or escapes the pursuer by increasing the distance to some specified value or reaching an escape region. Due to the nature that each player improves their cost by making an equal detriment to the cost of the other player, pursuit-evasion games are often modeled as zero-sum games. In games of this type, there exists a single cost function $G(x)$ where one player tries to maximize this function while the other one attempts to minimize it. This problem can also be modeled where each player tries to maximize or minimize their own cost function which is the negative of the opposing player's: $(G_1(x) = -G_2(x))$.

The ultimate outcome of the game can also be measured in a variety of ways. In some games, the only outcome that matters is whether the pursuer or evader achieves

either capture or escape. These pursuit-evasion games are called games of kind. For other games it is more important when or how the objective is met. These games of degree include games that want to minimize the time in which capture or escape occurs. [7]

Throughout this paper there will be a focus on zero-sum games of degree in which a single pursuer attempts to capture a single evader in the shortest time possible. The majority of this paper will detail how to solve these pursuit-evasion games.

1.3 Dynamic Programming

One method of solving pursuit-evasion games is to use dynamic programming(DP). This method attempts to solve control problems by providing a value for each state in a discretized state space. In a dynamic programming problem each state evolves according to:

$$x^{k+1} = f(x^k, u^k, w^k), \tag{1.4}$$

where k is the discrete time at which the control occurs, x^k is the current state of the system, u^k is the control, and w^k is a randomization parameter (this could be uncertainty in control or any other uncontrolled parameter). A problem in which w^k is some random value is called stochastic while a problem in which $w^k = 0\ \forall k$ is called deterministic. This paper focuses on problems that are deterministic and will often omit w^k.

In DP problems, at each state some cost $g^k(x^k, u^k)$ is incurred. Given that k takes a discrete number of time steps N, dynamic programming problems can be solved with:

$$g^N(x^N) + \sum_{k=0}^{N-1} g^k(x^k, u^k). \tag{1.5}$$

This equation can also be represented as a sequence of equations that determine an optimal cost at each time step J^k called the DP Algorithm:

$$J^N(x^N) = g^N(x^N), \tag{1.6}$$

$$J^k(x^k) = g^k(x^k, u^k) + J^{k+1}(f^k(x^k, u^k)). \tag{1.7}$$

This algorithm serves as the basis to solving problems with dynamic programming.[4]

Given a problem that ends after an undetermined number of time steps, there are multiple ways dynamic programming problems can still be solved. Two manners in which these types of problems can be solved include having a termination state with zero cost and using discounted problems. For problems with termination states, a series of these states are given a fixed and unchanging cost. Upon reaching any one of these states the problem arrives at completion and a final cost equal to the cost of the termination state is incurred. These termination states act in such a manner that some subsets of states x_T exist such that for all $x_T \in \mathcal{T}$:

$$g(x_T) = C, \tag{1.8}$$

where C is some constant value.

Progression to each successive state can also be given a discounting factor γ. For each state cost, as opposed to adding the entirety of the state cost of the following state,only a percentage of the subsequent state costs are incorporated into the current state cost.This transforms the DP algorithm from the one show in 1.6 and 1.7 to:

$$J^k(x^k) = g^k(x^k, u^k) + \gamma J^{k+1}(f^k(x^k, u^k)). \tag{1.9}$$

The logic behind using a discounted DP algorithm is that at each time step a percentage of the subsequent states have a cost of zero. As long as there is an upper bound on the value of anyone state, the discounted DP algorithm will converge. In order to solve problems of these types, also called infinite horizon problems, two different methods are generally used: value iteration and policy iteration.[4]

1.3.1 Value Iteration

Value iteration(VI) is a method of solving infinite horizon DP problems by determining the optimal state cost for every state. Given that the states are discretized to a

resolution h, where the discrete states are $z_i : z_i \in X$ and X is the continuous state space, a stochastic optimal control problem can be solved using value iteration. For VI an update function takes the form:

$$J^{k+1}(z_i) = \min_u [G(z_i, u) + \gamma \sum_j P(z_j|z_i, u) J^{(k)}(z_j)], \tag{1.10}$$

in which γ is the discount rate that ranges from $(0, 1)$. $J^{(k)}$ will converge to the optimal cost-to-go function as $k \to \infty$ giving the Bellman equation:

$$J^*(z_i) = \min_u [G(z_i, u) + \gamma \sum_j P(z_j|z_i, u) J^*(z_j)].$$

These equations can be modified for the deterministic case by setting $P(z_j|z_i, u) = 1$ where $z_j = f(z_i, u)$. Once the optimal cost for all states $J^*(z_i)$ has been determined, the optimal control can be ascertained by solving for $u^*(z_i)$:

$$u^*(z_i) = \operatorname*{argmin}_u [G(z_i, u) + \gamma \sum_j P(z_j|z_i, u) J^*(z_j)]. \tag{1.11}$$

1.3.2 Policy Iteration

Another method for solving infinite horizon DP problems is policy iteration. Policy iteration has the benefit of always terminating after a finite number of iterations. Where value iteration solves for the optimal cost at each state in order to determine the best control, policy iteration directly solves for the optimal control. Once again the states are discretized to a resolution h, where the discrete states are $z_i : z_i \in X$ and X is the continuous state space. After discretization, a stochastic optimal control problem can be solved using policy iteration and an initial policy u^0. First u^0 is evaluated in the policy evaluation step:

$$J_{u^k}(z_i) = G(z_i, u^k(z_i)) + \gamma \sum_j P(z_j|z_i, u^k(z_i)) J_{u^k}(z_j). \tag{1.12}$$

Next the policy is improved using the costs from the policy evaluation step. This step is called the policy improvement step:

$$u^{k+1}(z_i) = \underset{u}{\operatorname{argmin}}[G(z_i, u) + \gamma \sum_j P(z_j|z_i, u) J_{u^k}(z_j)]. \tag{1.13}$$

u^{k+1} will converge to the optimal policy u^* after a finite number of iterations. Once again these equations can be modified for the deterministic case by setting $P(z_j|z_i, u) = 1$ where $z_j = f(z_i, u)$.[4]

Chapter 2

Applications of Dynamic Programming and other methods for Solving Pursuit-Evasion Games

There are a variety of methods by which pursuit-evasion games can be solved. This chapter will explore two of those methods. The first method is applying dynamic programming in the case that the pursuit-evasion game is a boundary value problem (BVP). Bardi, Falcone, and Soravia detail a method they use to solve a one-dimensional pursuit-evasion game in "Fully Discrete Schemes for the Value Function of Pursuit-Evasion Games" [3]. A type of Rapidly-exploring Random Tree(RRT) algorithm called RRT* forms the basis of the second method. In "Incremental Sampling-Based Algorithms for a Class of Pursuit-Evasion Games", Karaman and Frazzoli detail how RRT* can be used to solve pursuit-evasion games.

2.1 One-Dimensional Boundary Value Problems

An attempt to solve pursuit-evasion games by dynamic programming was made by Martino Bardi, Maurizio Falcone, and Pierpaolo Soravia in the 1990's. In their paper, "Fully Discrete Schemes for the Value Function of Pursuit-Evasion Games," Bardi, Falcone, and Soravia present a method for solving pursuit-evasion games by discretiz-

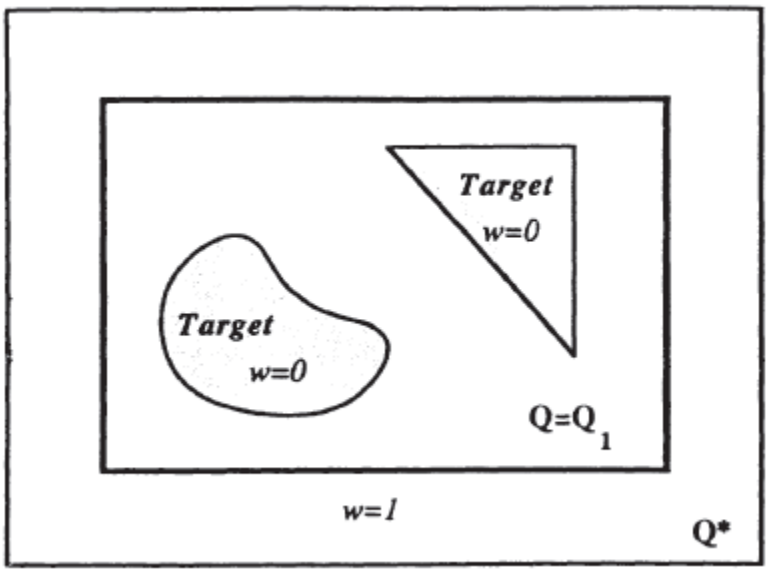


Figure 2-1: A state space divided into capture($w = 0$),escape($w = 1$), and regions of play($Q = Q_1$) [3]

ing the state space of a boundary value problem [3]. The first step in their scheme is to discretize the bounded state space. This bounded state space P is divided into a number of simplexes S_j that cover the entirety of the state space without overlapping. The vertices of these simplexes are the N discretized state nodes x_i. Furthermore, all states within the bounded state can be accounted for as a convex combination of the state nodes in S_j or:

$$x = \sum_{i=1}^{N} \lambda_i x_i \text{ where } \lambda_i \geq 0, \sum_{i=1}^{N} \lambda_i = 1, \lambda_i = 0 \text{ if } x_i \notin S_j \tag{2.1}$$

The state space is further discretized by adding points $z_i(a, b) \equiv x_i + hf(x_i, a, b) \in Q^*$, where h is some time step strictly greater than 0. These points are used to determine the optimal controls from each of the discretized nodes.

The discretized state space is then divided into regions such as those in Figure 2-1. For this problem, $\mathcal{T}$ denotes the nodes in the target region and is a subset of the nodes in Q. The target region is the area in which the pursuer captures the evader. Q_1 is the region of play and contains all nodes in Q that are not in the target region or the escape region. The escape region is denoted by Q_2 and may contain

nodes in Q and R^M, where R^M is the boundary of the bounded state space. All the nodes in the escape region represent states in which it is assumed that the evader has avoided capture by the pursuer. A common division of the state space is to set $\mathcal{T}$ = origin, $Q_1 = Q$, and $Q_2 = R^M$. The discretization of the state space results in a discrete version of the Hamilton-Jacobi-Isaacs equation for the boundary value problem(HJD):

$$w(x) = \sum_j \lambda_j w(x_j) \qquad \text{if } x = \sum_j \lambda_j x_j \tag{2.2}$$

$$w(x_i) = \gamma \max_b \min_a w(x_i + hf(x_i, a, b)) + 1 - \gamma \qquad \text{if } x_i \in Q \backslash \mathcal{T} \tag{2.3}$$

$$w(x_i) = 1 \qquad \text{if } x_i \in Q_2 \backslash Q \tag{2.4}$$

$$w(x_i) = 0 \qquad \text{if } x_i \in (\mathcal{T} \cap Q) \cup (R^M \backslash Q_2) \tag{2.5}$$

$$\text{where } \gamma = e^{-h} \tag{2.6}$$

Since the HJD is completely discretized into a finite number of states, it can easily be seen that by using dynamic programming $w(x_i)$ can be solved for any initial condition x_0. Furthermore, in a previous paper by Bardi and Soravia, "Hamilton-Jacobi equations with singular boundary conditions on a free boundary and applications to differential games,"[2] they are able to prove that the continuous boundary value problem:

$$v(x) + \min_{b \in B} \max_{a \in A} -f(x, a, b) \cdot Dv(x) - 1 = 0 \text{ in } R^M \backslash \mathcal{T} \equiv \mathcal{T}^C, \tag{2.7}$$

has a unique bounded viscosity solution v that meets the natural Dirichlet boundary condition:

$$v(x) = 0 \text{ for } x \in \delta\mathcal{T}, \tag{2.8}$$

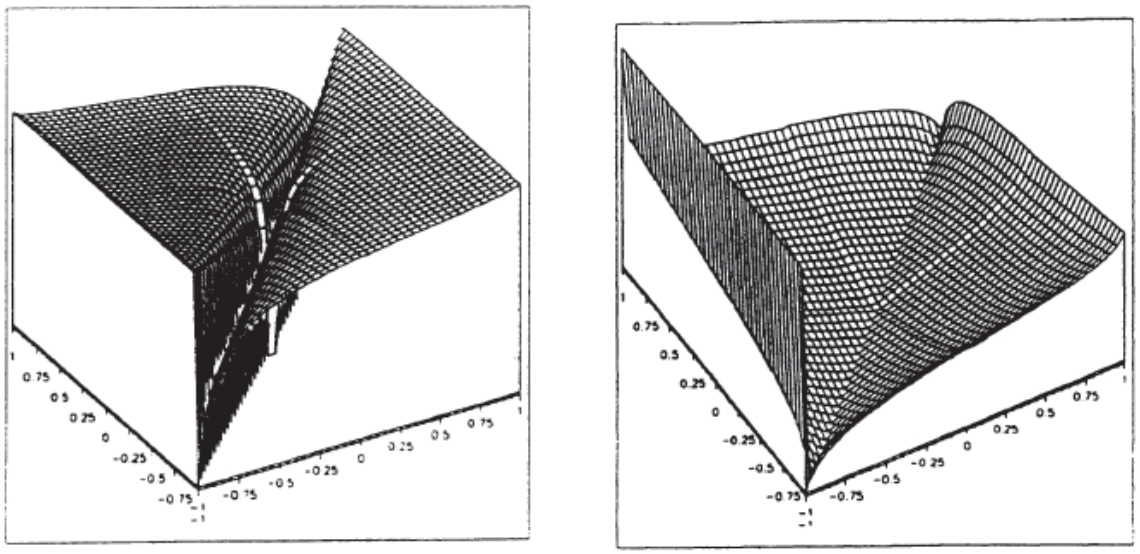
if v is continuous. They are further able to characterize this viscosity solution as:

$$v(x) \equiv \begin{cases} 1 - e^{-\mathrm{T}(x)}, & \text{if } \mathrm{T}(x) < +\infty, \\ 1, & \text{if } \mathrm{T}(x) = +\infty. \end{cases} \tag{2.9}$$

Denoting w_n as the unique solution to the HJD, Q^n as the discretization of the state space, h_n as the time step, and $|f(x_i, a, b)| \leq M_f$, Bardi, Falcone, and Soravia show that w_n will converge to the unique bounded viscosity solution v as Q^n and h_n become infinitely small. This is presented in:

Theorem 1 (Convergence of Bounded Viscosity Solution [3]) *Assume* $|f(x, a, b) - f(y, a, b)| \leq L|x - y|$ *for all* x, y, a, b; $|f(x_i, a, b)| \leq M_f$ *for all* $x \in \delta T$ *and all* a, b; T *is the closure of an open set with Lipschitz boundary ;* (Q^n, h_n) *is an admissible sequence ; there is a bounded continuous viscosity solution* v *of Equations* (2.7) *and* (2.8). *Then* w_n *converge to* v *as* $n \to \infty$, *uniformly on any compact set of* R^M.

In order to demonstrate the ability of these discretized boundary value problems, Bardi, Falcone, and Soravia implemented the system of Equations (2.2) to (2.6) with simple one-dimensional pursuit-evasion games. The results of these games can be seen in Figure 2-2.[3]



(a) Approximate value function when $v_1 = 1$, $v_2 = 1$

(b) Approximate value function when $v_1 = 5$, $v_2 = 1$

Figure 2-2: Results for one-dimensional pursuit-evasion games [3]

Bardi, Falcone, and Soravia's discrete method of solving pursuit-evasion games offers a number of advantages and disadvantages. As given in Theorem 1, their method

has the advantage of being mathematically sound. Since they focus on solving pursuit-evasion games with continuous viscosity solutions, they are able to prove that these games converge to a guaranteed optimal solution under certain conditions. A major disadvantage that arises from this is that the method does not guarantee convergence to an optimal solution if barriers or other factors that break the continuity of the solution exist. The restriction to a compact state space can also be problematic. Given too small of a state space the method might omit potential solutions that require leaving the state space. For example, given a pursuer with greater speed but less maneuverability it has been shown that an optimal strategy may require the pursuer to increase the distance between itself and the evader before making the capture [7]. Under these conditions the discrete boundary value problem may determine that under optimal control the evader can escape while in reality the pursuer has an optimal strategy that results in the capture of the evader.

The simple solution to this problem is to increase the state space. However, as the state space increases either the discretization must decrease or the number of discrete state nodes increases. This can be problematic when the state space encompasses a number of dimensions d that are evenly discretized into n states resulting in a total of n^d state nodes. In this case, any increase in the discretization of the state space results in order d polynomial growth of the state nodes. When d is large this is called the curse of dimensionality. Bardi, Falcone, and Soravia tested their algorithm for solving discrete boundary value problems on a state space composed of 1849 nodes. Given a state of just the x and y position of both the pursuer and the evader with each dimension discretized into twenty discrete values would result in a state space with a size of $20^4 = 1.6 \times 10^5$. Because of this further methods must be used to efficiently solve pursuit-evasion games.

2.2 RRT*

Besides dynamic programming, other methods have been used to solve pursuit-evasion games. Sertac Karaman and Emilio Frazzoli use a special version of Rapidly-exploring

Random Tree (RRT) algorithm called RRT* and Stackelberg strategies to solve pursuit-evasion games in "Incremental Sampling-Based Algorithms for a Class of Pursuit-Evasion Games" [8]. Due to the edge updating scheme used by RRT*, this method can be effectively used as a shortest path algorithm. Karaman and Frazzoli use this method to determine the shortest path to a goal region for an evader without being caught by multiple pursuers.

The RRT* algorithm is the cornerstone to Karaman and Frazzoli's ability to solve this pursuit-evasion problem. This algorithm can be used to determine the optimal shortest path as the number of iterations it is run for approaches infinity. To start the algorithm, a tree G composed of vertices and edges (V, E) is initialized to a single vertex at the vehicle's initial position z_{init}. Next, a single point z_{rand} is randomly sampled from the entirety of the continuous state space X. The nearest vertex $z_{nearest}$ to z_{rand} is then determined. A trajectory x_{new} between $z_{nearest}$ and z_{init} is formulated along with the controls u_{new} required to traverse the trajectory. If $z_{nearest}$ can be reached within some time t_{max} then t_{new} takes on the time required to traverse x_{new} to reach z_{init}. Otherwise, $t_{new} = t_{max}$. A new vertex z_{new} is now determined by following x_{new} for t_{new}. If x_{new} is obstacle free, then z_{new} is added to the list of vertices V.

If the edge between z_{new} and $z_{nearest}$ was added to the list of edges E, then this step along with the preceding steps would characterize the RRT algorithm. However, what makes RRT* special is the next two steps in determining edges. First, within some radius all the nearby vertices, Z_{near}, to z_{new} are used to determine the edge which reaches z_{new} in optimal time c_{min}. The edge between z_{new} and the optimal vertex z_{min} is then added to E. Secondly, all the nearby vertices that are not z_{min} are tested to determine if they can be reached in a more optimal time through z_{new}. If this is the case then the edge between z_{new} and the vertex z_{near} is added to E while the edge between z_{near} and the original parent, or vertex that connected z_{near} to z_{init}, is removed. This algorithm can be viewed in detail in Algorithms 1 and 2. [8]

The last two steps of RRT* are especially important because they convey upon the algorithm the eventuality of optimality that is not present in RRT. If the vertices

defined as nearby are all the vertices within a ball of volume $\gamma \frac{\log n}{n}$ centered at z_{new} and $n = |V|$, then RRT* will converge to optimality under the conditions of Theorem 2 given that $\mu(\cdot)$ is the Lebesgue measure.

Theorem 2 (Asymptotic Optimality of RRT* [8]) *If $\gamma > 2^d(1+1/d)\mu(X \backslash Xi_{obs})$, the event that for any vertex z that is in the tree in some finite iteration j the RRT* algorithm converges to a trajectory that reaches z optimally, i.e., in time $T^*(z)$, occurs with probability one. Formally,*

$$\mathbb{P}(\{\lim_{i \to \infty} T(z)[i+j] = T^*(z), \forall z \in V[j]\}) = 1, \qquad \forall j \in \mathbb{N}.$$

Algorithm 1 RRT* [8]

```
V ← z_init; E ← ; i ← 0;
while i < N do
    G ← (V, E);
    z_rand ← Sample(i);
    (V, E, z_new) ← Extend(G, z_rand);
    i ← i + 1;
end while
```

Karaman and Frazzoli further apply the RRT* algorithm to solve pursuit-evasion games. In order to do this, they implement a Stackelberg strategy in which the evader first choses its controls and then the pursuers follow the evader by choosing their controls to counter the strategy of the evader. These types of strategies provide for a worst case scenario for the evader. Therefore, if the RRT* pursuit-evasion algorithm finds a winning strategy for the evader this strategy will always result in a win for the evader no matter what strategy the pursuers choose.

