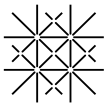


University of Basel

SOGBOFA as heuristic guidance for THTS

Master Thesis

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

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. Thomas Keller

Ferdinand Badenberg
ferdinand.badenberg@unibas.ch
2014-055-206

29.4.2020

Acknowledgments

I want to thank Prof. Dr. Malte Helmert for the opportunity to write this thesis in his research group and Dr. Thomas Keller for his supervision of the project, insight on the topic, and constant feedback. I also thank all my friends and family for being so understanding and for the moral support and the encouragement they always provided. Calculations were performed at sciCORE[1] scientific computing center at University of Basel.

[1] http://scicore.unibas.ch/

Abstract

Most well-known and traditional online planners for probabilistic planning are in some way based on Monte-Carlo Tree Search. SOGBOFA, symbolic online gradient-based optimization for factored action MDPs, offers a new perspective on this: it constructs a function graph encoding the expected reward for a given input state using independence assumptions for states and actions. On this function, they use gradient ascent to perform a symbolic search optimizing the actions for the current state. This unique approach to probabilistic planning has shown very strong results and even more potential. In this thesis, we attempt to integrate the new ideas SOGBOFA presents into the traditionally successful Trial-based Heuristic Tree Search framework. Specifically, we design and evaluate two heuristics based on the aforementioned graph and its Q value estimations, but also the search using gradient ascent. We implement and evaluate these heuristics in the PROST planner, along with a version of the current standalone planner.

Table of Contents

Acknowledgments ii

Abstract iii

1 Introduction 1

2 Background 3
- 2.1 Definitions . . . 3
  - 2.1.1 Markov Decision Process . . . 3
  - 2.1.2 Trial-based Heuristic Tree Search . . . 4
  - 2.1.3 Factored MDP . . . 4
  - 2.1.4 RDDL . . . 5
    - 2.1.4.1 Navigation Example . . . 6
- 2.2 Automatic Differentiation . . . 7
  - 2.2.1 Reverse Pass . . . 8
  - 2.2.2 Forward Pass . . . 10
  - 2.2.3 Reverse Mode versus Forward Mode . . . 10

3 Related Work 11
- 3.1 SOGBOFA . . . 11
  - 3.1.1 Building the Q Function Graph . . . 12
  - 3.1.2 Gradient Steps . . . 15
  - 3.1.3 Action Projection . . . 16
  - 3.1.4 Concrete Action Sampling . . . 16
- 3.2 Conformant SOGBOFA . . . 17

4 Methods 19
- 4.1 SOGBOFA as a Standalone Planner . . . 19
- 4.2 Action Preconditions as Reward Penalties . . . 20
- 4.3 Parameters of SOGBOFA . . . 21
  - 4.3.1 Dynamic Step Size . . . 22
  - 4.3.2 Dynamic Search Depth . . . 22
- 4.4 SOGBOFA as a Heuristic . . . 23
  - 4.4.1 Propagation Heuristic . . . 23

4.4.2 Conformant Heuristic 24

5 Experiments **26**
5.1 Experiment Setup 26
5.1.1 IPC Score 27
5.2 Standalone Planner 27
5.2.1 Search Depth 28
5.2.2 Step Size 29
5.2.3 Threshold 29
5.2.4 General Action Constraints 30
5.3 Conformant Standalone Planner 31
5.3.1 Search Depth 31
5.3.2 Step Size 32
5.3.3 Threshold 33
5.3.4 General Action Constraints 33
5.4 Propagation Heuristic 33
5.4.1 Search Depth Parameter 34
5.5 Conformant Heuristic 37
5.5.1 Parameters 37
5.6 Comparison to State-of-the-Art 44
5.6.1 Comparison of Standalone Planners 45
5.6.2 Comparison of Heuristics 45

6 Conclusion **47**
6.1 Results 47
6.2 Future Work 48

Bibliography **50**

Declaration on Scientific Integrity **52**

1 Introduction

Planning is a part of Artificial Intelligence concerned with achieving a goal from a given initial state. Probabilistic planning includes the additional challenge of non-deterministic actions, where the resulting next state from executing an action is tied to a certain probability. Online planning algorithms attempt to find a policy indicating which action should be taken in each state in order to reach a goal state by alternating between planning and execution steps. Therefore, planning steps are rather short and they have to be able to operate efficiently under tight time constraints. The space of states and actions from which they have to select good ones can be extremely large. A popular approach to tackle this challenge is Trial-based Heuristic Tree Search (THTS) [9], which searches the problem tree using trials with heuristic guidance. It closely based on Monte-Carlo Tree Search [2], where the search is conducted through repeated random samples (or trials) to find an approximation of the result. THTS introduces a common framework for well-known MCTS algorithms, such as UCT [10] or AO* [11].

Recently, a novel approach to MCTS based on concepts generally used in inference problem has shown very strong results: Symbolic Online Gradient-Based Optimisation for Factored Action MDPs (SOGBOFA) [4]. SOGBOFA symbolically approximates the expected reward as a differentiable function of the available actions based independence assumptions of multiple actions. This differentiable function for the expected reward allows it to perform gradient ascent to find a good action. This works very well for many problems, but can struggle when the symbolic representation is too simplistic: For future actions, a random policy is used. This assumption for the next planning steps is often insufficient, which negatively impacts the evaluation of the quality of the current state. This can be remedied with a conformant planning version of the algorithm [5] which optimizes the future actions in addition to the current one.

In this thesis, we attempt to combine the best of both approaches by adapting the SOGBOFA algorithm to be used as a heuristic to guide THTS-based algorithms. The aim is to retain the strong intuition SOGBOFA delivers with its Q value function and optimization thereof, but provides an independent search algorithm to fall back on. We will explore the potential synergies of this combination and attempt to exploit them based on empirical testing. In order to do this, we want to implement the standalone SOGBOFA procedure

and its conformant version as a search engine in the PROST planner [8], which is a probabilistic planning system following the THTS framework. This gives us the opportunity to evaluate the importance of the different components of SOGBOFA and potentially think about possible improvements to the SOGBOFA procedure itself, such as a way to include action preconditions in a more general way. But more importantly, it will give us a good understanding of the components, so that we can design a good heuristic from them. In particular, we will use a heuristic that calculates its value within a single forward pass through the SOGBOFA Q value graph and one where we optimize conformant actions for additional guidance.

In the following Chapter 2, we will present the theoretical background including all the definitions and techniques relevant to the application of the SOGBOFA algorithm in this context. In Chapter 3, we will outline the SOGBOFA algorithm in its existing form, followed by a discussion of the most relevant parts for our context and the changes necessary to accomodate an adaption of the procedure to the heuristic setting in Chapter 4. Finally, we will fine tune the parameters for all configurations and then test and discuss the performance of the conformant planner, the non-conformant planner, and, most importantly, the designed heuristics in Chapter 5.

2
Background

2.1 Definitions

This section provides basic definitions needed for the procedures discussed in the subsequent sections.

2.1.1 Markov Decision Process

A Markov decision process (MDP) is a formalisation of a planning task as a 6-tuple $M = \langle S, s_I, A, T, R, H \rangle$, where

- S is a finite set of states;
- $s_I \in S$ is the initial state;
- A is a finite set of actions;
- $T : S \times A \times S \to [0, 1]$ is the transition function where $T(s, a, s')$ models $p(s' \mid a, s)$;
- $R : S \times A \to \mathbb{R}$ is the reward function;
- $H \in \mathbb{N}$ is a finite horizon.

Given an MDP, a policy is a function $\pi : S \to A$ dictating for each state which action should be chosen. A policy can be evaluated by calculating the Q-value for the policy in the initial state, $Q_\pi(s_I, \pi(s_I), H)$.
The Q-value (or action-value) function $Q_\pi : S \times A \times \mathbb{N} \to \mathbb{R}$ for a given policy π is defined as:

$$Q_\pi(s, a, h) = \begin{cases} 0 & \text{if } h = 0 \\ R(s, a) + \sum_{s' \in S} T(s, a, s') Q_\pi(s, \pi(s), h - 1) & \text{otherwise} \end{cases}$$

This value stands for the expected reward for executing action a in state s under the policy π with h remaining steps. It is calculated by summing the immediate reward and the rewards in the following states weighted by the probability to reach them.

2.1.2 Trial-based Heuristic Tree Search

The Trial-based Heuristic Tree Search framework (THTS) [7, 9] is a more general version of the Monte-Carlo Tree Search framework [2]. The name Trial-based Heuristic Tree Search comes from the unfolding of the MDP used to define the problem into a tree, which is then searched in trials with the guidance of a heuristic.
The general idea of the THTS framework is for one to formulate search algorithms that combine guarantees of optimality when going towards the limit of the full horizon with efficient performance under strict time and memory constraints. Furthermore, the framework also attempts to define these algorithms succinctly and under a common denominator. It distinguishes different approaches to search algorithms based on their so-called ingredients, six components that together define a search algorithm. These ingredients are:

- initialisation and heuristic function
- backup function
- action selection
- outcome selection
- trial length
- recommendation function

A heuristic $h : S \times A \to \mathbb{R}$ attempts to provide guidance to a search algorithm by estimating the Q-value of a state-action pair. Most of the time, heuristics find a reasonable Q-value for a state-action pair by solving a partial or simplified version of the problem. In this regard, they are quite similar to a search algorithm themselves.

2.1.3 Factored MDP

In practice, MDPs are often very large with huge state and action spaces. Hence, they are often represented in the more compact *factored form* [1] An MDP as defined in Section 2.1.1 can be fully described using this factored form:

- A set of binary state variables $\mathcal{V}$ inducing $S = 2^{\mathcal{V}}$
- A valuation $v_I(\mathcal{V})$ representing the initial state $s_I \in S$
- A set of binary action variables $\mathcal{A}$ inducing $A = 2^{\mathcal{A}}$
- A set of transition functions $\mathcal{T}$ with a transition function $t \in \mathcal{T}$, over $\mathcal{V}, \mathcal{A}$, for all state variables $s^* \in \mathcal{V}$ and all action variables $a^* \in \mathcal{A}$. $\mathcal{T}$ induces $T(s, a, s')$ as a product of all transition functions t.
- A reward function $\mathcal{R}$ over $\mathcal{V}, \mathcal{A}, \mathbb{R}$ modelling R.
- $H \in \mathbb{N}$ is the finite horizon.

This is a much more compact representation of an MDP.

2.1.4 RDDL

Relational Dynamic Influence Diagram Language (RDDL) [12] is a relational definition language using parameterized variables and first-order logic to succinctly describe factored MDPs as follows:

- $\mathcal{V}$ is described using a set of parameterized state variables called state fluents.
- $\mathcal{A}$ is described using a set of parameterized action variables called action fluents.
- $\mathcal{T}$ is described using a set of transition functions, called conditional probability functions, over the state and action fluents. They encode the probability of reaching a next state, given previous states and actions.
- $\mathcal{R}$ is described using a reward function over the state and action fluents.

Transition and reward functions can use the following expressions:

- logical expressions
- arithmetic expressions
- equality and inequality comparisons
- conditional expressions
- probability distributions

The encoding of functions used by RDDL can also be described using a numerical representation rather than a logical one. The translation from the logical to the numerical representation is easiliy done following a few rules.

Converting RDDL to Arithmetic Expressions The RDDL description can be converted into arithmetic expressions in the following way:

Logical Formula	Arithmetic Expression
$a \wedge b$	$a \cdot b$
$a \vee b$	$1 - ((1-a)(1-b))$
$\neg a$	$1 - a$
$a = b$	$\sigma(a - b + \epsilon) - \sigma(a - b - \epsilon)$ with $\epsilon = 0.5$
$a \leq b$	$\sigma(\tau(a - b))$ with $\tau = 10$
if C_1 then E_1 elseif C_2 then $E_2 \ldots$ else E_n	$(C_1)E_1 + ((1 - C_1)C_2)E_2 + \cdots + (\ldots)E_n$

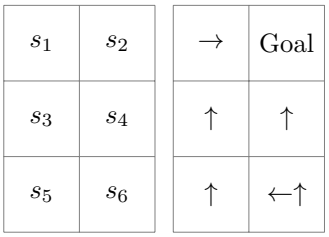
In this case, σ represents the sigmoid function:

$$\sigma = \frac{1}{1 + e^{-x}}$$

Other functions used in RDDL follow accordingly. Hence, we can say that, for all our intents and purposes, RDDL formulates transition functions t and the reward function $\mathcal{R}$ as arithmetic expressions over action fluents $\mathcal{A}$, state fluents $\mathcal{V}$ and $\mathbb{R}$.

