University of Basel

# Learning Heuristic Functions Through Supervised Learning

Master Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence
http://ai.cs.unibas.ch

Examiner: Prof. Dr. Malte Helmert
Supervisors: Dr. Thomas Keller and Cedric Geissmann

Đorđe Relić
dorde.relic@unibas.ch
15-050-321

June 26$^{th}$, 2017

# Acknowledgments

Firstly, I would like to thank Dr. Thomas Keller and Cedric Geissmann for their mentoring, advice and patience during the course of the last six months. I would also like to thank Prof. Dr. Malte Helmert for allowing me to write this thesis at the Artificial Intelligence research group. Last but not the least, I would like to thank my colleagues, my friends and especially my family for their support during this period and also during the entire time of my studies.

# Abstract

Probabilistic planning is a research field that has become popular in the early 1990s. It aims at finding an optimal policy which maximizes the outcome of applying actions to states in an environment that feature unpredictable events. Such environments can consist of a large number of states and actions which make finding an optimal policy intractable using classical methods. Using a heuristic function for a guided search allows for tackling such problems. Designing a domain-independent heuristic function requires complex algorithms which may be expensive when it comes to time and memory consumption.

In this thesis, we are applying the supervised learning techniques for learning two domain-independent heuristic functions. We use three types of gradient descent methods: *stochastic*, *batch* and *mini-batch gradient descent* and their improved versions using *momentum*, *learning decay rate* and *early stopping*. Furthermore, we apply the concept of feature combination in order to better learn the heuristic functions. The learned functions are provided to Prost, a domain-independent probabilistic planner, and benchmarked against the winning algorithms of the International Probabilistic Planning Competition held in 2014. The experiments show that learning an offline heuristic improves the overall score of the search for some of the domains used in aforementioned competition.

# Table of Contents

# 1 Introduction

Probabilistic planning has become a popular research field in the AI community since the early 1990s. It is applied in environments that feature unpredictable events. An environment is defined by a set of states, a set of actions and a reward function that maps states to actions. An environment features unpredictable events if each action leads from one state to one or more states with a given probability and a given outcome. The aim of probabilistic planning is finding a mapping from states to actions, which is called a policy. A policy that maximizes the outcome of taking actions is called an optimal policy. For a given starting state, finding an optimal policy is done through the exploration of the search space.

Environments that feature a large number of states and actions make the problem of finding an optimal policy intractable using classical methods due to resource constraints such as time and memory. Using heuristic search algorithms allows for tackling planning problems that include a large number of states and actions. It is based on the idea of using a heuristic function for guided exploration of the search space by predicting the outcome of actions while trading optimality for speed. Domain-dependent planners tend to use hand-crafted heuristic functions. Deep Blue [1] was one of the algorithms that used a highly complex hand-crafted heuristic function that contributed to the first ever win of a machine over a human professional player in chess. Silver et. all [2] in 2016 presented AlphaGo, an algorithm that uses artificial neural networks, a subset of machine learning, to learn a heuristic function for the traditional game Go. Due to the enormous search space of Go, AlphaGo's success in beating world's best professional Go players shows that the idea of using machine learning techniques for learning a heuristic function can be highly successful. However, domain dependence induces inapplicability of one heuristic function to different domains.

Generation of a domain-independent heuristic function requires complex algorithms which may result in expensive algorithms when it comes to time and memory consumption. In this thesis, we implement learning of heuristic functions for domain independent probabilistic planning tasks using machine learning techniques. We use gradient descent methods to approximate two different heuristic functions. Furthermore, we optimize the gradient descent by using the concepts of momentum, learning decay rate and early stopping. In order to learn the heuristic more accurately, we apply the concept of feature combination. We compare the results against the winning algorithm of the International Probabilistic Planning Competition (IPPC) held in 2014 and showcase the improved performance for certain domains.

This thesis is divided into five chapters. In the first chapter, we introduce the theoretical background needed for defining a planning task using the model of Markov Decision Process as our state space. Chapter 3 is the main contribution of this thesis where we introduce machine learning as a concept and showcase how we applied supervised learning to learn offline heuristic functions. In chapter 4 we evaluate the results by running experiments on 120 different problem setups including 12 different domains and 10 different instances of each of the domains used at the IPPC 2014. In the last chapter of this thesis, we conclude the work and provide suggestions for future work.

# 2 Background

This chapter will introduce the theoretical background needed to present the methods used throughout this thesis. We formally define the *Markov Decision Processes (MDPs)* and introduce the *Trial-based Heuristic Tree Search (THTS).*

## 2.1 Markov Decision Process

One of the approaches in modeling the problem of planning under uncertainty is using the Markov decision processes (MDPs) [3]. The earliest definition of MDPs goes at least back to the work of Bellman [4]. Here, we will use the notations from the work of Keller [5] and adapt it for the use in this thesis.

**Definition 2.1.1** (Markov Decision Process). *A MDP is a mathematical framework defined by a 6-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, H, s_0 \rangle$ where we have:*

- $\mathcal{S}$ *– the finite set of* ***states***
- $\mathcal{A}$ *– the finite set of* ***actions***
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *– the* ***transition function***
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *– the* ***reward function***
- $H \in \mathbb{N}$ *– the* ***finite horizon***
- $s_0 \in \mathcal{S}$ *– the* ***initial state***

*where $\mathcal{T}$ gives the probability $\mathbb{P}_{\mathcal{T}}[s'|s, a]$ that applying an action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ leads to state $s' \in \mathcal{S}$. $\mathcal{R}$ represents the reward given when action $a \in \mathcal{A}$ is applied in state $s \in \mathcal{S}$.*

**Definition 2.1.2** (Policy). *For a given MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, H, s_0 \rangle$ a* ***policy*** *is a mapping $\pi : \mathcal{S} \times \{1, ..., H\} \to \mathcal{A}$.*

**Definition 2.1.3** (Value functions). *Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, H, s_0 \rangle$ be a MDP, $s \in \mathcal{S}$ a state and $\pi$ a policy. The* ***state-value*** *$V_\pi(s, d)$ of $s$ under $\pi$ with $d$ steps to go is defined as*

$$V_\pi(s,d) = Q_\pi(s,\pi(s,d))$$

*where the **action-value** $Q_\pi(s,\pi(s,d))$ under $\pi$ is defined as*

$$Q_\pi(s,a) = \begin{cases} \mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} (\mathbb{P}_\mathcal{T}[s'|s,a] \cdot V_\pi(s', d-1)) & ,d > 0 \\ \mathcal{R}(s,a) & ,\ \textit{otherwise} \end{cases}$$

*for all state-action pairs $(s,a)$.*

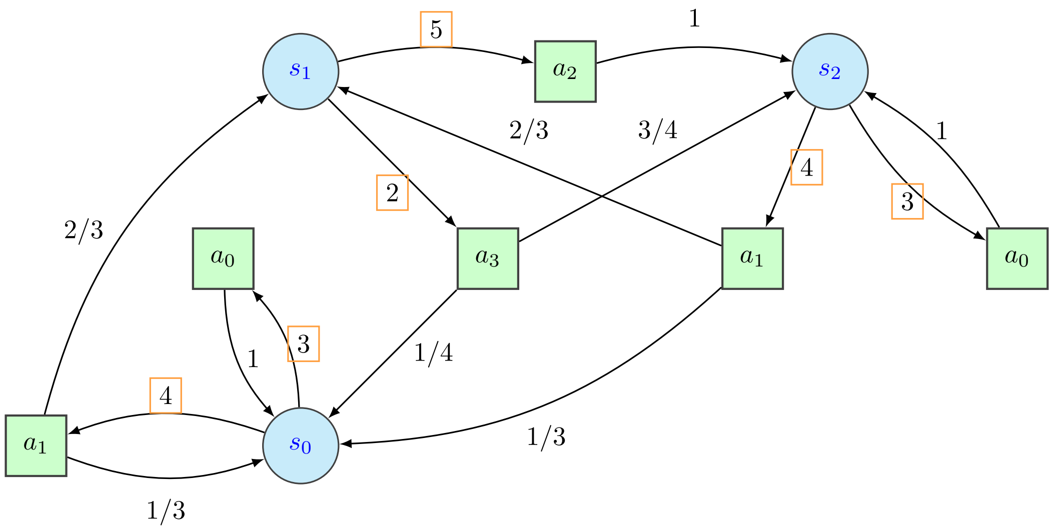


Figure 2.1: Example of a MDP with 3 states and 2 actions.

Choosing an optimal policy is the core problem of MDPs. Since the number of policies in a MDP is in $O(|\mathcal{A}|^{(|\mathcal{S}|\cdot H)})$, the problem of finding an optimal policy is intractable for MDPs with a large number of states and actions. The *Bellman Optimality Equation* [4] describes the optimal policy.

**Definition 2.1.4** (Optimal policy)**.** *Let the Bellman optimality equation for a state $s \in \mathcal{S}$ be a set of equations that describe $V^*(s,d)$, where*

$$V^*(s,d) = max_{a \in \mathcal{A}} Q^*(s,d,a)$$

$$Q^*(s,d,a) = \begin{cases} \mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} (\mathbb{P}_\mathcal{T}[s'|s,a] \cdot V^*(s', d-1)) & ,d > 0 \\ \mathcal{R}(s,a) & ,\ \textit{otherwise} \end{cases}$$

*A policy $\pi^*$ is an **optimal policy** if $\pi^* \in arg\ max_{a \in \mathcal{A}} Q^*(s,d,a)$ for all $s \in \mathcal{S}$*

**Example 2.1.1.** *Figure 2.1 showcases a simple MDP where we have $\mathcal{S} = \{s_0, s_1, s_2\}$ and $\mathcal{A} = \{a_0, a_1, a_2, a_3\}$. Taking an action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ is rewarded with the value in the orange square, and it leads from $s$ to a successor state with the given probability. An example policy is "always pick an action with the maximum reward", i.e. $\pi(s_0,d) = a_1$, $\pi(s_1,d) = a_2$, $\pi(s_2,d) = a_1$, for any $d$. If the horizon is $H = 4$ and we set the starting state*

*$s_0$, we have different outcomes due to the probabilistic property of action outcome. The best case scenario is the following sequence $s_0 \to a_1 \to s_1 \to a_2 \to s_2 \to a_1 \to s_1 \to a_2 \to s_2$ with total reward of 18 while one of the worst case scenarios results in the sequence $s_0 \to a_0 \to s_0 \to a_0 \to s_0 \to a_0 \to s_0 \to a_0 \to s_0$ and total reward of 12.*

Depending on the domain, MDPs can become huge in terms of the number of states. Enumerating states and keeping track of them becomes hard. In order to define a state in a different way, we define a *finite-domain variable.* Usage of finite-domain variables allows for a compact representation of a large number of states.

**Definition 2.1.5** (Finite-domain variable)**.** *A **finite-domain variable** is a mathematical object $v$, associated with a finite set of values, the **domain** $\mathcal{D}_v$ of $v$. The set $\mathcal{D}_v^+ := \mathcal{D}_v \cup \{\bot\}$ is called the **extended domain** of $v$, where $\bot \notin \mathcal{D}_v$ is the **undefined** value.*

**Definition 2.1.6** (State)**.** *A **partial variable assignment** over a finite set of variables $\mathcal{V}$ is a mapping $s : \mathcal{V} \to \cup_{v\in\mathcal{V}}\mathcal{D}_v^+$ such that for every finite-domain state variable $v \in \mathcal{V}$, $s[v]$ is defined in $\mathcal{D}_v^+$. The **scope** of $s$ is the set of variables where $s$ does not have the undefined value $\bot$; it is denoted as $vars(s) := \{v \in \mathcal{V} | s[v] \neq \bot\}$. A partial variable assignment $s$ is called a **state** iff $vars(s) = \mathcal{V}$.*

**Definition 2.1.7** (Fact)**.** *A **fact** is a tuple $\langle v, d\rangle$ where $v \in \mathcal{V}$ and $d \in \mathcal{D}_v$. We define the **set of all facts** as $\mathcal{F} = \{\langle v, d\rangle | v \in \mathcal{V} \wedge d \in \mathcal{D}_v\}$ and the **set of facts in state** $s$ as $\mathcal{F}(s) = \{\langle v, d\rangle | v \in \mathcal{V} \wedge d \in \mathcal{D}_v \wedge s[v] = d\} \subset \mathcal{F}$ where $s \in \mathcal{S}$.*

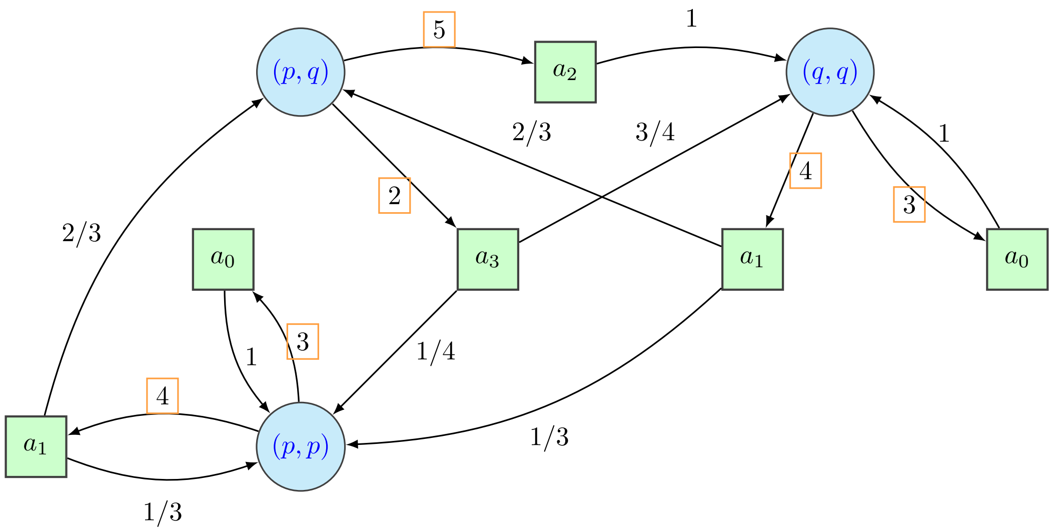


Figure 2.2: Example of MDP with 3 states and 2 actions.