The pursuit-evasion algorithm modifies upon RRT* by creating trees for both the evader G_e and the pursuer G_p. Besides creating two separate trees, the pursuit-evasion algorithm also checks all the nearby vertices of the other player. If the new evader vertex can be reached in less time by a nearby pursuer vertex or a new pursuer vertex can reach nearby evader vertices in less time, then those evader vertices along with their connecting edges are removed from G_e. This algorithm can be examined

Algorithm 2 Extend(G, z) [8]

$V' \leftarrow V; E' \leftarrow E$;
$z_{nearest} \leftarrow \text{Nearest}(G, z)$;
$(x_{new}, u_{new}, t_{new}) \leftarrow \text{Steer}(z_{nearest}, z); z_{new} \leftarrow x_{new}(t_{new})$;
if ObstacleFree(x_{new}) **then**
    $V' \leftarrow V' \cup \{z_{new}\}$;
    $z_{min} \leftarrow z_{nearest}; c_{min} \leftarrow T(z_{new})$;
    $Z_{nearby} \leftarrow \text{Near}(G, z_{new}, |V|)$;
    for all $z_{near} \in Z_{nearby}$ **do**
        $(x_{near}, u_{near}, t_{near}) \leftarrow \text{Steer}(z_{near}, z_{new})$;
        if ObstacleFree(x_{near}) and $x_{near}(t_{near}) = z_{new}$ and $T(z_{near}) +$ EndTime$(x_{near}) < T(z_{new})$ **then**
            $c_{min} \leftarrow T(z_{near}) + \text{EndTime}(x_{near})$;
            $z_{min} \leftarrow z_{near}$;
        end if
    end for
    $E' \leftarrow E' \cup \{(z_{min}, z_{new})\}$;
    for all $z_{near} \in Z_{nearby}\ \{z_{min}\}$ **do**
        $(x_{near}, u_{near}, t_{near}) \leftarrow \text{Steer}(z_{new}, z_{near})$;
        if ObstacleFree(x_{near}) and $x_{near}(t_{near}) = z_{near}$ and $T(z_{near}) > T(z_{new}) +$ EndTime(x_{near}) **then**
            $z_{parent} \leftarrow \text{Parent}(z_{near})$;
            $E' \leftarrow E'\ \{(z_{parent}, z_{near})\}; E' \leftarrow E' \cup \{(z_{new}, z_{near})\}$;
        end if
    end for
else
    $z_{new} = \text{NULL}$;
end if
return $G' = (V', E', z_{new})$

in detail in Algorithm 3.[8]

Algorithm 3 Pursuit-Evasion RRT* [8]

$V_e \leftarrow x_{e,init}; E_e \leftarrow ; V_p \leftarrow x_{p,init}; E_p \leftarrow ; i \leftarrow 0;$
while $i < N$ **do**
  $G_e \leftarrow (V_e, E_e); G_p \leftarrow (V_p, E_p)$
  $z_{e,rand} \leftarrow \text{Sample}_e(i);$
  $(V_e, E_e, z_{e,new}) \leftarrow \text{Extend}_e(G_e, z_{e,rand});$
  if $z_{e,new} \neq$ NULL **then**
    $Z_{p,near} \leftarrow \text{NearCapture}_e(G_p, z_{e,new}, |V_p|);$
    for all $z_{p,near} \in Z_{p,near}$ **do**
      if $\text{Time}(z_{p,near}) \leq \text{Time}(z_{e,new})$ **then**
        $\text{Remove}(G_e, z_{e,new});$
      end if
    end for
  end if
  $z_{p,rand} \leftarrow \text{Sample}_p(i);$
  $(V_p, E_p, z_{p,new}) \leftarrow \text{Extend}_p(G_p, z_{p,rand});$
  if $z_{p,new} \neq$ NULL **then**
    $Z_{e,near} \leftarrow \text{NearCapture}_p(G_e, z_{p,new}, |V_e|);$
    for all $z_{e,near} \in Z_{e,near}$ **do**
      if $\text{Time}(z_{p,new}) \leq \text{Time}(z_{e,near})$ **then**
        $\text{Remove}(G_e, z_{e,near});$
      end if
    end for
  end if
  $i \leftarrow i + 1;$
end while
return G_e, G_p

In order to demonstrate the abilities of the pursuit-evasion RRT*, Karaman and Frazzoli implement this algorithm on two different examples. In the first example, simplified dynamics are used. For the evader, the dynamics are:

$$\frac{d}{dt}x_e(t) = \frac{d}{dt}\begin{bmatrix} x_{e,1}(t) \\ x_{e,2}(t) \end{bmatrix} = u_e(t) = \begin{bmatrix} u_{e,1}(t) \\ u_{e,2}(t) \end{bmatrix},$$

with $\|u_e(t)\|_2 \leq 1$, while the pursuers have the following dynamics:

$$\frac{d}{dt}x_p(T) = \frac{d}{dt}\begin{bmatrix} x_{p_1}(t) \\ x_{p_2}(t) \\ x_{p_3}(t) \end{bmatrix} = \frac{d}{dt}\begin{bmatrix} x_{p_1,1}(t) \\ x_{p_1,2}(t) \\ x_{p_2,1}(t) \\ x_{p_2,2}(t) \\ x_{p_3,1}(t) \\ x_{p_3,1}(t) \end{bmatrix} = \begin{bmatrix} u_{p_1}(t) \\ u_{p_2}(t) \\ u_{p_3}(t) \end{bmatrix} = \begin{bmatrix} u_{p_1,1}(t) \\ u_{p_1,2}(t) \\ u_{p_2,1}(t) \\ u_{p_2,2}(t) \\ u_{p_3,1}(t) \\ u_{p_3,1}(t) \end{bmatrix},$$

with $\|u_{p_1}(t)\|_2 \leq 1$, $\|u_{p_2}(t)\|_2 \leq 0.5$, and $\|u_{p_3}(t)\|_2 \leq 0.5$. The second example uses Dubins dynamics where the evader can be modeled by the following differential equations:

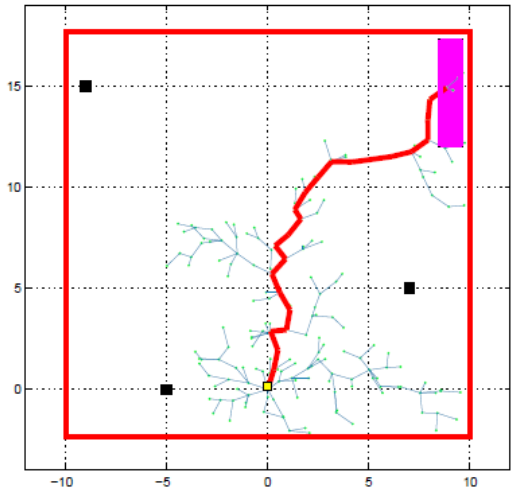
$$\begin{aligned}
&x_e(t) = [x_{e,1}(t), x_{e,2}(t), x_{e,3}(t), x_{e,4}(t), x_{e,5}(t)]^T, \\
&f(x_e(t), u_e(t)) = [v_e\cos(x_{e,3}(t)), v_e\sin(x_{e,3}(t)), u_{e,1}(t), x_{e,5}(t), u_{e,2}(t)]^T, \\
&\dot{x}_e(t) = f(x_e(t), u_e(t)), \\
&v_e = 1, |u_{e,1}(t)| \leq 1, |u_{e,2}(t)| \leq 1, |x_{e,5}| \leq 1.
\end{aligned}$$

The pursuer also follows Dubins dynamics with the following differential equations:

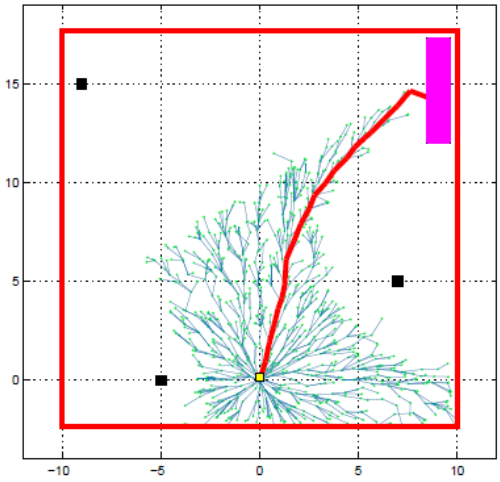
$$\begin{aligned}
&x_p(t) = [x_{p,1}(t), x_{p,2}(t), x_{p,3}(t), x_{p,4}(t), x_{p,5}(t)]^T, \\
&f(x_p(t), u_p(t)) = [v_p\cos(x_{p,3}(t)), v_p\sin(x_{p,3}(t)), u_{p,1}(t), x_{p,5}(t), u_{p,2}(t)]^T, \\
&\dot{x}_p(t) = f(x_p(t), u_p(t)), \\
&v_p = 2, |u_{p,1}(t)| \leq 1/3, |u_{p,2}(t)| \leq 1, |x_{p,5}| \leq 1.
\end{aligned}$$

The results of these examples can be seen in Figures 2-3 to 2-5.[8]

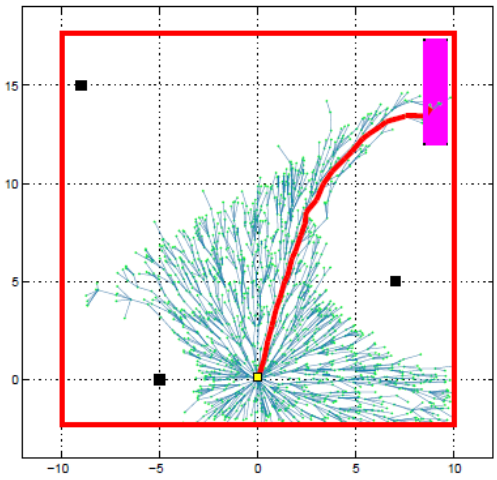
Just as with dynamic programming, there are numerous advantages and disadvantages to using RRT* to solve pursuit-evasion games. The computational speed of RRT* is the biggest advantage to using this algorithm. 10000 iterations of the simple dynamics example were computed in about 3 seconds, while 3000 iterations



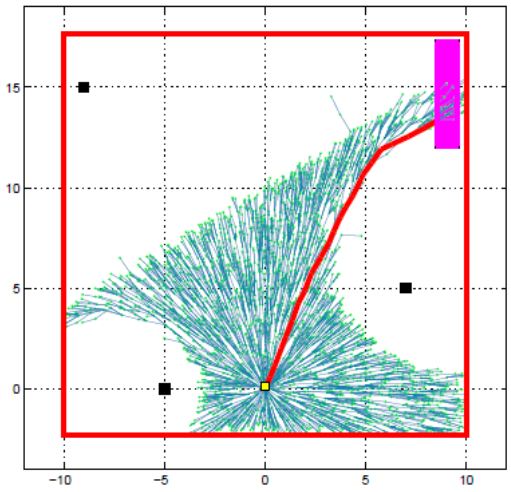
(a) 500 iterations



(b) 3000 iterations

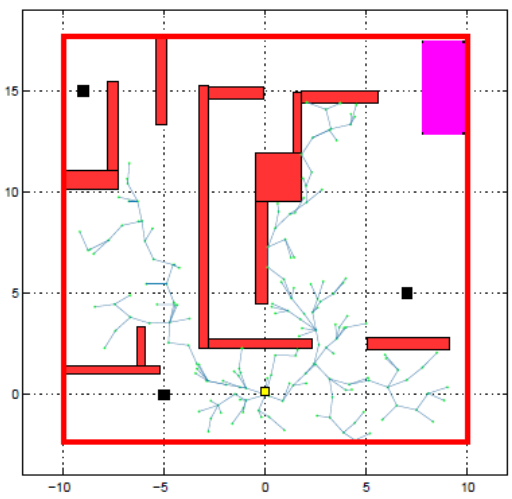


(c) 5000 iterations

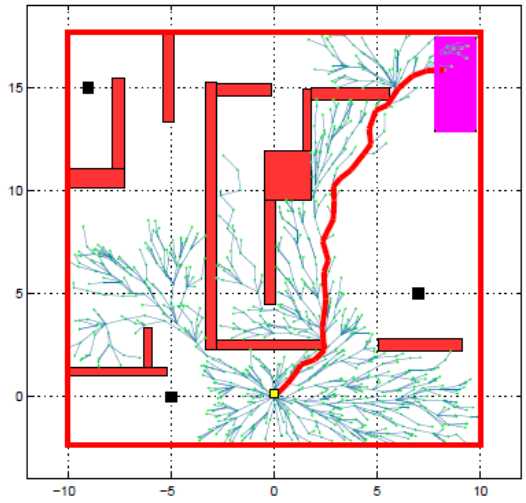


(d) 10000 iterations

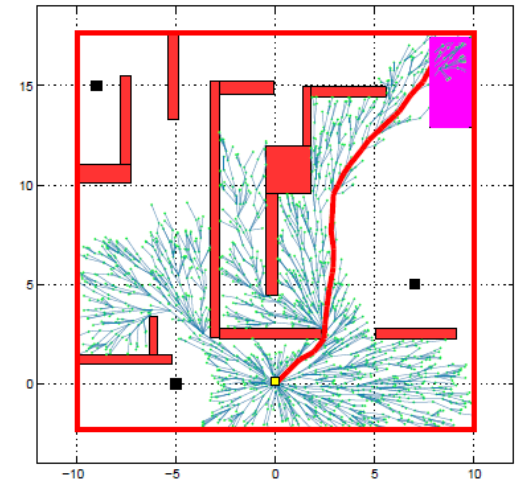
Figure 2-3: Pursuit-Evasion RRT* at various iterations [8]



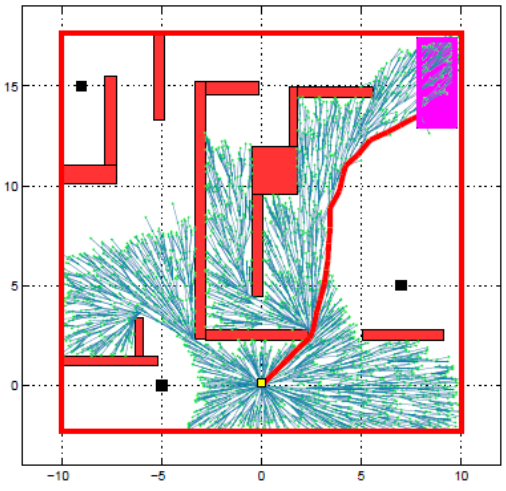
(a) 500 iterations



(b) 3000 iterations

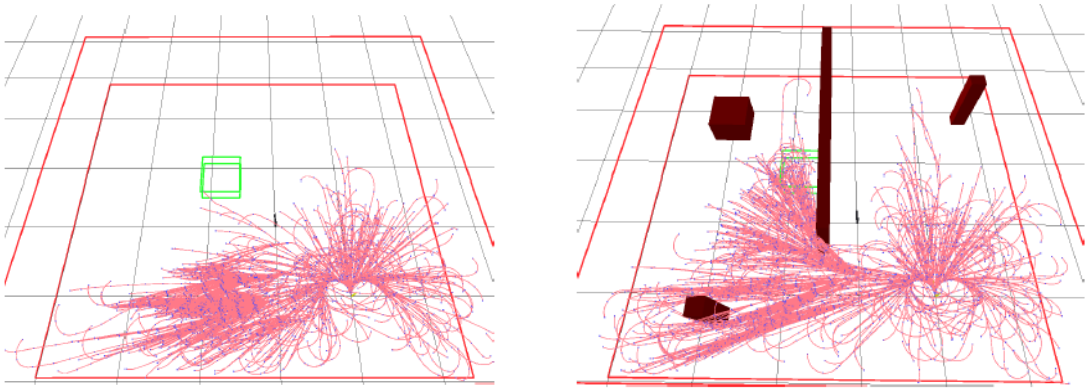


(c) 5000 iterations



(d) 10000 iterations

Figure 2-4: Pursuit-Evasion RRT* in a field with obstacles at various iterations[8]



(a) Pursuit-Evasion RRT* run for 3000 iterations

(b) Pursuit-Evasion RRT* run for 3000 iterations in a field with obstacles

Figure 2-5: Pursuit-Evasion RRT* on a problem with Dubins dynamics [8]

of the Dubins dynamics example were computed in about 20 seconds [8]. Unlike the dynamic programming method presented in Section 2.1, RRT* also handles obstacles very well as can be seen in Figures 2-4 and 2-5b.

The major disadvantage of RRT* is in its inability to handle sub-optimality very well. If either the pursuer or evader ends up taking suboptimal actions, the results of the RRT* algorithm could be completely useless. For example, suppose that the evader moves into the region of potential pursuer capture by accident. However, the pursuer is elsewhere expecting evader optimal control. In this case the only way to determine a new path for the evader is to recompute the entire solution.

Chapter 3

Tensor-Train Decomposition

One method of handling the curse of dimensionality is to reduce the number of states that must be examined. If a problem has a low-rank representation, tensor-train decomposition may be used to reduce the number of states employed to compute the solution to a problem. Tensor-train decomposition, as outlined in Oseledets's "Tensor-train decomposition" and "TT-cross approximation for multidimensional arrays," takes advantage of the structure of tensors to reduce the state space [9, 10]. This method takes advantage of single value decomposition (SVD) to create compact tensors. The start of this chapter will provide a description of tensors in general. Next, the chapter will explain how tensor-train decomposition works and under what circumstances this method works best. Finally, a series of methods for maintaining low-rank will be provided along with the TT-SVD algorithm.

3.1 Tensor-Train Decomposition

Tensors are another way of representing a multidimensional array. Given such an array or tensor $\mathbf{A}$, the element at $A(i_1, i_2, ..., i_d)$, where d is the dimensionality of the tensor and i_j is the i^{th} entry of the j^{th} dimension of the tensor, can be given by:

$$A(i_1, ..., i_d) = \sum_{\alpha=1}^{r} U_1(i_1, \alpha) U_2(i_2, \alpha) ... U_d(i_d, \alpha). \tag{3.1}$$

The d matrices $U_k = [U_k(i_k, \alpha)]$ are called canonical factors and the value r is the tensor rank. The tensor rank represents the minimal number of summands required to represent the entire tensor $\mathbf{A}$ in the form of Equation (3.1).

A tensor $\mathbf{B}$ can be approximated with a tensor-train $\mathbf{A}$ created from the product of a series of matrices:

$$A(i_1, i_2, ..., i_d) = G_1(i_1)G_2(i_2)...G_d(i_d). \tag{3.2}$$

$G_k(i_k)$, also called a tensor core, is an $r_{k-1} \times r_k$ matrix where $r_0 = r_d = 1$ is imposed. These tensor cores can also be represented by a three-dimensional array of size $r_{k-1} \times n_k \times r_k$ that has elements $G_k(\alpha_{k-1}, i_k, \alpha_k) = G_k(i_k)_{\alpha_{k-1},\alpha_k}$. Therefore any individual element of a tensor-train can be calculated by:

$$A(i_1, ..., i_d) = \sum_{\alpha_0,...,\alpha_{d-1},\alpha_d} G_1(\alpha_0, i_1, \alpha_1)G_2(\alpha_1, i_2, \alpha_2)...G_d(\alpha_{d-1}, i_d, \alpha_d) \tag{3.3}$$

As can be seen in Equation (3.3), in order to compute the individual elements of the tensor all the tensor cores are multiplied and then summed over the auxiliary indices a_k. It is from this process that the decomposition gets the name tensor-train. Tensor-train decomposition makes it possible to represent all elements in the tensor $\mathbf{A}$ with d three-dimensional arrays.[9]

Since the size of the tensor cores are determined by ranks r_k, determining these ranks are pivotal to the compressibility of tensor-train decomposition. The rank r_k is dependent on the rank of the k^{th} unfolding matrix A_k which has the form:

$$A_k = A_k(i_1, ..., i_k; i_{k+1}, ..., i_d) = A(i_1, ..., i_d). \tag{3.4}$$

These unfolding matrices can be determined by the MATLAB reshape function as such:

$$A_k = \text{reshape}(\mathbf{A}, \prod_{s=1}^{k} n_s, \prod_{s=k+1}^{d} n_s).$$

By determining the rank of these unfolding matrices, a bound on the tensor-train

ranks can be determined with respect to Theorem 3. [9]

Theorem 3 (Bound on TT-Ranks [9]) *If for each unfolding matrix A_k of form 3.4 of a d-dimensional tensor $\boldsymbol{A}$*

$$\text{rank } A_k = r_k,$$

then there exists a decomposition 3.3 with TT-ranks not higher than r_k.

3.2 Methods of Maintaining Low Order

Given certain tensors it may be the case that the unfolding matrices do not meet the low rank requirements of tensor-train decomposition. The unfolding matrices can be replaced by approximate low rank unfolding matrices R_k such that:

$$A_k = R_k + E_k, \text{rank } R_k = r_k, \|E_k\| = \varepsilon_k, k = 1, ..., d-1. \tag{3.5}$$

This results in a new tensor $\mathbf{B}$ where the norm of the difference between the new tensor and the old tensor is bounded by an error constant as given by:

Theorem 4 (TT Approximation [9]) *Suppose that the unfoldings A_k of the tensor $\boldsymbol{A}$ satisfy 3.5. Then TT-SVD computes a tensor $\boldsymbol{B}$ in the TT-format with TT-ranks r_k and*

$$\|\boldsymbol{A} - \boldsymbol{B}\| \leq \sqrt{\sum_{k=1}^{d-1} \varepsilon_k^2}.$$

The results of Theorem 4 can be used to create a tensor in TT-format as detailed in Algorithm 4.

The largest benefit of tensor-trains is that they greatly reduce the complexity of high-dimensional calculations. Generally these computations have complexity that is linear with dimension but polynomially with rank. However, many of these tensor-train computations have the unfortunate effect of increasing the rank of the final tensor-train. For example, adding two tensor-trains with the same rank takes next

Algorithm 4 TT-SVD [9]

Require: d-dimensional tensor $\mathbf{A}$; Prescribed accuracy ε.