2.1.4.1 Navigation Example

An example of such an RDDL description can be built from a simplified toy instance of the navigation domain from the International Planning Competition (IPC) 2011 [13]. It is modified to reduce the complexity of the example by only allowing a choice of actions in the initial state and not allowing a noop action. The reward is also modified to give an immediate reward. The idea is to reach the goal state at the top right from the initial state at the bottom right. The tiles in between have a chance of blowing up, leading to an unsuccessful terminal state. The path through the left is safer (with only a 0.2 chance to explode) but also a detour, while the direct path is more dangerous (with a 0.8 chance to explode). Figure 2.1 shows the states, $s_1, \ldots, s_6$, encoding where the traveller currently is and the applicable actions, moving left, right or up. Here, the actions other than the first choice of paths are fixed. Upon reaching the goal state, a large immediate reward is received, while a penalty is applied as long as the goal state is not yet reached.



(a) States (b) Actions

Figure 2.1: Toy Navigation

For each state of this toy example, RDDL would have a transition function describing how the state variable is updated in each step based on previous states and actions $\mathcal{A} = \{$*go-up, go-left, go-right*$\}$.

- *at-*s_1' = if *at-*$s_3 \wedge$ *go-up* then true else false;
- *at-*s_2' = if *at-*$s_1 \wedge$ *go-right* then true else if *at-*$s_4 \wedge$ *go-up* then true else if *at-*s_2 then true else false;
- *at-*s_3' = if *at-*$s_5 \wedge$ *go-up* then Bernoulli(0.8) else false;
- *at-*s_4' = if *at-*$s_6 \wedge$ *go-up* then Bernoulli(0.2) else false;
- *at-*s_5' = if *at-*$s_6 \wedge$ *go-left* then true else false;
- *at-*s_6' = false;

The initial state can never be reached again, while the goal state can be reached from two different paths and is never left again. The two fields in the middle are only reached with a certain probability.
Furthermore, the reward could be described as:

$$\text{reward} = 10 \cdot (\textit{at-}s_1 \wedge \textit{go-right}) + 10 \cdot (\textit{at-}s_4 \wedge \textit{go-up}) - \sim \textit{at-}s_2$$

This gives a reward of 10 once for reaching the goal state while applying a penalty of 1 as long as the goal state is not reached
However, this description still contains logical and conditional expressions as well as probability distributions and is not yet in the numerical representation we need. Once translated into arithmetic expressions, transitions and reward would look like this:

- $\textit{at-s}'_1 = \textit{at-s}_3 \cdot \textit{go-up}$;
- $\textit{at-s}'_2 = (\textit{at-s}_1 \cdot \textit{go-right}) + (1 - \textit{at-s}_1 \cdot \textit{go-right}) \cdot (\textit{at-s}_4 \cdot \textit{go-up}) + (1 - \textit{at-s}_1 \cdot \textit{go-right}) \cdot (1 - \textit{at-s}_4 \cdot \textit{go-up}) \cdot (\textit{at-s}_2)$;
- $\textit{at-s}'_3 = 0.6 \cdot \textit{at-s}_5 \cdot \textit{go-up}$;
- $\textit{at-s}'_4 = 0.4 \cdot \textit{at-s}_6 \cdot \textit{go-up}$;
- $\textit{at-s}'_5 = \textit{at-s}_6 \cdot \textit{go-left}$;
- $\textit{at-s}'_6 = 0$;
- reward $= 10 \cdot (\textit{at-s}_1 \cdot \textit{go-right}) + 10 \cdot (\textit{at-s}_4 \cdot \textit{go-up}) + \textit{at-s}_2 - 1$

2.2 Automatic Differentiation

This section describes the process of automatic differentiation independent of SOGBOFA. How it is used in the context of SOGBOFA is described in Section 3.1.
Automatic Differentiation is a set of common techniques to find derivatives of functions in computer programs which are particularly suited to gradient calculation. As described by Griewank and Walther [6], automatic differentiation assumes and exploits that the function to be derived is a composition of differentiable elementary operations and functions. Using this input, automatic differentiation essentially describes the repeated use of the chain rule to find derivatives at working precision. The function to be derived can also be represented as a directed, acyclic graph, with nodes for each elementary operation. The most commonly used variants are forward mode and reverse mode, which are simply the two extreme cases of traversing the chain rule, inside out and outside in, respectively (or bottom to top and top to bottom for the graph representation). We will have a closer look at reverse mode, as this is what we will use later on.
In order to understand the process behind reverse mode, a small toy example is presented in the following, which is already very similar to its application for the SOGBOFA procedure. Let us consider the function:

$$z = x_1 x_2 + 2^{x_1} \tag{2.1}$$

We are now interested in finding the partial derivatives with regards to x_1 and x_2:

$$\frac{\partial z}{\partial x_1}$$

and

$$\frac{\partial z}{\partial x_2}$$

Equation (2.1) can be decomposed into several sub-expressions:

$$w_1 = x_1 \tag{2.2}$$

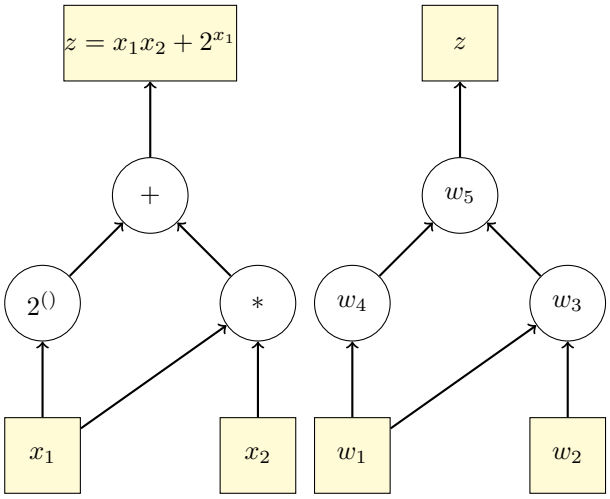
$$w_2 = x_2 \tag{2.3}$$

$$w_3 = w_1 w_2 \tag{2.4}$$

$$w_4 = 2^{w_1} \tag{2.5}$$

$$w_5 = w_3 + w_4 \tag{2.6}$$

where $w_5 = z$. For each of these expressions, the differentiation rules are known. Furthermore, Figure 2.2 shows how this expression z can be represented as a DAG, both as the original encoding of z and using these sub-expressions.



(a) Encoding of original expression (b) Decomposition into sub-expressions

Figure 2.2: Expression Graph

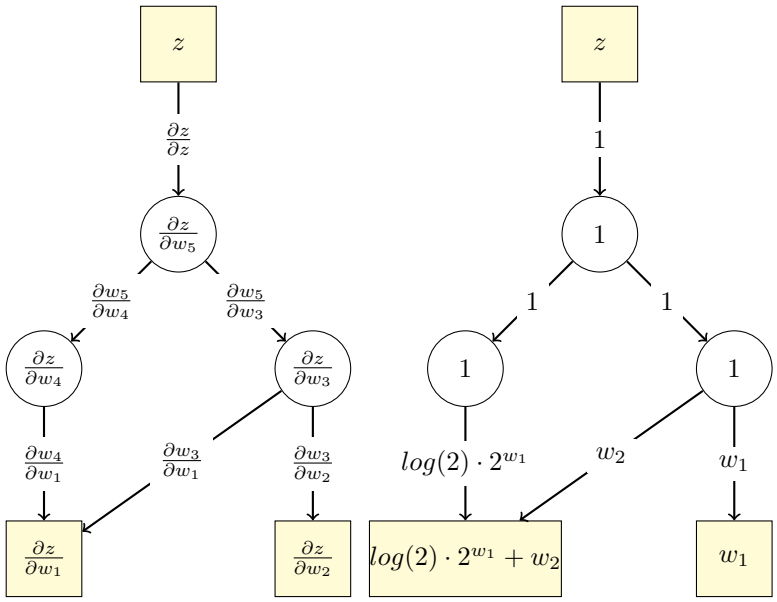
2.2.1 Reverse Pass

The reverse pass is based on the chain rule for derivatives, which leads to the following statement in this graph context:

$$\frac{\partial z}{\partial w_i} = \sum_{p \in Parents(i)} \frac{\partial z}{\partial w_p} \frac{\partial w_p}{\partial w_i} \tag{2.7}$$

where z is our topmost node containing the full expression and w_i are the input nodes $x_1, \ldots, x_n$ with respect to which we want to find the partial derivatives. In order to calculate this at the leaf nodes, we need to sweep through the complete graph from top to bottom,

always applying Equation (2.7) to calculate the partial derivatives in the intermediary nodes. Notably, as Equation (2.1) is a scalar valued function, we only need to go through the graph once. The complete traversal of the graph using chain rule is also demonstrated in Figure 2.3.



(a) Reverse pass graph traversal (b) Results of reverse pass calculation

Figure 2.3: Computational Graph

In our example, starting at the top would yield us:

$$\frac{\partial z}{\partial w_5} = \frac{\partial z}{\partial z}\frac{\partial z}{\partial z} = 1 \cdot 1 = 1$$

as we know that $w_5 = z$ and $\frac{\partial z}{\partial z} = 1$.
The next calculation would be:

$$\frac{\partial z}{\partial w_4} = \frac{\partial z}{\partial w_5}\frac{\partial w_5}{\partial w_4}$$

The first part of which, $\frac{\partial z}{\partial w_5}$, we just calculated. The second part is easy to calculate because we only have to calculate:

$$\frac{\partial w_5}{\partial w_4} \overset{(2.6)}{=} \frac{\partial (w_3 + w_4)}{\partial w_4} = 1$$

Hence, the whole expression yields:

$$\frac{\partial z}{\partial w_4} = \frac{\partial z}{\partial w_5}\frac{\partial w_5}{\partial w_4} = 1 \cdot 1 = 1$$

Continue by repeatedly reusing the previous derivatives and the derivatives of simple expressions:

$$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial w_5}\frac{\partial w_5}{\partial w_3} = 1 \cdot \frac{\partial (w_3 + w_4)}{\partial w_3} = 1 \cdot 1 = 1$$

$$\frac{\partial z}{\partial w_2} = \frac{\partial z}{\partial w_3}\frac{\partial w_3}{\partial w_2} = 1 \cdot \frac{\partial w_1 w_2}{\partial w_2} = 1 \cdot w_1 = w_1$$

$$\begin{aligned}\frac{\partial z}{\partial w_1} &= \frac{\partial z}{\partial w_4}\frac{\partial w_4}{\partial w_1} + \frac{\partial z}{\partial w_3}\frac{\partial w_3}{\partial w_1} \\ &= 1 \cdot \frac{\partial 2^{w_1}}{\partial w_1} + 1 \cdot \frac{\partial w_1 w_2}{\partial w_1} \\ &= 1 \cdot log(2) \cdot 2^{w_1} + 1 \cdot w_2 \\ &= log(2) \cdot 2^{w_1} + w_2\end{aligned}$$

Reverse pass so far is independent of the forward pass and needs to be done only once.

2.2.2 Forward Pass

During the forward pass, we simply evaluate the basic expressions for specific inputs by forward propagation through the graph. Let's say these inputs are $x_1 = 2$ and $x_2 = 3$. Then, we would save the following values during the forward pass:

$$w_1 = x_1 = 2$$

$$w_2 = x_2 = 3$$

$$w_3 = x_1 x_2 = 6$$

$$w_4 = 2^{w_1} = 4$$

$$w_5 = w_3 + w_4 = 10$$

Depending on the results of the reverse pass, we might not need to calculate the full forward pass, even though it is very fast and straight-forward. In our case, for example, we only need the values for w_1 and w_2:

$$\frac{\partial z}{\partial w_2} = w_1 = 2$$

$$\frac{\partial z}{\partial w_1} = log(2) \cdot 2^{w_1} + w_2 = 5.77$$

2.2.3 Reverse Mode versus Forward Mode

The above procedure of the reverse mode automatic differentiation showed us that we can calculate the full gradients of the function in only one traversal of the graph, for which we can then simply plug in our input variable assignments.
In contrast, forward mode directly follows the flow of derivative information in its evaluation by saving the derivative evaluation of each expression directly with the expression, starting at the input variables. As such, it is conceptionally simpler than reverse mode, but only calculates the derivative of one input variable during each traversal of the graph. This leads to slower calculation times if the full gradient is needed and the graph is large.

3
Related Work

3.1 SOGBOFA

Online symbolic gradient-based optimisation for factored action MDPs, short SOGBOFA is a successful, recent stochastic online planning algorithm by Hao Cui and Roni Khardon [4] that symbolically represents an approximation of the Q value function as a function of the action variables in order to perform gradient-based search over actions. This assumes independence of state and action variables to reduce the complexity.