**Definition 2.1.8** (Planning task)**.** *A **planning task** is a 4-tuple $T = \langle \mathcal{V}, \mathcal{A}, H, s_0\rangle$ where:*

- *$\mathcal{V}$ – the finite set **of finite-domain variables** $v$ with domain $\mathcal{D}_v$*
- *$\mathcal{A}$ – the finite set of **actions**. An action $a \in \mathcal{A}$ is a tuple $\langle effect_a, reward_a\rangle$ where*

  - *$\mathit{effect}_a$ – is a probability distribution over partial variable assignment $\{(p_i^a, e_i^a)\}_{i=1}^n$ where $p_i^a$ is a probability, $e_i^a$ is a partial variable assignment and $\sum_{i=1}^{n} p_i^a = 1$*
  - *$\mathit{reward}_a$ – the reward of applying action $a$*

- *$H \in \mathbb{N}$ – the **finite horizon***
- *$s_0 \in \mathcal{V}$ – the **initial state***

The application of a partial variable assignment $e^a$ to a state $s \in \mathcal{S}$ is the state $s' \in \mathcal{S}$ where

$$s'[v] = \begin{cases} s[v], & \text{if } e[v] = \bot \\ e[v], & \text{otherwise} \end{cases} \tag{2.1}$$

for all $v \in \mathit{vars}(s')$.

Definition 2.1.8 enables us to formalize the problem we are trying to solve. A planning task $T = \langle \mathcal{V}, \mathcal{A}, H, s_0 \rangle$ induces MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, H, s_0 \rangle$ in notation $T_\mathcal{M}$ where:

- $\mathcal{V}$ induces set of states $\mathcal{S} = \mathcal{D}_v^\mathcal{V}$
- $\mathcal{R}(s, a) = \mathit{reward}_a$ for $a \in \mathcal{A}$ and all $s \in \mathcal{S}$
- $\mathcal{T}(s, a, s') = \sum_{(p^a, e^a) \in \mathit{effect}_a} p^a$ where $p^a$ is the probability that applying partial variable assignment $e^a$ in state $s$ results in state $s'$

**Example 2.1.2.** *Let us describe the Example 2.1.1 by using definition 2.1.8. We define set of variables $\mathcal{V} = \{v_0, v_1\}$ and domain $\mathcal{D}_v = \{p, q\}$. In Figure 2.2 we have a factored representation of states states $s_0 = [p, p]$, $s_1 = [p, q]$ and $s_2 = [q, q]$. We have set of actions $\mathcal{A} = \{a_0, a_1, a_2, a_3\}$ defined in tuples:*

- $a_0 = \langle \mathit{effect}_{a_0}, \mathit{reward}_{a_0} \rangle$
  - $\mathit{effect}_{a_0} = \{(e_1^{a_0}, p_1^{a_0})\}$ *where* $e_1^{a_0} = \{v_0 = \bot, v_1 = \bot\}$ *and* $p_1^{a_0} = 1$
  - $\mathit{reward}_{a_0} = 3$
- $a_1 = \langle \mathit{effect}_{a_1}, \mathit{reward}_{a_1} \rangle$
  - $\mathit{effect}_{a_1} = \{(e_1^{a_1}, p_1^{a_1}), (e_2^{a_1}, p_2^{a_1})\}$ *where*
    - $e_1^{a_1} = \{v_0 = p, v_1 = q\}$ *and* $p_1^{a_1} = 2/3$
    - $e_2^{a_1} = \{v_0 = p, v_1 = p\}$ *and* $p_2^{a_1} = 1/3$
  - $\mathit{reward}_{a_1} = 4$
- $a_2 = \langle \mathit{effect}_{a_2}, \mathit{reward}_{a_2} \rangle$
  - $\mathit{effect}_{a_2} = \{(e_1^{a_2}, p_1^{a_2})\}$ *where* $e_1^{a_2} = \{v_0 = q, v_1 = q\}$ *and* $p_1^{a_2} = 1$
  - $\mathit{reward}_{a_2} = 5$
- $a_3 = \langle \mathit{effect}_{a_3}, \mathit{reward}_{a_3} \rangle$

- $effect_{a_3} = \{(e_1^{a_3}, p_1^{a_3}), (e_2^{a_3}, p_2^{a_3})\}$ *where*
  - $e_1^{a_3} = \{v_0 = q, v_1 = \perp\}$ *and* $p_1^{a_3} = 3/4$
  - $e_2^{a_3} = \{v_0 = \perp, v_1 = p\}$ *and* $p_2^{a_3} = 1/4$
- $reward_{a_3} = 2$

*From Example 2.1.1 we know what the best case scenario for applying policy "always pick an action with the maximum reward" is. In terms of effects, the best case scenario looks like this:* $(p,p) \rightarrow e_1^{a_1} \rightarrow (p,q) \rightarrow e_1^{a_2} \rightarrow (q,q) \rightarrow e_1^{a_1} \rightarrow (p,q) \rightarrow e_1^{a_2} \rightarrow (q,q)$*. Furthermore, from Definition 2.1.7 we have* $\mathcal{F}(s_0) = \{\langle v_0, p\rangle, \langle v_1, p\rangle\}$ *as the set of facts in state* $s_0$ *and a similarly we derive a set for each of the states in* $\mathcal{S}$*.*

## 2.2 Trial-based Heuristic Tree Search

*Trial-based Heuristic Tree Search (THTS)* is a framework that can model different approaches for solving finite horizon MDPs. THTS operates a *search tree*, which consists of two types of nodes: *decision nodes* and *chance nodes.* A **decision node** is a 4-tuple $node_d = \langle s, d, \hat{V}, n\rangle$ where $s \in \mathcal{S}$ represents a state, $d \leq H \in \mathbb{N}$ represents number of steps to go, $\hat{V} \in \mathbb{R}$ state-value estimate and $n$ the number of times $node_d$ was visited. A **chance node** is a 5-tuple $node_c = \langle s, d, a, \hat{Q}, n\rangle$ where $s \in \mathcal{S}$ represents a state, $d < H \in \mathbb{N}$ represents number of steps to go, $a \in \mathcal{A}$ an action, $\hat{Q} \in \mathbb{R}$ action-value estimate and $n$ is the number of times $node_c$ was visited. $\hat{V}$ and $\hat{Q}$ are estimates of value functions from Definition 2.1.3. THTS is split into five parts: *heuristic function*, *backup function*, *action selection*, *outcome selection* and *trial length.* Since THTS supports modeling many different approaches, we will describe only the parts of the framework we use in this thesis. For more details about THTS, consult the work of Keller and Helmert [6].

The **heuristic function** is used in the expansion phase. Unvisited chance nodes are initialized using a heuristic function $h : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ and added to the search tree.

The **backup function** updates nodes based on the values collected from one or more of their successor nodes. To deal with the problems that include a large number of outcomes, the decomposed representation of chance nodes presented in the work of Keller and Eyerich [7] is used. The backup function that we use in this thesis is the *partial Bellman backup.* The partial Bellman backup implements a partial version of the *full Bellman backup* [6], with difference of not requiring all successor nodes to be visited. The backup of the nodes is done by applying the following updates:

$$\hat{V}_{node_d} = \max_{node_c \in \text{ child}(node_d)} \hat{Q}(node_c) \tag{2.2}$$

While $\hat{Q}$ values in chance nodes $node_c$ are updated with:

$$\hat{Q}_{node_c} = \mathcal{R}(node_c) + \frac{\sum\limits_{node_d \in \text{ child}(node_d)} \mathbb{P}[node_d | node_c] \cdot \hat{V}_{node_d}}{\sum\limits_{node_d \in \text{ child}(node_d)} \mathbb{P}[node_d | node_c]} \tag{2.3}$$

In other words, $\hat{V}$ in decision nodes is updated with the maximum $\hat{Q}$ value of a child node $node_c$ while $\hat{Q}$ values are updated using the weighted outcome proportional to the probability of transition to its child nodes.

The **action selection** in THTS covers any action selection strategy including the ones present in algorithms such as RTDP [8] and AO* [9] (which prioritize greedy approach) as well as UCT [10] and AOT [11] (which prioritize balanced approach). The action selection that we use in this thesis is the UCB1 [12].

$$UCB1(node_d) = \underset{node_c \in \text{ child}(node_d)}{\text{argmax}} \hat{Q}_{node_c} + B\sqrt{\frac{\ln n_{node_d}}{n_{node_c}}} \tag{2.4}$$

where $B$ is a bias parameter set to $\hat{V}_{node_d}$.

During the **outcome selection**, the outcome of a chosen action has to be determined. Monte-Carlo sampling methods [13] are used which sample an outcome according to the probability.

The **trial length** is determined by the time or the number of trials needed for reaching a leaf node in the search tree.

The main contribution of this thesis is improving the **heuristic function** of THTS.

### 2.2.1 UCT*

UCT* is the algorithm presented in the work of Keller and Helmert [6]. Its basis is the *Upper Confidence Bound for Trees (UCT)* [10] and it converges to an optimal policy if given enough time [5]. The main differences to UCT is the usage of partial Bellman backup instead of Monte-Carlo backup and limiting the search using the horizon. More details on UCT* can be found in the work of Keller and Helmert [6]. Here, we will explain UCT* with an example.