Ensure: Cores $G_1, ..., G_d$ of the TT-approximation $\mathbf{B}$ to $\mathbf{A}$ in the TT-format with TT-ranks $\hat{r_k}$ equal to the δ-ranks of the unfoldings A_k of $\mathbf{A}$, where $\delta = \frac{\varepsilon}{\sqrt{d-1}}\|A\|_F$.
The computed approximation satisfies

$$\|\mathbf{A} - \mathbf{B}\|_F \leq \varepsilon \|\mathbf{A}\|_F.$$

1: {Initialization}
Compute truncation parameter $\delta = \frac{\varepsilon}{\sqrt{d-1}}\|A\|_F$.
2: Temporary tensor: $\mathbf{C} = \mathbf{A}, r_0 = 1$.
3: **for** $k = 1$ to $d-1$ **do**
4: $C := \text{reshape}(C, [r_{k-1}n_k, \frac{\text{numel}(C)}{r_{k-1}n_k}])$.
5: Compute δ-truncated SVD: $C = USV + E, \|E\|_F \leq \delta, r_k = \text{rank}_\delta(C)$.
6: New core: $G_k := \text{reshape}(U, [r_{k-1}, n_k, r_k])$.
7: $C := SV^T$.
8: **end for**
9: $G_d = C$
10: Return tensor $\mathbf{B}$ in TT-format with cores $G_1, ..., G_d$.

to no computations but doubles the number of ranks in the resulting tensor-train. Taking the scalar product of two tensor-trains takes $O(dnr^4)$ computations but results in $O(r^2)$ ranks. In order to keep the tensor-train low-rank, TT-rounding can be used to create a new tensor $\mathbf{B}$ with δ-ranks such that $\delta = \frac{\varepsilon}{\sqrt{d-1}}\|A\|$. This new approximation maintains an ε-level accuracy and only requires $O(dnr^2 + dr^4)$ computations:

$$\|\mathbf{A} - \mathbf{B}\| \leq \varepsilon \|\mathbf{A}\|. \tag{3.6}$$

A detailed explanation for TT-rounding can be found in Algorithm 5. [9]

[9, 10] can be referenced for detailed algorithms and further proofs.

Algorithm 5 TT-Rounding [9]

Require: d-dimensional tensor $\mathbf{A}$ in the TT-format; Required accuracy ε.

Ensure: $\mathbf{B}$ in the TT-format with TT-ranks $\hat{r_k}$ equal to the δ-ranks of the unfoldings A_k of $\mathbf{A}$, where $\delta = \frac{\varepsilon}{\sqrt{d-1}}\|A\|_F$. The computed approximation satisfies

$$\|\mathbf{A} - \mathbf{B}\|_F \leq \varepsilon \|\mathbf{A}\|_F.$$

Let $G_k, k = 1, ..., d$, be cores of $\mathbf{A}$.
{Initialization}
Compute truncation parameter $\delta = \frac{\varepsilon}{\sqrt{d-1}}\|A\|_F$.
{Right-to-left orthogonalization}
for $k = d$ to 2 step -1 **do**
    $[G_k(\beta_{k-1}; i_k\beta_k), R(\alpha_{k-1}, \beta_{k-1})] := QR_{rows}(G_k(\alpha_{k-1}; i_k\beta_k))$.
    $G_{k-1} := G_k \times_3 R$
end for
{Compression of the orthogonalized representation}
for $k = 1$ to $d - 1$ **do**
    {Compute δ-truncated SVD}
    $[G_k(\beta_{k-1}i_k; \gamma_k), \Lambda, V(\beta_k, \gamma_k)] := SVD_\delta[G_k(\beta_{k-1}i_k; \beta_k)]$.
    $G_{k+1} := G_{k+1} \times_1 (V\Lambda)^T$
end for
Return $G_k, k = 1, ..., d$, as cores of $\mathbf{B}$.

Chapter 4

Tensor-based Value Iteration

The advantages of tensor-train decomposition have already been applied to solving optimal control problems with dynamic programming. Gorodetsky, Karaman, and Marzouk have come up with a version of value iteration that takes advantage of tensor-train decomposition in "Efficient High-Dimensional stochastic Optimal Motion Control using Tensor-Train Decomposition" [6]. They call this new version of value iteration Tensor-based value iteration(TVI). This chapter will begin by providing a quick background into stochastic optimal motion control for which Gorodetsky et.al designed TVI. Next, the TVI algorithm will be provided in detail. The chapter will end with an example problem Gorodetsky et. al. solved in [6] along with their results.

4.1 Stochastic Optimal Motion Control

Gorodetsky, Karaman, and Marzouk focus on problems of stochastic optimal motion control. While deterministic pursuit-evasion games have many similarities to problems of this type, a brief explanation will be given for problems of stochastic optimal motion control in order to fully appreciate the results of TVI when applied to these problems. In problems of stochastic optimal motion control some system has the differential form:

$$dx(t) = b(x(t), u(t))dt + F(x(t), u(t))dw(t), \tag{4.1}$$

in which $d, d_u, d_w \in Z_+, X \subset R^d$ and $U \subset R^{d_u}$ are compact sets with smooth boundaries and non-empty interiors. $w(t) : t \geq 0$ is a d_w-dimensional Brownian motion defined on some probability space $(\omega, \mathcal{F}, P)$, where ω is a sample space. $\mathcal{F}$ is σ-algebra, and P is a probability measure while $b : X \times U \to R^d$ and $F : X \times U \to R^{d \times d_w}$ are measurable, continuous, and bounded functions as detailed in [6].

Just as with the discrete boundary value problem in [3], the first step is to discretize the state space. The state space is discretized using discrete Markov Decision Processes (MDPs) that follow the local consistency conditions as given by:

Theorem 5 (Local Consistency Conditions [6]) *Suppose the sequence $M_l : l \in N$ of MDPs and the sequence $\Delta t_l : l \in N$ holding times satisfy the following conditions: For any sequence of inputs $u_i^l : i \in N$ and the resulting sequence of trajectories $\xi_i^l : i \in N$,*

- *for all $z \in X$,*

$$\lim_{l \to \infty} \Delta t_l(z) = 0,$$

- *for all $z \in X$ and $v \in U$,*

$$\lim_{l \to \infty} \frac{E[\xi_{i+1}^l - \xi_i^l | \xi_i^l = z, u_i^l = v]}{\Delta t_l(z)} = f(z, v),$$

$$\lim_{l \to \infty} \frac{Cov[\xi_{i+1}^l - \xi_i^l | \xi_i^l = z, u_i^l = v]}{\Delta t_l(z)} = F(z, v),$$

Then, the sequence $(\xi_i^l, u^l) : l \in N$ of interpolations converges in distribution to (x, u) that solves the integral equation with differential form given by 4.1. Let J_l^* denote the optimal cost-to-go function for the MDP M_l. Then, for all $z \in S$,

$$\lim_{l \to \infty} |J_l^*(z) - J^*(z)| = 0.$$

Given that the states are discretized to a resolution h, where the discrete states are $z_i : i \in l$, the stochastic optimal control problem can be solved using value iteration

and the update function:

$$J_h^{k+1}(z_i) = \min_u [G(z_i, u) + \gamma_h \sum_j P(z_j|z_i, u) J_h^{(k)}(z_j)], \tag{4.2}$$

in which γ_h is the discount rate that ranges from $(0, 1)$. $J_h^{(k)}$ will converge to the optimal cost-to-go function as $k \to \infty$ giving the Bellman equation:

$$J_h^*(z_i) = \min_u [G(z_i, u) + \gamma_h \sum_j P(z_j|z_i, u) J_h^*(z_j)].[6] \tag{4.3}$$

4.2 Tensor-based Value Iteration Algorithm

The curse of dimensionality that is inherent in this problem can be solved by applying tensor-train decomposition. Instead of using normal tensor-train decomposition, a further reduction in size can be made by applying tensor-train decomposition on the skeleton of the unfolding matrices, called tensor-train cross. The skeleton unfolding matrix A_k can be written as:

$$A_k = A_k[:, C] A_k[I, C]^{-1} A_k[C, :], \tag{4.4}$$

where I is the set of rows where $|I| \geq r$ and C is the set of columns where $|C| \geq r$. Tensor-train cross can be used to solve the tensor-based value iteration algorithm as given in Algorithm 6.

Algorithm 6 Tensor-based Value Iteration [6]

Require: Termination criterion δ_{max}; TT-cross accuracy ϵ; Initial cost function $\hat{J}_h^{(0)}$
Ensure: Residual $\delta = \|\hat{J}_h^{(k)} - \hat{J}_h^{(k-1)}\|^2 < \delta_{max}$

```
δ = δ_max + 1
k = 0
while δ > δ_max do
    Ĵ_h^(k+1) = TT-cross ((4.2), ε)
    k = k + 1
    δ = ||Ĵ_h^(k) − Ĵ_h^(k−1)||^2
end while
```

In this algorithm, a max residual and accuracy setting are given along with an initialized guess of the cost function $\hat{J}_h^{(0)}$ in tensor-train form. The initial residual is set such that the algorithm enters the loop that continues while $\delta > \delta_{max}$. Taking as input the update function, Equation (4.2), and the accuracy setting ϵ, the TT-cross function returns an updated cost function that maintains the accuracy setting. Finally the iteration and residual value are updated. This algorithm continues until $\delta < \delta_{max}$.

It is further shown in Theorem 6 that the error of tensor-based value iteration can be bounded.

Theorem 6 (TVI Approximation Error [6]) *When the proposed interpolation method are run for k iterations on a grid with resolution h, and TT-cross accuracy ϵ, the number of computational operations performed by the algorithm is $O(\frac{kdr^3}{h})$ and the resulting approximation error satisfies:*

$$\|\hat{J}_j^{(k)} - J_{u_h^*}\| \leq \epsilon(\frac{R_{max} + \gamma}{1 - \gamma}) + \gamma^{k+1}\epsilon\|\tilde{J}_h^{(0)}\| + \gamma^k\|\tilde{J}_h^{(0)} - J_{u_h^*}\|.$$

Where $R_{max} = \max_{u(x),x} r(x, u(x))$ and $\tilde{J}_h^{(k)}$ is the cost-to-go approximation of the k^{th} iteration of normal value iteration.

4.3 Test and Results

Gorodetsky, Karaman, and Marzouk perform this tensor-based value iteration on a seven dimensional perching glider problem. In this problem a glider navigates a two dimensional plane under the influence of the seven following state variables: $(x, y, \theta, \phi, v_x, v_y, \dot{\theta})$. The variables in order are the x position, y position, angle of attack, elevator angle, horizontal speed, vertical speed, and the angle of attacks rate of change. Elevator angle rate of change $\dot{\phi}$ is the only control variable present in this problem. The optimization problem for this perching glider problem is represented in [6] as:

$$J^*(z) = \min_{u(t)} E[\int_0^T \bar{x}'Q\bar{x} + 0.1u^2dt + \bar{x}(T)'Q_f\bar{x}(T)], \tag{4.5}$$

s.t.,

$$
\begin{aligned}
x_w &= [x - l_w c_\theta, y - l_w s_\theta], \\
\dot{x}_w &= [\dot{x} + l_w \dot{\theta} s_\theta, \dot{y} - l_w \dot{\theta} c_\theta], \\
x_e &= [x - l c_\theta - l_e, c_{\theta_{+\phi}}, y - l s_\theta - l_e s_{\theta_{+\phi}}], \\
\dot{x}_e &= [\dot{x} + l\dot{\theta} s_\theta + l_e(\dot{\theta} + u) s_{\theta_{+\phi}}, \dot{y} - l\dot{\theta} c_\theta - l_e(\dot{\theta} + u) c_{\theta_{+\phi}}], \\
\alpha_w &= \theta - \tan^{-1}(\dot{y}_w, \dot{x}_w), \\
\alpha_e &= \theta + \phi - \tan^{-1}(\dot{y}_e, \dot{x}_e), \\
f_w &= \rho S_w |\dot{x}_w|^2 \sin(\alpha_w), \\
f_e &= \rho S_e |\dot{x}_e|^2 \sin(\alpha_e), \\
m\ddot{x} &= -f_w s_\theta - f_e s_{\theta_{+\phi}} + Fdw, \\
m\ddot{y} &= f_w c_\theta - f_e c_{\theta_{+\phi}} - mg + Fdw, \\
I\ddot{\theta} &= -f_w l_w - f_e(l c_\phi + l_e) + Fdw.
\end{aligned}
$$

In the above dynamics ρ represents the air density, m is the glider mass, and I is the glider's moment of inertia. S_w and S_e are the wing and tail control surface areas respectively. The length from the center of gravity to the elevator is l. l_w is the wing half cord and l_e is the elevator half cord. Finally, c_γ represents the $\cos(\gamma)$ and s_γ represents the $\sin(\gamma)$. By uniformly discretizing each variable into 32 states, tensor-based value iteration is capable of performing value iteration on a problem with a discretized state space of 3.4×10^{10} states with a maximal error of $\epsilon = 0.1$. The resulting glide path and vertical and horizontal velocity of the optimal solution can be seen in Figure 4-1.

The key to tensor-train value iterations success is the reduction in the number of states examined. Of the 3.4×10^{10} states making up the initial discretized state space, only approximately 10^6 of these states are used as can be seen in Figure 4-2.[6]

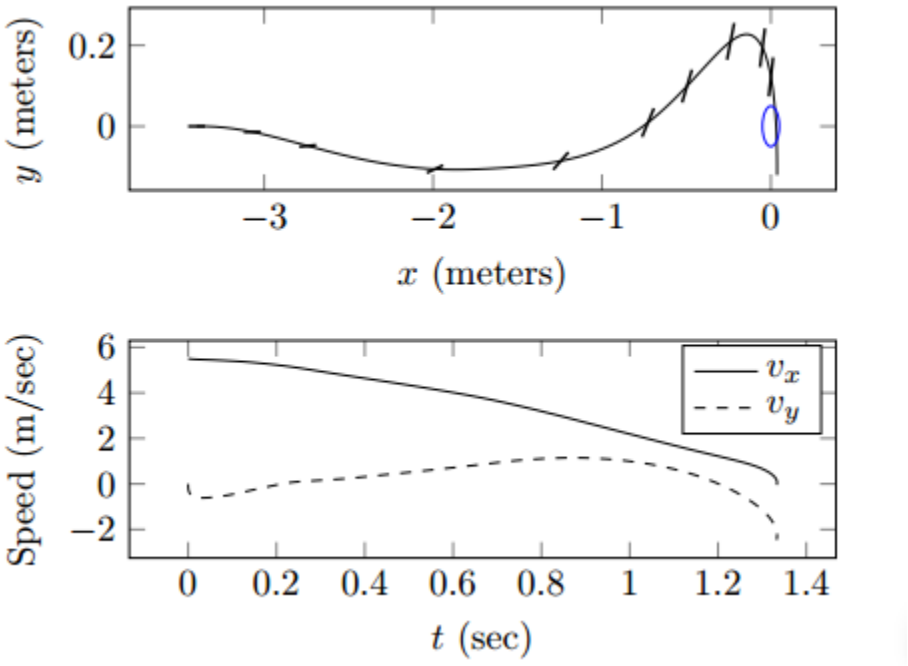


Figure 4-1: Optimal glide path with vertical and horizontal velocities [6]

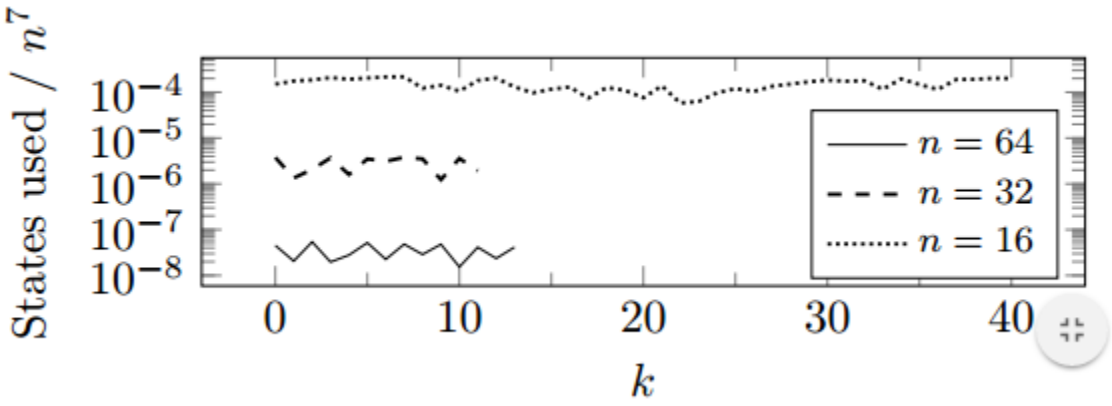


Figure 4-2: Fraction of states evaluated by TTVI in optimal glide problem [6]

Chapter 5

Tensor-Based Value Iteration for Pursuit-Evasion Games

Building on the work presented in Chapter 4, this chapter will present an algorithm that uses TVI to solve pursuit-evasion games. This chapter will begin by describing the general dynamic programming problem used to model a pursuit-evasion game. Next, the chapter will detail how best response can be used along with dynamic programming to solve for the optimal control of both the pursuer and the evader. The Best Response Dynamic Programming algorithm will then be presented in detail. Finally, the chapter will culminate in the presentation of the Best-Response Tensor-Train-decomposition-based Value Iteration algorithm.

5.1 Description of Problem

As demonstrated in Section 1.2, solving pursuit-evasion games requires determining optimal control values for two or more players with opposing outcomes. In order to keep the problem as simple as possible, there will be a focus on zero-sum games in which the cost function of the pursuer is the negative of the evaders cost function or $(G_1(x) = -G_2(x))$. Despite this constraint, it is possible to apply TVI to nonzero-sum games or even simply use predetermined controls as either the pursuer or evaders control values. Before exploring the modifications brought about by adjusting TVI

for pursuit-evasion games, a description of how dynamic programming may be used with best response will be provided.

Value iteration can be modified to solve pursuit-evasion games using a best response dynamic by solving for the optimal cost at each state. The problem is modeled as a discounted, deterministic, shortest path problem where γ is the discount factor. First, the state space Z is discretized into evenly spaced states $z_i \in Z$ where the space between states are h just as in [3]. Given that z_i is the current state, u_1 is the pursuer control, and u_2 is the evader control, $P(z_j|z_i, u_1, u_2) = 1$ where z_j is the deterministic result of $f(z_i, u_1, u_2)$. Even if $z_j \notin z_i$, the state cost at z_j can be determined by:

$$J(z_j) = \sum_i \lambda_i J(z_i), \tag{5.1}$$

where:

$$\lambda_i = \begin{cases} \dfrac{1 - |z_j - z_i|}{h}, & \forall i|\ |z_i - z_j| \leq h, \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

Each run of best response K results in a control policy u^K which is used as a constant control policy for the opponent.

Instead of using a single update function as in Equation (4.2), I model the problem as solving two separate update functions over the same state-space:

$$J_p^{(k+1,K)}(z_i) = \min_{u_1}[G(z_i, u_1, u_2^K) + \gamma J_p^{(k,K)}(z_j|z_i, u_1, u_2^K)], \tag{5.3}$$

$$J_e^{(k+1,K)}(z_i) = \max_{u_2}[G(z_i, u_1^K, u_2) + \gamma J_e^{(k,K)}(z_j|z_i, u_1^K, u_2)]. \tag{5.4}$$

By running the update functions until $k \to \infty$, the optimal value function is achieved for a given opponents control policy u^K:

$$J_p^{(*,K)}(z_i) = \min_{u_1}[G(z_i, u_1, u_2^K) + \gamma J_p^{(*,K)}(z_j|z_i, u_1, u_2^K)], \tag{5.5}$$

$$J_e^{(*,K)}(z_i) = \max_{u_2}[G(z_i, u_1^K, u_2) + \gamma J_e^{(*,K)}(z_j|z_i, u_1^K, u_2)]. \tag{5.6}$$

The optimal value functions can then be used to solve for the optimal control values $u_1^{(*,K)}$ and $u_2^{(*,K)}$:

$$u_1^{(*,K)}(z_i) = \underset{u_1}{\operatorname{argmin}}[G(z_i, u_1, u_2^K) + \gamma J_p^{(*,K)}(z_j|z_i, u_1, u_2^K)], \tag{5.7}$$

$$u_2^{(*,K)}(z_i) = \underset{u_2}{\operatorname{argmax}}[G(z_i, u_1^K, u_2) + \gamma J_e^{(*,K)}(z_j|z_i, u_1^K, u_2)]. \tag{5.8}$$

5.2 Best Response

In "Dynamic Noncooperative Game Theory", Tamer Basar and Geert Olsder prove that under certain conditions pursuit-evasion games will converge at a saddle point or Nash equilibrium. A major condition is that the value function V satisfies the partial differential equation:

$$-\frac{\delta V}{\delta t} = \min_{u_1 \in S_1} \max_{u_1 \in S_2} [\frac{\delta V}{\delta x} f(t, x, u_1, u_2) + g(t, x, u_1, u_2)] \tag{5.9}$$

$$= \max_{u_1 \in S_2} \min_{u_1 \in S_1} [\frac{\delta V}{\delta x} f(t, x, u_1, u_2) + g(t, x, u_1, u_2)]. \tag{5.10}$$

Furthermore, it can be shown that if the value function is continuously differentiable then the saddle point equilibrium exist:

Theorem 7 (Existence of Saddle Point [1]) *If (i) a continuously differentiable function $V(t,x)$ exists that satisfies the Isaacs equation 5.9, (ii) $V(T,x) = q(T,x)$ on the boundary of the target set, defined by $(t,x) = 0$, and (iii) either $u_1^*(t) = \gamma_1^*(t,x)$, or $u_2^*(t) = \gamma_2^*(t,x)$ as derived from 5.9, generates trajectories that terminate in finite time (whatever γ_2 , respectively γ_1, is) then $V(t,x)$ is the value function and the pair (γ_1^*, γ_2^*) constitutes a saddle point.*

Using the intuition from Theorem 7, if each iteration of best response is run until $K \to \infty$ and $u_1^{(*,K)}$ and $u_2^{(*,K)}$ each converge to a single set of control values, then those control values are considered a Nash equilibrium and:

$$J_p^{(*,*)}(z_i) = \min_{u_1}[G(z_i, u_1, u_2^K) + \gamma J_p^{(*,*)}(z_j|z_i, u_1, u_2^*)], \tag{5.11}$$

$$J_e^{(*,*)}(z_i) = \max_{u_2}[G(z_i, u_1^K, u_2) + \gamma J_e^{(*,*)}(z_j|z_i, u_1^*, u_2)]. \tag{5.12}$$

The optimal control values can be determined by:

$$u_1^{(*,*)}(z_i) = \operatorname*{argmin}_{u_1}[G(z_i, u_1, u_2^K) + \gamma J_p^{(*,*)}(z_j|z_i, u_1, u_2^*)], \tag{5.13}$$

$$u_2^{(*,*)}(z_i) = \operatorname*{argmax}_{u_2}[G(z_i, u_1^K, u_2) + \gamma J_e^{(*,*)}(z_j|z_i, u_1^*, u_2)]. \tag{5.14}$$

These equations result in a dynamic programming best response algorithm 7.