Algorithm 1: SOGBOFA

```
Result: action
optimizedActions = {};
qFunction = buildQFunctionTree(States, Actions);
while time remaining do
    currentActions ← randomRestart();
    while time remaining and not converged do
        gradient ← calculateGradient(currentStates, currentActions, qFunction);
        currentActions ← makeUpdates(gradient);
        currentActions ← projection(currentActions);
        optimizedActions.add(sampleConcreteAction(currentActions));
    end
end
return best(optimizedActions);
```

SOGBOFA (Algorithm 1) uses a rollout algorithm [15] to symbolically build a directed, acyclic graph, which succinctly approximates the Q value function in terms of the action variables. Once such a graph is built, it is used extensively to find a good action. This is done by repeatedly sampling a legal initial action for the state, which is then optimised using gradient ascent.

After each gradient step, a concrete action is generated from the action variables and saved. This is repeated with new initial actions until the time is up and the best concrete action found so far is selected.

From Algorithm 1, we can identify these core functions:

- Building the Q function graph
- Gradient steps
- Action projection
- Concrete action sampling

The details of these different steps are explained in the following.

3.1.1 Building the Q Function Graph

The first step in the algorithm is the construction of the directed, acyclic graph representing the Q value function, i.e. the transformation of the Q value function into a symbolic representation. As we have seen in Section 2.2, this graph is vital to the procedure as it allows for automatic differentiation to be used to perform efficient gradient calculations.
The idea behind the graph follows from the MCTS rollouts. These rollouts sample trajectories for of action variables over future steps to get an estimation of the Q value of each action. With such a strategy, problems can occur if the sampling of trajectories is very costly. Then, this procedure has a high variance due to only having few samples. This was the motivation for a strategy of aggregate simulation, an algebraic process where only one sample of the trajectory is performed, but this one sample gives an estimation of many trajectories [3]. In this one sample, the marginal probabilities of state variables in the next state are calculated based on the marginals of state and action variables in the previous state. If we make the assumption of independence among state and action variables, these marginals can be used as an approximation of the state distribution at the next state. From this, we can calculate (an approximated) evaluation of the reward function for this state. If done over multiple future states, this leads to estimation of the Q value, based on the current state and action variables. Even better, we can represent this estimation of the Q value as a symbolic function of the marginals of action variables at the root level (i.e. the current state). It is imporant to understand that the calculations for the marginals is very much correct, but we disregard the dependence between them. For example, we can calculate exact marginal probabilities of an elevator in the elevators-2011 domain[2] going up and going down, but we disregard that he can not, in fact, do both at the same time.
Both formally and its construction this graph is very much akin to expression trees. It contains the following components:

- State variables (current state)
- Action variables (subject to updates)
- Next state variables (algebraic expression)

[2] In this domain, one or more elevators have to transport people arriving at random timesteps to their target floor. For more information, see [13].

- Action constants (fixed)
- Reward (algebraic expression)
- Q value (sum of rewards)

This is very similar to the RDDL encoding of the MDP as described in Section 2.1.4. Thus, building the graph is a rather straight-forward compilation from the RDDL encoding, using its description of the variables, transition functions and the reward function in their numeric representation. The Q value function graph for the toy navigation example is shown in Figure 3.1 for the direct path with two actions, and in Figure 3.2 for the safer path with four actions.

Graph construction from RDDL: Given a factored MDP $M = \langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R}, H \rangle$, with a horizon H, encoding a planning task, we construct the SOGBOFA Q-function graph as follows:

- Add a node $\langle v^*, h \rangle$ for all $v^* \in \mathcal{V}$ and $h \in 1, \ldots, H$ encoding the state fluents at that step.
- Add a node $\langle a^*, h \rangle$ for all $a^* \in \mathcal{A}$ and $h \in 1, \ldots, H$ encoding the action fluents at that step.
- Add a node $\langle \mathcal{R}, h \rangle$ for all $h \in 1, \ldots, H$ endcoding the reward at that step.
- Add a node $\forall h \in 1, \ldots, H$ encoding the Q value at that step.
- Add a directed, acyclic subgraph for all state fluent nodes with $h \neq 1$, encoding the transition function in $\mathcal{T}$ updating that state fluent based on the previous layer. This subgraph connects this state fluent node with the relevant nodes at step $h - 1$ and contains operator nodes for operators used by the arithmetic expressions in the RDDL transition function it encodes. The same is true for constants. Analogously, a subgraph is introduced for each reward node to encode the reward function, connecting the reward node with the relevant state and action nodes of the same layer.
- Add an edge from each node encoding a Q value to the reward node at that layer and Q value node on the previous layer, connecting the Q values to the rewards as a sum of the current reward and the previous Q value.

For this graph construction all functions are assumed to only contain arithmetic expressions. The standard RDDL transition and reward functions can be converted to arithmetic expressions following the rules described in Section 2.1.4.

Q function graph for the navigation example: This graph is built from bottom up, starting at the leaves with state variables initiated as given by the current state (either 0 or 1) and action variables initiated with valid values derived from a concrete, random action. Constant nodes can be set as necessary. Which state and action variables or constants there are directly follows the RDDL encoding of the problem, as demonstrated in Figure 3.1.

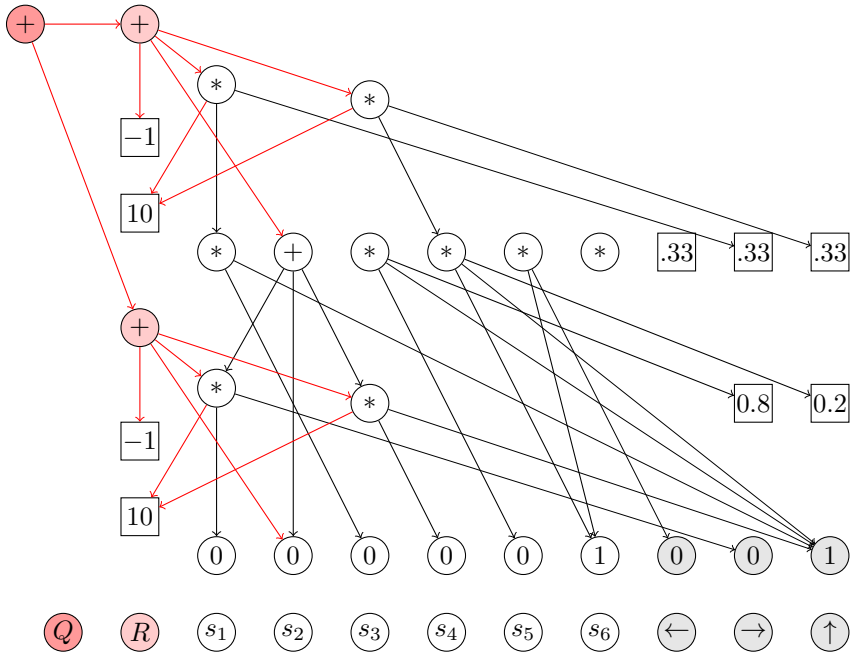


Figure 3.1: DAG approximating the Q value function for the navigation example from Section 2.1.4 over two steps. Reward and its summation to the Q value in red, actions to be optimised in grey.

There are six state variables for the possible positions on the field, three action variables for the movement, and constant nodes for the probability and reward calculation. On the internal layers, the next states are represented as algebraic operation nodes over previous states and actions. They are again derived from the algebraic expression for the next state in the RDDL transition functions. The RDDL transition functions for our example are shown in Section 2.1.4. In addition to these internal state nodes, these layers also introduce additional action constants representing the action choices made in the next planning step. These are assigned a uniformly distributed probability value. For each layer with state and action variables, there is also a node encoding the reward formula, as compiled from the algebraic expression for the reward in RDDL. Finally, the Q value node is simply a sum over all reward nodes.

The height of the DAG represents the search depth. In theory, of course, one could build the graph for the full horizon, i.e. such that the search depth is equal to the horizon depth. As the size of the DAG scales only linearly with search depth, this would be feasible. However, this is generally not ideal due to time constraints on the overall search. As such, there is an inherent tradeoff between the search depth and the number of actions that can be explored. We will discuss how to find a good search depth dynamically in detail in Section 4.3.2. The general idea is to allocate time for a minimum number of updates beforehand and then building the graph with as many layers as possible while ensuring the specified number of updates can still be performed afterwards.

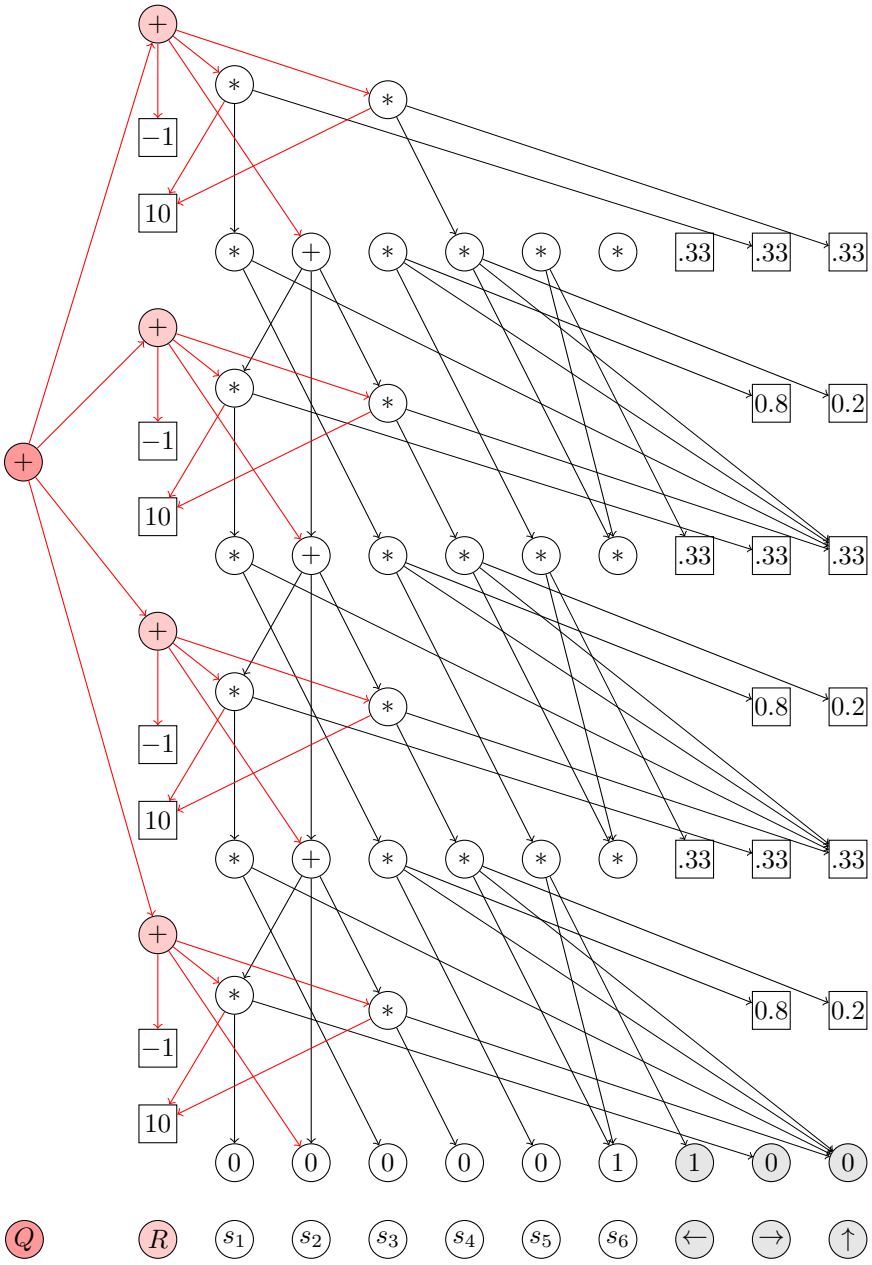


Figure 3.2: DAG approximating the Q value function for the running navigation example over four steps. Reward and its summation to the Q value in red, actions to be optimised in grey.

3.1.2 Gradient Steps

The gradient ascent is repeated for as many randomly chosen initial actions as possible within the time remaining, after which a best action is selected from the actions found during the random restarts of gradient ascent. Gradient ascent for each starting configuration is itself a loop over gradient calculation, action updates, action projection, and concrete action sampling. Gradient ascent is stopped once the largest change in a single action value is under a certain threshold, with the assumption that we have reached a local optimum.
The first part of gradient ascent is to take a step in the direction of the gradients.