**Example 2.2.1.** *Figure 2.3 shows an unfolding of the Example 2.1.2 into a search tree. Furthermore, part of the UCT* search tree is showcased. Every node has additional information attached to it which shows the value functions from Definition 2.1.3 as well as the number of times the node was visited. The decision nodes $node_d$ contain the state-value estimate $\hat{V}$ and chance nodes $node_c$ contain the action-value estimate $\hat{Q}$. Information about remaining steps to go is omitted from the graphical representation. If the root node $(p, p)$ has $d$ steps to go then child action nodes from the root node and their children have $d - 1$ steps to go. Since Figure 2.3 showcases 3 steps, we know that $d - 3 > 0$ holds. One trial of UCT* consists of the following phases:*

- *The **action selection** is done using the UCB1 formula (2.4).*
- *In the **expansion** phase, all child nodes of a $node_d$ are added to the search tree and initialized using a heuristic function.*

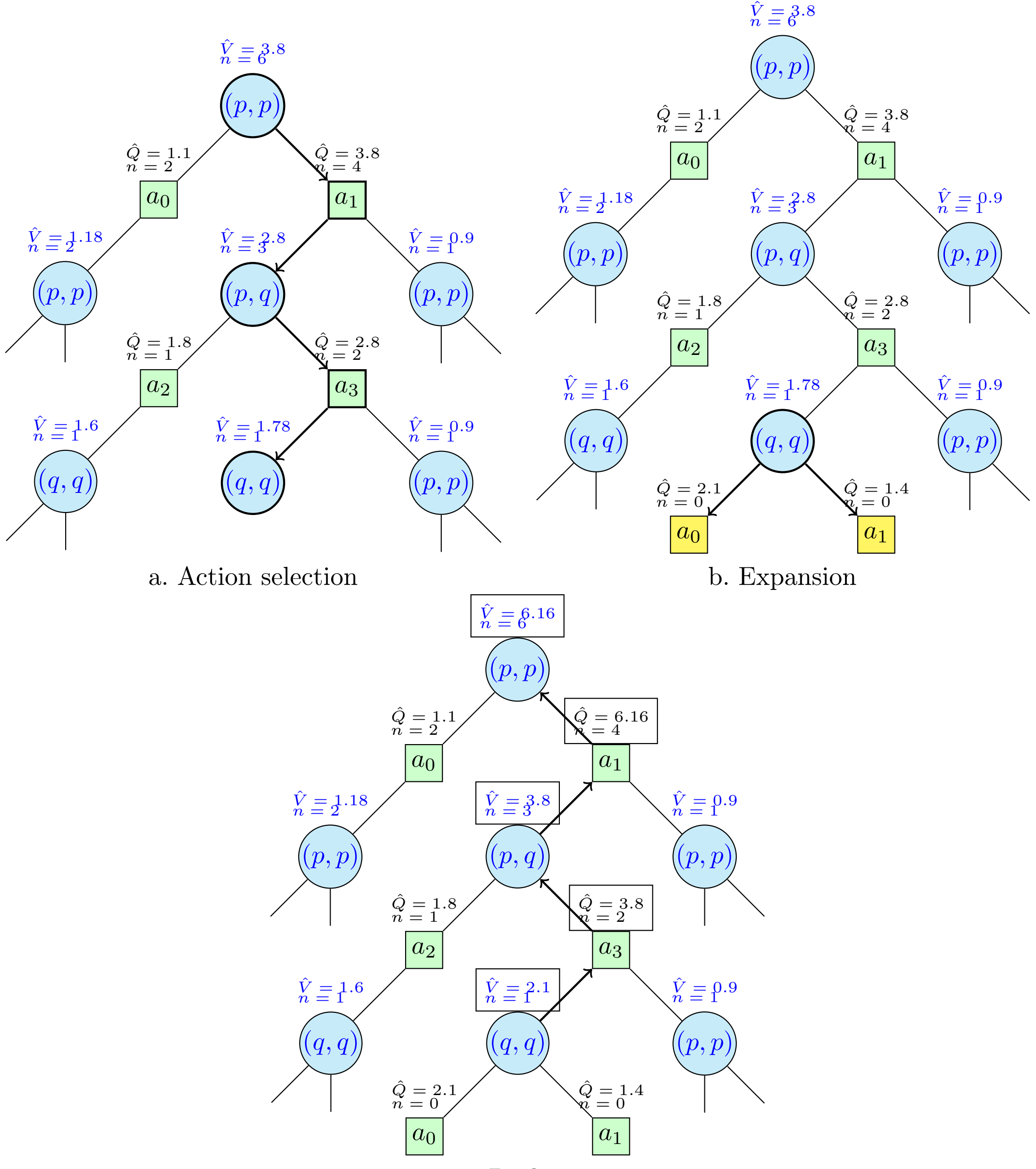


Figure 2.3: Example of UCT* algorithm

- *During the **backup** phase, all the nodes are updated using the partial Bellman backup presented in the Equation (2.2) for the decision nodes and in the Equation (2.3) for the chance nodes.*

*UCT* builds an asymmetric search tree and focuses on the more promising actions while also taking all the actions into consideration. It also focuses on states that are close to the root of the search tree, which results in convergence to an optimal policy when given enough trials.*

# 3 Learning the Heuristic

In Chapter 2, while describing THTS framework, we introduced the **heuristic function**. The heuristic function is used in the initialization of new nodes in the search tree and it greatly influences the performance of the search. However, general heuristic functions tend to be expensive when it comes to resource consumption such as time and memory. The main contribution of this thesis is learning a heuristic functions offline so that more resources can be allocated to the UCT* algorithm during the search.

## 3.1 Heuristic Function

Let $h^*$ be the heuristic function we want to learn. In Section 2.2 we defined a chance node $node_c = \langle s, d, a, \hat{Q}, n \rangle$. The heuristic function $h^*$ initializes $node_c$ with action-value estimate $\hat{Q}$. For a given state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ in $node_c$ we have the following formula for the action-value estimate:

$$h^*_a(s) = \hat{Q} \tag{3.1}$$

We can learn any function that, for a given state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, estimates the action-value $\hat{Q} \in \mathbb{R}$. In this thesis we will learn two heuristic functions: *Iterative Deepening Search (IDS)* and *Trial-based Heuristic Tree Search (THTS).*

**Iterative Deepening Search (IDS)** [7] is a breadth first search, which for a given state $s \in \mathcal{S}$ applies all actions $a \in \mathcal{A}$ and estimates the action-values using reward function $\mathcal{R}$ defined in previous chapter. The algorithm is recursively applied until a predefined maximum depth is reached. IDS results in a list of actions with corresponding action-value estimates.

**Trial-based Heuristic Tree Search (THTS)** [6] is described in Chapter 2. The configuration which we use here includes the IDS as a heuristic function, UCT* as action selection and partial Bellman backup as backup function. For a given state $s \in \mathcal{S}$, THTS applies all actions and returns the corresponding action-value estimates.

## 3.2 Data Set

The key ingredient we use for learning a heuristic function is the *data set*. From the data set we get the information how the heuristic function $h^*_a$ behaves for a given state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$. We describe a state using a set of facts defined in Definition 2.1.7. From set $\mathcal{F}(s)$ we derive **state features**.

Every data set entry consists of a given state features $\mathcal{F}(s)$ for a given state $s \in \mathcal{S}$ and action-value estimate $\hat{Q} = h^*_a(s)$. Since we want to learn a heuristic function for every action $a \in \mathcal{A}$ we have the following definition of a data set:

**Definition 3.2.1** (Data set)**.** *The data set for an action $a \in \mathcal{A}$ is a set $\mathfrak{D}_a = \{(s, \hat{Q})\}_{i=1}^{m}$, where $s \in \mathcal{S}$ represents the* ***state*** *which we also call input data, $\hat{Q}$ represents* ***action-value estimate*** *which we also call the response value and $m \in \mathbb{N}$ represents the size of data set.*

## 3.3 Machine Learning

We use *machine learning (ML)* techniques for learning the heuristic function from a given data set. ML is one of the most used techniques for *automated learning* from data. The result of applying ML techniques for automated learning is a function $\hat{h}_a$ that predicts the behavior of the function we want to learn, which is denoted with $h^*_a$ in our case. ML is a whole sub-field of computer science and more details can be found in the textbooks of Murphy[14] and Burch[15]. In this thesis we use the concept of **supervised learning**.

Depending on the properties of the response value, **supervised learning** is divided into *classification* and *regression*. Classification is learning where the response values are categorized and the number of categories is known while regression is learning where the response values are continuous. In this thesis, the response values in our data sets are continuous, hence we use regression.

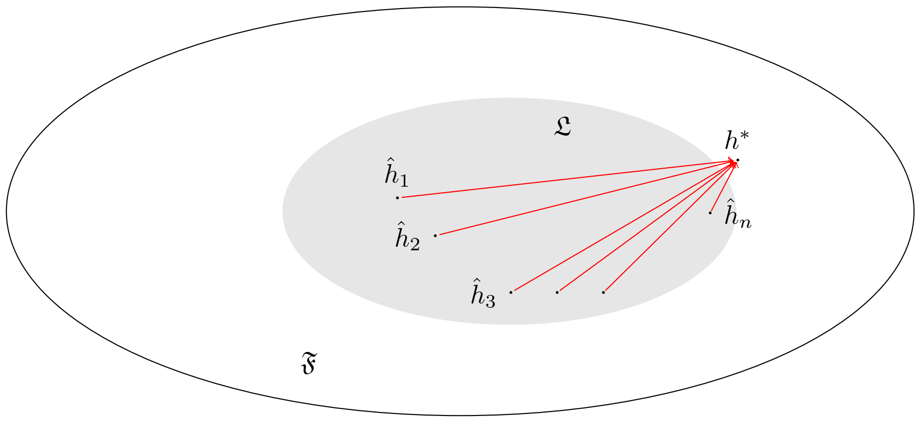


Figure 3.1: Approximating function $h^*$ with function $\hat{h}$ in space of linear functions $\mathfrak{L}$

Let $\mathfrak{F} = \{h \mid h : \mathcal{S} \to \mathbb{R}\}$ be the *space of all possible functions* that map states to real values. Function $h^*_a$ could be any function from space $\mathfrak{F}$. In this thesis, we approximate function $h^*_a$ with a linear function $\hat{h}_a \in \mathfrak{L} \subset \mathfrak{F}$. Learned function $\hat{h}_a$ is represented as a weighted sum of all features in $\mathcal{F}(s)$:

$$\hat{h}_a(s) = \sum_{w_f \in \mathcal{W}_a} w_f, f \in \mathcal{F}(s) \tag{3.2}$$

where $w_f$ is the weight assigned to state feature $f \in \mathcal{F}(s)$.

We start our learning by choosing a random function $h_a \in \mathfrak{L}$. Through an iterative process of learning we approximate function $h_a^*$ by updating the weights assigned to state features in $\hat{h}_a$. One iteration of learning consists of updating weights and is called an **epoch**. Function $\hat{h}_a$ is the resulting function after the update in the last epoch.

We use the *objective function* $J(\mathcal{W}_a, s)$, also called the **error function** for estimating how close are we in approximating function $h_a^*$ with function $\hat{h}_a$. The error function $J(\mathcal{W}_a, s)$ is a function of weights and the most commonly used one is the *mean squared error (MSE)* function.

$$J(\mathcal{W}_a, s) = (\hat{h}_a(s) - h_a^*(s))^2 \tag{3.3}$$

Figure 3.1 is a graphical representation of a linear value function approximation. After $n$ epochs, we get the function $\hat{h}_n$ with the lowest value of the error function, denoted in red. For the remainder of this thesis we will omit the notation of action when denoting a set of weights $\mathcal{W}$, the error function $J(\mathcal{W}, s)$, a data set $\mathfrak{D}$, the functions $\hat{h}$ and $h^*$. If not stated otherwise, those terms are related to the action $a \in \mathcal{A}$.

### 3.3.1 Gradient Descent

One of the most used optimization algorithms is gradient descent [16]. It is used to minimize the error function $J(\mathcal{W}, s)$. Minimization is performed in an iterative update of the weights $\mathcal{W}$ in direction opposite to the gradient of the error function $\nabla_{\mathcal{W}} J(\mathcal{W}, s)$ w.r.t. the weights. The **step size** $\alpha$, also denoted as the learning rate, determines how impacting the update is in every epoch. One epoch of gradient descent performs the following:

$$w_f := w_f - \alpha \frac{\partial J(\mathcal{W}, s)}{\partial w_f}, \forall w_f \in \mathcal{W} \tag{3.4}$$

The error function $J(\mathcal{W}, s)$, together with the step size $\alpha$, defines the rate of change of parameters $\mathcal{W}$. In Equation 3.3 we defined the error function for a single entry in the data set. However, we can use more than one data set entry for computing the error function. A more general form of the error function, which takes more data set entries into consideration is:

$$J(\mathcal{W}, s) = \frac{1}{2m} \sum_{(s,\hat{Q}) \in \mathfrak{D}} (\hat{h}(s) - \hat{Q})^2 \tag{3.5}$$

and by deriving we get:

$$\frac{\partial}{\partial w_f} J(\mathcal{W}, s) = \frac{1}{m} \sum_{(s,\hat{Q}) \in \mathfrak{D}} (\hat{h}(s) - \hat{Q}) \cdot \frac{\partial}{\partial w_f} \hat{h}(s), \forall w_f \in \mathcal{W} \tag{3.6}$$

where $m$ denotes the size of the data set used for computing the error function. If we look at the definition of function $\hat{h}$ in Equation 3.2, we get the following:

$$\frac{\partial}{\partial w_f}\hat{h}(s) = \begin{cases} 1, & \text{if } f \in \mathcal{F}(s) \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

where weight $w_f$ is assigned to state feature $f$. In other words, the weight is being updated with the derivative *if and only if* the feature to which the weight is assigned is present in set of facts of state $s$. In Equation 3.5, you can note the constant $\frac{1}{2}$ in front of the expression. Even when multiplied by a constant, the error function keeps its properties, however, multiplying the error function with $\frac{1}{2}$ enables a cleaner look of the derivative of the error function.