Algorithm 7 Best Response Dynamic Programming

Require: Algorithm termination criterion Δ_{max};Value Iteration termination criterion δ_{max}; Initial cost functions $J_p^{(0,0)}, J_e^{(0,0)}$

Ensure: Residual $\delta = \|J_p^{(k,K)} - J_p^{(k-1),K}\|^2 < \delta_{max}$;Residual $\delta = \|J_e^{(k,K)} - J_e^{(k-1),K}\|^2 < \delta_{max}$;Residual $\Delta_p = \|J_p^{(*,K)} - J_p^{*,(K-1)}\|^2 < \Delta_{max}$;Residual $\Delta_e = \|J_e^{(*,K)} - J_e^{*,(K-1)}\|^2 < \Delta_{max}$

$\Delta_p = \Delta_{max} + 1$
$K = 1$
while $\Delta_p > \Delta_{max} \cup \Delta_e > \Delta_{max}$ **do**
    $\delta = \delta_{max} + 1$
    $k = 0$
    while $\delta > \delta_{max}$ **do**
        for $\forall z_i$ **do**
            5.3
        end for
        $k = k + 1$
        $\delta = \|J_p^{(k,K)} - J_p^{(k-1),K}\|^2$
    end while
    $\delta = \delta_{max} + 1$
    $k = 0$
    while $\delta > \delta_{max}$ **do**
        for $\forall z_i$ **do**
            5.4
        end for
        $k = k + 1$
        $\delta = \|J_e^{(k,K)} - J_e^{(k-1),K}\|^2$
    end while
    $\Delta_p = \|J_p^{(*,K)} - J_p^{*,(K-1)}\|^2$
    $\Delta_e = \|J_e^{(*,K)} - J_e^{*,(K-1)}\|^2$
    $K = K + 1$
end while

5.2.1 Modifications to Tensor-based Value Iteration

Tensor-based value iteration can be modified to solve pursuit-evasion games using a best response dynamic. As in [6], tensor-train cross can be used to provide an approximation of value iteration within an error bound ϵ. By applying tensor-train cross to Algorithm 7, a best response TVI algorithm results as in Algorithm 8. This algo-

Algorithm 8 Best-Response Tensor-Train-decomposition-based Value Iteration

Require: Algorithm termination criterion Δ_{max};Value Iteration termination criterion δ_{max}; Initial cost functions $J_p^{(0,0)}, J_e^{(0,0)}$;TT-cross accuracy ϵ;

Ensure: Residual $\delta = \|J_p^{(k,K)} - J_p^{(k-1),K}\|^2 < \delta_{max}$;Residual $\delta = \|J_e^{(k,K)} - J_e^{(k-1),K}\|^2 < \delta_{max}$;Residual $\Delta_p = \|J_p^{(*,K)} - J_p^{*,(K-1)}\|^2 < \Delta_{max}$;Residual $\Delta_e = \|J_e^{(*,K)} - J_e^{*,(K-1)}\|^2 < \Delta_{max}$

$\Delta_p = \Delta_{max} + 1$
$K = 1$
while $\Delta_p > \Delta_{max} \cup \Delta_e > \Delta_{max}$ **do**
  $\delta = \delta_{max} + 1$
  $k = 0$
  while $\delta > \delta_{max}$ **do**
    $J_p^{(k+1),K} =$ TT-cross $((5.3), \epsilon)$
    $k = k + 1$
    $\delta = \|J_p^{(k,K)} - J_p^{(k-1),K}\|^2$
  end while
  $\delta = \delta_{max} + 1$
  $k = 0$
  while $\delta > \delta_{max}$ **do**
    $J_e^{(k+1),K} =$ TT-cross $((5.4), \epsilon)$
    $k = k + 1$
    $\delta = \|J_e^{(k,K)} - J_e^{(k-1),K}\|^2$
  end while
  $\Delta_p = \|J_p^{(*,K)} - J_p^{*,(K-1)}\|^2$
  $\Delta_e = \|J_e^{(*,K)} - J_e^{*,(K-1)}\|^2$
  $K = K + 1$
end while

rithm can be used to efficiently solve a variety of problems including those presented in Chapter 6.

Chapter 6

Application of Best-Response Tensor-Train-Decomposition-Based Value Iteration

Using the algorithms and problem definitions presented in the previous chapter, this chapter will detail how two example problems were solved. The first problem, a four-dimensional pursuit-evasion problem will be solved with both a traditional value iteration algorithm(VI), Algorithm 7, and the BR-TT-VI algorithm, Algorithm 8. This will allow for comparisons between the two algorithms to demonstrate the benefits of using BR-TT-VI. Next a six-dimensional pursuit-evasion problem will be solved that demonstrates BR-TT-VI's ability to efficiently compute a solution for a problem that would take VI days if not weeks to solve. The end of this chapter will also begin to detail a number of shortcomings with the BR-TT-VI algorithm.

6.1 Four-Dimensional Problem

The four-dimensional pursuit-evasion problem used here is simple enough to be solved analytically, with VI, and with BR-TT-VI. This problem is used to show the accuracy of the BR-TT-VI approximation and demonstrate the advantages of using BR-TT-VI over VI. This section will start with a definition of the four-dimensional pursuit-

evasion problem as well as a brief explanation for the optimal analytic solution to the problem. Next, it will be explained how VI was accomplished and the results of this algorithm. Afterwards it will be detailed how the BR-TT-VI was used and the results of this algorithm. Finally, a comparison will be made between the results of VI and BR-TT-VI.

6.1.1 Problem Definition

The four-dimensional pursuit-evasion problem used here has the pursuer attempting to reduce the distance to or capture the evader, while the evader attempts to maximize the distance between itself and the pursuer. A pursuer and evader are modeled in a 10×10 two-dimensional state space with simple Euclidean dynamics. The four dimensions are the x position of the pursuer x_1, the y position of the pursuer y_1, the x position of the evader x_2, and the y position of the evader y_2. This results in a state space such that $z_i = (x_1, y_1, x_2, y_2)$. Each of the dimensions are bounded as follows:

$$\begin{aligned} x_1 &\in [0, 10] \\ y_1 &\in [0, 10] \\ x_2 &\in [0, 10] \\ y_2 &\in [0, 10]. \end{aligned}$$

Both the pursuer and the evader have two controls for their x and y velocity respectively, $u_1 = (Vx_1, Vy_1)$ and $u_2 = (Vx_2, Vy_2)$. These velocities are also bounded:

$$\begin{aligned} Vx_1 &\in [-1, 1] \\ Vy_1 &\in [-1, 1] \\ Vx_2 &\in [-0.5, 0.5] \\ Vy_2 &\in [-0.5, 0.5]. \end{aligned}$$

Furthermore, the following system dynamics are used:

$$x_1^{k+1} = x_1^k + Vx_1 dt, \tag{6.1}$$

$$y_1^{k+1} = y_1^k + Vy_1 dt, \tag{6.2}$$

$$x_2^{k+1} = x_2^k + Vx_2 dt, \tag{6.3}$$

$$y_2^{k+1} = y_2^k + Vy_2 dt. \tag{6.4}$$

Each dimension is discretized into 21 equally spaced states for a total of $21^4 \approx 2 \cdot 10^5$ discrete states. This discretization was chosen as it creates a state for every 0.5 units:

$$\begin{aligned} x_1 &\in \{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10\} \\ y_1 &\in \{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10\} \\ x_2 &\in \{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10\} \\ y_2 &\in \{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10\}. \end{aligned}$$

Value iteration can be used to solve for the optimal cost at each state. The problem is modeled as in Section 5.1. A state cost function based on the distance between the pursuer and evader was chosen:

$$G(z_i, u_1, u_2) = 10 + (x_1 - x_2)^2 + (y_1 - y_2)^2. \tag{6.5}$$

The constant of 10 is added to provided a buffer for the BR-TT-VI approximation in order to prevent negative values. A discount factor of $\gamma = 0.7$ was chosen. Applying the state cost function and discount factor to Equation (5.3) results in the update functions for the four-dimensional pursuit-evasion problem:

$$J_p^{(k+1,K)}(z_i) = \min_{u_1}[10 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + 0.7J_p^{(k,K)}(z_j|z_i, u_1, u_2^K)], \tag{6.6}$$

$$J_e^{(k+1,K)}(z_i) = \max_{u_2}[10 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + 0.7J_e^{(k,K)}(z_j|z_i, u_1^K, u_2)]. \tag{6.7}$$

By running Algorithm 7 with Equations (6.6) and (6.7) results in the given optimal

value functions:

$$J_p^{(*,*)}(z_i) = \min_{u_1}[10 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + 0.7J_p^{(*,*)}(z_j|z_i, u_1, u_2^*)], \tag{6.8}$$

$$J_e^{(*,*)}(z_i) = \max_{u_2}[10 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + 0.7J_e^{(*,*)}(z_j|z_i, u_1^*, u_2)]. \tag{6.9}$$

These optimal value functions 6.8 6.9 can be used to solve the following pursuit-evasion optimization problem:

$$\min_{u_1}\max_{u_2}[\int_0^T e^{-0.7t}(10 + (x_1(t) - x_2(t))^2 + (y_1(t) - y_2(t))^2)dt] \tag{6.10}$$

$$\text{s.t.} \quad 6.1. \tag{6.11}$$

6.1.2 Analytic Solution

The problem also has a simple analytic solution since the dimensions are separable. This problem has separable dimensions because any change in one dimension does not affect a change in the other dimension. Furthermore, the controls are completely separate in regard to dimension, choosing a certain control in the x dimension does not prevent choosing a control in the y dimension. Because of this the problem can be viewed as simultaneously solving two one-dimensional problems where the pursuer is trying to minimize $|x_1 - x_2|$ and $|y_1 - y_2|$ while the evader attempts to maximize these functions. Because of this the optimal solution can be analytically found. For the pursuer, the optimal solution is:

$$Vx_1^* = \underset{Vx_1}{\text{argmin}}[((x_1 + Vx_1dt) - (x_2 + Vx_2dt))^2], \tag{6.12}$$

$$Vy_1^* = \underset{Vy_1}{\text{argmin}}[((y_1 + Vy_1dt) - (y_2 + Vy_2dt))^2]. \tag{6.13}$$

The optimal solution for the evader is then:

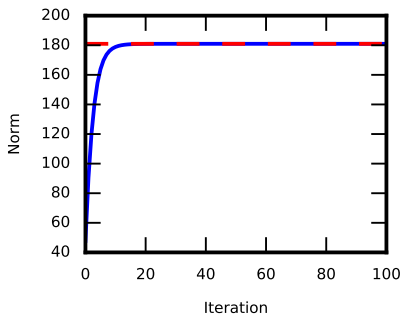
$$Vx_2^* = \underset{Vx_2}{\text{argmax}}[((x_1 + Vx_1dt) - (x_2 + Vx_2dt))^2], \tag{6.14}$$

$$Vy_2^* = \underset{Vy_2}{\text{argmax}}[((y_1 + Vy_1 dt) - (y_2 + Vy_2 dt))^2]. \tag{6.15}$$

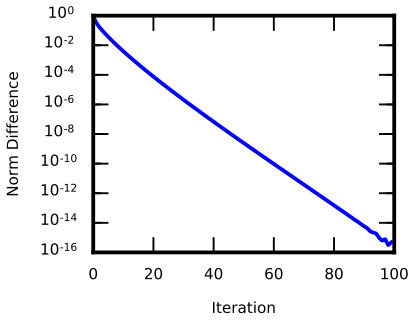
These analytically solutions can be used to check the validity of the results of both VI and BR-TT-VI.

6.1.3 Traditional Value Iteration

Before running the traditional value iteration algorithm, Algorithm 7, both the pursuer and evader algorithm were run for 100 iterations to better characterize the algorithm when run past completion. First, both the pursuer and evader state cost were set to initialized such that $J_p^{(0,0)}(z_i) = (x_1 - x_2)^2 + (y_1 - y_2)^2$ and $J_e^{(0,0)}(z_i) = (x_1 - x_2)^2 + (y_1 - y_2)^2$. Next, the pursuer value iteration is run for one hundred iterations. It took approximately an hour and a half to run these one hundred iterations of pursuer value iteration, Table 6.1. The evolution of the pursuer state cost can be seen in Figure 6-2. Take note of the minimal difference in the state cost between ten and a hundred iterations. Notice that the average norm converges to the optimal average norm, Figure 6-1a, and the difference between the average norm logarithmically decreases, Figure 6-1b. The breakdown of the logarithmic decrease around the 95^{th} iteration is due to rounding error caused by the precision of float values.



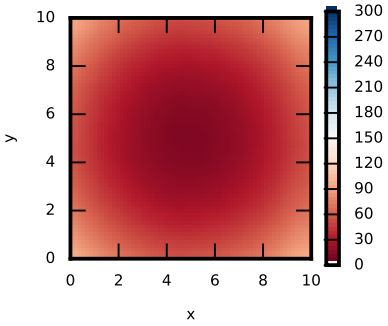
(a) Average norm of the pursuer state cost with red dashed line as the average norm of the optimal value



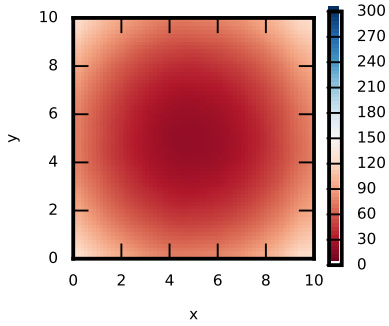
(b) Normalized difference in average norm between sequential pursuer state costs

Figure 6-1: Diagnostic results of running 100 Iterations of Pursuit Value Iteration

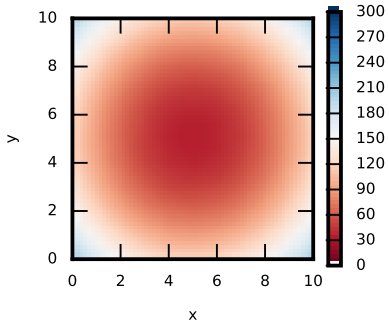
After the pursuer value iteration is run for a hundred iterations, the evader value iteration is also run for a hundred iterations with the updated pursuer cost, $J_p^{(100,1)}$.



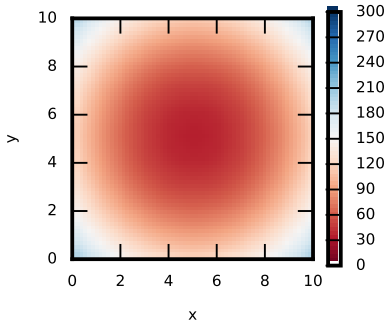
(a) 1 Iteration



(b) 2 Iterations



(c) 10 Iterations



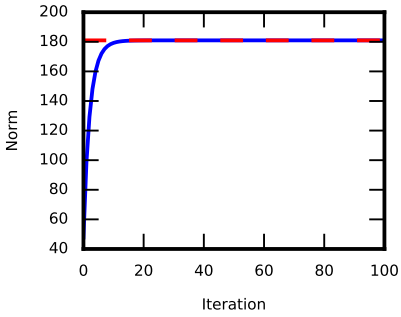
(d) 100 Iterations

Figure 6-2: State cost evolution when evader is at (5,5)

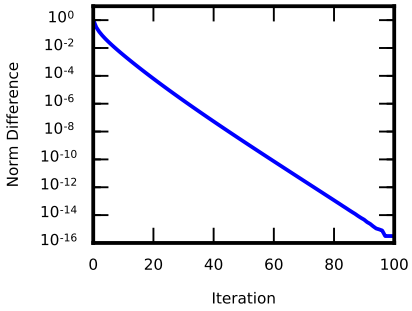
Table 6.1: Value Iteration Program Run Times(sec)

	1 Iteration	100 Iterations
Pursuit	57.6885	5942.2932
Evasion	58.3154	5865.4880

Just like with the pursuer value iteration, it took approximately an hour and a half to run one hundred iterations of evader value iteration, Table 6.1. The evolution of the evader state cost can be seen in Figure 6-4. Take note of the minimal difference in the state cost between ten and a hundred iterations. Notice that the average norm converges to the optimal average norm, Figure 6-3a, and the difference between the average norm logarithmically decreases, Figure 6-3b. Notice that despite maximizing over the same cost function, the average norm converges to approximately the same value as in Figure 6-1a.



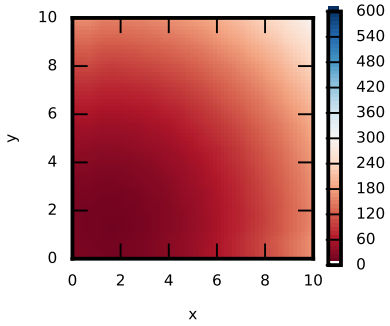
(a) Average norm of the evader state cost with red dashed line as the average norm of the optimal value



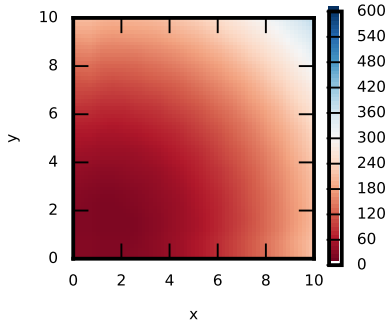
(b) Normalized difference in average norm between sequential evader state costs

Figure 6-3: Diagnostic results of running 100 Iterations of Evader Value Iteration

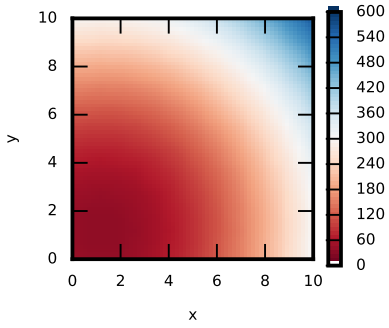
Traditional value iteration was also implemented with Algorithm 7. This algorithm was ran twice, once at $\Delta_{max} = 10^{-2}$ and $\delta_{max} = 10^{-2}$ for an equal comparison with the BR-TT-VI algorithm and once where $\Delta_{max} = 10^{-5}$ and $\delta_{max} = 10^{-5}$ to determine optimality to a point where changes in the state cost were no longer noticeable. For the $(\Delta_{max} = 10^{-2}, \delta_{max} = 10^{-2})$ case, on the first best response run both the pursuer and evader state costs converge to an average norm of 179 in about



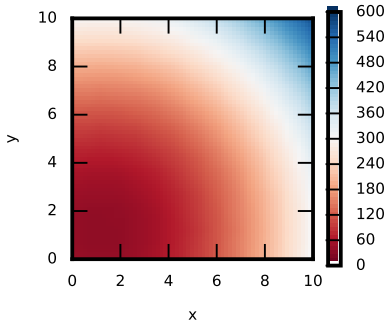
(a) 1 Iteration



(b) 2 Iterations



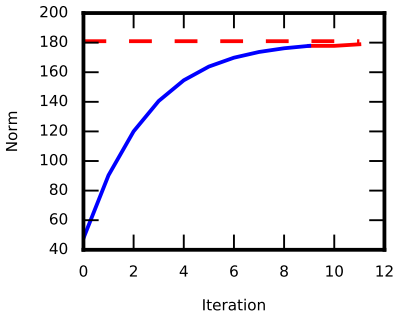
(c) 10 Iterations



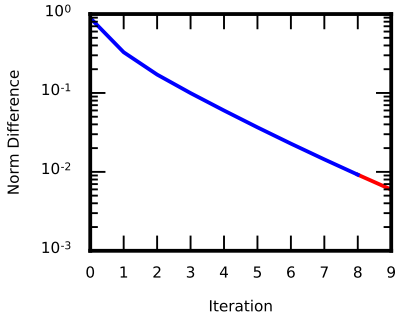
(d) 100 Iterations

Figure 6-4: State cost evolution when pursuer is at (1,1)

10 iterations as can be seen by the blue line in Figures 6-5a and 6-6a. Changes to the control on this run are minor enough that there are no major changes in state costs on the second run as can be seen by the red line segment in the same figures. This causes the algorithm to end after only two runs. To run the entire algorithm it took almost twenty minutes, Table 6.4.