Gradient calculation: With the symbolic graph of the Q value function built, SOGBOFA uses reverse mode automatic differentiation (as explained in Section 2.2) to calculate gradients for all action variables. This yields values to use for updates of the action variables in a time efficient manner: only one sweep through the Q function graph is needed in this case due to the scalar-valued Q value function. Of course, we also calculate the Q value itself together with the gradients.

Action updates: Once we know the gradients of the actions, we use them to update the action variables with a certain step size by adding the product of the step size and the respective derivative. While we assume a fixed step size for now, we will discuss possibilities to find the step size dynamically in Section 4.3.1. The idea behind this is that some step sizes might be better suited than others depending on the problem or the particular gradient steps.

3.1.3 Action Projection

While the gradient based updates work well to find the right direction for action updates, there is no guarantee that the updated actions still maintain the $a \in [0, 1]$ bound on their domain due to probability constraints or any constraints induced by what constitutes a legal action. This can propagate through the Q value function graph, leading to very nonsensical and wrong results. It is possible to fix this by projecting the actions back to a legal range after any gradient update step in what is called lazy gradient ascent. [14]: If any actions are outside their $[0, 1]$ range after the gradient updates, they can be simply scaled back by subtracting the smallest value among the actions and then dividing by the difference of the largest and the smallest value among the actions. This keeps the correct relation between the values But this only projects them to be between zero and one again, it does not help to make sure no action constraints are violated in the gradient step.
Therefore, SOGBOFA also uses a second projection: The actions are projected again such that they adhere to action constraints (for example being able to play only one action at a time). They do this by repeatedly subtracting any excess amount in violation of the action constraints from all actions equally, with the lower bound of 0 for all actions. While this is a very helpful step, potentially providing a lot of guidance, such a solution only works for sum constraints in the form of

$$\sum_i a_i \leq B$$

with $a_i \in$ Actions and a bound B. Any constraints on actions in another form would have to be manually added in a similar manner.

3.1.4 Concrete Action Sampling

With the gradient steps and the following projection, gradient ascent over the actions means that a continuous search is applied to a discrete action space. In other words, we generate marginal action probabilities for all factored action variables somewhere between zero and one. Yet, when we have to take an action, a valid action only has values of zero or one in

its factored action variables. Therefore, we need to apply a process of sampling a concrete, discrete action from our continuous assignments. SOGBOFA does this after every gradient step and saves the resulting concrete state together with the estimated Q value, from which the best one is selected in the end. This also means that it is less essential at which exact threshold we stop the gradient ascent as long as we sample the same concrete value afterwards.

With regards to how the action state is sampled, SOGBOFA uses a greedy highest marginal probability heuristic, meaning they repetedly set the highest action values to one for as long as the maximum number of legal actions is not surpassed and its probability is not smaller than a uniform distribution. All other actions are set to zero. The resulting action state is the concrete action state.

This approach is exploiting the way the planning domains are generally encoded, in that most of the time, taking an action is beneficial, so rounding up to a one for an action can often prove useful. But the same problem could theoretically also be encoded the other way around, and then this greedy assumption would fall flat.

3.2 Conformant SOGBOFA

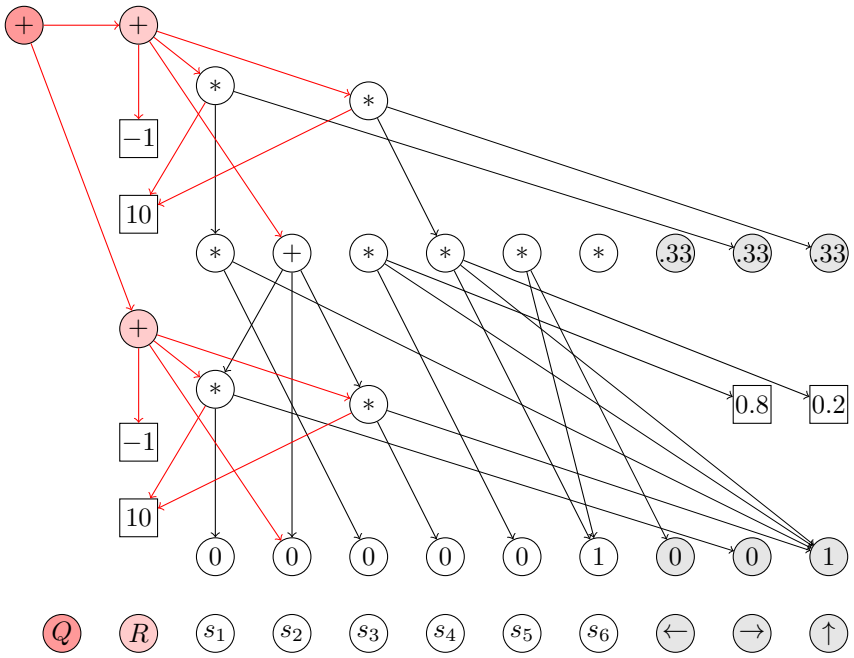


Figure 3.3: DAG of the conformant SOGBOFA approximating the Q value function for the running navigation example over two steps. Reward and its summation to the Q value in red, actions to be optimised in grey. Note that action constants are now action variables.

In addition to the algorithm described in Section 3.1, there is an extension of it called conformant SOGBOFA [5]. It offers an improvement to the algorithm by improving the policy assumed for future actions.

The conformant algorithm for SOGBOFA introduces a change to handling action variables

in layers other than the leaves. Normally, SOGBOFA would simply assign probabilities to all actions in deeper layers as action constants based on a uniform distribution. This simulates an execution of all actions at the same time, but to a small extent, for any future action up to the search depth. While this works well in many cases, there are definitely domains where such a uniform simulation is insufficient to properly evaluate the quality of the current state.

This is where the conformant algorithm offers a large improvement: In conformant SOGBOFA, the aforementioned action constants are replaced with uniformly initialised action variables, which are then optimized together with the action variables at the leaf level. This means that their gradient is calculated and they are updated just like the other action variables, but of course the selection of a concrete action remains to be only based on the action variables in the initial state. How this changes the Q value function graph is shown in Figure 3.3. The implication here is that the optimized conformant actions represent a good choice of future actions, so SOGBOFA can recognize good rewards from a very specific set of choices that is not apparent from a random policy for future actions. It focuses the many estimated trajectories of the current actions towards the ones containing sensible future actions.

Due to the gradient calculation in reverse mode automatic differentiation, the many additional gradients for the conformant actions can still be calculated in one traversal of the graph, thus not really impacting the calculation time.

4
Methods

In this thesis, we are at first looking to test the SOGBOFA search algorithm, understanding the contributions and necessity of the different parts involved, and identify room for improvement.
However, the main contribution of this thesis is the design of heuristics to be used by a THTS framework, such as the Prost planner, based on the ideas from the SOGBOFA. Specifically, we incorporate the guidance from the Q value function graph and the gradient-based optimization techniques performed on it into a THTS heuristic.
In the following, we discuss the techniques, parameters, and concepts used in the heuristics by explaining them in the context of a standalone SOGBOFA planner as a search engine in THTS. Then, we will discuss the key changes in the heuristic setting and how SOGBOFA can be adapted as a heuristic.

4.1 SOGBOFA as a Standalone Planner

A more general overview of the SOGBOFA procedure can be found in Section 3.1. Here, we will focus on details relevant to our implementation and fine tuning of the algorithm with parameters. We will also present an idea for a potential improvement of the algorithm.
As explained in Section 3.1, the SOGBOFA search engine repeats random restarts of gradient ascent steps finding optimal values for its Q-function. Now, we look at the algorithm in more detail, including the parameter settings Cui and Khardon [4] suggest and changes made in our implementation. Algorithm 2 demonstrates the main loop over random restarts and gradient ascent steps, but with some changes to Algorithm 1. These changes reflect the additions discussed in this chapter, providing their context in SOGBOFA.
The first notable change in Algorithm 2 is the calculation of a dynamic search depth for the construction of the Q function. Then, we calculate a dynamic step size for each gradient step. Lastly, we can see that our implementation only samples concrete actions once in the end, and instead always saves the best continuous action after each gradient step.
Another change is hidden in the way we sample concrete actions (not shown in the algorithm). In the Prost planner, we have the concrete action states available. Thus, it is easy to calculate the distance of our desired action state given by the gradient ascent to all

Algorithm 2: SOGBOFA Search Engine

```
Result: Estimates best actions
bestAction;
findDynamicSearchDepth();
qFunction = buildQFunctionTree(States, Actions);
while time remaining do
    currentActions ← randomRestart();
    while time remaining and not converged do
        gradient ← calculateGradient(currentStates, currentActions, qFunction);
        findDynamicStepSize(currentActions, gradient);
        currentActions ← makeUpdates(gradient);
        currentActions ← projection(currentActions);
        if currentActions have better Q value then
            bestAction ← currentActions;
        end
    end
end
return sampleConcreteAction(bestAction);
```

concrete, legal and applicable action states. Consequently, we currently use the action state minimizing this distance as the corresponding concrete action.

This complicates direct comparisons, but should only beneficial to the procedure, as we can guarantee to pick a legal action. It should also dampen the effect of violated action constraints, as working with illegal actions can surely impact the quality of the decisions, but does not lead to a final illegal action.

Algorithm 3 shows how the Q-function to be optimized is built. It closely follows the procedure outlined in Section 3.1, building the Q function level by level by calculating reward and Q value while updating the state and action fluents accordingly.

The only difference from the procedure described in Section 3.1.1 is that we subtracted a penalty from the Q value in the first layer. This penalty adds a large cost to the Q value for all the action preconditions that are violated with the current actions. The idea is to force SOGBOFA to stay within the legal actions with its optimization, as we explain in the following Section 4.2.

There are some slight changes for the conformant procedure, in particular the function *calculateGradient* also takes conformant action fluents as input to be optimized, which replace the uniform actions in the Q function.

4.2 Action Preconditions as Reward Penalties

We have seen in Section 3.1.3 how constraints on actions are handled in the original SOGBOFA procedure. Constraints in the form of $\sum_i a_i \leq B$ are supported. In general, however, constraints can take the form of any arbitrary arithmetic formula. Hence, SOGBOFA can not represent or take into account action constraints in a different form than the ones implemented. Following this implementation, the way to support new action precondition formulas would be to manually add a similar constraint handling for every type of formula encountered. For this reason, we only implemented this for the constraint on the sum of

Algorithm 3: qFunction

```
Input: currentStates, currentActions
Result: qValue
while searchDepth not reached do
    if first layer then
        reward ← rewardFormula.evaluate(currentStates, currentActions);
        penalty ← 0;
        for all actionPreconditions do
            penalty ← penalty - 1000 * (1 -
              actionPreconditionFormula.evaluate(currentState, currentActions));
        end
        qValue ← reward + penalty;
        nextStates ← transitionFormulas.evaluate(currentState, currentAction);
        nextActions ← 1 / currentActions.size();
    end
    else
        nextStates ← currentStates;
        reward ← rewardFormula.evaluate(nextStates, nextActions);
        qValue ← qValue + reward;
    end
end
return qValue;
```

legal actions specifically, and opted for a different method to represent other constraints.
As an alternative, we propose a generalized way to handle action constraints, where any constraint in the form of an arithmetic expression can automatically be taken into account. The idea is to integrate the action precondition formulas, which are in the form of an arithmetic expression, directly into the SOGBOFA Q value function graph. This is done by adding a multiplication node for each constraint formula to the reward node of the actions we want to force to adhere to the constraints. This multiplication node has a large negative penalty as one child and the negation of the constraint formula as the other child. Using sigmoid functions for the equality expressions, this remains differentiable (which is of course critical for the gradient calculation).

4.3 Parameters of SOGBOFA

Now, we have a look at the most important parameters of the algorithm, which can all have significant effects on the overall performance.
The following parameters can be tuned (regardless of conformant or non-conformant procedure):

- **Search depth:** The depth for the Q-function tree, i.e. the number of layers. Ways to calculate this dynamically are discussed below.
- **Step size:** The step size scaling the gradient ascent updates. Ways to calculate it dynamically are discussed below.
- **Threshold:** The number of gradient steps per random restart can be indirectly controlled through the sensitivity of the threshold.