**Algorithm 1** Error function

```
1: function ERROR($weights$, $qValue$)
2:     Declare $qValue_{est} \leftarrow 0$
3:
4:     for $w$ in $weights$ do
5:         $qValue_{est} \leftarrow qValue_{est} + w$
6:     end
7:     return $qValue_{est} - qValue$
```

Depending on the number of data set entries used for computing the gradient, there are three types of gradient descent: *stochastic*, *batch* and *mini-batch gradient descent*. From Equation 3.6 the variable that is different for each of the gradient descent types is the number of data set entries used for computing the derivative. The common function for all gradient descents types is the ERROR function. The implementation of the ERROR function is presented in Algorithm 1.

#### 3.3.1.1 Stochastic Gradient Descent

In every epoch, stochastic gradient descent (SGD) performs an update of the weights for every single data set entry $(s, \hat{Q}) \in \mathfrak{D}$. The derivative in SGD is calculated in the following way:

$$\frac{\partial}{\partial w_f}J(\mathcal{W}, s) = (\hat{h}(s) - \hat{Q}) \cdot \frac{\partial}{\partial w_f}\hat{h}(s), \forall w_f \in \mathcal{W} \tag{3.8}$$

Algorithm 2 showcases a single epoch of stochastic gradient descent. The SGD approach results in high oscillation of the error function. On one hand, this enables SGD avoiding local minimums by jumping to other, possibly better, minimums. On the other hand, high oscillation might lead SGD to jumping out of global minimums and consequently ending in local minimums or even overshooting and ultimately diverging.

#### 3.3.1.2 Batch Gradient Descent

The batch gradient descent(BGD) takes the whole data set into consideration when calculating the derivative for a single update. The derivative in BGD is calculated in the same way as Equation 3.6:

**Algorithm 2** Stochastic gradient descent algorithm

```
1: function SGD($weights$, $dataSet$, $alpha$)
2:     Declare $\Delta_f \leftarrow 0$
3:
4:     for $s, qValue$ in $dataSet$ do
5:         $\Delta_f \leftarrow$ Error($weights, qValue$)
6:         for $f \in \mathcal{F}(s)$ do
7:             $weight[f] \leftarrow weight[f] + alpha \cdot \Delta_f$
8:         end
9:     end
```

$$\frac{\partial}{\partial w_f} J(\mathcal{W}, s) = \frac{1}{m} \sum_{(s,\hat{Q}) \in \mathfrak{D}} (\hat{h}(s) - \hat{Q}) \cdot \frac{\partial}{\partial w_f} \hat{h}(s), \forall w_f \in \mathcal{W} \tag{3.6}$$

where $m$ is the size of the data set $\mathfrak{D}$.

**Algorithm 3** Batch gradient descent algorithm

```
1: function BGD($weights$, $dataSet$, $alpha$)
2:     Declare $\Delta[\mathcal{F}(\mathcal{S})]$
3:
4:     for $s, qValue$ in $dataSet$ do
5:         for $f \in \mathcal{F}(s)$ do
6:             $\Delta[f] \leftarrow \Delta[f] +$ Error($weights, qValue$)
7:         end
8:     end
9:
10:    for $f \in \mathcal{F}(s)$ do
11:        $weights[f] \leftarrow weights[f] + alpha \cdot (\frac{\Delta[f]}{size(dataSet)})$
12:    end
```

BGD converges to the a minimum more slowly that SGD due to the fact that the changes depend on whole data set in a single update per epoch. Unlike in SGD, fluctuation of the error function is minimized. Algorithm 3 showcases an update of weights in a single epoch.

#### 3.3.1.3 Mini-Batch Gradient Descent

Mini-batch gradient descent (MBGD) is the mix of the two previous types. It takes a fixed part of the data set, called a *batch* and computes the derivatives using data set entries from the batch. The upper limit in the sum is $k < m$ while the lower limit is $j < k$. From Equation 3.6 we derive the following form of the derivative:

$$\frac{\partial}{\partial w_f} J(\mathcal{W}, s) = \frac{1}{k-j} \sum_{(s,\hat{Q}) \in \mathfrak{B}} (\hat{h}(s) - \hat{Q}) \cdot \frac{\partial}{\partial w_f} \hat{h}(s), \forall w_f \in \mathcal{W} \tag{3.9}$$

where $\mathfrak{B} = \{(s, \hat{Q})\}_j^k \subset \mathfrak{D}$ is a part of the data set with size of $(k - j)$ where $k > j$.

**Algorithm 4** Mini-batch gradient descent algorithm

```
function MBGD($weights$, $dataSet$, $alpha$, $eta$)
    Declare $\Delta[\mathcal{F}(\mathcal{S})]$
    Declare $batches \leftarrow$ GetBatches($dataSet, eta$)

    for $batch$ in $batches$ do
        for $s, qValue$ in $batch$ do
            for $f \in \mathcal{F}(s)$ do
                $\Delta[f] \leftarrow \Delta[f] +$ Error($weights, qValue$)
            end
        end

        $con \leftarrow$
        for $f \in \mathcal{F}(s)$ do
            $weights[f] \leftarrow weights[f] + alpha \cdot (\frac{\Delta[f]}{size(batch)})$
        end
    end
```

**Algorithm 5** Get batches

```
function GetBatches($dataSet$, $eta$)
    Declare $batches \leftarrow []$
    Declare $start \leftarrow 0$
    Declare $end \leftarrow \eta$

    while $start < size(dataSet)$ do
        $batches.append(dataSet[start : end])$
        $start \leftarrow end$
        $end \leftarrow end + \eta$

        if $end > size(dataSet)$ then
            $end \leftarrow size(dataSet)$
        end
    end
     return $batches$
```

One epoch consists of as many updates as there are batches. Algorithm 4 showcases an update done in one epoch of MBGD. The generation of batches is presented in Algorithm 5.

### 3.3.2 Summary and Challenges

If a function that we want to learn is complex, it can be really hard to converge to a global minimum using gradient descent methods. This is due to the fact that there is no best way to escape local minimums. While the SGD can be the fastest method to converge, due to the big jumps in updates, the high variation of the error function can lead to overshooting and divergence. The BGD, even though it is the slowest in convergence, reaches the minimum with a lower chance of overshooting. Whereas MBGD combines both approaches and minimizes the overshooting while converging to the minimum at a faster rate than BGD.

All methods share parameters such as the *step size* $\alpha$ and the number of *epochs*. Challenges lay in choosing the right values for these parameters. Setting the right value for **step size** $\alpha$ is challenging because small $\alpha$ leads to slow convergence and demands a high number of **epochs** while large $\alpha$ increases the variation of the error function and may even lead to divergence. There are many optimization variants of gradient descent, which are summarized in work of Ruder [16]. In this thesis, we apply three modifications: *momentum*, *learning decay rate* and *early stopping*.

**Algorithm 6** Optimized stochastic gradient descent algorithm

```
function OSGD($weights$, $dataSet$, $alpha$)
    Declare $\Delta_f \leftarrow 0$
    Declare $change \leftarrow 0$

    for $s, r$ in $dataSet$ do
        $\Delta_f \leftarrow$ ERROR($weights, qValue$)
        $change \leftarrow change + \Delta_f$
        for $f \in \mathcal{F}(s)$ do
            $weight[f] \leftarrow weight[f] + alpha \cdot \Delta_f + \gamma \cdot \Delta_{prev}[f]$
            $\Delta_{prev}[f] \leftarrow \Delta_f$
        end
    end

    if $change < \epsilon$ then
        return False
    end
    return True
```

**Momentum** [17], denoted with $\gamma$, optimizes gradient descent by using concepts from physics mechanics. The idea is in adding a value from the previous update $\Delta_{prev}$ to the new update value. This approach helps escaping local minimums. It also addresses the problem gradient descent methods face while navigating ravines [18]. Introducing **learning decay rate** [19], denoted with $\sigma$, also addresses the challenge of dealing with fluctuations of the objective function. The step size is reduced in a predefined schedule by a faction $\sigma$ of its value. Furthermore, we implement **early stopping** which halts the process of updating weights when the change in the objective function falls below the predefined threshold $\epsilon$. Changes in the implementation are minimal and are showcased for the SGD Algorithm in 6.

Problems that machine learning methods are facing are **overfitting** and **underfitting**. Both overfitting and underfitting occur when the chosen model is too weak to describe the data provided by the data set. Overfitting is detected when the approximated function $\hat{h}$ evaluates low error function values on the seen data (one seen during learning) while for the unseen data (one during prediction) it evaluates too high error function values. Overfitting is characteristic for models which contain a large number of parameters. Problem of underfitting is resembled in high error function values both on seen and unseen data.

Detecting the problem of overfitting and underfitting is done through evaluation of the learned function using the data from the given data set. The idea is splitting the data set into *training set* $\mathfrak{T}$ and *validation set* $\mathfrak{V}$ with following properties:

$$\mathfrak{D} = \mathfrak{T} \cup \mathfrak{V}$$
$$\emptyset = \mathfrak{T} \cap \mathfrak{V}$$
$$|\mathfrak{T}| > |\mathfrak{V}|$$

In this thesis, the ratio in which we split the data set is 7:3 in favour of the training set. After using the training set to learn, the validation is performed by comparing evaluated response values from the learned function against the response values from the validation test. Comparing the error derived from training and the error derived from validation, we can see the changes in learning and adjust the parameters so that we minimize the error. Another mechanism for improving the learning of a heuristic function is **shuffling** the training set before the start of every epoch. This is important because the weights are not updated with values in the same order in every epoch but rather mixed order so that the updates are not applied by a pattern.

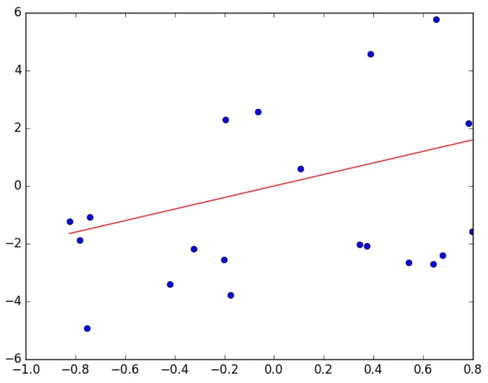


Figure 3.2: The model of linear functions is too weak to learn a heuristic function from the given data set represented in blue dots.

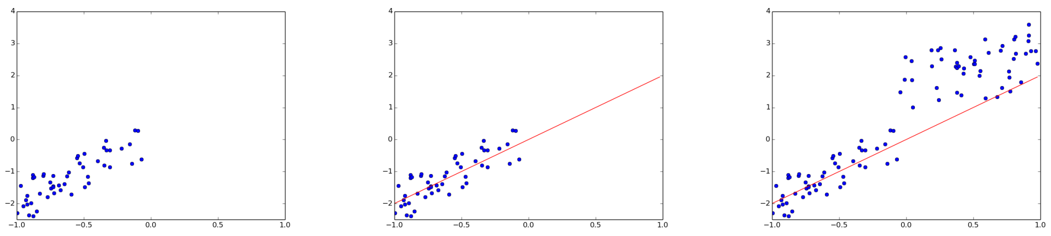


Figure 3.3: This figure showcases three stages of learning: a. Data set, b. Learning from a given data set and c. Validating the learned function. We can see that the initial data set did not provide enough information for the unseen data. The model, even though it is weak for the data set given, was subject to overfitting and produced different error function values for the training set and validation set.

**Example 3.3.1.** *Figures 3.3 and 3.2 showcase the overfitting and underfitting. Denoted in red color is the approximation function which is learned from the data set. We can see from Figure 3.2 that the function cannot be properly learned due to the fact that the model of linear function is too weak for learning from the provided data set. An example for overfitting is showcased in Figure 3.3 where we see that on the seen data (Figure 3.3 b), the function approximates the response values closely while on the unseen data (Figure 3.3 c), it performs far worse.*

### 3.3.3 Feature Combination

**Algorithm 7** Optimized SGD algorithm with combined features

```
function CFOSGD($weights$, $dataSet$, $alpha$)
    Declare $\Delta_f \leftarrow 0$
    Declare $change \leftarrow 0$

    for $s, r$ in $dataSet$ do
        $\Delta_f \leftarrow$ Error($weights, r$)
        $change \leftarrow change + \Delta_f$
        for $f \in \mathcal{F}^2(s)$ do
            $weight[f] \leftarrow weight[f] + alpha \cdot \Delta_f + \gamma \cdot \Delta_{prev}[f]$
            $\Delta_{prev}[f] \leftarrow \Delta_f$
        end
    end

    if $change < \epsilon$ then
        return False
    end
    return True
```

Combining features allows for more accurate function approximation. Using features of a given state $s \in \mathcal{S}$ we generate new features that can be used for learning a heuristic function. The number of newly generated features depends on the number of features we want to combine. Using new features, increases the complexity of the resulting learned function. In this thesis, we combined two features. New features are in the the set of facts $\mathcal{F}^2 = \{\langle v_1, d_1, v_2, d_2 \rangle \mid v_1, v_2 \in \mathcal{V} \wedge d_1, d_2 \in \mathcal{D}_v\}$. Hence, for a given state $s$, we have a set of facts for state $s$:

$$\mathcal{F}^2(s) = \{\langle v_1, d_1, v_2, d_2 \rangle \mid v_1, v_2 \in \mathcal{V} \wedge d_1, d_2 \in \mathcal{D}_v \wedge s[v_1] = d_1 \wedge s[v_2] = d2\}$$

Using the new features, we applied SGD, BGD as well as MBGD for learning. Algorithm 7 showcases the changes in the SGD. Again, every feature $f \in \mathcal{F}^2$ has a weight $w_f \in \mathcal{W}$ assigned to it.