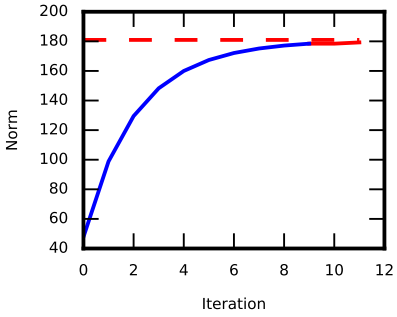


(a) Average norm of the pursuer state cost with red dashed line as the average norm of the optimal value

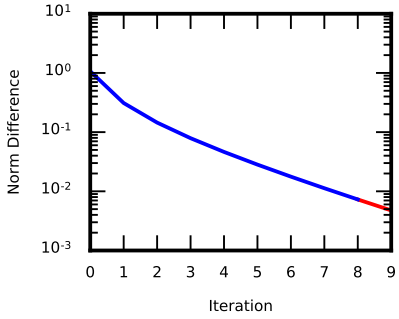


(b) Normalized difference in average norm between sequential pursuer state costs

Figure 6-5: Pursuer diagnostic results of running Best Response Value Iteration Algorithm with ($\Delta_{max} = 10^{-2}, \delta_{max} = 10^{-2}$). Blue solid line is the first run, while the red solid line is the second run



(a) Average norm of the evader state cost with red dashed line as the average norm of the optimal value

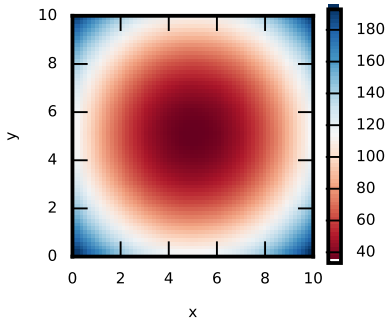


(b) Normalized difference in average norm between sequential evader state costs

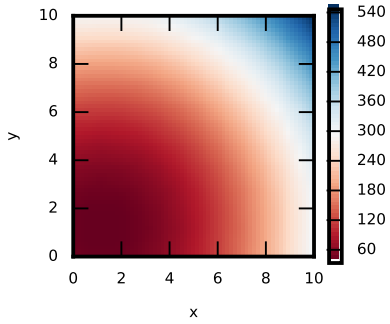
Figure 6-6: Evader diagnostic results of running Best Response Value Iteration Algorithm with ($\Delta_{max} = 10^{-2}, \delta_{max} = 10^{-2}$). Blue solid line is the first run, while the red solid line is the second run

The pursuer state costs monotonically decrease toward the evaders position, Figure 6-7a, while the evader's state costs monotonically decrease toward the pursuers

position,Figure 6-7b. The game was simulated with the pursuer starting at eight different positions and the evader starting at (5,5) as can be seen in Figure 6-8. In each of these cases, the pursuer heads directly toward the evader while the evader heads in the opposite direct of the pursuer. Due to the pursuers faster speed in both the x and y axis, the distance between the pursuer and evader decreases directly to the origin, Figure 6-9. It should be noted that due to the separation of the dynamics in the x and y axis, when the pursuer and evader have the same value on only one axis an oscillation occurs. This is due to the evader randomly darting to one or the other direction with the pursuer quickly following and overcoming the evader.



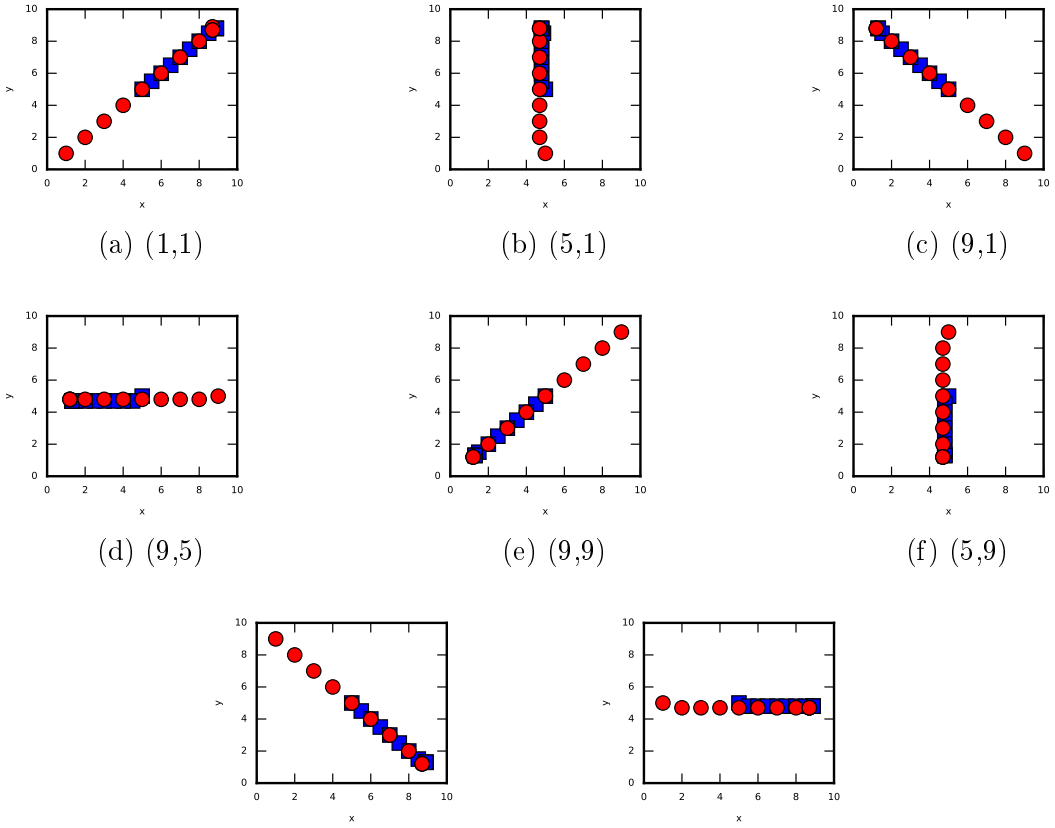
(a) Pursuer state cost when evader is at (5,5)



(b) Evader state cost when pursuer is at (1,1)

Figure 6-7: State Costs after running Best Response Value Iteration Algorithm with $(\Delta_{max} = 10^{-2}, \delta_{max} = 10^{-2})$

A $(\Delta_{max} = 10^{-5}, \delta_{max} = 10^{-5})$ case was also implemented to ensure that the $(\Delta_{max} = 10^{-2}, \delta_{max} = 10^{-2})$ case ran to an appropriate termination. On the first best response run both the pursuer and evader state costs converge to an average norm of 180.97 in just over 25 iterations as seen by the blue solid line in Figures 6-10a and 6-11a. The second run of best response again produces no notable change in the state cost of either the pursuer or the evader as seen by the red solid line in the same figures. Because of this the average norm of 180.97 will be considered optimal for the four-dimensional problem. In order to get this more accurate response, the algorithm ran for almost an hour, Table 6.4. Despite the much longer run time, the results of the pursuit-evasion game do not noticeably change as can be seen by Figures 6-12 to 6-14 mirroring Figures 6-7 to 6-9.

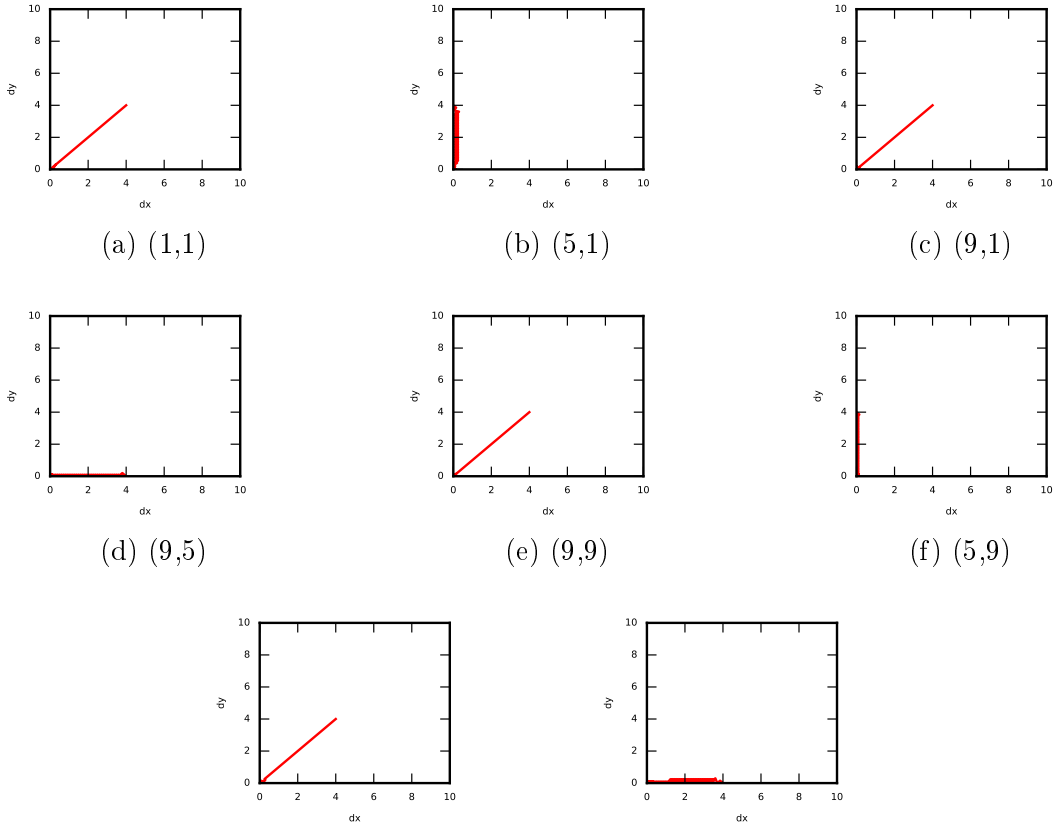


(a) (1,1) (b) (5,1) (c) (9,1)

(d) (9,5) (e) (9,9) (f) (5,9)

(g) (1,9) (h) (1,5)

Figure 6-8: Pursuit-Evasion game with evader at (5,5) and pursuer at varying positions. 1 second intervals are used between each marker

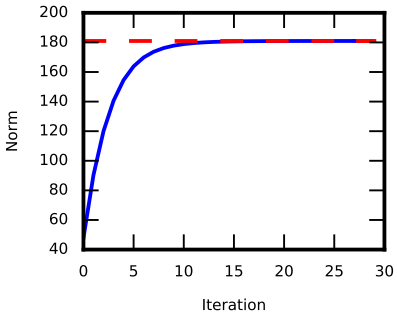


(a) (1,1) (b) (5,1) (c) (9,1)

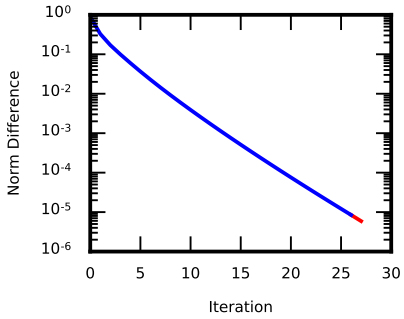
(d) (9,5) (e) (9,9) (f) (5,9)

(g) (1,9) (h) (1,5)

Figure 6-9: Relative position of pursuer and evader in pursuit-Evasion game with evader at (5,5) and pursuer at varying positions

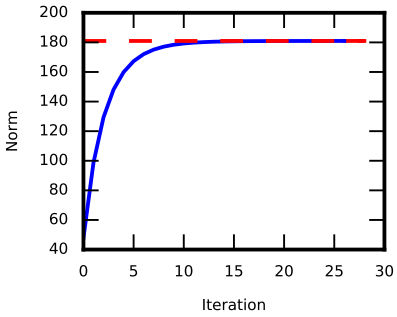


(a) Average norm of the pursuer state cost with red dashed line as the average norm of the optimal value

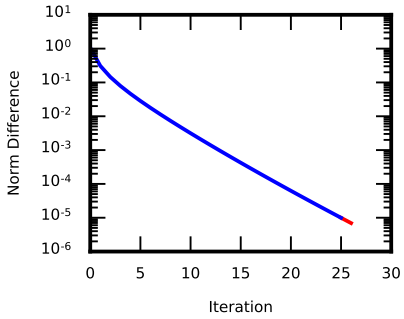


(b) Normalized difference in average norm between sequential pursuer state costs

Figure 6-10: Pursuer diagnostic results of running Best Response Value Iteration Algorithm with ($\Delta_{max} = 10^{-5}, \delta_{max} = 10^{-5}$). Blue solid line is the first run, while the red solid line is the second run



(a) Average norm of the evader state cost with red dashed line as the average norm of the optimal value

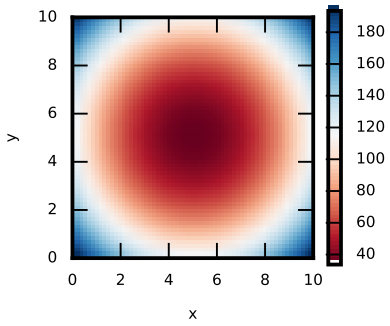


(b) Normalized difference in average norm between sequential evader state costs

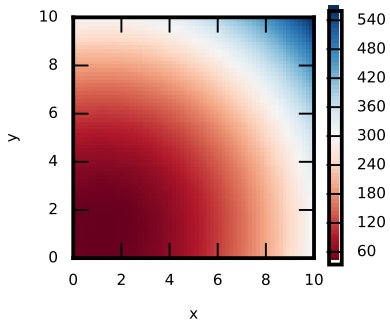
Figure 6-11: Evader diagnostic results of running Best Response Value Iteration Algorithm with ($\Delta_{max} = 10^{-5}, \delta_{max} = 10^{-5}$). Blue solid line is the first run, while the red solid line is the second run

6.1.4 Tensor-Train-Decomposition-Based Value Iteration

Tensor-Train-Decomposition-Based Value Iteration(TT-VI) can also be used to solve the pursuit-evasion game with simple Euclidean dynamics. Just as with VI, before running Algorithm 8 both the pursuer and evader TT-VI were run for 100 iterations to characterize the algorithm when run past completion. Once again both the pursuer and evader state cost were set to initialized such that $J_p^{(0,0)}(z_i) = (x_1 - x_2)^2 + (y_1 - y_2)^2$ and $J_e^{(0,0)}(z_i) = (x_1 - x_2)^2 + (y_1 - y_2)^2$. The pursuer TT-VI is run first for one hundred



(a) Pursuer state cost when evader is at (5,5)



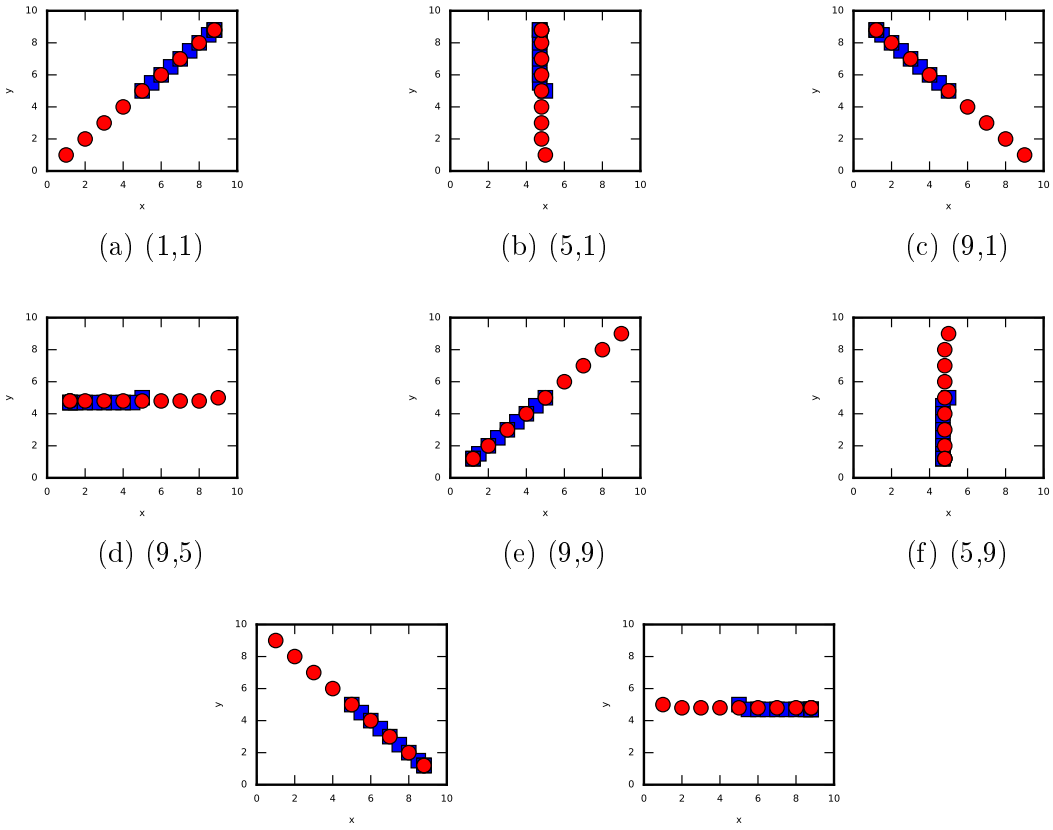
(b) Evader state cost when pursuer is at (1,1)

Figure 6-12: State Costs after running Best Response Value Iteration Algorithm with ($\Delta_{max} = 10^{-5}, \delta_{max} = 10^{-5}$)

iterations. These one hundred iterations only take about eight minutes, Table 6.2. The evolution of the pursuer state cost can be seen in Figure 6-16. Once again there is very little difference between the state cost at ten iterations and a hundred iterations. Notice that the average norm converges to the optimal average norm, Figure 6-15a. The difference between the average norm logarithmically decreases until a difference of about 10^{-3} at which point the plot fluctuates about this point Figure 6-15b. Average tensor rank remains constant at 3.60, Figure 6-15c, while the fraction of states used fluctuates between about 0.020 and 0.013 of the original 21^4 states, Figure 6-15d.

After the pursuer TT-VI is run for a hundred iterations, the evader TT-VI is run for a hundred iterations with the updated pursuer cost, $J_p^{(100,1)}$. Just like the pursuer TT-VI, the evader TT-VI only takes about eight minutes to complete a hundred iterations, Table 6.2. The evolution of the evader state cost can be seen in Figure 6-18. The average norm still converges to the optimal average norm, Figure 6-17a. Just as with pursuer TT-VI the difference between the average norm logarithmically decreases until a difference of about 10^{-3} at which point the plot fluctuates about this point Figure 6-17b. Average tensor rank also remains constant at 3.60, Figure 6-17c, while the fraction of states used fluctuates between about 0.020 and 0.013 of the original 21^4 states, Figure 6-17d. Just as with VI, the average norm converges to approximately the same value as in Figure 6-15a.