4.3.1 Dynamic Step Size

Cui describes a process of dynamically finding an appropriate step size for every newly calculated gradient. This process is shown in Algorithm 4 and works as follows: We find the maximum value among the gradients. This defines our maximal step size, such that we make sure not to push the action values beyond $[-1, 2]$. From zero to this maximal step size, we perform a search over ten evenly spaced potential step sizes. During this search, we estimate the potential Q value for each step size through an evaluation of the Q value function with action values after a gradient step with that step size (i.e., we simulate a gradient step at that step size). We then pick the step size with the best Q value, but repeat the procedure up to five times if the smallest step size was picked, indicating our search window had too large step sizes. In that case, we set the next maximal step size to the current minimal step size for the next iteration.

Algorithm 4: findDynamicStepSize

Input: currentActions, gradient
Result: step size α
$u_{max} \leftarrow \max(\text{gradients})$;
$\alpha_{max} \leftarrow \frac{1}{u_{max}}$;
alphas $\leftarrow$ calculate 10 evenly spaced $\alpha_0, \ldots, \alpha_9 \in [0, \alpha_{max}]$;
do
  $\alpha \leftarrow$ find α_i from alphas with the best Q value;
while α *is* α_0*, but at most 5 times*;
return α;

While the intuition provided by a search over possible step sizes seems potentially very useful, it seems somewhat costly to calculate a new step size with every gradient update. It might be more efficient to calculate a suitable step size for each problem once at the beginning.

4.3.2 Dynamic Search Depth

Algorithm 5: FindSearchDepth

Result: searchDepth
$t_i \leftarrow$ stop the time for one gradient ascent step at a minimum searchDepth;
$t_i' \leftarrow t_i + \frac{t_i}{\text{searchDepth}}$;
while *minUpdates* $\cdot t_i' <$ *remainingTime* **do**
  searchDepth $\leftarrow$ searchDepth + 1;
  $t_i \leftarrow t_i'$;
  $t_i' \leftarrow t_i + \frac{t_i}{\text{searchDepth}}$;
end

For SOGBOFA, a dynamic calculation of the search depth, i.e. the height of the search tree, is suggested. Cui and Khardon suggest the calculation of one gradient update at a specified minimal search depth to find the average time increase for gradient calculation that comes with adding another layer (as this time increase linearly scales with the search depth). Then,

additional layers are added as long as the remaining time is enough for k gradient updates at that search depth.

4.4 SOGBOFA as a Heuristic

Due to the setting, a heuristic will naturally be different to the standalone planner. Most importantly, the action variables will always be fixed at the input layer (leaf level), because together with the state variables they are given by the planner. Rather than optimizing these actions, we now have to calculate a good estimation of their Q value. But there are also other differences, some implicitly given. For example the heuristic calculation has to be much faster than the standalone planner, as it is expected to be calculated many times in a single step. Based on these constraints, we have designed two heuristic approaches in line with the ideas from the standalone SOGBOFA procedure. Both are presented in the following.

4.4.1 Propagation Heuristic

The first idea was to create a heuristic from the SOGBOFA procedure by simply forward propagating the action state values given by the planner once. The resulting Q value is returned to the planner.

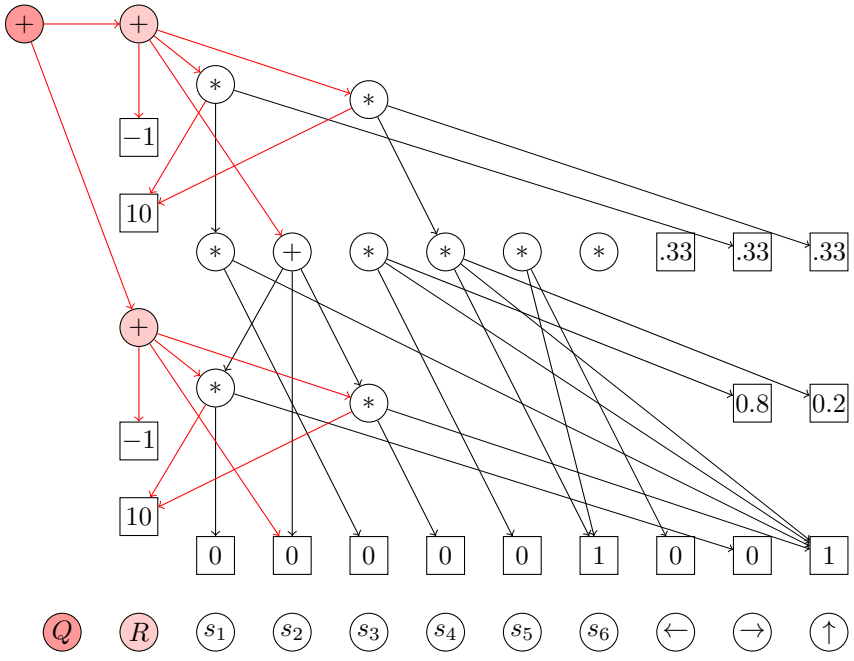


Figure 4.1: DAG approximating the Q value function for the navigation example from Section 2.1.4 over two steps as used for a simple heuristic calculation. Reward and its summation to the Q value in red. Note that both the state variables and action variables are now constants at the leaf level.

The idea behind this heuristic is to create a very minimal integration of the guidance provided by the SOGBOFA Q value function graph into an existing planner, without relying on

the finely tuned process of gradient based optimisation. An example of a Q value function graph for this type of heuristic evaluation can be see in Figure 4.1. The biggest potential advantage of this heuristic is that it would be very fast to calculate. It also has only one parameter to tune: the depth of the Q value function graph. Nonetheless, this parameter allows some control over calculation speed and the information the graph can provide.

4.4.2 Conformant Heuristic

The second heuristic also integrates the gradient-based optimization steps. As the actions at the input layer are fixed, we can not do any optimization on them. We can, however, optimize the other action variables not at the lowest level, in the same way they are optimized in the conformant version of the Sobgofa standalone planner. This should provide a more accurate approximation of the Q value function, as we gain additional information about good future actions. It also means we have more parameters that can influence the exact settings of the heuristic. Next to the search depth, we now also have a maximum number of gradient steps, a threshold, and a step size.
What this conformant heuristic procedure means for the structure of the Q value function graph is shown in Figure 4.2.

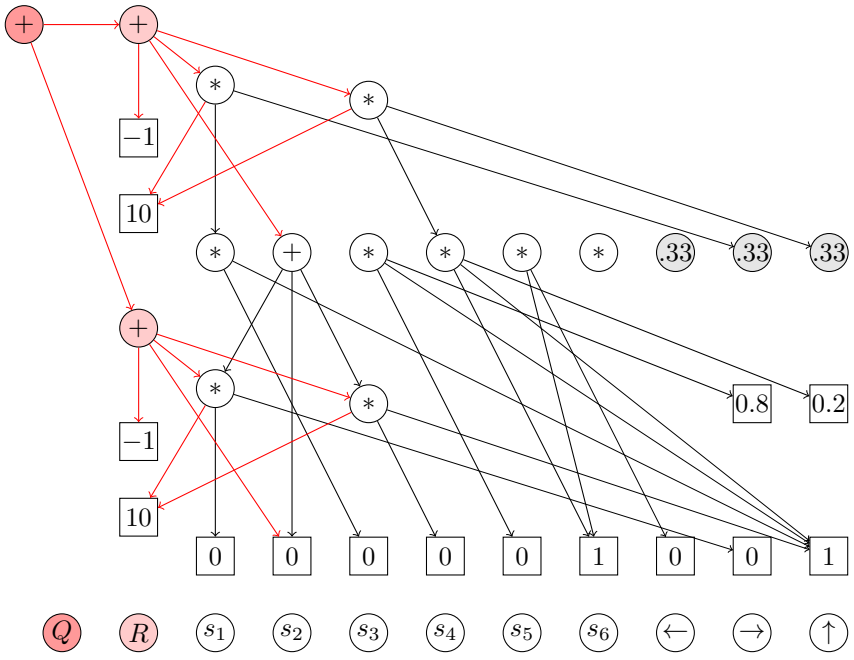


Figure 4.2: DAG approximating the Q value function for the navigation example from Section 2.1.4 over two steps as used for a more advanced conformant heuristic calculation. Reward and its summation to the Q value in red. Note that while the state variables and action variables are still constants at the leaf level, the action variables in higher levels are now being optimized.

Lastly, we will have a look at an overview of the algorithm for these heuristics in Algorithm 6. We cache the states we visit to increase performance if we have few or very similar states. We initially set state fluents and search depth, only we can not afford to calculate a search

depth dynamically. Then we perform the actual calculation of the Q values, depending on the chosen heuristic. In the conformant case, we initialize the conformant actions in addition to the action fluents at the leaf level in the same way as for the standalone planner. Then, we optimize only the conformant action fluents until we have converged or reached a maximum number of gradient ascent steps, returning the last Q value we calculated to the planner. In the propagation case, we simply perform a forward propagation step by evaluating the Q value function once.

Algorithm 6: Heuristic

```
Input: state, actionsToExpand
Result: qValues
qValues =  if state is already cached then
    return qValues(state);
else
    sf ← state.stateFluents;
    maxSearchDepth ← min(maxSearchDepth, state,stepsToGo);
    for all actionsToExpand do
        af ← action.state.actionFluents;
        q;
        if conformant then
            afConformant ← 1 / action.state.size();
            while steps remaining and not converged do
                q ← gradientAscentHeuristic(af, afConformant);
            end
        else
            q ← qFunction(af);
        end
        qValues(state) = q;
        return qValues(state);
    end
end
```

5
Experiments

This section describes the experimental evaluation of our methods. We first look at the setup for the experiment and then discuss strengths and weaknesses of the procedures, key findings, and trends. To do so, we initially look at the individual parameters of the different setups. Then, we compare our standalone planner and heuristics. We also compare and contrast our algorithms to the other closely related state-of-the-art planners PROST [8].

5.1 Experiment Setup

The setup for our experiments closely followed the setup used for the International Planning Competition (IPC) 2018. [16] Notable differences are that the algorithms were tested on the planning domains from the IPC 2011, 2014, and 2018 over 100 runs for each configuration with 2.5 seconds for each step. Calculations were performed at sciCORE[3].

An issue with the SOGBOFA planner [4] in general is that it demands several restrictions from the domain and instance encoding. For one, it assumes that there are no enums in the domain encoding. While these can be replaced in a precompilation step and instead be modelled with interm fluents, SOGBOFA will not work if they are still existent. Furthermore, SOGBOFA can not handle action constraints that are not in the form of a sum constraint over actions. This, again, means that SOGBOFA will not work if it encounters action constraints it does not recognize. The latter is not as problematic in the PROST setting, as we always have concrete actions calculated and use a more general way to handle all action constraints. But, our implementation does not work with enums either. Also, the PROST planner does not support interm fluents. This becomes a problem as enums are compiled away using interm fluents to generate binary domains from the finite domains with enums. As there is currently no version of all IPC 2018 domains without both enums and interm fluents available, we unfortunately have to restrict our benchmark setup by excluding the problematic IPC 2018 domains.

Similarly, the SOGBOFA planner used at the IPC 2018 is unable to parse the 2011 and 2014 instances we used. This leaves us with an overlap of very few domains where we can

[3] http://scicore.unibas.ch/

directly compare the scores between our setup and the original SOGBOFA. Hence, it does not make sense to do so. Instead, we can use the PROST IPC2014 configuration to have a state-of-the-art planning benchmark to compare to. Still, we will closely relate our results to the ones of the original SOGBOFA and see which trends and attributes of the planner we can also notice.

5.1.1 IPC Score

A very commonly used measurement for the performance of a probabilistic planner is the IPC score.
It is the performance score calculated in the planning competitions to evaluate the performance of a planner and defined as

$$\frac{R - R_0}{R^*}$$

with R being the average accumulated reward of the planner over all runs, R_0 the average accumulated reward of a minimal reference policy (the better of a random and a noop policy), and R^* the highest average accumulate reward of planners tested. This can be calculated for every instance, and then summed up over all instances and domains. It is, however, very context sensitive based on the compared configurations, because the difference in accumulated reward is divided by the reward of the best configuration. Unless otherwise specified, IPC scores can only be directly compared between results of the same experiment. Note that we follow the recent trend of the IPC 2018 to use the sum of the IPC scores over all instances and domains as total score instead of the previously used average. As a side effect of this, the 2018 planning domains are weighted stronger as they have 20 instances instead of 10, but we will mention this again if it should have an effect on the experiments. In the following, we will do an in-depth discussion of our different configurations, the tuning of their parameters, and their strengths and weaknesses.