# 4 Experimental Results

This chapter is divided into four sections. In the first section we present the experimental environment, in which the learned heuristic functions were evaluated. The second section covers the details of different parameters used for learning an offline heuristic function. The performance of UCT*, using the heuristic functions learned using single features is evaluated in the third section, while fourth section evaluates the learned heuristic functions using combined features in the same setup.

The goal of this thesis was to find a parameter configuration for learning an offline heuristic function, which would show significant results in all the tests. From the experiments we conducted, such a parameter configuration was not found. However, we derived a set of offline learned heuristic functions using different parameter configurations which when combined with UCT* showcased significant results. Furthermore, while examining the influence of some parameter values on the total score of the experimental evaluation, we showcase the best possible results for the values of the parameter in question in the following setup: the value of the parameter in question is fixed while the set of best parameter combinations of other parameters is taken into consideration for the task where such configuration performs best. Tables that contain such results have a tag *SBPC* (abbreviation for *Set of Best Parameter Configurations*) next to the name of the offline heuristic function, which is being evaluated.

## 4.1 Experimental Environment

Mechanism for learning an offline heuristic is integrated in the state-of-the-art domain-independent probabilistic planner Prost[7] which implements THTS framework described in Section 2.2. Learned heuristic functions were provided to the UCT* algorithm as the heuristic function for the initialization of new nodes in the search tree.

Evaluation was done on 12 different domains including 10 instances per domain from the International Probabilistic Planning Competition 2014[1]. The results are averaged accumulated rewards of 100 runs, scaled between 0 and 1. The artificial minimum derived

[1] https://cs.uwaterloo.ca/ mgrzes/IPPC_2014/

| | | | | |
|---|---|---|---|---|
| **Data set size** | 200 | 2000 | | |
| **Methods** | SGD | BGD | MBGD | |
| **Step size** | 0.0001 | 0.0005 | 0.00001 | 0.00005 |
| **Epochs** | 1000 | 10000 | | |
| **Learning rate decay** | 0 | 0.05 | | |
| **Momentum** | 0 | 0.1 | 0.3 | |
| **Heuristic function** | IDS | THTS | | |
| **Features combined** | 1 | 2 | | |

Table 4.1: All the parameters used used in the experimental evaluation.

from IPPC 2014 has 0 assigned as a relative score while 1 is assigned to the algorithm that achieves the highest durning search reward. We compare the results of using a learned heuristic to the scores of the best algorithms from IPPC 2011[2] and IPPC 2014 as well as to the DP-UCT[6] using the Uniform heuristic function, in the remainder of this thesis denoted as the *baseline algorithms*.

Experiments were run on Intel Xeon E5-2660 CPUs running at 2.2 GHz. Each algorithm was allowed 1 second to perform trials before choosing the best action.

## 4.2 Parameter Configurations

In this thesis we use 8 different parameters for learning an offline heuristic function. Parameters and parameter values we used are listed in Table 4.1. The early stopping parameter is fixed to $\epsilon = 0.00001$ for all configurations. The parameter values showcased represent a subset of the parameter values used during the experimental evaluation which achieved significant results compared to baseline algorithms.

### 4.2.1 Data Sets

A domain-instance pair represents a *planning task*, or shortly just a **task**. With 12 domains and 10 instances per domain there are 120 tasks in total. For each task, a **data set** was generated by sampling random states, providing them to the heuristic we want to learn and storing the resulting action-value estimates for a given state. In order to test the impact of the data set's size on learning, we set two different upper limits to the number of states sampled. Not all tasks can provide the desired number of sampled states. For example, Navigation 1 tasks feature only 13 states while Elevators 10 features 6397 states.

We experimented with two upper limits for the number of states sampled for generating a data set: 200 and 2000. Experiments were run using parameter configuration including all three gradient descent types with momentum $\gamma = 0$ and learning decay rate $\sigma = 0.05$ for all combinations of step size and number of epochs mentioned in Table 4.1. Learning using a data set generated of 2000 sampled states was superior to learning using a data set generated by sampling 200 states in all but the following 15 planning tasks: Academic (1, 5, 7), Elevators (1), Tamarisk (1, 5, 7), Sysadmin (8, 9), Game (3), Skill (1, 4, 9) and

[2] http://users.cecs.anu.edu.au/ ssanner/IPPC_2011/

| DP-UCT | IPPC2011 | IPPC2014 | SGD | BGD | MBGD |
|---|---|---|---|---|---|
| 0.0 | 0.0 | 0.75 | **1.0** | 0.63 | 0.81 |

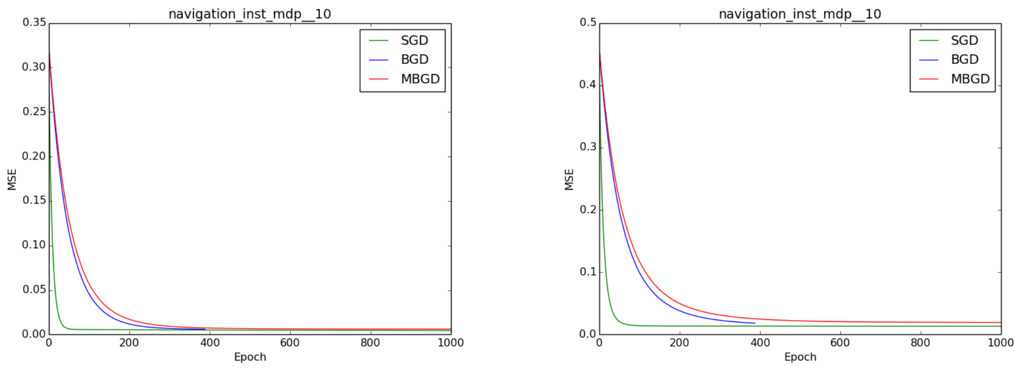
Table 4.2: Results in task Navigation 10 using: $\alpha = 0.0001$, $\sigma = 0$, $\gamma = 0$, 1000 epochs and learning from the IDS heuristic function.

Navigation (9, 10) out of 120 planning tasks in total. Considering these results, for the remainder of the experiments, all data sets were generated using the upper limit of 2000 sampled states. More details on particular tasks where using smaller data set for learning was more successful can be found in the Appendix A.1.

### 4.2.2 Gradient Descent Methods

Batch size in MBGD method was set to 10% of the data set, if the data set was large enough, else MBGD performed the same as BGD.

Even though MBGD is found to be the best type of gradient descent, in our experimental evaluation we found that different methods show better results for different domains. In average, SGD is the method that achieved the best results. However, there are tasks where other methods perform better.



a. Training b. Testing

Figure 4.1: MSE graph for learning a heuristic function from IDS in task Navigation 10 with parameters: $\alpha = 0.0001$, $\sigma = 0$, $\gamma = 0$ and number of epochs 1000.

Figure 4.1 showcases the curve of the error function during the learning of the IDS heuristic. Step size is set to $\alpha = 0.0001$ and the number of epochs to 1000 with learning decay rate of $\sigma = 0$ and momentum of $\gamma = 0$. This is an example where BGD applies early stopping after 388 epochs while MBGD continues to learn with a slower pace but not falling under threshold $\epsilon$. Since SGD converged the fastest, it evaluated the lowest error function and hence performed best during the search. The scores achieved are presented in Table 4.11. In the Appendix A.2, examples where other gradient descent types performed superior can be found with short discussion.

| Step size / Epochs | 0.0005 | 0.0001 | 0.00005 | 0.00001 |
|---|---|---|---|---|
| 1000 | 53 | 53 | 53 | 44 |
| 10000 | 60 | 59 | 54 | 53 |

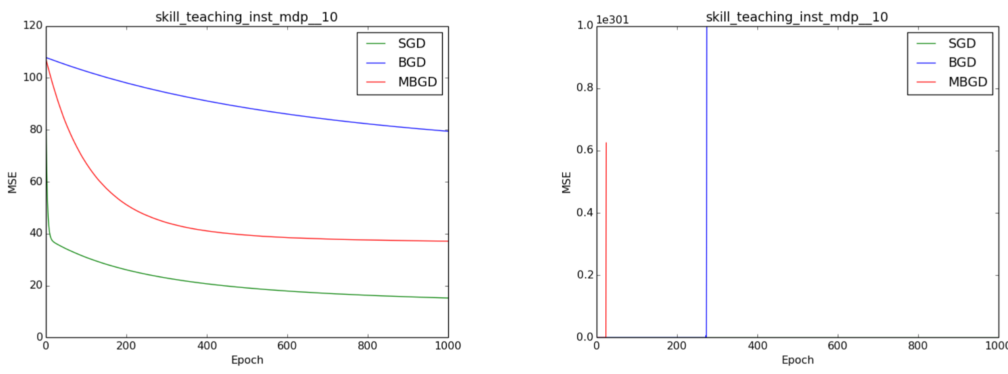
Table 4.3: Number of results for epoch-step size combination over 120 tasks where an offline learned heuristic function scored as well as or higher total score values than baseline algorithms.

### 4.2.3 Step Size and Number of Epochs

During the experimental evaluation, step sizes used ranged from 0.000001 to 0.1. Step sizes showcased in Table 4.1 are the ones that showcased the best results. Smaller step size leads to slower convergence and demands a larger number of epochs while larger step size tends to overshot and even lead to divergence. The number of epochs which was used during the experiments varied from 100 to 10000. During the final stages of conducting experiments, we fixed two numbers of epochs which showcased the best results. Table 4.3 shows the success of different step sizes in combination with different number of epochs where numbers in cells show the number of domains where given combination scored equal or higher scores to the baseline algorithms of all 120 planning tasks. The rest of the parameters belong to the *set of best parameter configurations* as noted in the beginning of this chapter.

Since different combinations of the number of epochs and step sizes work better for different domains, further experiments are carried out with all combinations.

### 4.2.4 Momentum and Learn Decay Rate



a. $\gamma = 0$ b. $\gamma = 0.3$

Figure 4.2: MSE graph for learning a heuristic function from IDS in task Skill 10 with parameters: $\alpha = 0.00001$, $\sigma = 0.05$ and number of epochs 1000. Values on the y-axis showcase the difference in the learning success. For the value of $\gamma = 0.3$ we can notice the problem of overshooting.

Momentum takes values from range $\gamma \in [0, 1]$. The recommended value for momentum is $\gamma = 0.9$ [16]. Using such a high value for momentum resulted in overshooting in our experiments. Therefore, we used smaller momentum values from the set $\{0, 0.1, 0, 3\}$ which

| | DP-UCT | IPPC 2011 | IPPC 2014 | SBPC Offline Heuristic from IDS |
|---|---|---|---|---|
| Wildfire | 0.79 | 0.83 | 0.69 | **0.85** |
| Triangle | 0.42 | 0.39 | **0.88** | 0.67 |
| Academic | **0.69** | 0.3 | 0.31 | 0.47 |
| Elevators | 0.6 | **0.97** | 0.96 | 0.83 |
| Tamarisk | 0.64 | **0.9** | 0.88 | 0.88 |
| Sysadmin | 0.61 | 0.75 | 0.81 | **0.99** |
| Recon | 0.59 | **0.99** | 0.95 | 0.95 |
| Game | 0.71 | 0.91 | **0.97** | 0.96 |
| Traffic | 0.86 | 0.93 | **0.98** | 0.69 |
| Crossing | 0.42 | 0.81 | **0.99** | 0.83 |
| Skill | 0.94 | 0.95 | 0.96 | **1.0** |
| Navigation | 0.67 | 0.58 | 0.93 | **0.99** |
| Total | 0.66 | 0.78 | **0.86** | 0.84 |

Table 4.4: The score of learning the offline heuristic function from IDS while choosing the other parameters from the *Set of Best Parameter Configurations* of parameters.

produced significant results. Not using the momentum, or rather setting the value $\gamma = 0$, proved to perform the best. Furthermore, with exception of the SGD, none of the methods produced significant results. Learning decay rate is usually set to $\sigma < 0.1$. In this thesis we used two values for learning decay rate $\sigma \in \{0, 0.05\}$. Step size $\alpha$ was decreased by a fraction $\sigma$ of its value every 50 epochs.