The four-dimensional problem was also solved using Algorithm 8. $\Delta_{max} = 10^{-2}$



(a) (1,1) (b) (5,1) (c) (9,1)

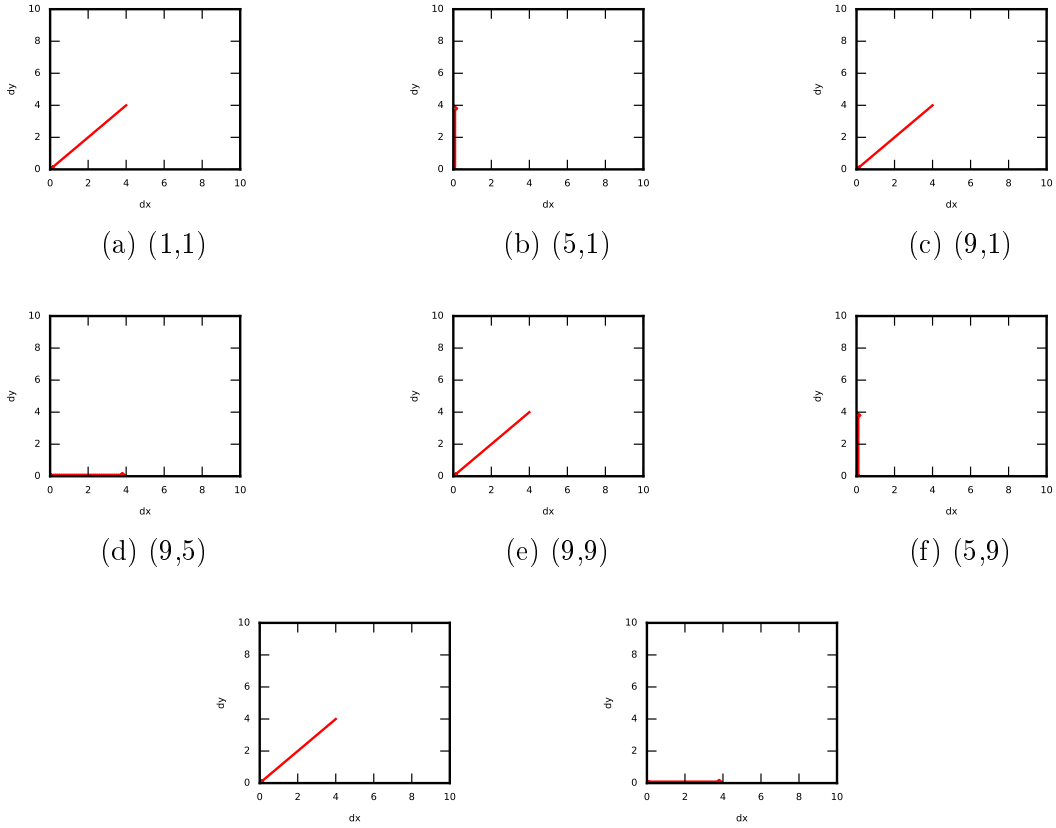
(d) (9,5) (e) (9,9) (f) (5,9)

(g) (1,9) (h) (1,5)

Figure 6-13: Pursuit-Evasion game with evader at (5,5) and pursuer at varying positions. 1 second intervals are used between each marker

Table 6.2: 4D TT-VI Program Run Times(sec)

	1 Iteration	100 Iterations
Pursuit	4.3525	473.1945
Evasion	4.6160	483.1202

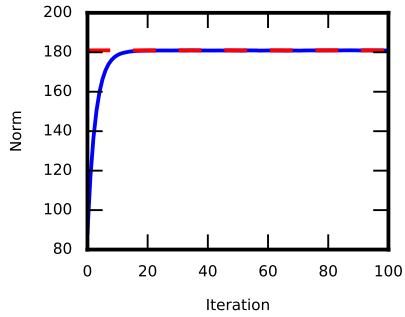


(a) (1,1) (b) (5,1) (c) (9,1)

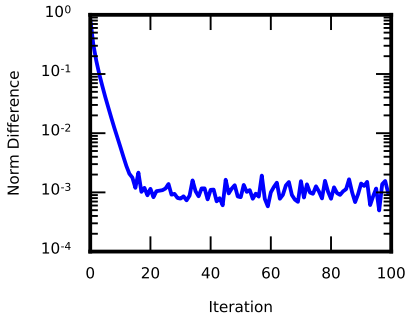
(d) (9,5) (e) (9,9) (f) (5,9)

(g) (1,9) (h) (1,5)

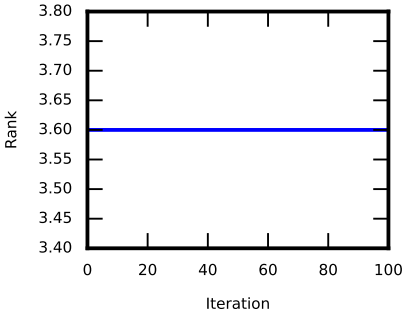
Figure 6-14: Relative position of pursuer and evader in pursuit-Evasion game with evader at (5,5) and pursuer at varying positions



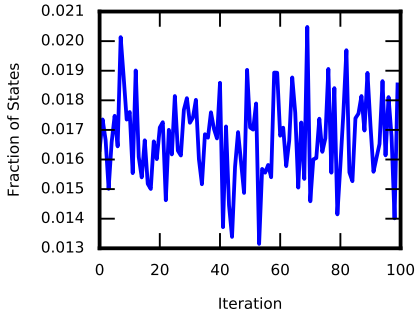
(a) Average norm of the pursuer state cost with red dashed line as the average norm of the optimal value



(b) Normalized difference in average norm between sequential pursuer state costs



(c) Average rank of the pursuer state cost tensor

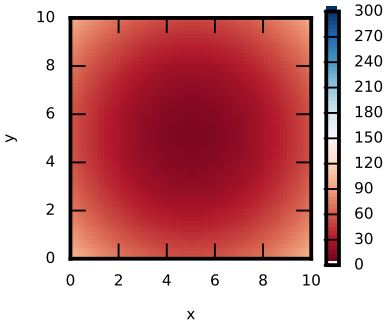


(d) Fraction of states used for the pursuer state cost tensor

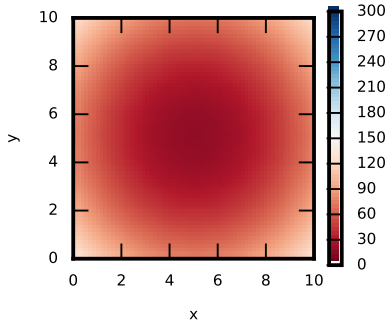
Figure 6-15: Diagnostic results of running 100 Iterations of Pursuit TT-VI

and $\delta_{max} = 10^{-2}$ were used as the best response and the value iteration accuracy respectively. On the first best response run both the pursuer and evader state costs converge to an average norm of 179 in about 10 iterations as can be seen by the solid blue line in Figures 6-19a and 6-20a. The average rank for both the pursuer and evader stays at 3.6 throughout, Figures 6-19c and 6-20c. In determining the new state costs, less than 1/50 of the total states is used for each iteration of TT-VI, Figures 6-19d and 6-20d. On the second run of best response, the changes remain minor resulting in virtually no change to the norm, rank, or states used for both the pursuer and the evader as can be seen by the solid red line in the previous figures, Figures 6-19 and 6-20. The entire BR-TT-VI algorithm for the four-dimensional problem takes less than two minutes to complete, Table 6.4.

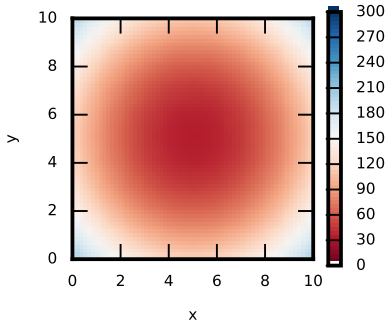
The pursuer state costs monotonically decrease toward the evaders position, Fig-



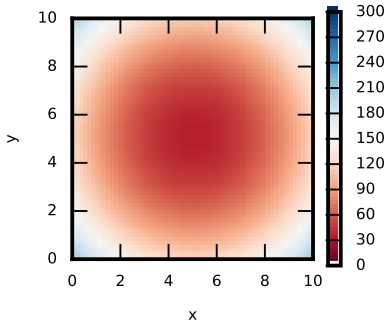
(a) 1 Iteration



(b) 2 Iterations

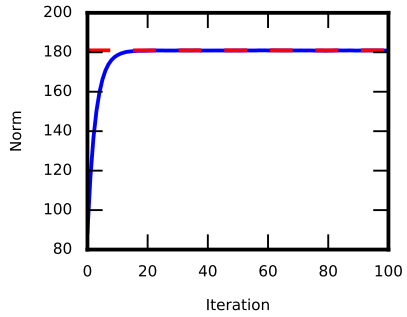


(c) 10 Iterations

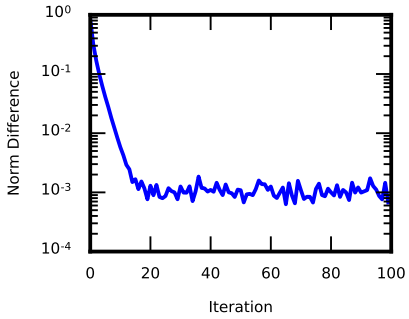


(d) 100 Iterations

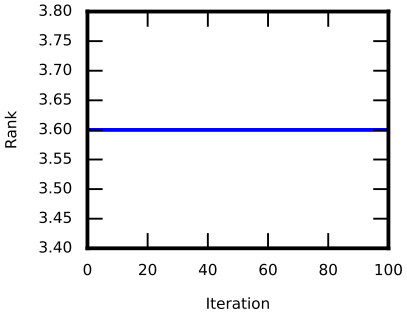
Figure 6-16: State cost evolution when evader is at (5,5)



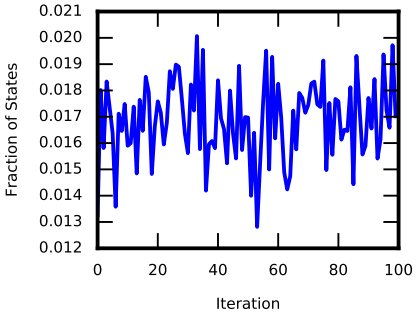
(a) Average norm of the evader state cost with red dashed line as the average norm of the optimal value



(b) Normalized difference in average norm between sequential evader state costs



(c) Average rank of the evader state cost tensor



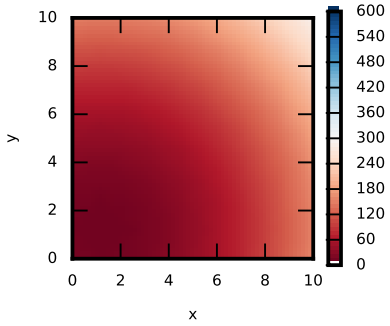
(d) Fraction of states used for the evader state cost tensor

Figure 6-17: Diagnostic results of running 100 Iterations of Evade TVI

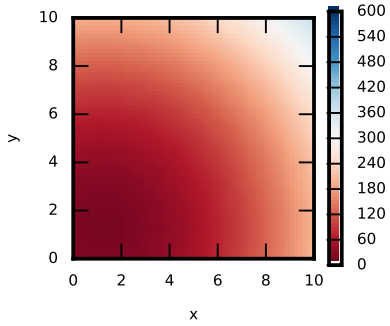
ure 6-21a, while the evader's state costs monotonically decrease toward the pursuers position,Figure 6-21b. The pursuer heads directly towards the evader, while the evader heads in the opposite direction of the pursuer as can be seen in Figure 6-22. Due to the pursuer's greater speed, the game has the pursuer constantly decreasing the distance between itself and the evader until capture as can be seen in Figure 6-23.

6.1.5 Comparison of Traditional Value Iteration and TT-VI

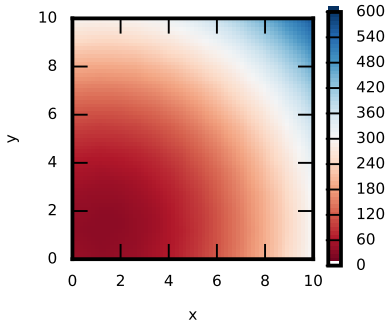
Both VI and TT-VI can be used to solve the four-dimensional simple euclidean dynamics problem, however each method has its advantages and drawbacks. For the four-dimensional pursuit-evasion problem both VI and TT-VI successfully solve the



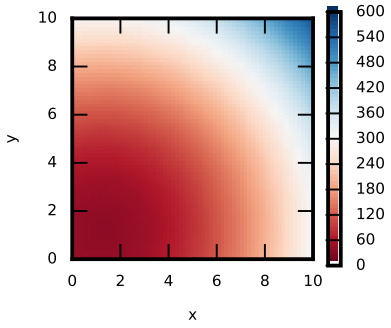
(a) 1 Iteration



(b) 2 Iterations

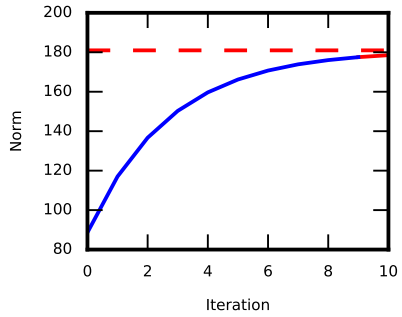


(c) 10 Iterations

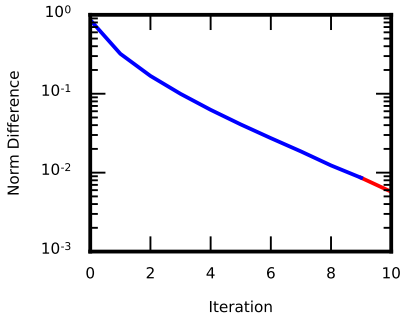


(d) 100 Iterations

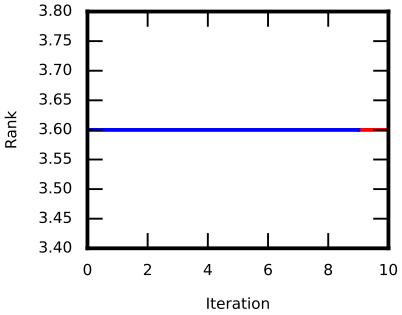
Figure 6-18: State cost evolution when pursuer is at (1,1)



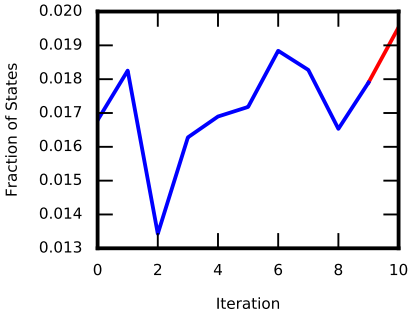
(a) Average norm of the pursuer state cost with red dashed line as the average norm of the optimal value



(b) Normalized difference in average norm between sequential pursuer state costs



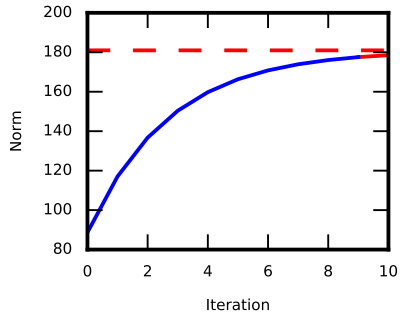
(c) Average rank of the pursuer state cost tensor



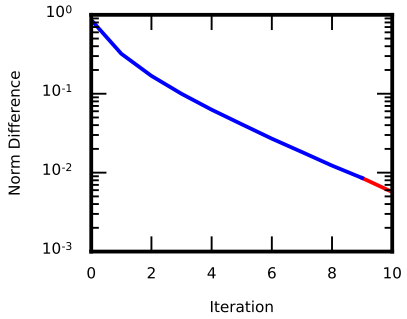
(d) Fraction of states used for the pursuer state cost tensor

Figure 6-19: Pursuer diagnostic results of running BR-TT-VI Algorithm with ($\Delta_{max} = 10^{-2}, \delta_{max} = 10^{-2}$). Blue solid line is the first run, while the red solid line is the second run

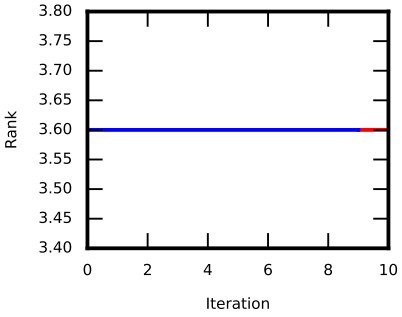
problem. This can be seen by the nearly identical solutions from all eight starting positions for both the state cost results from VI and the state cost result from TT-VI as seen in Figures 6-8, 6-13 and 6-22. All average norms of the state costs also converge to the average norm of the optimal solution of 180.97, Figures 6-5a, 6-6a, 6-10a, 6-11a, 6-19a and 6-20a. However, the precision of VI can be more exact than TT-VI. This can be seen in the difference between Figures 6-1b and 6-3b and Figures 6-15b and 6-17b. While the traditional value iteration continues to become more precise until limited by the precision of the variable (about 10^{-16}), TVI is constrained in its precision by the accuracy of the tensor approximation (for the given example about 10^{-3}). Furthermore, TVI can only tighten its precision at the potential expense in



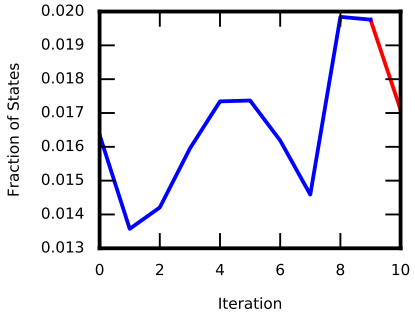
(a) Average norm of the evader state cost with red dashed line as the average norm of the optimal value



(b) Normalized difference in average norm between sequential evader state costs

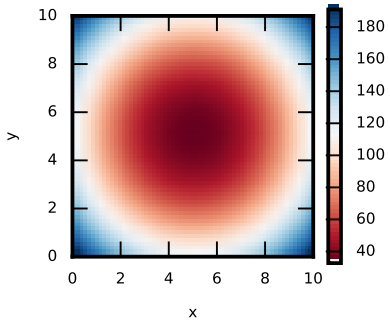


(c) Average rank of the evader state cost tensor

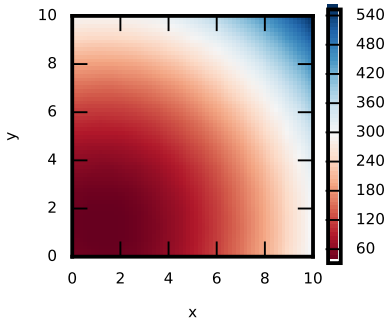


(d) Fraction of states used for the evader state cost tensor

Figure 6-20: Evader diagnostic results of running Best Response TVI Algorithm with ($\Delta_{max} = 10^{-2}, \delta_{max} = 10^{-2}$). Blue solid line is the first run, while the red solid line is the second run

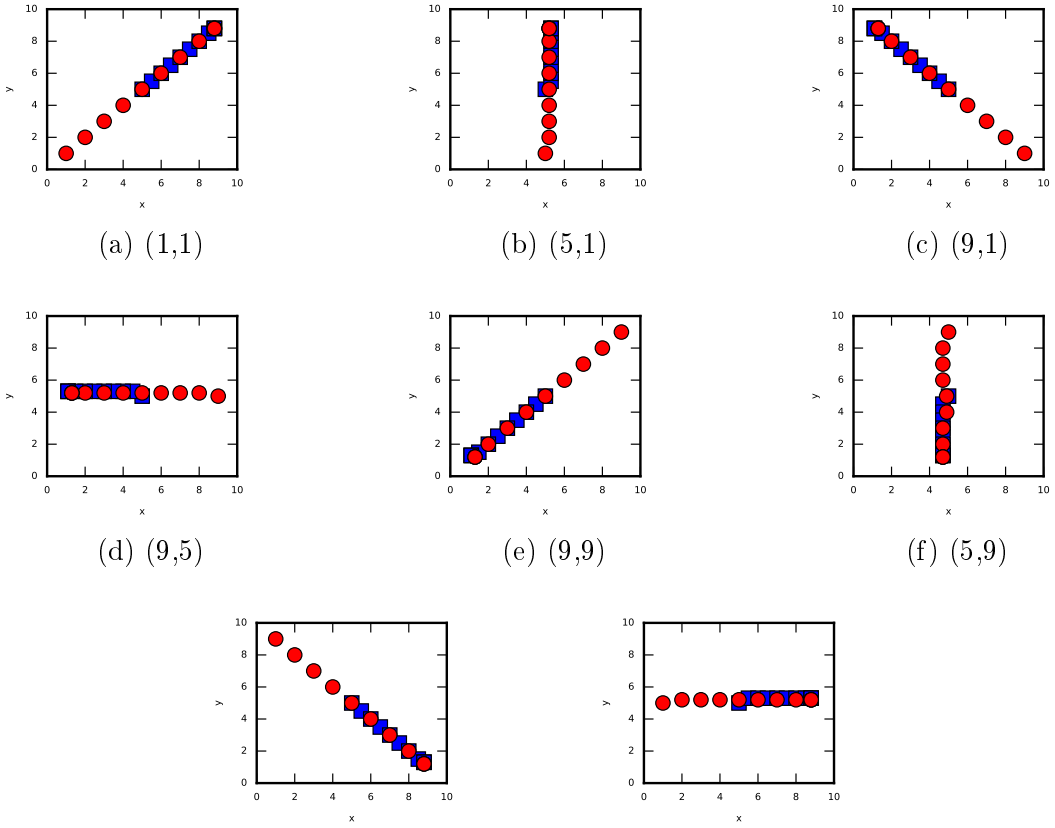


(a) Pursuer state cost when evader is at (5,5)



(b) Evader state cost when pursuer is at (1,1)

Figure 6-21: State Costs after running BR-TT-VI Algorithm with ($\Delta_{max} = 10^{-2}, \delta_{max} = 10^{-2}$)

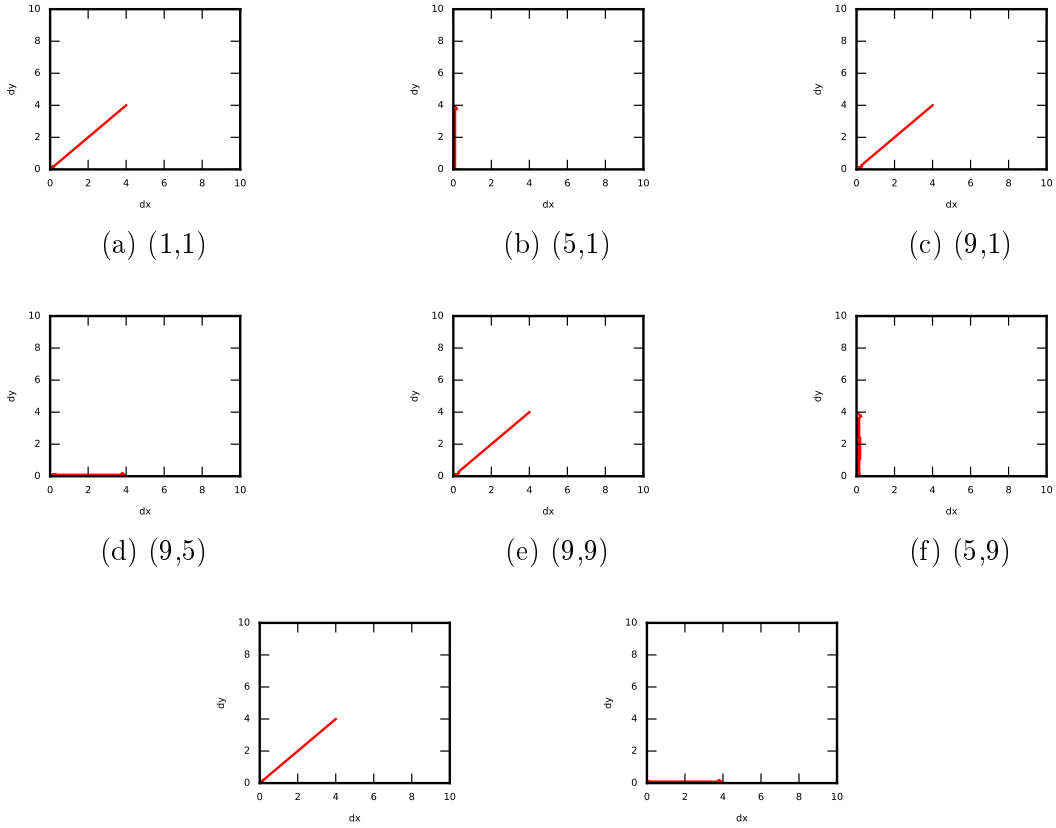


(a) (1,1) (b) (5,1) (c) (9,1)

(d) (9,5) (e) (9,9) (f) (5,9)

(g) (1,9) (h) (1,5)

Figure 6-22: Pursuit-Evasion game with evader at (5,5) and pursuer at varying positions. 1 second intervals are used between each marker



(a) (1,1) (b) (5,1) (c) (9,1)

(d) (9,5) (e) (9,9) (f) (5,9)

(g) (1,9) (h) (1,5)

Figure 6-23: Relative position of pursuer and evader in pursuit-Evasion game with evader at (5,5) and pursuer at varying positions

Table 6.3: State Cost Storage

	Value Iteration	TT-VI
Pursuit	8 MB	883.3 kB
Evasion	8 MB	1.1 MB

Table 6.4: Best Response Program Run Times

	4D VI(10^{-2})	4D VI(10^{-5})	4D BR-TT-VI(10^{-2})	6D BR-TT-VI (10^{-2})
Time(s)	1152.3172	3123.4174	102.0020	593.2975

increased run time.