5.2 Standalone Planner

In a first step, we take a look at the results for the standalone planner. For the most part, we are interested in confirming the trends discussed by Cui and Khardon [4] to show that the procedure is solid and our implementation is sound. Furthermore, we want to provide an evaluation of all components discussed in Chapter 4, and discuss which components could merit a further investigation. As a default configuration for the experiments, we used the the one which corresponds to the original SOGBOFA planner:

- Using a dynamic search depth.
- Using a dynamic step size.
- Using a threshold of 0.1.
- Using sum constraints in the sampling of concrete actions and in the action projection to adhere to action constraints.

Since we want to know the contribution of these settings to the performance of the procedure, we tested other options for each of these settings. This was done one setting at a time, with always one setting different from the default to show its effect. The tested differences to the default are:

- Using fixed search depths.
- Using fixed step sizes.
- Using a different threshold.
- Using general reward penalties to adhere to action constraints.

The results are presented in the following.

5.2.1 Search Depth

Table 5.1: IPC Scores for the non-conformant standalone planner at varying search depths (between 3 and the horizon), including the dynamic selection of a search depth. These scores are comparable with Table 5.5. Notable results in bold.

Domain	3	7	10	14	20	30	40	Dynamic
crossing-traffic-2011	**4.42**	9.84	9.84	9.84	9.84	9.84	9.84	9.84
elevators-2011	1.26	0.81	0.81	0.81	0.81	0.81	0.81	0.81
game-of-life-2011	7.92	8.99	9.07	8.76	8.78	8.64	8.77	9.09
navigation-2011	3.22	2.25	2.25	2.25	2.25	2.25	2.25	2.25
recon-2011	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
skill-teaching-2011	9.63	9.23	9.30	9.11	9.16	9.25	9.08	9.09
sysadmin-2011	9.40	9.71	9.76	9.69	9.78	9.67	9.70	9.69
academic-advising-2014	0.00	0.00	0.00	0.00	0.00	0.62	1.23	1.23
tamarisk-2014	9.45	9.30	9.49	9.43	9.24	9.37	9.30	9.45
triangle-tireworld-2014	4.55	3.04	3.09	3.10	2.59	2.57	3.06	4.28
wildfire-2014	**7.61**	9.45	9.59	9.53	9.45	9.56	9.64	9.67
academic-advising-2018	**3.68**	6.26	6.56	6.76	6.44	6.52	5.95	6.37
cooperative-recon-2018	1.40	1.27	1.23	1.47	1.60	1.10	1.44	1.75
Sum	62.54	70.15	70.99	70.75	69.94	70.19	71.08	**73.53**

We have tested different fixed search depths and the dynamic approach of SOGBOFA. From the scores in Table 5.1, we can see that a dynamic search depth leads to an improvement of the procedure in general. Of course, fixed search depths can still work well if they are well suited for the problem, as the dynamic search depth simply fixes a search depth to be used for the whole instance based on an estimation of the needed time. Overall, it is not an enormous increase, but the fixed search depths seems to find a rather reasonable balance between the number possible random restarts and a more informative gradient ascent.
As to the effect of the search depth in general, it is interesting to look at specific domains. Then, we can see that a search depth of three can be very insufficient for certain problems, such as crossing-traffic-2011, wildfire-2014, or academic advising-2018, as we fail to model key mechanisms of that domain with such a small Q value function graph. We can also see

that SOGBOFA does not appear to work at all on some domains, such as elevators-2011, or recon-2011

5.2.2 Step Size

Table 5.2: IPC Scores for the non-conformant standalone planner at varying step sizes (between 0.01 and 0.5), including the dynamic selection of a step size. These scores are comparable with Table 5.6. Notable results in bold.

Domain	0.01	0.05	0.1	0.3	0.5	Dynamic
crossing-traffic-2011	9.86	9.86	9.90	9.86	9.86	9.86
elevators-2011	1.05	1.05	1.05	1.05	1.05	1.05
game-of-life-2011	9.18	9.06	8.47	7.45	6.91	**8.96**
navigation-2011	2.30	2.30	2.30	2.30	2.30	2.30
recon-2011	0.00	0.00	0.00	0.00	0.00	0.00
skill-teaching-2011	9.14	9.15	8.63	8.22	7.63	**9.16**
sysadmin-2011	9.67	9.69	9.79	9.50	9.59	9.65
academic-advising-2014	1.21	1.23	1.38	1.67	1.60	1.23
tamarisk-2014	9.61	9.23	8.72	8.26	8.36	9.56
triangle-tireworld-2014	4.35	4.83	4.49	4.80	5.39	**5.71**
wildfire-2014	9.57	8.30	7.95	7.62	7.58	**9.69**
academic-advising-2018	5.79	5.54	5.59	5.48	5.39	**6.53**
cooperative-recon-2018	3.99	4.60	2.59	0.71	0.21	**1.75**
Sum	75.73	74.83	70.85	66.92	65.88	75.45

We tested several fixed step sizes in addition to the dynamic calculation. The results are shown in Table 5.2. As a general trend, small step sizes seem more suitable, as they lead to a smoother convergence of the algorithm. The dynamic step size appears to find a suitable step size for all domains. Especially when there is a large difference between the step sizes, the dynamic procedure selects one which is among the best of that domain, which leads to a very successful overall performance. This behavior of the dynamic step size is consistent with the findings by Cui and Khardon [4].
An exception is the cooperative-recon-2018 domain, where the dynamic step size is only mediocre, which hurts the total score.

5.2.3 Threshold

In a very similar fashion to the findings of Cui and Khardon, we can see that the choice of thresholds does not have clear trend for the different values or too much overall impact on the score; neither overall nor in specific domains (other than maybe that a 0.1 threshold underperforms in cooperative-recon-2018).
The range of reasonable thresholds seems to be generously large. But, as a threshold of 0.3 yields the best results, we will keep that for our best configuration.

Table 5.3: IPC Scores for the non-conformant standalone planner at varying thresholds (between 0.05 and 0.7). These scores are comparable with Table 5.7. Notable results in bold.

Domain	0.05	0.1	0.3	0.5	0.7
crossing-traffic-2011	9.79	9.79	9.79	9.79	9.79
elevators-2011	0.29	0.29	0.29	0.29	0.29
game-of-life-2011	9.04	8.52	9.01	9.11	9.06
navigation-2011	2.89	2.89	2.89	2.89	2.89
recon-2011	0.00	0.00	0.00	0.00	0.00
skill-teaching-2011	9.31	9.15	9.38	9.32	9.22
sysadmin-2011	9.89	9.87	9.87	9.88	9.88
academic-advising-2014	1.96	2.00	1.98	1.99	1.97
tamarisk-2014	9.32	9.21	9.36	9.33	9.16
triangle-tireworld-2014	4.48	4.28	5.01	4.37	5.18
wildfire-2014	9.66	9.69	9.71	9.74	9.73
academic-advising-2018	5.96	7.36	6.49	6.25	5.48
cooperative-recon-2018	3.17	**1.74**	3.18	3.45	2.76
Sum	75.77	74.77	76.97	76.42	75.40

5.2.4 General Action Constraints

We tested a generalized way to include the essential information from action constraints, without having to implement them by hand for each new type of constriant that is encountered, but not yet supported. Recall that SOGBOFA also uses a specific handling for the sum constraint on the maximum number of concurrent actions allowed, as it is included in the concrete action sampling in addition to the action projection.

In the PROST framework, on the other hand, concrete actions are always generated. While this leads to increased difficulty for the parser, it means that we always have the legal, concrete actions available for us to simply pick the closest one to our symbolic action state. As such, we should be less reliant on the information introduced through the maximum number of concurrent actions during concrete action sampling based on highest marginal probabilities. This lessens the need to have all encountered action preconditions integrated in the procedure, as we could still find legal states in the end. This might therefore also dampen the effect of our generalized action constraint implementation in our version, but we would still gain a lot of information from them regardless.

As our default implementation also adopts the special handling of the maximum number of concurrent actions allowed used in action projection and concrete action sampling, we disable this in the configuration with generalized action constraints. The aim is to test our generalized action constraints, through the integration of action preconditions directly into the Q value function graph, without giving it specific, additional information, which would weight some constraints as more important than others.

We can see in Table 5.4 that the general integration of action constraints gives similar results to the use of the *maximum number of concurrent actions* constraint only. This is somewhat expected, as many domains here do not use a large number of additional action preconditions. One example of a domain with many action preconditions is cooperative-recon-2018. And indeed, we can see that the general action constraints work significantly

Table 5.4: IPC Scores for the non-conformant standalone planner with and without our generalized action constraint integration. These scores are comparable with Table 5.8. Notable results in bold.

Domain	Generalized Constraints	Special Handling of Allowed Actions
crossing-traffic-2011	9.83	9.79
elevators-2011	0.29	0.29
game-of-life-2011	6.86	8.52
navigation-2011	2.89	2.89
recon-2011	0.00	0.00
skill-teaching-2011	8.94	9.19
sysadmin-2011	8.39	9.75
academic-advising-2014	1.23	1.23
tamarisk-2014	9.19	9.27
triangle-tireworld-2014	6.18	4.25
wildfire-2014	9.02	9.67
academic-advising-2018	4.36	**7.42**
cooperative-recon-2018	**3.93**	1.52
Sum	71.12	73.79

better on this domain than only including the maximum number of concurrent actions. On the other hand, we can see in academic-advising-2018, where there is only one other constraint beside the maximum of actions, that the integration of the action preconditions into the Q value function graph can cost a lot of calculation time on such large domains. This does not pay off if it does not lead to additional information.

5.3 Conformant Standalone Planner

The experiment setup for the conformant standalone planner follows the setup for the non-conformant planner.

5.3.1 Search Depth

Table 5.5 shows the effect of the search depth on the performance of SOGBOFA. Note that these IPC scores can be directly compared to the ones presented in Table 5.1.

As a general trend, similar to the non-conformant version, we can see that a very small search depth is inqadequate for many problems. We also have to note that the dynamic search depth calculation does not generate the best results. While the dynamic search depth appears to select a very reasonable search depth on most domains, cooperative-recon-2018 catches the eye as a domain where it does not work at all. With a better performance on this domain, it would represent the best results overall. We have to also keep in mind that the 2018 domains have twice the number of instances of the other domains, which widens the gap in the sum over all domains this deficit causes.

Table 5.5: IPC Scores for the conformant standalone planner at varying search depths (between 3 and the horizon), including the dynamic selection of a search depth. These scores are comparable with Table 5.1. Notable results in bold.

Domain	3	7	10	14	20	30	40	Dynamic
crossing-traffic-2011	**4.04**	9.84	9.84	9.84	9.79	8.63	7.55	9.81
elevators-2011	**1.79**	4.69	3.94	3.43	4.38	3.35	3.06	3.93
game-of-life-2011	7.92	9.08	9.05	9.07	8.88	8.77	8.84	8.64
navigation-2011	3.25	2.25	2.25	2.95	2.95	4.24	4.70	2.69
recon-2011	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
skill-teaching-2011	9.78	8.95	9.02	9.22	8.90	7.98	7.47	9.10
sysadmin-2011	9.04	9.04	9.16	9.12	8.72	8.33	8.27	8.85
academic-advising-2014	0.00	0.00	0.23	0.69	0.86	2.17	3.14	0.00
tamarisk-2014	9.22	8.57	7.82	7.51	6.31	5.58	5.36	8.85
triangle-tireworld-2014	4.53	3.19	3.53	2.84	2.73	4.15	4.68	3.90
wildfire-2014	8.21	9.57	9.51	8.98	9.30	9.53	9.59	9.57
academic-advising-2018	**3.80**	5.19	4.50	4.09	4.01	3.78	3.80	5.36
cooperative-recon-2018	**1.20**	2.07	1.57	2.77	5.46	5.25	3.92	**0.03**
Sum	62.75	72.43	70.42	70.5	72.29	71.76	71.39	70.73

5.3.2 Step Size

In Table 5.6 we can see that we generally need larger step sizes for the conformant SOGBOFA compared to the non-conformant SOGBOFA.

Table 5.6: IPC Scores for the conformant standalone planner at varying step sizes (between 0.01 and 0.5), including the dynamic selection of a step size. These scores are comparable with Table 5.2. Notable results in bold.