### 4.2.5 Heuristic Functions

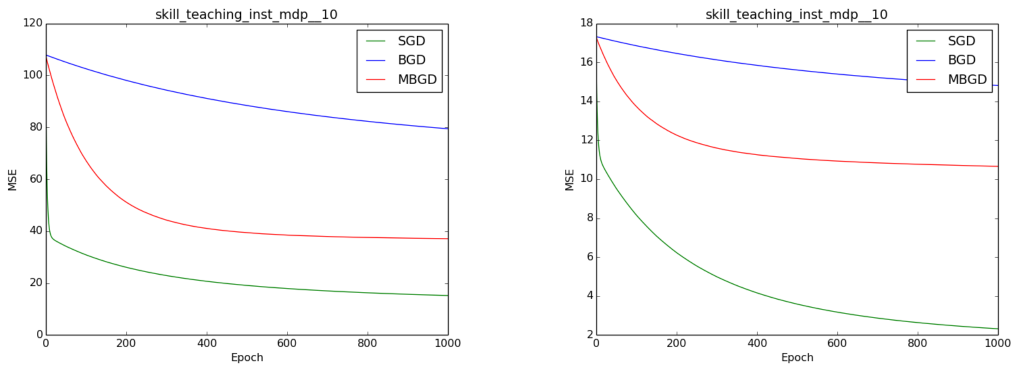
In this thesis we learned two heuristic functions: IDS and THTS. Learning of the heuristic functions was mostly possible in all tasks using at least one of the gradient descent types.

**IDS** is the heuristic function presented in Section 3.1. The UCT* algorithm using IDS as the heuristic function came to be the winning algorithm of IPPC 2014. The base idea for this thesis was to improve the winning algorithm by providing a better or resource-wise cheaper heuristic function. Table 4.4 showcases the scores of the best parameter configurations used for learning the IDS heuristic functions.

**THTS**, in the setup we were using to learn a heuristic function, is much more complex than IDS and hence is not feasible to be used as a heuristic function during search. It was the initial expectation that results of an offline heuristic function learned from THTS would achieve better results in the experimental evaluation. For most of the tasks, this was not the case. However, learning of an offline heuristic function from THTS on tasks such as Skill 10, was more successful than learning the IDS heuristic function. Figure 4.3 showcases the comparison of MSE in training an offline heuristic function for Skill 10 task between learning THTS and IDS heuristic functions. The assumption for the reason of this unexpected behavior is that the model of linear functions is too weak for learning the THTS on some tasks.

| | DP-UCT | IPPC 2011 | IPPC 2014 | SBPC Offline Heuristic from THTS |
|---|---|---|---|---|
| Wildfire | 0.85 | **0.86** | 0.73 | 0.36 |
| Triangle | 0.42 | 0.39 | **0.88** | 0.58 |
| Academic | **0.68** | 0.3 | 0.31 | 0.48 |
| Elevators | 0.6 | **0.97** | 0.96 | 0.89 |
| Tamarisk | 0.66 | **0.92** | 0.91 | 0.85 |
| Sysadmin | 0.64 | 0.78 | 0.84 | **0.99** |
| Recon | 0.58 | **0.98** | 0.94 | **0.98** |
| Game | 0.71 | 0.91 | **0.97** | 0.96 |
| Traffic | 0.87 | 0.93 | **0.98** | 0.32 |
| Crossing | 0.42 | 0.81 | **1.0** | 0.74 |
| Skill | 0.94 | 0.94 | 0.95 | **1.0** |
| Navigation | 0.67 | 0.58 | 0.94 | **0.99** |
| Total | 0.67 | 0.78 | **0.87** | 0.76 |

Table 4.5: The score of learning the offline heuristic function from THTS while choosing the other parameters from the *Set of Best Parameter Configurations*.



a. IDS    b. THTS

Figure 4.3: Comparison of the MSE graph for learning a heuristic function: a. IDS and b. THTS in task Skill 10 with parameters: $\alpha = 0.00001$, $\sigma = 0.05$, $\gamma = 0$ and number of epochs 1000. From the values of y-axis, it can be noted that learning the THTS was more successful than learning the IDS heuristic function for the given parameter configuration. In this particular task, the algorithm that learned from IDS scored 0.7 while the one learning from THTS scored 0.93. The highest score achieved by the baseline algorithms was 0.85.

Table 4.5 showcases the combined scores from multiple parameter configurations and compares them against the baseline algorithms.

## 4.3 Evaluation of Learning using Single Features

There are several parameter configurations that score the same total score. In Table 4.6 we showcase averaged scores for one of them: SGD with step size $\alpha = 0.0005$ and number of epochs 1000. Learning decay rate and momentum are fixed to $\sigma = 0$ and $\gamma = 0$ for all parameter configurations used for learning the offline heuristic functions used in the search. Parameter configurations that achieve the same score are showcased in Table 4.7.

| | DP-UCT | IPPC 2011 | IPPC 2014 | SGD from IDS |
|---|---|---|---|---|
| Wildfire | 0.79 | **0.83** | 0.69 | 0.72 |
| Triangle | 0.42 | 0.39 | **0.88** | 0.51 |
| Academic | **0.68** | 0.3 | 0.31 | 0.38 |
| Elevators | 0.59 | **0.96** | **0.96** | 0.63 |
| Tamarisk | 0.64 | **0.9** | 0.88 | 0.72 |
| Sysadmin | 0.61 | 0.74 | 0.8 | **0.86** |
| Recon | 0.58 | **0.98** | 0.94 | 0.88 |
| Game | 0.71 | 0.91 | **0.97** | 0.9 |
| Traffic | 0.86 | 0.93 | **0.98** | 0.3 |
| Crossing | 0.42 | 0.81 | **0.99** | 0.7 |
| Skill | 0.93 | 0.93 | **0.94** | 0.92 |
| Navigation | 0.67 | 0.58 | 0.92 | **0.93** |
| Total | 0.66 | 0.77 | **0.86** | 0.71 |

Table 4.6: Total scores for UCT* using an offline heuristic function learned from IDS with SGD using $\alpha = 0.0005$, $\sigma = 0$, $\gamma = 0$ over 1000 epochs.

| **Heuristic learned** | **Method** | **Epochs** | **Step size** |
|---|---|---|---|
| IDS | SGD | 1000 | 0.0005 |
| IDS | SGD | 10000 | 0.0005 |
| IDS | SGD | 1000 | 0.0001 |
| IDS | SGD | 10000 | 0.0001 |
| IDS | SGD | 10000 | 0.00005 |

Table 4.7: Combination of parameters for learning the offline heuristic functions that achieved the highest score. Learning decay rate and momentum are fixed to $\sigma = 0$ and $\gamma = 0$.

| | DP-UCT | IPPC 2011 | IPPC 2014 | SBPC Offline Heuristic 1 |
|---|---|---|---|---|
| Wildfire | 0.79 | 0.83 | 0.69 | **0.87** |
| Triangle | 0.42 | 0.39 | **0.88** | 0.68 |
| Academic | **0.68** | 0.3 | 0.31 | 0.57 |
| Elevators | 0.59 | **0.96** | **0.96** | 0.9 |
| Tamarisk | 0.64 | **0.9** | 0.88 | **0.9** |
| Sysadmin | 0.61 | 0.74 | 0.8 | **0.99** |
| Recon | 0.58 | **0.98** | 0.94 | **0.98** |
| Game | 0.71 | 0.91 | **0.97** | 0.96 |
| Traffic | 0.86 | 0.93 | **0.98** | 0.69 |
| Crossing | 0.42 | 0.81 | **0.99** | 0.83 |
| Skill | 0.93 | 0.93 | 0.94 | **1.0** |
| Navigation | 0.67 | 0.58 | 0.92 | **0.99** |
| Total | 0.66 | 0.77 | **0.86** | **0.86** |

Table 4.8: Scores for UCT* using an offline heuristic function learned using single features with parameters from the *Set of Best Parameter Configurations.* These results are achieved by using a set of heuristic functions(learned with different parameters) where the offline heuristic function was assigned to the task on which it performs the best.

| | DP-UCT | IPPC 2011 | IPPC 2014 | Offline Heuristic 2 |
|---|---|---|---|---|
| Wildfire | 0.85 | **0.86** | 0.73 | 0.1 |
| Triangle | 0.42 | 0.39 | **0.88** | 0.53 |
| Academic | **0.69** | 0.3 | 0.31 | 0.17 |
| Elevators | 0.6 | **0.97** | **0.97** | 0.74 |
| Tamarisk | 0.66 | **0.92** | 0.91 | 0.32 |
| Sysadmin | 0.75 | 0.91 | **0.99** | 0.38 |
| Recon | 0.6 | **1.0** | 0.96 | 0.37 |
| Game | 0.72 | 0.92 | **0.98** | 0.64 |
| Traffic | 0.87 | 0.93 | **0.98** | 0.21 |
| Crossing | 0.42 | 0.81 | **1.0** | 0.29 |
| Skill | 0.96 | 0.96 | **0.97** | 0.77 |
| Navigation | 0.67 | 0.58 | 0.94 | **0.95** |
| Total | 0.68 | 0.8 | **0.88** | 0.46 |

Table 4.9: Scores of the offline heuristic function learned from IDS using BGD with combined features, $\sigma = 0.05$, momentum $\gamma = 0$ and the number of epochs is 1000.

If for each of the tasks, UCT* was assigned the offline heuristic function which performs best for that particular task, the overall score meets the score of the winning algorithm of IPPC 2014. Table 4.8 showcases the averaged scores for all domains.

## 4.4 Evaluation of Learning using Combined Features

By combining existing features we create new features. In this thesis we combined every two features of a state in order to create a new feature. If a state in a planning task is described with $n$ features, we have total of $\binom{n}{2} - \frac{n}{2}$ new features and the same number of weights assigned to them. The idea of using new features is to more accurately apply linear value function approximation since every new feature consist of two features whose impact on the result of the weighted sum depends on the presence of both of them. For feature combination, we conducted the same experiments using the same parameter configurations as for the learning using only 1 feature per weight.

The results of using combined features for learning an offline heuristic function were not as good as we initially expected. The offline heuristic functions learned using combined features were outperformed by both baseline algorithms and heuristic functions learned using single features in all but two planning tasks: Elevators 1 and Navigation 9. Two parameter configurations achieved a total score of 0.46. Learning decay rate is set to $\sigma = 0.05$, momentum $\gamma = 0$ and the number of epochs is 1000. BGD used the step size of $\alpha = 0.0005$ while SGD used the step size $\alpha = 0.00005$. Table 4.9 showcases the averaged scores for parameter configuration using BGD method.

Table 4.10 showcases the averaged scores of multiple parameter configurations that achieved the best results over all planning tasks. Learning using combined features outperforms baseline algorithms only in the Navigation domain.

We failed to find the reason for the unexpected results in the evaluation of the offline heuristic functions learned using combined features in our experiments. During the analysis, we came to the conclusion that the algorithm using an offline heuristic function learned using

| | DP-UCT | IPPC 2011 | IPPC 2014 | SBPC Offline Heuristic 2 |
|---|---|---|---|---|
| Wildfire | 0.85 | **0.86** | 0.73 | 0.21 |
| Triangle | 0.42 | 0.39 | **0.88** | 0.63 |
| Academic | **0.69** | 0.3 | 0.31 | 0.2 |
| Elevators | 0.6 | **0.97** | **0.97** | 0.80 |
| Tamarisk | 0.66 | **0.92** | 0.91 | 0.37 |
| Sysadmin | 0.75 | 0.91 | **0.99** | 0.44 |
| Recon | 0.6 | **1.0** | 0.96 | 0.4 |
| Game | 0.72 | 0.92 | **0.98** | 0.7 |
| Traffic | 0.87 | 0.93 | **0.98** | 0.27 |
| Crossing | 0.42 | 0.81 | **1.0** | 0.31 |
| Skill | 0.96 | 0.96 | **0.97** | 0.83 |
| Navigation | 0.67 | 0.58 | 0.94 | **1.0** |
| Total | 0.68 | 0.8 | **0.88** | 0.51 |

Table 4.10: Scores of the offline heuristic functions learned using combined features and parameters from the *Set of Best Parameter Configurations.*

| | $\sigma$ | $\gamma$ | Heuristic function | Method | $\alpha$ | Epochs | Number of features |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | THTS | BGD | 0.00005 | 1000 | 1 |
| 2 | 0 | 0 | THTS | BGD | 0.00005 | 1000 | 1 |
| 3 | 0 | 0 | THTS | BGD | 0.00005 | 1000 | 1 |
| 4 | 0.05 | 0 | IDS | MBGD | 0.00005 | 10000 | 2 |
| 5 | 0 | 0 | THTS | SGD | 0.0001 | 10000 | 1 |
| 6 | 0 | 0 | IDS | SGD | 0.00001 | 10000 | 1 |
| 7 | 0 | 0 | IDS | BGD | 0.00005 | 10000 | 1 |
| 8 | 0.05 | 0 | IDS | BGD | 0.00001 | 10000 | 1 |
| 9 | 0 | 0 | IDS | BGD | 0.00005 | 1000 | 1 |
| 10 | 0 | 0 | IDS | SGD | 0.0001 | 1000 | 1 |

Table 4.11: Set of the best parameter configuration per instance for learning an offline heuristic function which in combination with UCT* outperforms the baseline algorithms in the Navigation domain.

combined features initially exploits a lower number of states than the the algorithm using an offline heuristic function with the same parameters but using single features to learn (task Navigation 10). Furthermore, in the Traffic 1 task we noted that the number of trials performed by the algorithm using an offline heuristic function learned using combined features is about 20% lower than the number of trials performed by the algorithm using an offline heuristic function learned using single features but the same parameters. The reason for the later anomaly might be the implementation of learning using combined features. However, we did not investigate this issue fully.