Although traditional value iteration has benefits when it comes to precision, TT-VI can compute the solution more efficiently. Using the 10^{-2} error bounds, TT-VI runs more than ten times faster than the traditional method as can be seen in Tables 6.1, 6.2 and 6.4. Use of the TT-VI algorithm also conserves memory. As seen in Table 6.3, the state cost representations of TT-VI are approximately an eighth the size of the four-dimensional arrays used to store the state costs in VI. This four-dimensional example shows that TT-VI provides many benefits for problems already solvable by VI.

6.2 Six-Dimensional Problem

For the six dimensional problem, the pursuer and evader adapt Dubins car dynamics as outlined in [5]. The pursuer and evader remain in a 10×10 two-dimensional state space with the addition of a heading state for both. The six dimensions are the x position of the pursuer x_1, the y position of the pursuer y_1, the heading of the pursuer θ_1, the x position of the evader x_2, the y position of the evader y_2, and the heading of the evader θ_2. This results in a state space such that $z_i = (x_1, y_1, \theta_1, x_2, y_2, \theta_2)$. Each

of the dimensions are bounded as follows:

$$\begin{aligned} x_1 &\in [0, 10] \\ y_1 &\in [0, 10] \\ \theta_1 &\in [0, 2\pi] \\ x_2 &\in [0, 10] \\ y_2 &\in [0, 10] \\ \theta_2 &\in [0, 2\pi]. \end{aligned}$$

Both the pursuer and the evader have a single control for their change in heading, respectively u_1 and u_2. These controls are bounded between $[-\frac{\pi}{4}, \frac{\pi}{4}]$. The pursuer and evader each have a constant velocity as defined by V_1 and V_2 respectively. For this problem $V_1 = 1$ and $V_2 = 0.5$. This results in the following system dynamics:

$$\dot{x}_1 = x_1 + V_1 cos(\theta_1)dt, \tag{6.16}$$

$$\dot{y}_1 = y_1 + V_1 sin(\theta_1)dt, \tag{6.17}$$

$$\dot{\theta}_1 = \theta_1 + V_1 tan(u_1)dt \tag{6.18}$$

$$\dot{x}_2 = x_2 + V_2 cos(\theta_2)dt, \tag{6.19}$$

$$\dot{y}_2 = y_2 + V_2 sin(\theta_2)dt, \tag{6.20}$$

$$\dot{\theta}_2 = \theta_2 + V_2 tan(u_2)dt. \tag{6.21}$$

Each dimension is again discretized into 21 equally spaced states for a total of $21^6 \approx 9 \cdot 10^7$ discrete states. This discretization creates the same discrete state space as for the four-dimensional problem for the x and y states. The theta values are discretized such that the four cardinal directions are included and 0 and 2π are redundantly kept in the discretization as can be seen:

$$\begin{aligned} \theta_1 &\in [0, \frac{\pi}{10}, \frac{\pi}{5}, \frac{3\pi}{10}, \frac{2\pi}{5}, \frac{\pi}{2}, \frac{3\pi}{5}, \frac{7\pi}{10}, \frac{4\pi}{5}, \frac{9\pi}{10}, \pi, \frac{11\pi}{10}, \frac{6\pi}{5}, \frac{13\pi}{10}, \frac{7\pi}{5}, \frac{3\pi}{2}, \frac{8\pi}{5}, \frac{17\pi}{10}, \frac{9\pi}{5}, \frac{19\pi}{10}, 2\pi] \\ \theta_2 &\in [0, \frac{\pi}{10}, \frac{\pi}{5}, \frac{3\pi}{10}, \frac{2\pi}{5}, \frac{\pi}{2}, \frac{3\pi}{5}, \frac{7\pi}{10}, \frac{4\pi}{5}, \frac{9\pi}{10}, \pi, \frac{11\pi}{10}, \frac{6\pi}{5}, \frac{13\pi}{10}, \frac{7\pi}{5}, \frac{3\pi}{2}, \frac{8\pi}{5}, \frac{17\pi}{10}, \frac{9\pi}{5}, \frac{19\pi}{10}, 2\pi]. \end{aligned}$$

Value iteration can also be used to solve for the optimal cost at each state. The problem is modeled as in Section 5.1. A state cost function based on the distance between the pursuer and evader was chosen:

$$G(z_i, u_1, u_2) = 10 + (x_1 - x_2)^2 + (y_1 - y_2)^2. \tag{6.22}$$

The constant of 10 is added to provided a buffer for the BR-TT-VI approximation in order to prevent negative values. A discount factor of $\gamma = 0.7$ was chosen. Applying the state cost function and discount factor to Equation (5.3) results in the update functions for the six-dimensional pursuit-evasion problem:

$$J_p^{(k+1,K)}(z_i) = \min_{u_1}[10 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + 0.7J_p^{(k,K)}(z_j|z_i, u_1, u_2^K)], \tag{6.23}$$

$$J_e^{(k+1,K)}(z_i) = \max_{u_2}[10 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + 0.7J_e^{(k,K)}(z_j|z_i, u_1^K, u_2)]. \tag{6.24}$$

By running Algorithm 8 with Equations (6.23) and (6.24) results in the given optimal value functions:

$$J_p^{(*,*)}(z_i) = \min_{u_1}[10 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + 0.7J_p^{(*,*)}(z_j|z_i, u_1, u_2^*)], \tag{6.25}$$

$$J_e^{(*,*)}(z_i) = \max_{u_2}[10 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + 0.7J_e^{(*,*)}(z_j|z_i, u_1^*, u_2)]. \tag{6.26}$$

These optimal value functions 6.25 6.26 can be used to solve the following pursuit-evasion optimization problem:

$$\min_{u_1}\max_{u_2}[\int_0^T e^{-0.7t}(10 + (x_1(t) - x_2(t))^2 + (y_1(t) - y_2(t))^2)dt] \tag{6.27}$$

$$\text{s.t.} \quad 6.16. \tag{6.28}$$

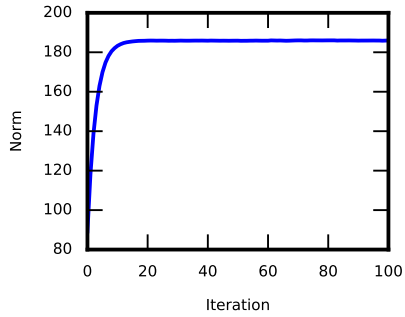
6.2.1 Solving the Six-Dimensional Pursuit-Evasion Problem

The excessive size of the six-dimensional pursuit-evasion game makes this problem hard to solve with traditional value iteration. One iteration of VI on the pursuit problem takes over thirteen hours to complete. However, BR-TT-VI may be used

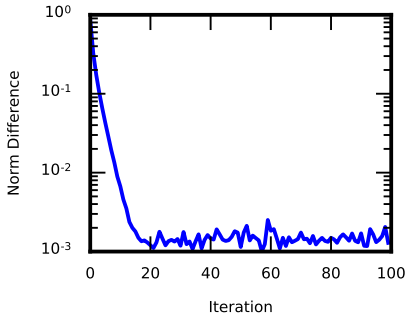
to solve the problem within some accuracy ε in relatively short periods of time. Just as with the four-dimensional problem, before applying the BR-TT-VI algorithm a hundred iterations of the pursuer and evader TT-VI were conducted on the six-dimensional problem.

Once again both the pursuer and evader state cost were set to initialized such that $J_p^{(0,0)}(z_i) = (x_1 - x_2)^2 + (y_1 - y_2)^2$ and $J_e^{(0,0)}(z_i) = (x_1 - x_2)^2 + (y_1 - y_2)^2$. The pursuer TT-VI is run first for one hundred iterations. These hundred iterations only took about fifty minutes which is less than half the time it took VI to run a hundred iterations of the four-dimensional pursuit problem, Table 6.5. The evolution of the pursuer state cost in relation to the x and y position of the pursuer can be seen in Figure 6-25. Once again there is very little difference between the state cost at ten iterations and a hundred iterations. The average norm converges to a little over 180 as seen in Figure 6-24a. The difference between the average norm logarithmically decreases until a difference of just over 10^{-3} at which point the plot fluctuates about this point, Figure 6-24b. Average tensor rank starts at 3.85 but quickly jumps up to just below 4.3, Figure 6-24c. Fluctuating at about 0.0001 of the original 21^6 states, the fraction of states used is even smaller than in the four-dimensional problem, Figure 6-24d.

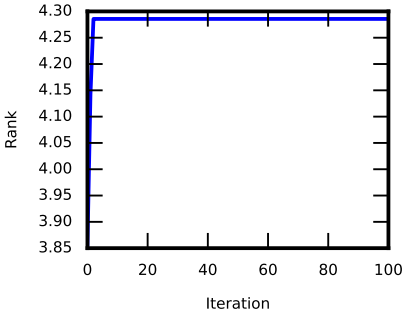
After the pursuer TT-VI is run for a hundred iterations, the evader TT-VI is also run for a hundred iterations with the updated pursuer cost, $J_p^{(100,1)}$. Once again the evader TT-VI only takes about fifty minutes to complete a hundred iterations, Table 6.5. The evolution of the evader state cost in relation to the x and y position of the evader can be seen in Figure 6-27. Also of note is the evolution of the evader state cost in relation to the heading of both the pursuer and the evader as seen in Figure 6-28. Notice how the state cost evolves from uniform in iteration one to having distinct minimum and maximum points in iteration 10. This pattern is especially impressive as the θ-values do not have an inherent cost in Equation (6.22). The average norm of the evader, much like the pursuer, converges to just over 180, Figure 6-26a. Just as with the pursuer, the difference between the average norm logarithmically decreases until a difference of just above 10^{-3} at which point the plot fluctuates about this



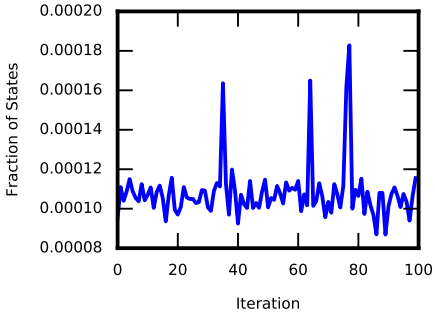
(a) Average norm of the pursuer state cost



(b) Normalized difference in average norm between sequential pursuer state costs



(c) Average rank of the pursuer state cost tensor



(d) Fraction of states used for the pursuer state cost tensor

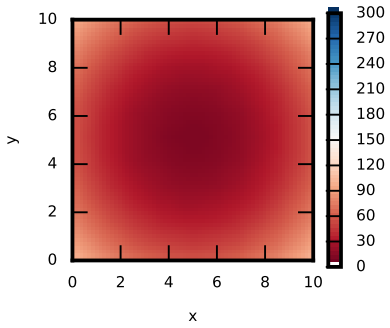
Figure 6-24: Diagnostic results of running 100 Iterations of Pursuit TT-VI

point Figure 6-26b. Average tensor rank behaves the same as in the pursuer case by starting at about 3.85 and jumping to a point just below 4.3, Figure 6-26c. The fraction of states used fluctuates again at about 0.0001 of the original 21^6 states, Figure 6-26d. Just as with the four-dimensional problem, the average norm converges to approximately the same value as in Figure 6-24a.

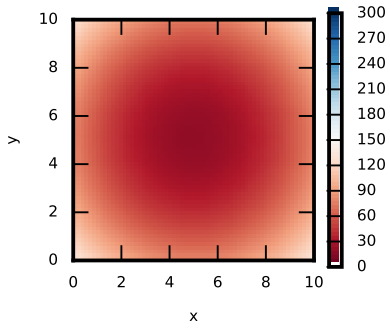
Using Algorithm 8, the six-dimensional pursuit-evasion problem was solved. Just as with the four-dimensional problem, $\Delta_{max} = 1E-2$ and $\delta_{max} = 1E-2$ were used

Table 6.5: 6D BR-TT-VI Program Run Times(sec)

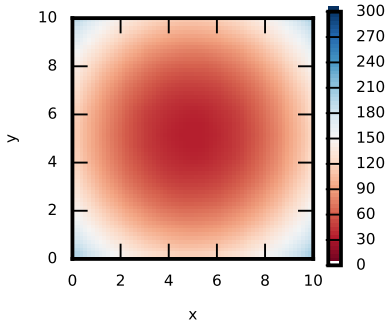
	1 Iteration	100 Iterations
Pursuit	22.7885	2883.9859
Evasion	22.6637	2838.0722



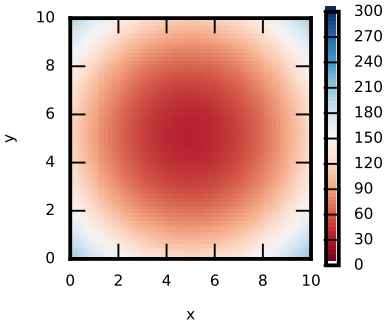
(a) 1 Iteration



(b) 2 Iterations

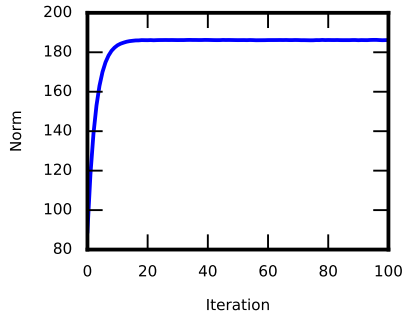


(c) 10 Iterations

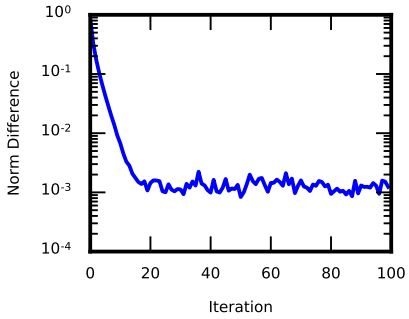


(d) 100 Iterations

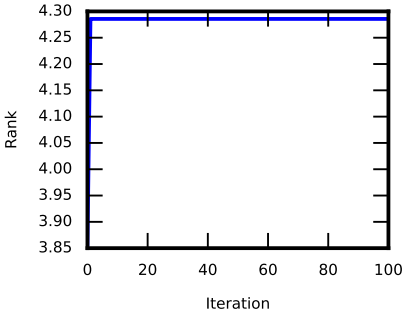
Figure 6-25: State cost evolution with respect to pursuer x and y position when evader is at (5,5) and ($\theta_1 = 0, \theta_2 = 0$)



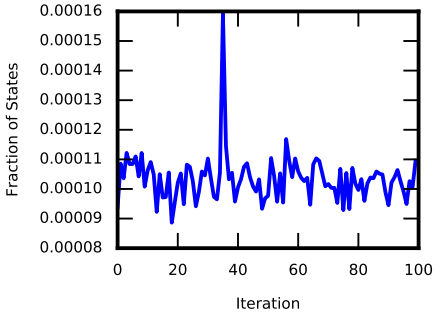
(a) Average norm of the evader state cost



(b) Normalized difference in average norm between sequential evader state costs



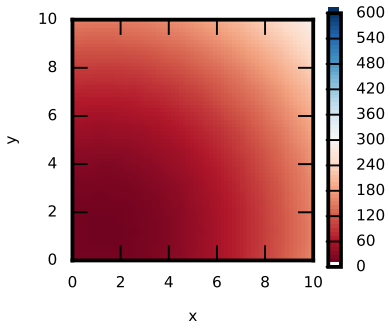
(c) Average rank of the evader state cost tensor



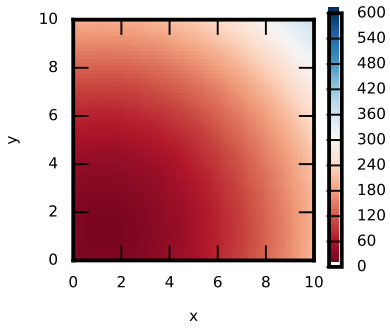
(d) Fraction of states used for the evader state cost tensor

Figure 6-26: Diagnostic results of running 100 Iterations of Evade TVI

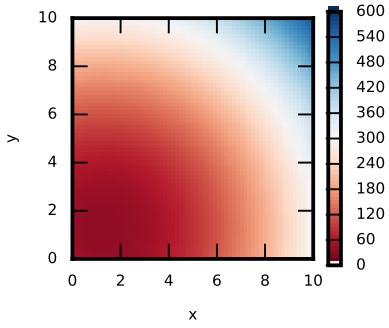
as the best response and the value iteration accuracy respectively. On the first best response run both the pursuer and evader state costs converge to an average norm of about 180 in about ten iterations as can be seen by the blue line in Figures 6-29a and 6-30a. For both the pursuer and the evader, the average rank starts at 3.85 and quickly climbs to just below 4.30 before leveling off, Figures 6-29c and 6-30c. Generally only about 1/10000 of the total states are used for each iteration of TT-VI, Figures 6-29c and 6-30c. On the second run of best response, the changes remain minor resulting in virtually no change to the norm, rank, or states used for both the pursuer and the evader as can be seen by the red line in Figures 6-29 and 6-30. The entire algorithm takes just under ten minutes to complete, Table 6.4. This is about half the time that the VI took for the four-dimensional problem and a much shorter time than the thirteen plus hours that a single iteration of VI takes for the



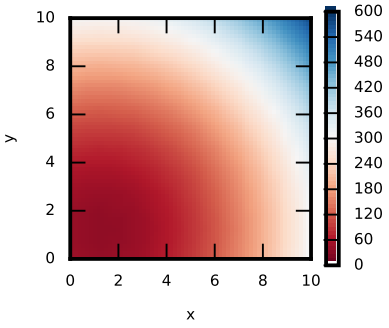
(a) 1 Iteration



(b) 2 Iterations

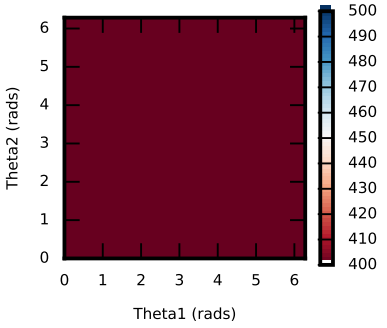


(c) 10 Iterations

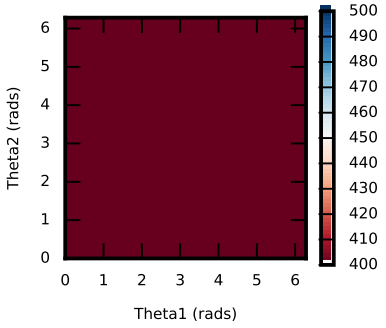


(d) 100 Iterations

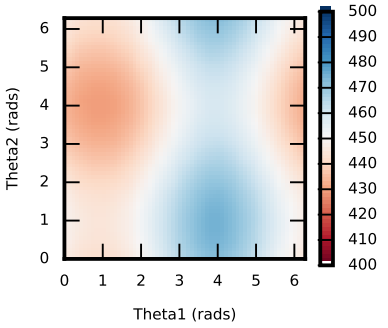
Figure 6-27: State cost evolution with respect to evader x and y position when pursuer is at (1,1) and $(\theta_1 = 0, \theta_2 = 0)$



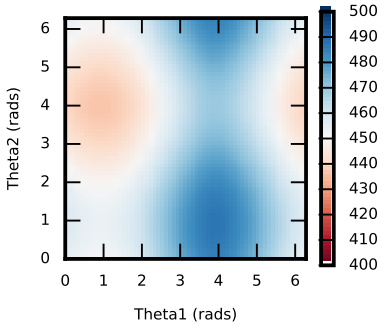
(a) 1 Iteration



(b) 2 Iterations



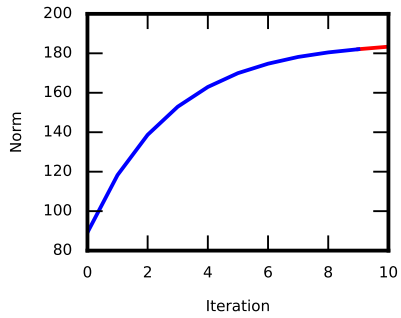
(c) 10 Iterations



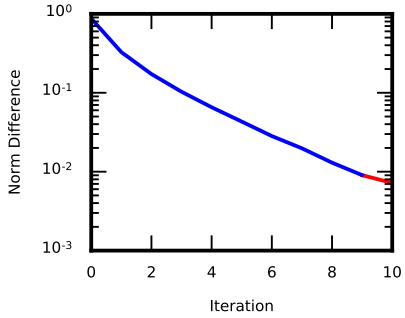
(d) 100 Iterations

Figure 6-28: State cost evolution with respect to heading when pursuer is at (1,1) and the evader is at (9,9)

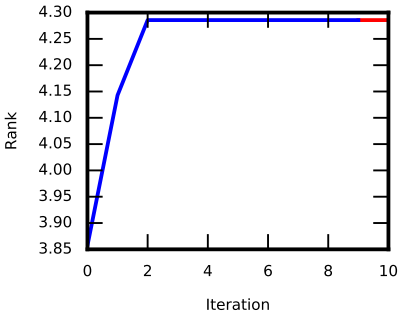
six-dimensional problem.