Domain	0.01	0.05	0.1	0.3	0.5	Dynamic
crossing-traffic-2011	9.86	9.93	9.88	9.83	9.68	9.83
elevators-2011	0.05	3.30	3.96	4.69	3.28	4.61
game-of-life-2011	9.16	8.43	8.09	7.17	6.80	8.51
navigation-2011	2.30	2.30	2.30	3.89	4.25	**2.94**
recon-2011	0.00	0.00	0.00	0.00	0.00	0.00
skill-teaching-2011	9.15	9.14	8.91	8.92	8.04	9.17
sysadmin-2011	9.45	9.34	9.23	9.22	9.44	8.81
academic-advising-2014	0.00	0.94	2.41	2.02	1.35	**0.00**
tamarisk-2014	9.50	9.09	8.85	**3.74**	**3.85**	8.96
triangle-tireworld-2014	4.57	4.62	4.46	4.33	5.19	5.31
wildfire-2014	9.29	9.23	8.95	8.81	8.78	9.59
academic-advising-2018	5.57	5.58	5.51	5.00	5.20	5.46
cooperative-recon-2018	1.95	0.00	0.00	0.25	0.16	**0.03**
Sum	70.85	71.91	72.56	67.88	66.01	**73.20**

Still, step sizes too large are can be conterproductive, the most prominent example for this is tamariks-2014. But the best configuration is, again, the dynamic step size. With the exception of navigation-2011, academic-advising-2014, and cooperative-recon-2018, it selects a very good step size for each domain.

5.3.3 Threshold

Table 5.7: IPC Scores for the conformant standalone planner at varying thresholds (between 0.05 and 0.7). These scores are comparable with Table 5.3. Notable results in bold.

Domain	0.05	0.1	0.3	0.5	0.7
crossing-traffic-2011	9.82	9.59	9.86	9.83	9.86
elevators-2011	0.29	**3.77**	0.29	0.29	0.29
game-of-life-2011	8.84	8.06	8.81	8.86	8.83
navigation-2011	2.89	**4.00**	2.89	2.95	2.89
recon-2011	0.00	0.00	0.00	0.00	0.00
skill-teaching-2011	9.17	8.91	9.26	9.41	9.33
sysadmin-2011	9.76	8.93	9.87	9.69	9.73
academic-advising-2014	0.00	0.00	0.00	0.00	0.00
tamarisk-2014	9.47	8.89	9.31	9.45	9.65
triangle-tireworld-2014	4.88	4.84	6.28	5.98	4.26
wildfire-2014	9.68	9.71	9.66	9.63	9.65
academic-advising-2018	4.40	5.42	4.33	4.26	4.35
cooperative-recon-2018	4.48	**0.67**	3.77	3.88	4.10
Sum	73.68	72.79	74.33	74.25	72.93

The results for the effect different thresholds can have on conformant SOGBOFA are shown in Table 5.7. It is difficult to see significant differences between them, even when looking at specific domains. The only notable outliers appear to be generated by the threshold of 0.1. It has much better results than the other thresholds in the elevators-2011 and the navigation-2011 domain, but much worse results than the others in cooperative-recon-2018, although it remains unclear why that is.
Again, the threshold of 0.3 produces the best results overall.

5.3.4 General Action Constraints

Table 5.8 shows our general version of integrating action precondition information compared to using the information on the maximum number of concurrent actions only.
We can still see that the general version performs better on cooperative-recon-2018. But it also has much better results on the elevators-2011 domain with the conformant version.
In general (and on tamarisk-2014 specifically), however, the limited information gain still does not appear to be worth the increased complexity of the Q value function tree.

5.4 Propagation Heuristic

We now do an in-depth evaluation of the SOGBOFA heuristics, starting with the propagation heuristic. Recall that the propagation heuristic uses a forward propagation through the SOGBOFA Q value function graph to give a Q value estimate for a specific action.
The key attributes to good performance of any heuristic are how informative it is and how fast it can be calculate. It is necessary to balance a tradeoff between these two attributes: a slow, but informative and a fast, but uninformative heuristic might both be outperformed by a heuristic in between. Regarding the propagation heuristic, the idea was to design a

Table 5.8: IPC Scores for the conformant standalone planner with and without our generalized action constraint integration. These scores are comparable with Table 5.4. Notable results in bold.

Domain	Generalized Constraints	Special Handling of Allowed Actions
crossing-traffic-2011	9.81	9.59
elevators-2011	5.82	3.77
game-of-life-2011	7.59	8.07
navigation-2011	4.79	4.00
recon-2011	0.00	0.00
skill-teaching-2011	6.28	8.96
sysadmin-2011	8.45	8.82
academic-advising-2014	0.00	0.00
tamarisk-2014	5.39	8.97
triangle-tireworld-2014	5.00	4.80
wildfire-2014	9.47	9.69
academic-advising-2018	4.38	5.37
cooperative-recon-2018	2.25	0.67
Sum	69.23	72.71

faster, but less informed heuristic, particularly in comparison to the conformant heuristic, which is strictly slower to calculate. But the aforementioned tradeoff is not only dependent on the version of the heuristic, but it also heavily depends on the parameters we use. Hence, we need to discuss the effect of our parameters on the interplay of informativeness, speed, and the resulting performance of our heuristics. Luckily, in the case of the propagation heuristic, we only have one parameter: the search depth.

5.4.1 Search Depth Parameter

The search depth parameter is very important to our analysis. It is the only parameter for the propagation heuristic and important part of the conformant heuristic. Based on Section 3.1 we would assume that a higher search depth leads to a more informed heuristic, as more future steps are taken into account. In some cases, this can have very drastic effects: consider the simple case where the reward does not contain any action fluents directly in its formula (such as the elevators-2011 domain). Then, we would need at least a search depth of two to detect that the actions have an effect on the Q value. This means that we would expect a higher search depth to lead to a very large increase in informativeness. But this, of course, comes at a price: a higher search depth should, at least in theory, increase the evaluation time of the Q value function linearly. As we have seen in Section 5.2.1 and Section 5.3.1, a good search depth is a tradeoff of speed and information.

Heuristic guidance: Now, we want to verify if our experiments agree with our theoretical assumptions of guidance and calculation time for the search depth. First of all, that means testing the guidance our heuristic provides at different search depths. This can be detected by evaluating the results for always playing the action with the highest Q value, as given by the heuristic. Such a procedure effectively considers our heuristic to be our search enigne: We evaluate each action with our heuristic, and return the one with the highest Q value as

the best action. In this case, the very harsh time constraints for the heuristic are mostly eliminated, as we now have the 2.5 seconds for a planning step to calculate our heuristic for the applicable actions.[4] The resulting IPC score gives a good estimate for the guidance quality of the heuristic.

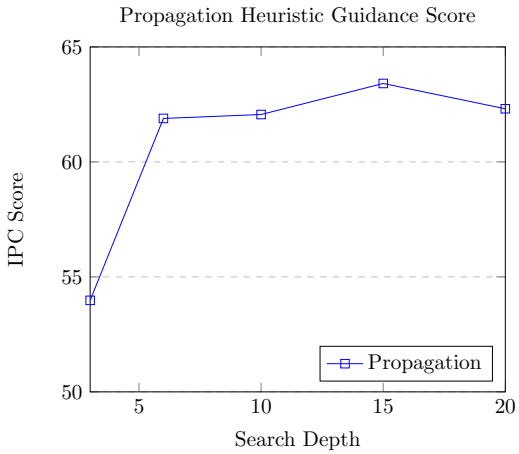


Figure 5.1: IPC Score of the propagation heuristic when used as a search engine to indicate the guidance provided at varying search depths.

Figure 5.1 shows the guidance of the propagation heuristic at varying search depths. And, indeed, the guidance increases with a rising search depth. While this is not a drastic increase over all domains, it does support our assumptions. From the tested domains, the crossing-traffic-2011, the wildfire-2014, and the academic-advising-2018 domains profit most from an increased search depth, in particular the increase from three to six (as shown in Table 5.9). This agrees with our assumption that a very small search depth is sometimes insufficient to see basic connections between the actions and their effects.

Table 5.9: IPC Scores the propagation heuristic on interesting domains at different search depths.

Domain	SD 3	SD 6	SD 10	SD 15	SD 20
crossing-traffic-2011	3.41	8.82	9.05	9.05	9.05
wildfire-2014	5.07	9.42	9.64	9.70	9.76
academic-advising-2018	3.28	4.49	4.62	5.32	4.32
Sum (all domains)	53.98	61.89	62.06	63.41	62.31

Calculation time: In a second step, we are interested in the cost in calculation time that comes with an increased search depth. Generally, in order to evaluate how fast a heuristic can be calculated in practice, it is a good idea to look at the performed trials. Specifically, the number of trials a planner can perform using this particular heuristic. While this does have a certain bias through cache hits during heuristic calculation when states are very similar

[4] In extreme cases, timeouts can still apply. But in that case, the heuristic would not usable in practical applications with multiple trials anyway.

due to limited stochasticity, other options include their own bias. For example, looking at the number of different states visited to filter out cache hits, includes the non-negligible time used by the planner.
Figure 5.2 shows the sum of performed trials during the first planning step over all domains and instances for the propagation heuristics at varying search depths. And indeed, the cost of adding more layers to the Q function to reach a higher search depth is mostly linear, reflecting our theoretical assumptions very closely.

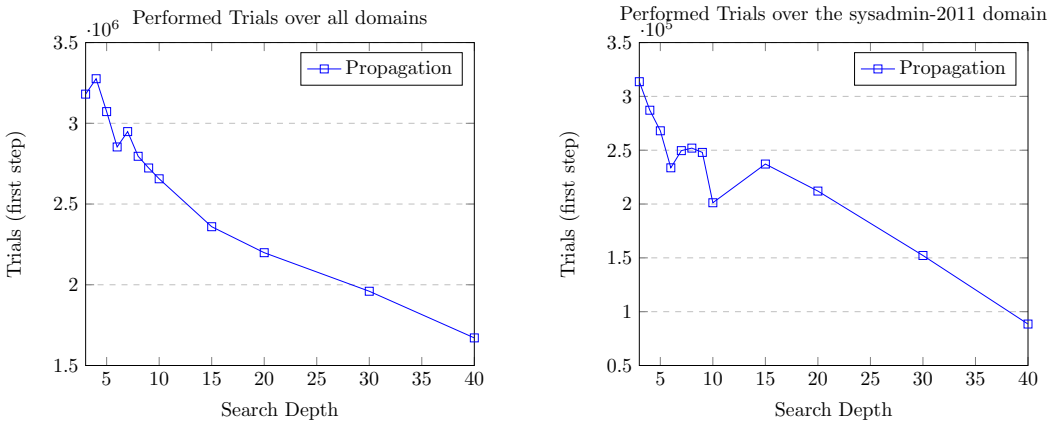


Figure 5.2: Performed trials of the propagation heuristic at varying search depths. For each instance, the number of performed trials in the first planning step is used and summed up over instances.

Of course, exact curves for the relationship between search depth and performed trials, shown here over all domains, will vary from domain to domain, but the overall trend is representative of the results by domain. Still, it is also worthwhile to look at an extreme case in addition to the overall sum of performed trials, such as the last instance of the sysadmin-2011 domain (shown in Figure 5.2). Sysadmin-2011 is a particularly large and stochastic domain, which makes it ideal to evaluate the number of performed trials. This is because it can happen (as mentioned before) in smaller and less stochastic domains, where only few different states are reached, that cache hits of previously visited states heavily boost the number of trials that can be performed which can mitigate and mask the effects from slower heuristic calculations.
Here, the overall linear trend is even clearer, with some outliers for small changes in search depth, which is to be expected with fewer samples.

Performance as a heuristic: We have seen an increased search depth lead to a better heuristic guidance, with diminishing returns, but also a linearly degrading calculation speed. The best performing search depth parameter for our heuristic has to strike a balance between a higher search depth for better quality and a lower search depth for a faster calculation.
Figure 5.3 shows this search depth to be at seven for our experiments. Note that the summed trend shown here is representative of the trends in the different domains.

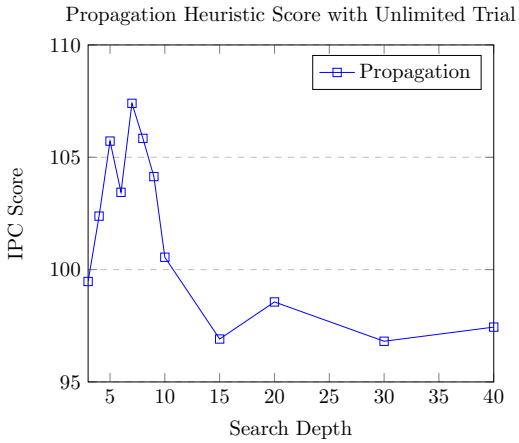


Figure 5.3: IPC Score of the propagation heuristic at varying search depths.

5.5 Conformant Heuristic

Recall that the conformant heuristic optimizes conformant actions using gradient ascent to achieve receive a better heuristic estimation from the Q value function graph.