## 4.5 Final Result

From the experimental results presented we conclude that, among the parameter configurations used in this thesis, there is **no single parameter configuration** that outperforms baseline algorithms for all tasks. However, there exists a set of offline heuristic functions which, when provided to UCT*, outperform the baseline algorithms for particular tasks. Set

| | DP-UCT | IPPC 2011 | IPPC 2014 | SBPC Offline Heuristic |
|---|---|---|---|---|
| Wildfire | 0.79 | 0.83 | 0.69 | **0.86** |
| Triangle | 0.42 | 0.39 | **0.88** | 0.7 |
| Academic | **0.68** | 0.3 | 0.31 | 0.6 |
| Elevators | 0.59 | **0.96** | **0.96** | 0.9 |
| Tamarisk | 0.64 | **0.9** | 0.88 | **0.9** |
| Sysadmin | 0.61 | 0.74 | 0.8 | **0.99** |
| Recon | 0.58 | **0.98** | 0.94 | **0.98** |
| Game | 0.71 | 0.91 | **0.97** | 0.96 |
| Traffic | 0.86 | 0.93 | **0.98** | 0.7 |
| Crossing | 0.42 | 0.81 | **0.99** | 0.83 |
| Skill | 0.93 | 0.93 | 0.94 | **1.0** |
| Navigation | 0.67 | 0.58 | 0.92 | **1.0** |
| Total | 0.66 | 0.77 | 0.86 | **0.87** |

Table 4.12: The best scores for learned heuristic function over all possible configurations.

of parameter configurations which are used for learning the set of offline heuristic functions is called *Set of Best Parameter Configurations (SBPC)*. Table 4.12 showcases the averaged score of UCT* using the offline heuristic function learned with parameters from SBPC. Set of parameter configurations used for the Navigation domain is showcased in Table 4.11. Sets of parameter configurations for rest 11 domains are presented in the Appendix A.3.

# 5 Conclusion and Future Work

## 5.1 Conclusion

In this thesis we present a framework for learning an offline heuristic function in domain-independent probabilistic planning. Learning of an offline heuristic function is performed by supervised learning using the gradient descent methods. The framework supports three different gradient descent types: stochastic gradient descent (SGD), batch gradient descent (BGD) and mini-batch gradient descent (MBGD). The framework allows for different configuration for total of 5 parameters including: data set size, step size, number of epochs, learning decay rate and momentum. The data set can be generated using any heuristic function, which for a given state, evaluates the action-value for all actions in a given planning task. Furthermore, the framework supports learning using new features generated by combining the existing state features into pairs.

We showcase that learning an offline heuristic function using the gradient descent methods is possible in the context of domain-independent probabilistic planning. In our experiments, we provided the offline heuristic functions to UCT* and compared the results of the search against the state-of-the-art probabilistic planning algorithms on 120 planning tasks used in IPPC 2014.

Experiments we conducted showcase that it is possible to find a parameter configuration for each planning task so that the UCT* algorithm using an offline learned heuristic function outperforms the state-of-the-art planners. However, the process of finding the best parameter configuration for each planning task requires a lot of experiments and fine tuning of the parameters.

## 5.2 Future Work

Even though through using gradient descent methods for learning proved successful, there are other machine learning techniques which could improve the learning process even more. Among others, possible options are an improved version of the mini batch gradient descent method ADADELTA[20] or artificial neural networks (ANN)[21]. The framework allows for an easy integration of other learning techniques.

Combination of two features for generating a new feature did not prove to be as successful as we expected in our experiments. Exploration of the reason why combination of features underperformed would be one of the important directions in future works. Furthermore, the concept of feature selection[22] was not addressed in this thesis. Preprocessing the data set and determining the significant features in the states set would allow for more informed updates of the weights assigned to the features and therefore more informed learning of an offline heuristic function.

Data set size was only briefly addressed in this thesis. Although, the idea of this work is learning an offline heuristic in domain-independent probabilistic planning, deeper analysis of domains could allow for better understanding of the importance of the size of the data set as well as the sampled states that are used for generating a data set.

In this thesis, only offline learning was applied. Another possible improvement to the performance of the learned heuristic function would be if online learning would be implemented. Online learning would mean updating the learned heuristic function during the search. Furthermore, Gelly and Silver propose the combination of online and offline learning and show the success of such an approach in learning a heuristic function for the traditional game Go[23].

# Bibliography

[1] Campbell, M., Hoane, A. J., and Hsu, F.-h. Deep Blue. *Artificial Intelligence*, 134(1-2):57–83 (2002).

[2] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489 (2016).

[3] Boutilier, C., Dean, T., and Hanks, S. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11(1):94 (1999).

[4] Bellman, R. A Markovian decision process. Technical report, Defense Technical Information Center (1957).

[5] Keller, T. *Anytime optimal MDP planning with trial-based heuristic tree search*. Ph.D. thesis, Dissertation, Albert-Ludwigs-Universität Freiburg, 2015 (2015).

[6] Keller, T. and Helmert, M. Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*. AAAI Press (2013).

[7] Keller, T. and Eyerich, P. PROST: Probabilistic planning based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*. AAAI Press (2012).

[8] Barto, A. G., Bradtke, S. J., and Singh, S. P. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138 (1995).

[9] Pearl, J. *Heuristics: Intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc. (1984).

[10] Kocsis, L. and Szepesvári, C. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*. Springer (2006).

[11] Berry, D. A. and Fristedt, B. *Bandit problems: Sequential allocation of experiments (Monographs on statistics and applied probability)*. Springer (1985).

[12] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43 (2012).

[13] Abramson, B. Expected-outcome: A general model of static evaluation. *IEEE transactions on pattern analysis and machine intelligence*, 12(2):182–193 (1990).

[14] Murphy, K. P. *Machine learning: A probabilistic perspective*. The MIT Press (2012).

[15] Burch, C. A survey of machine learning. Technical report, Pennsylvania Governor's School for the Sciences (2001).

[16] Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[17] Qian, N. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151 (1999).

[18] Sutton, R. S. Two problems with backpropagation and other steepest-descent learning procedures for networks. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society*. Erlbaum (1986).

[19] Robbins, H. and Monro, S. A stochastic approximation method. *The annals of mathematical statistics*, 22(3):400–407 (1951).

[20] Zeiler, M. D. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).

[21] Nielsen, M. A. *Neural networks and deep learning*. Determination Press (2015).

[22] Guyon, I. and Elisseeff, A. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(Mar):1157–1182 (2003).

[23] Gelly, S. and Silver, D. Combining online and offline knowledge in UCT. In *Proceedings of the 24th International Conference on Machine Learning*. ACM (2007).

# A Appendix

## A.1 Data Sets

For a total of 15 out of 120 tasks using a smaller data set for learning an offline heuristic function was more successful than using a larger data set. Those tasks are: Academic (1, 5, 7), Elevators (1), Tamarisk (1, 5, 7), Sysadmin (8, 9), Game (3), Skill (1, 4, 9) and Navigation (9, 10). Apart from the scores achieved on Academic (5, 7), a function learned using the larger data set was not outperformed by a high margin. However, for tasks Academic (5, 7), an offline heuristic function learned using a smaller data set outperformed baseline algorithms and offline heuristic functions using larger data set by $>$0.48 for Academic 2 and $>$0.95. The maximum number of states that can be sampled in Academic 5 and 7 is 4099 and 2934 respectively. Being able to "see" only 5% and 14% of the states respectively, the offline heuristic performed very good. Since this is an isolated case for 2 tasks, we did not pay too much attention on researching for reasons for such behavior. Moreover, one of the future works we mention is gathering more information on the influence of data set size to learning an offline heuristic function.

## A.2 Gradient Descent methods

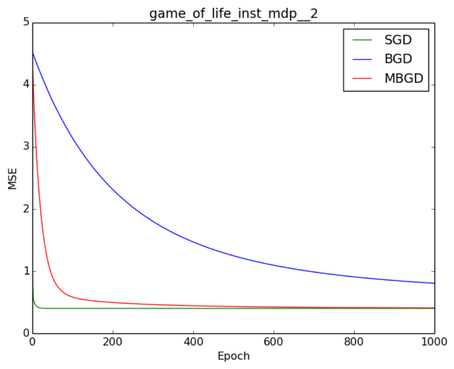
In this section, we cover an example where BGD outperformed the baseline algorithms as well as the other gradient descent methods.

### A.2.1 Batch Gradient Descent

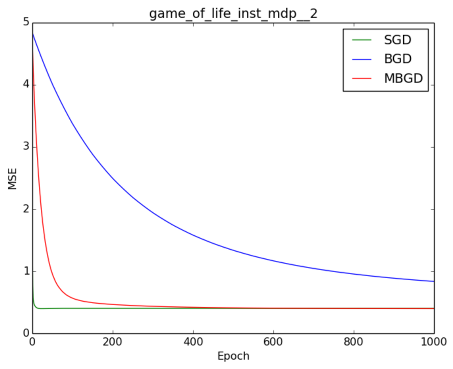
Figure A.1 showcases the graph of MSE function during learning on the planning task Game of Life 2. This example showcases how both SGD and MBGD are performed worse than BGD even though they evaluated lower MSE values. Early stopping was not applied

| DP-UCT | IPPC2011 | IPPC2014 | SGD | BGD | MBGD |
|---|---|---|---|---|---|
| 0.71 | 0.93 | 0.75 | 0.91 | **1.0** | 0.85 |

Table A.1: Results for Game of Life 2



a. Training error



b. Testing error

Figure A.1: Error function graph for learning a heuristic function for task Game of Life 2

| | $\sigma$ | $\gamma$ | Heuristic function | Method | $\alpha$ | Epochs | Number of features |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | IDS | SGD | 0.00005 | 10000 | 1 |
| 2 | 0.05 | 0 | IDS | SGD | 0.00005 | 10000 | 1 |
| 3 | 0.05 | 0 | IDS | SGD | 0.00005 | 1000 | 1 |
| 4 | 0.05 | 0 | IDS | MBGD | 0.0005 | 10000 | 1 |
| 5 | 0.05 | 0.3 | IDS | SGD | 0.0005 | 10000 | 1 |
| 6 | 0.05 | 0 | IDS | SGD | 0.00005 | 10000 | 1 |
| 7 | 0.05 | 0 | IDS | SGD | 0.0001 | 1000 | 1 |
| 8 | 0 | 0 | IDS | SGD | 0.00005 | 10000 | 1 |
| 9 | 0 | 0 | IDS | MBGD | 0.0005 | 1000 | 1 |
| 10 | 0.05 | 0.1 | IDS | SGD | 0.00005 | 1000 | 1 |

Table A.2: Set of best parameter configurations per instance for learning an offline heuristic function which in combination with UCT* outperforms the baseline algorithms in the Wildfire domain

due to the fact that the change in the error function did not fall bellow the threshold of $\epsilon$. From Figure A.1 we cannot derive that SGD and MBGD were subject to overfitting, however, we can conclude based on MSE function values that BGD failed to converge as fast as SGD and MBGD and hence was more general which was better approach for this particular task. The scores BGD achieved against baseline algorithms and SGD and MBGD are showcased in Table A.9. Step size was $\alpha = 0.0005$ over 1000 iterations.