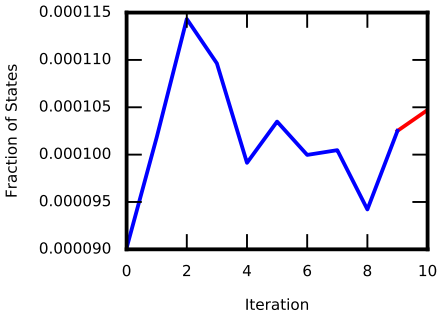
(a) Average norm of the pursuer state cost with red dashed line as the average norm of the optimal value



(b) Normalized difference in average norm between sequential pursuer state costs



(c) Average rank of the pursuer state cost tensor

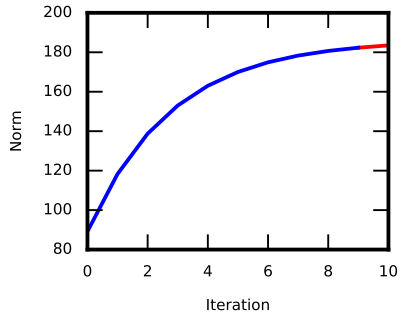


(d) Fraction of states used for the pursuer state cost tensor

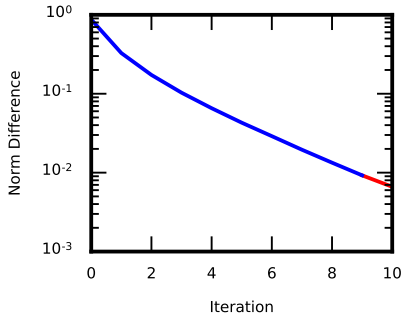
Figure 6-29: Pursuer diagnostic results of running BR-TT-VI Algorithm with ($\Delta_{max} = 10^{-2}, \delta_{max} = 10{-}2$). Blue solid line is the first run, while the red solid line is the second run

Just as in the four-dimensional problem, the pursuer state costs monotonically decrease toward the evaders position, Figure 6-31a, while the evader's state costs monotonically decrease toward the pursuers position,Figure 6-31b. The state costs follow a similar pattern with regard to the heading of the pursuer and the evader. It can be seen in Figure 6-31c that the state cost monotonically decreases when the pursuer or evader heading is toward the other player.

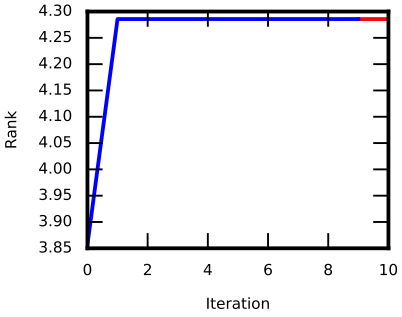
Using the optimal state costs $J_p^{(*,*)}$ and $J_e^{(*,*)}$, two different problems are solved. The first is a standard problem where the pursuer starting at (1,1) and with a head $\theta_1 = 0$ chases an evader starting at (5,5) with a heading of $\theta_2 = 0$. The pursuer and



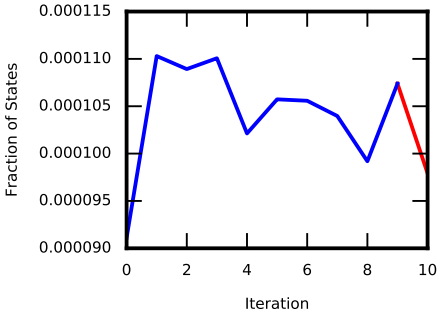
(a) Average norm of the evader state cost with red dashed line as the average norm of the optimal value



(b) Normalized difference in average norm between sequential evader state costs

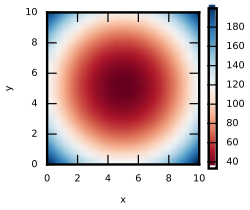


(c) Average rank of the evader state cost tensor

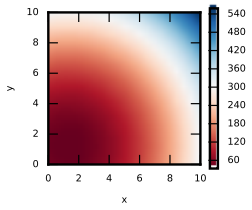


(d) Fraction of states used for the evader state cost tensor

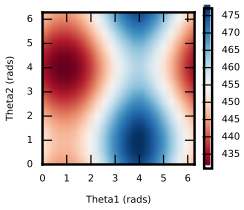
Figure 6-30: Evader diagnostic results of running BR-TT-VI Algorithm with ($\Delta_{max} = 10^{-2}, \delta_{max} = 10^{-2}$). Blue solid line is the first run, while the red solid line is the second run



(a) Pursuer state cost in respect to the pursuer's x and y position when evader is at (5,5) and ($\theta_1 = 0, \theta_2 = 0$)



(b) Evader state cost in respect to the evader's x and y position when pursuer is at (1,1) and ($\theta_1 = 0, \theta_2 = 0$)

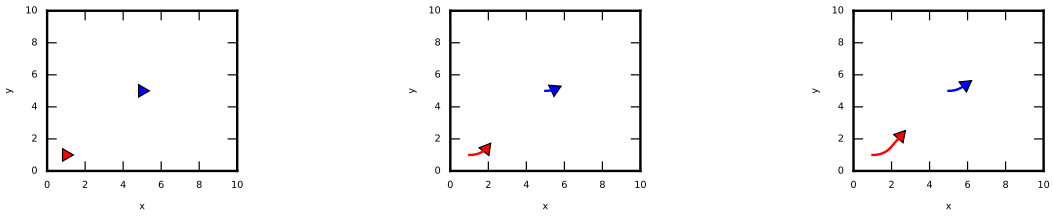


(c) Evader state cost with respect to θ_1 and θ_2 when pursuer is at (1,1) and the evader is at (9,9)

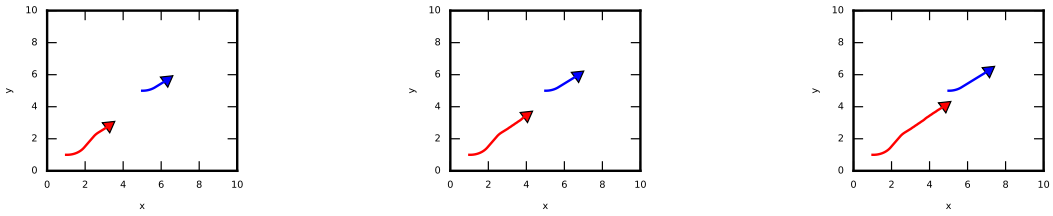
Figure 6-31: State Costs after running Best Response TVI Algorithm with ($\Delta_{max} = 1E-2, \delta_{max} = 1E-2$)

evader both adjust their headings to the optimal solution of approximately $\pi/4$. Due to the pursuer's greater speed, the game has the pursuer constantly decreasing the distance between itself and the evader until capture as can be seen in Figures 6-32 and 6-33.

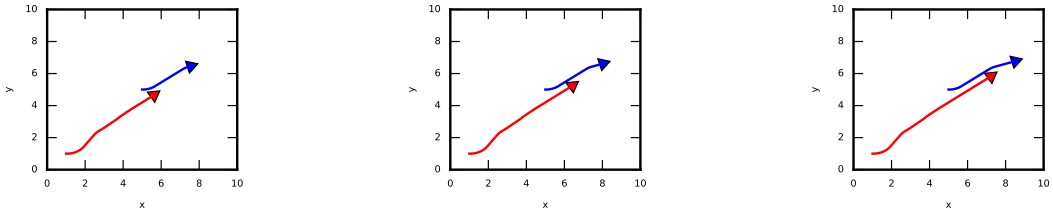
The second problem involves the pursuer starting at (7,5) with a heading of $\theta_1 = 0$ and the evader behind the pursuer at (5,5) and a heading of $\theta_2 = 0$. In this game the pursuer has to make a wide sweep around in order to get on the tail of the evader while the evader dodges below the pursuer,Figures 6-34 and 6-35a. In this case the pursuer actually has to increase the distance between itself and the evader before it can close the distance as seen in Figure 6-35b. This is a counter intuitive movement for the pursuer when examining the value function Equation (6.23). This shows that BR-TT-VI can solve non-intuitive problems.



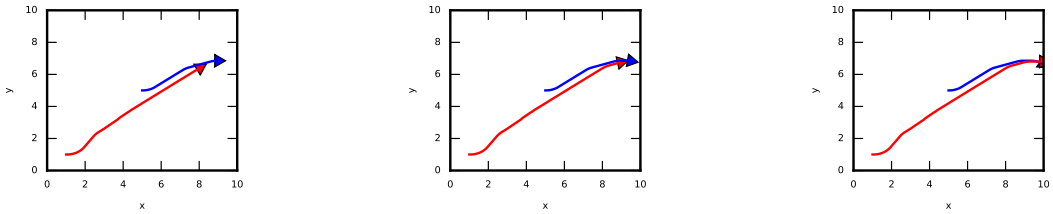
(a) 0 s. (b) 1 s. (c) 2 s.



(d) 3 s. (e) 4 s. (f) 5 s.

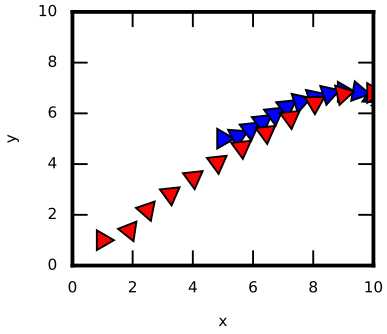


(g) 6 s. (h) 7 s. (i) 8 s.

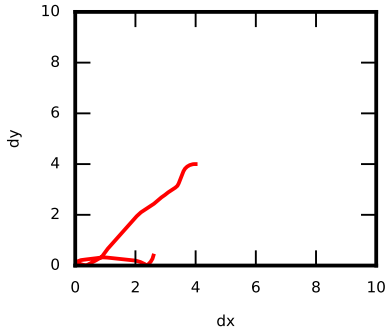


(j) 9 s. (k) 10 s. (l) 11 s.

Figure 6-32: Pursuit-Evasion Game with Pursuer starting at (1,1) with $\theta_1 = 0$ heading and Evader starting at (5,5) with $\theta_2 = 0$ heading

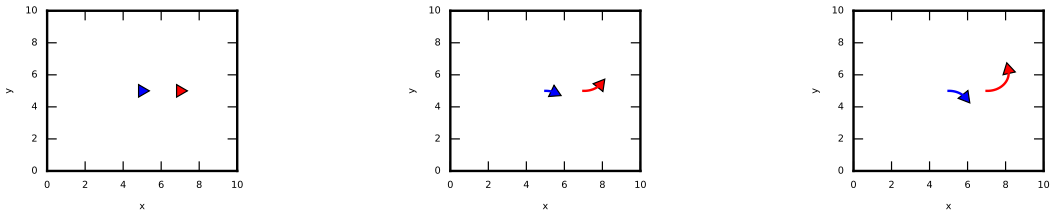


(a) Position of Pursuer and Evader with one second intervals and heading

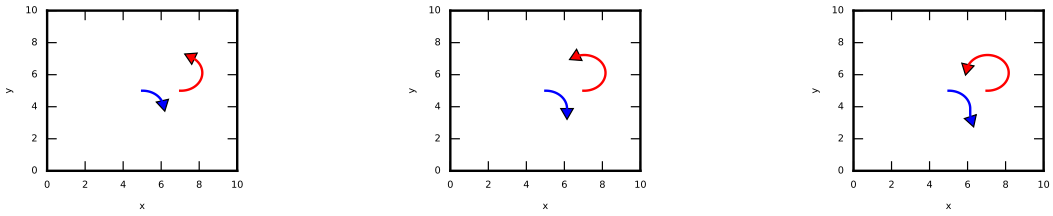


(b) Relative distance between pursuer and evader

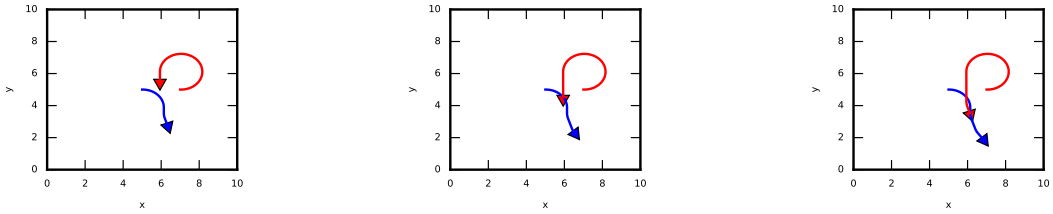
Figure 6-33: Pursuit-Evasion Game with Pursuer starting at (1,1) with $\theta_1 = 0$ heading and Evader starting at (5,5) with $\theta_2 = 0$ heading



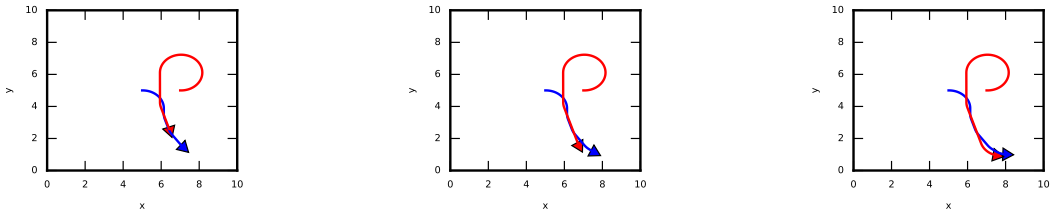
(a) 0 s. (b) 1 s. (c) 2 s.



(d) 3 s. (e) 4 s. (f) 5 s.

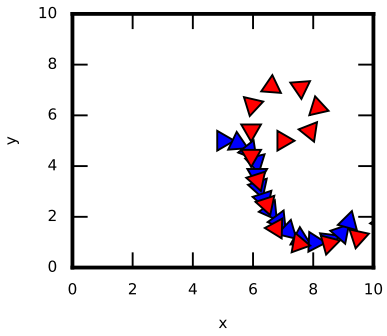


(g) 6 s. (h) 7 s. (i) 8 s.

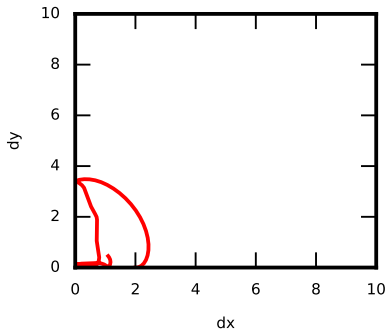


(j) 9 s. (k) 10 s. (l) 11 s.

Figure 6-34: Pursuit-Evasion Game with Pursuer starting at (7,5) with $\theta_1 = 0$ heading and Evader starting at (5,5) with $\theta_2 = 0$ heading



(a) Position of Pursuer and Evader with one second intervals and heading



(b) Relative distance between pursuer and evader

Figure 6-35: Pursuit-Evasion Game with Pursuer starting at (7,5) with $\theta_1 = 0$ heading and Evader starting at (5,5) with $\theta_2 = 0$ heading

Chapter 7

Conclusion

Best-Response Tensor-Train-Decomposition-Based Value Iteration and the underlying TT-VI algorithm have a number of advantages and disadvantages. This chapter will begin by exploring both the advantages and disadvantages of TT-VI as they relate to solving pursuit-evasion problems. A reflection on the constraints imposed by TT-VI's disadvantages will also be included in the first section. After detailing the benefits and constraints of TT-VI, a number of potential improvements for future research will be suggested. Finally, last remarks will be given on the importance of the research presented in this paper.

7.1 Summary of Results

Tensor-train-decomposition-based value iteration's ability to compute a solution in an efficient manner is the method's single greatest benefit. This method reduces the computational time of a four-dimensional problem to approximately a tenth of the time and makes a six-dimensional problem efficiently computable. However, TT-VI and the BR-TT-VI algorithm still have a variety of problems. First, as stated in Chapter 6, TT-VI only provides an approximation which can be held to some accuracy ϵ. While ϵ can be reduced to some very small value, doing so can reduce the benefits of TT-VI by resulting in approximate tensors with large tensor ranks. As noted in Chapter 3, the larger the ranks become the computational complexity of

using tensor-trains increases dramatically. For this reason, using TT-VI often resulted in a balancing act of accuracy and complexity. Because of this, using TT-VI often required manually determining an accuracy that would maintain low-rank.

Maintaining low-rank often created restrictions on the modeling of a problem. Creating large capture regions or other discontinuity producing areas often strained TT-VI in finding a low-rank approximation. This was especially troublesome considering that the approximations often took the largest toll when the pursuer and evader were in close distance of each other. These restrictions required a relatively simple game in which the pursuer tried to minimize the distance between itself and the evader while the evader attempted to maximize this value.

The approximate nature of TT-VI also created a problem with potentially negative values. Often when the state cost for a particular state was either zero or close to zero, TT-VI would approximate this to a negative value. These values could potentially spiral out of control making the entirety of the state space converge to $-\infty$ as the number of iterations approached ∞. For this reason a constant of 10 was added to the state cost in Equations (6.5) and (6.22).

Despite a variety of shortcomings, BR-TT-VI makes numerous problems efficiently solvable as detailed in Chapter 6. As noted for the six-dimensional problem, using TT-VI reduces the time required to find a solution from weeks for VI to less than half an hour. For some problems, such as the four-dimensional problem in Chapter 6, the results forBR-TT-VI and VI are indistinguishable with BR-TT-VI taking only a tenth of the time.

7.2 Future Directions

The BR-TT-VI algorithm provides a great basis for improvement. As already noted, BR-TT-VI requires manual input of three different accuracy values, one for the tensor, value iteration, and best response. An algorithm capable of adjusting each of these accuracies to ensure the most accurate solution possible without causing either the rank to become too large or for the algorithm to enter a never ending loop would

greatly assist in the usability of the algorithm. This modified algorithm could self-adjust to determine the optimal accuracy of the problem without requiring prior research or guess work to determine sufficient accuracies.

Numerous potential methods could be used to increase the accuracy of the solution by combining BR-TT-VI with other methods. One such method could be to use the TT-VI approximation to create a full state array and continue the algorithm with traditional value iteration. Another method could be to use TT-VI to approximate the majority of the state space while using VI to determine small areas of the state space that either require more precise accuracy or are not naturally low rank such as absorption regions.

Finally there are numerous ways to test BR-TT-VI with a more complex state space or dynamics. This paper has constrained its use of BR-TT-VI to vehicles navigating a two-dimensional space such as with cars. There is potential that best response can be used to navigate three-dimensional space such as with aircraft that can have variable altitude. More complex dynamics could include adding the effects of friction or wind resistance. Applying BR-TT-VI to an environment with obstacles would also require overcoming a variety of complications due to discontinuities in the state space. These are just a few of the many future opportunities to conduct research with BR-TT-VI.

7.3 Final Remarks

Best-Response Tensor-Train-Decomposition-Based Value Iteration builds on the work of a variety of methods in order to efficiently solve pursuit-evasion games. The computational efficiency of BR-TT-VI enables platforms with limited computational power to solve complex problems in an efficient manner. This is especially true for autonomous vehicles in pursuit-evasion environments. For example, suppose BR-TT-VI was ran overnight to produce controls for an autonomous spy vehicle to tail a certain target using a vehicle that has Dubins dynamics with a certain maximum speed and turning radius. However, halfway through the mission the target decides to change

vehicles to one with a different maximum speed or turning radius. BR-TT-VI would allow the autonomous spy vehicle to quickly approximate new optimal controls and continue with the mission. BR-TT-VI holds the potential to solve problems that would take days in hours and problems that would take hours in minutes.

Bibliography

[1] T. BaÅ§ar and G. Olsder. *Dynamic Noncooperative Game Theory, 2nd Edition*. Society for Industrial and Applied Mathematics, New York, 1998.

[2] M. Bardi and P. Soravia. Hamilton-jacobi equations with singular boundary conditions on a free boundary and applications to differential games. *Transactions of the American Mathematical Society*, 325(1):205–229, 1991.

[3] M. Bardi, P. Soravia, and M. Falcone. Fully discrete schemes for the value function of pursuit-evasion games. In T. Basar and A. Haurie, editors, *Advances in Dynamic Games and Applications, part II*, pages 89–105. Birkhäuser Boston, 1994.

[4] D. Bertsekas. *Dynamic Programming and Optimal Control, Volume 1*. Athena Scientific, Belmont, Massachusetts, 2005.

[5] L.E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

[6] A. Gorodetsky, S. Karaman, and Y. Marzouk. Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition. *Robotics: Science and Systems*, 2015.

[7] R. Isaacs. *Differential Games*. Wiley, Chichester, 1965.

[8] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for a class of pursuit-evasion games. In David Hsu, Volkan Isler, Jean-Claude Latombe, and Ming C. Lin, editors, *Algorithmic Foundations of Robotics IX: Selected Contributions of the Ninth International Workshop on the Algorithmic Foundations of Robotics*, pages 71–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[9] I.V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[10] I.V. Oseledets and E. Tyrtyshnikov. Tt-cross approximation for multidimensional arrays. *Linear Algebra and its Applications*, 432(2010):70–88, 2009.