## A.3 Final Result

Together with Table 4.11, Tables A.2 to A.12 showcase the best learning parameter configurations for each of the domain-instance pairs.

| | $\sigma$ | $\gamma$ | Heuristic function | Method | $\alpha$ | Epochs | Number of features |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | IDS | BGD | 0.0001 | 1000 | 1 |
| 2 | 0 | 0 | IDS | BGD | 0.0001 | 1000 | 1 |
| 3 | 0 | 0 | IDS | BGD | 0.0001 | 1000 | 1 |
| 4 | 0 | 0 | IDS | BGD | 0.0001 | 1000 | 1 |
| 5 | 0 | 0 | THTS | BGD | 0.00005 | 1000 | 1 |
| 6 | 0.05 | 0 | IDS | BGD | 0.00001 | 1000 | 1 |
| 7 | 0.05 | 0 | THTS | MBGD | 0.00005 | 1000 | 2 |
| 8 | 0.05 | 0 | THTS | SGD | 0.0001 | 1000 | 2 |
| 9 | 0 | 0 | IDS | BGD | 0.0001 | 1000 | 1 |
| 10 | 0.05 | 0 | IDS | SGD | 0.00005 | 1000 | 1 |

Table A.3: Set of best parameter configurations per instance for learning an offline heuristic function which in combination with UCT* outperforms the baseline algorithms in the Triangle domain

| | $\sigma$ | $\gamma$ | Heuristic function | Method | $\alpha$ | Epochs | Number of features |
|---|---|---|---|---|---|---|---|
| 1 | 0.05 | 0.1 | THTS | SGD | 0.00005 | 1000 | 1 |
| 2 | 0.05 | 0 | IDS | SGD | 0.0001 | 10000 | 1 |
| 3 | 0 | 0 | THTS | SGD | 0.00001 | 10000 | 1 |
| 4 | 0 | 0 | THTS | SGD | 0.0001 | 10000 | 1 |
| 5 | 0.05 | 0 | IDS | MBGD | 0.0001 | 1000 | 1 |
| 6 | 0 | 0 | IDS | SGD | 0.0001 | 10000 | 1 |
| 7 | 0 | 0 | THTS | SGD | 0.0001 | 10000 | 1 |
| 8 | 0.05 | 0 | IDS | BGD | 0.0005 | 1000 | 2 |
| 9 | 0.05 | 0.1 | THTS | SGD | 0.00005 | 1000 | 1 |
| 10 | 0.05 | 0.1 | THTS | SGD | 0.00005 | 1000 | 1 |

Table A.4: Set of best parameter configurations per instance for learning an offline heuristic function which in combination with UCT* outperforms the baseline algorithms in the Academic domain

| | $\sigma$ | $\gamma$ | Heuristic function | Method | $\alpha$ | Epochs | Number of features |
|---|---|---|---|---|---|---|---|
| 1 | 0.05 | 0 | IDS | MBGD | 0.00001 | 10000 | 1 |
| 2 | 0 | 0 | THTS | SGD | 0.00001 | 10000 | 1 |
| 3 | 0 | 0 | IDS | SGD | 0.00005 | 10000 | 1 |
| 4 | 0 | 0 | IDS | MBGD | 0.00001 | 10000 | 1 |
| 5 | 0.05 | 0 | THTS | SGD | 0.00005 | 1000 | 2 |
| 6 | 0.05 | 0 | IDS | SGD | 0.0001 | 1000 | 1 |
| 7 | 0 | 0 | THTS | MBGD | 0.0005 | 1000 | 1 |
| 8 | 0 | 0 | THTS | MBGD | 0.00001 | 10000 | 1 |
| 9 | 0 | 0 | THTS | BGD | 0.00001 | 1000 | 1 |
| 10 | 0 | 0 | THTS | MBGD | 0.0005 | 1000 | 1 |

Table A.5: Set of best parameter configurations per instance for learning an offline heuristic function which in combination with UCT* outperforms the baseline algorithms in the Elevators domain

| | $\sigma$ | $\gamma$ | Heuristic function | Method | $\alpha$ | Epochs | Number of features |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | IDS | BGD | 0.00005 | 10000 | 1 |
| 2 | 0.05 | 0 | IDS | MBGD | 0.00005 | 10000 | 1 |
| 3 | 0 | 0 | IDS | BGD | 0.00005 | 10000 | 1 |
| 4 | 0 | 0 | IDS | MBGD | 0.00001 | 10000 | 1 |
| 5 | 0.05 | 0 | IDS | MBGD | 0.00005 | 10000 | 1 |
| 6 | 0.05 | 0 | IDS | MBGD | 0.0005 | 1000 | 1 |
| 7 | 0.05 | 0 | IDS | MBGD | 0.0001 | 1000 | 1 |
| 8 | 0 | 0 | IDS | BGD | 0.0001 | 10000 | 1 |
| 9 | 0.05 | 0 | IDS | BGD | 0.0005 | 10000 | 1 |
| 10 | 0.05 | 0 | THTS | MBGD | 0.0005 | 10000 | 1 |

Table A.6: Set of best parameter configurations per instance for learning an offline heuristic function which in combination with UCT* outperforms the baseline algorithms in the Tamarisk domain

| | $\sigma$ | $\gamma$ | Heuristic function | Method | $\alpha$ | Epochs | Number of features |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | IDS | SGD | 0.0001 | 10000 | 1 |
| 2 | 0.05 | 0.1 | IDS | SGD | 0.0005 | 10000 | 1 |
| 3 | 0.05 | 0 | THTS | SGD | 0.0001 | 1000 | 1 |
| 4 | 0 | 0 | THTS | MBGD | 0.0005 | 10000 | 1 |
| 5 | 0 | 0 | THTS | MBGD | 0.0005 | 1000 | 1 |
| 6 | 0.05 | 0 | THTS | SGD | 0.00001 | 1000 | 1 |
| 7 | 0.05 | 0 | IDS | MBGD | 0.00001 | 10000 | 1 |
| 8 | 0.05 | 0 | IDS | MBGD | 0.00005 | 1000 | 1 |
| 9 | 0 | 0 | IDS | MBGD | 0.0001 | 1000 | 1 |
| 10 | 0.05 | 0 | IDS | MBGD | 0.00005 | 1000 | 1 |

Table A.7: Set of best parameter configurations per instance for learning an offline heuristic function which in combination with UCT* outperforms the baseline algorithms in the Sysadmin domain

| | $\sigma$ | $\gamma$ | Heuristic function | Method | $\alpha$ | Epochs | Number of features |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | THTS | SGD | 0.00001 | 10000 | 1 |
| 2 | 0.05 | 0 | THTS | SGD | 0.00001 | 10000 | 1 |
| 3 | 0.05 | 0 | THTS | MBGD | 0.0005 | 10000 | 1 |
| 4 | 0 | 0 | IDS | SGD | 0.00005 | 1000 | 1 |
| 5 | 0 | 0 | THTS | MBGD | 0.0005 | 10000 | 1 |
| 6 | 0 | 0 | THTS | SGD | 0.00005 | 1000 | 1 |
| 7 | 0 | 0 | THTS | SGD | 0.00005 | 10000 | 1 |
| 8 | 0 | 0 | THTS | SGD | 0.00001 | 10000 | 1 |
| 9 | 0 | 0 | THTS | SGD | 0.0005 | 10000 | 1 |
| 10 | 0 | 0 | THTS | SGD | 0.00005 | 10000 | 1 |

Table A.8: Set of best parameter configurations per instance for learning an offline heuristic function which in combination with UCT* outperforms the baseline algorithms in the Recon domain

| | $\sigma$ | $\gamma$ | Heuristic function | Method | $\alpha$ | Epochs | Number of features |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | THTS | SGD | 0.00001 | 1000 | 1 |
| 2 | 0.05 | 0 | THTS | BGD | 0.0005 | 1000 | 1 |
| 3 | 0.05 | 0 | IDS | BGD | 0.0005 | 10000 | 1 |
| 4 | 0 | 0 | IDS | MBGD | 0.0005 | 10000 | 1 |
| 5 | 0 | 0 | IDS | SGD | 0.00001 | 10000 | 1 |
| 6 | 0 | 0 | IDS | MBGD | 0.0005 | 1000 | 1 |
| 7 | 0 | 0 | IDS | MBGD | 0.00005 | 1000 | 1 |
| 8 | 0 | 0 | IDS | BGD | 0.0005 | 1000 | 1 |
| 9 | 0 | 0 | IDS | SGD | 0.0001 | 1000 | 1 |
| 10 | 0 | 0 | IDS | BGD | 0.00005 | 10000 | 1 |

Table A.9: Set of best parameter configurations per instance for learning an offline heuristic function which in combination with UCT* outperforms the baseline algorithms in the Game domain

| | $\sigma$ | $\gamma$ | Heuristic function | Method | $\alpha$ | Epochs | Number of features |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | IDS | SGD | 0.0001 | 10000 | 1 |
| 2 | 0 | 0 | THTS | MBGD | 0.0001 | 10000 | 1 |
| 3 | 0 | 0 | IDS | MBGD | 0.00005 | 1000 | 1 |
| 4 | 0.05 | 0 | IDS | BGD | 0.0005 | 10000 | 1 |
| 5 | 0 | 0 | IDS | BGD | 0.0001 | 1000 | 1 |
| 6 | 0.05 | 0 | IDS | MBGD | 0.0005 | 10000 | 1 |
| 7 | 0.05 | 0 | IDS | SGD | 0.00001 | 10000 | 2 |
| 8 | 0.05 | 0 | IDS | SGD | 0.00001 | 10000 | 1 |
| 9 | 0.05 | 0.1 | IDS | SGD | 0.0005 | 10000 | 1 |
| 10 | 0.05 | 0 | IDS | MBGD | 0.00001 | 10000 | 1 |

Table A.10: Set of best parameter configurations per instance for learning an offline heuristic function which in combination with UCT* outperforms the baseline algorithms in the Traffic domain

| | $\sigma$ | $\gamma$ | Heuristic function | Method | $\alpha$ | Epochs | Number of features |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | IDS | SGD | 0.0005 | 10000 | 1 |
| 2 | 0 | 0 | IDS | SGD | 0.0005 | 10000 | 1 |
| 3 | 0 | 0 | IDS | SGD | 0.0005 | 10000 | 1 |
| 4 | 0 | 0 | THTS | SGD | 0.00001 | 10000 | 1 |
| 5 | 0 | 0 | IDS | SGD | 0.0005 | 10000 | 1 |
| 6 | 0 | 0 | IDS | SGD | 0.0005 | 10000 | 1 |
| 7 | 0 | 0 | IDS | SGD | 0.0005 | 10000 | 1 |
| 8 | 0.05 | 0 | THTS | BGD | 0.0005 | 1000 | 1 |
| 9 | 0 | 0 | IDS | SGD | 0.00005 | 10000 | 1 |
| 10 | 0 | 0 | IDS | BGD | 0.00001 | 1000 | 1 |

Table A.11: Set of best parameter configurations per instance for learning an offline heuristic function which in combination with UCT* outperforms the baseline algorithms in the Crossing domain

| | $\sigma$ | $\gamma$ | Heuristic function | Method | $\alpha$ | Epochs | Number of features |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | IDS | BGD | 0.0001 | 1000 | 1 |
| 2 | 0 | 0 | IDS | SGD | 0.00005 | 1000 | 1 |
| 3 | 0 | 0 | IDS | MBGD | 0.00005 | 10000 | 1 |
| 4 | 0 | 0 | IDS | MBGD | 0.00005 | 10000 | 1 |
| 5 | 0.05 | 0 | THTS | SGD | 0.0001 | 10000 | 1 |
| 6 | 0 | 0 | THTS | SGD | 0.0005 | 1000 | 1 |
| 7 | 0 | 0 | THTS | SGD | 0.0001 | 10000 | 1 |
| 8 | 0.05 | 0 | IDS | MBGD | 0.0001 | 1000 | 1 |
| 9 | 0 | 0 | THTS | SGD | 0.0005 | 1000 | 1 |
| 10 | 0 | 0 | THTS | SGD | 0.0005 | 1000 | 1 |

Table A.12: Set of best parameter configurations per instance for learning an offline heuristic function which in combination with UCT* outperforms the baseline algorithms in the Skill domain

# Declaration on Scientific Integrity
# Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

**Author — Autor**
Đorđe Relić

**Matriculation number — Matrikelnummer**
15-050-321

**Title of work — Titel der Arbeit**
Learning Heuristic Functions Through Supervised Learning

**Type of work — Typ der Arbeit**
Master Thesis

**Declaration — Erklärung**
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, June $26^{th}$, 2017

[illegible]

**Signature — Unterschrift**