5.5.1 Parameters

Above, we have shown the effects of the search depth parameter on the guidance, calculation time, and finally the resulting performance score for the propagation heuristic. In the following, we want to use the same approach to discuss and tune the parameters of the conformant heuristic. In addition to the search depth parameter, we now also have to consider the number of gradient steps, step size, and threshold for convergence.
Previously, we have seen algorithms for a dynamic calculation of search depth, step size, and even the gradient steps[5] in the context of the standalone SOGBOFA planner. Accordingly, a natural thought would be to simply apply these to find good parameters now. Unfortunately, the dynamic calculations of search depth and step size do not make sense in the heuristic setting: They are calculated by forward propagation or gradient calculations on the Q function graph. This would lead to an extreme overhead for only one heuristic trial. As such, we have to find good all around values for these parameters. In particular,this means that we will need to look at the interplay of these parameters. To look at the overall trends of a single parameter, we can average all combinations of parameters tested at each value of that parameter. But, we have to keep in mind that such an overall trend only gives a very limited view of the complete picture. Hence, we will also discuss noteworthy combinations of parameters.

Heuristic guidance: While we have mentioned the threshold for convergence as a potential parameter to tune, in this setting we will mostly disregard it. This is because the only thing it does is indirectly affect the number of gradient steps performed. But, now we already give an explicit step size. Then, the threshold only potentially reduces the number

[5] Through the use of the threshold.

of gradient steps to save calculation time. However, in this experiment, we are not interested in the performance optimization as we have mostly eliminated the time aspect anyway. Instead we rather want to know the potential guidance the heuristic can provide and how this is influenced by the number of gradient steps. This would be negatively impacted when less gradient steps are performed than we set out to test for any specific configuration. Therefore, we will simply use a very small threshold for this evaluation.

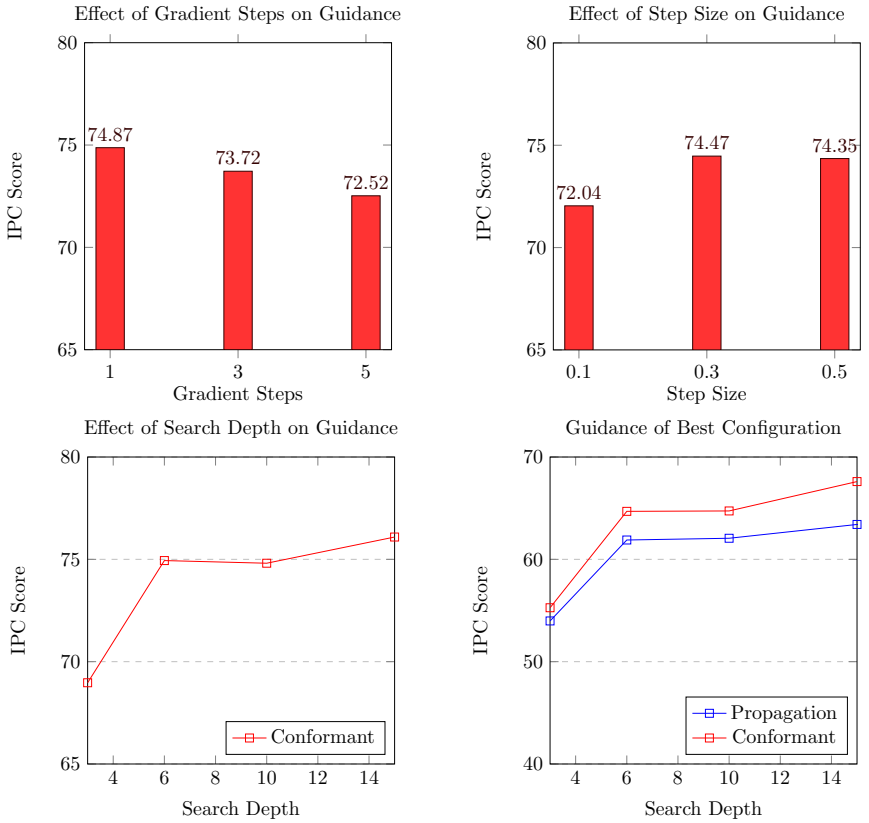


Figure 5.4: IPC scores for varying parameters with the conformant heuristic as search engine to give an estimate of the parameters on heuristic guidance. The last figure compares the guidance at different search depth with the respective best configurations against the propagation heuristic. Note that the scores from the last figure are not directly comparable to the other scores mentioned here.

Figure 5.6 shows the effect of the different parameters on heuristic guidance averaged over all combinations.

We can see that, as expected, search depth plays an important role for the heuristic quality. Similar to the results for the propagation heuristic, a higher search depth clearly leads to a better guidance, and expectedly so. It gets harder to tell a trend for the number of gradient steps and step size. From the overall trends, we can see a slight decrease in score with more gradient steps. While this is rather unexpected, it is only a very small difference, which makes it hard to draw confident conclusions from this. This is trend is understandable if we keep in mind that the step sizes used are very large compared to standard gradient ascent methods, which leads to a high variance in the results of the optimization, especially with

more gradient steps. Yet, a high step size seems to be necessary from the results, as a very small step size will be unable to exploit the gradient information in the few steps.
Even when looking at the full combination of parameter configurations, there is no clear trend favoring one particular value of the parameter.
The different step sizes seem to make only very slight differences on guidance when the search depth is low, but they appear to have a stronger impact at higher search depths, as shown in Table 5.10. The same can be observed for the gradient steps.

Table 5.10: Effect of different step sizes and gradient steps on the heuristic guidance given by IPC scores for the conformant heuristic as a search engine at different search depths.

Configuration	Step Size 0.1	Step Size 0.3	Step Size 0.5
Search Depth 3	53.71	53.83	53.88
Search Depth 10	60.69	63.85	62.42
Configuration	**1 Gradient Step**	**3 Gradient Steps**	**5 Gradient Steps**
Search Depth 3	54.18	53.72	53.47
Search Depth 10	63.29	62.77	61.31

This makes sense, as at a low search depth, the gradients themselves provide much less guidance. When the gradients are more meaningful at higher search depths, it is more important to use an adequate step size. But, even at a high search depth, the differences between the step sizes is low enough to be potentially caused by only a single domain where a certain step size works particularly well. And indeed, for example the at search depth 15, a step size of 0.3 with 5 gradient steps appears to work particularly well for the elevators domain, which then pushes its total score over all domains to be higher than the other step sizes. That the choice of a good step size is rather domain dependent is not necessarily news, though. Cui and Khardon [4] already showed for the original SOGBOFA that a dynamic step size payed off on certain problems, which we confirmed earlier for the standalone planner.
But we also have to consider the interplay of the step size with the number of gradient steps. A good step size generally also tuned to the number of gradient steps taken. When we are only interested in the quality of our heuristic and not in the time it takes to calculate, we would prefer a very large number of gradient steps with a low step size.
Table 5.11 shows additional testing in that direction: We have added configurations with ten and twenty gradient steps to test a smaller step size of 0.05 on. There appears to be a tradeoff between the smallest step size not being able to exploit the gradient information enough and a larger step size causing too much variance. This indicates that the procedure follows our theoretical assumptions in practice as well, but only if the search depth and the gradient steps are high enough.

Table 5.11: Effect of different step sizes on the heuristic guidance given by IPC scores for the conformant heuristic as a search engine. Here, the search depth is fixed at 15, but we use a higher number of 10 and 20 gradient steps and include a smaller step size.

Configuration	Step Size 0.05	Step Size 0.1	Step Size 0.3	Step Size 0.5
Average Score	67.35	69.30	65.38	65.88

Unforunately, with the aim of an at least somewhat realistic performance for a heuristic in mind, we have to limit our gradient steps heavily. Already for the tests shown in Table 5.11, with ten and twenty gradient steps at high search depths, the heuristic was sometimes unable to even finish one trial in particularly large domains. This is also the reason why the tested step sizes are all rather large. But, with a larger step size and fewer trials, it is hard to reach optimal results consistently.
Nonetheless, we have seen some configuration which work very well together and, in fact, lead to a strong improvement in guidance over the propagation heuristic. Specifically, we have previously seen a conformant configuration that works particularly well for the elevators domain. This domain was specifically mentioned by Cui et al. [5], because here, the conformant variant of SOGBOFA lead to a large improvement over the non conformant procedure. The assumption of the non-conformant procedure that all future actions are partially executed at the same time can sometimes be very problematic when a random policy for future actions provides poor guidance. The elevators domain is a good example of this. Because of this assumption, the elevator has a hard time picking up passengers, as he is always also facing the wrong direction, stationed at the wrong floor, or has his doors closed. Optimizing the future actions so he, for example, only moves in one direction means that the planner recognizes how to pick up passengers, which is the only way to increase the reward. And indeed, we have also seen in Table 5.9 that the propagation heuristic provides a very poor guidance on the elavators domain. This, similar to the original SOGBOFA planner, can also be fixed with the conformant version (as can be seen in Table 5.12). But, the results for the conformant elevators domain do not show such a large improvement over all configurations. This indicates that the gradient based conformant procedure is rather sensitive to the chosen configuration and parameters. Still, even without the ideal configuration of parameters, we can see a better heuristic guidance from the conformant heuristic compared to the propagation heuristic, but not as pronounced.

Table 5.12: IPC scores for some domains showing an improvement in guidance of the conformant heuristic over the propagation heuristic (as a search engine). We compare the best propagation and conformant heuristics over all domains with the conformant heuristic that gives the best score on that specific domain. As such scores here are not directly comparable with the rest of this section.

Domain	Propagation Heuristic	Conformant Heuristic	Best Settings
elevators-2011	0.24	1.99	4.00
navigation-2011	0.14	0.47	0.49
recon-2011	0.00	1.78	3.00
triangle-tireworld-2014	0.88	1.05	1.34

Calculation time: While we have seen that the conformant procedure generally provides better guidance, we can also expect a strictly worse calculation speed. This is, of course, because we always have a final forward propagation step of the Q value for the optimized conformant actions, which is identical to the propagation heuristic. But, before that, we have to optimize the conformant actions in a set number of gradient steps, each of them

taking more time than just the forward propagation. As such, the main question is whether the calculation of the heuristic is fast enough to be able to profit from the better guidance for better overall results.
Accordingly, the first configuration to test would be the fastest one, which is the very minimal configuration of one gradient step. As we have found that a large search depth is the most important parameter to a better heuristic guidance, we now examine how fast the configuration with one gradient step is to calculate at varying search depths, as seen in Figure 5.5.

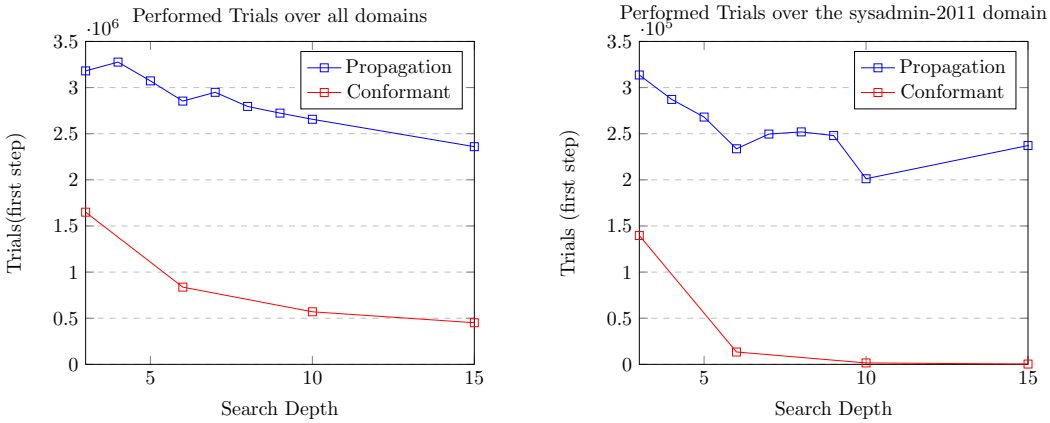


Figure 5.5: Performed trials of the conformant heuristic at varying search depths, compared with the propagation heuristic. For each instance, the number of performed trials in the first planning step is used and summed up over instances.

We can see that, initially, the number of performed trials looks reasonable, as we have already explained that we would expect a a number a little less than 50% of the performed trials the propagation heuristic can achieve. But, as soon as we increase the search depth, which would provide much more guidance, the performed trials of the conformant heuristic drop unproportionally to the propagation heuristic. This is even worse when we minimize the effect of caching by looking again at the very stochastic sysadmin-2011 domain. On the last instances of the sysadmin-2011 domains, only one trial is performed when using a good search depth. But even before that point, we reach a very problematic amount of trials, that makes it almost impossible to attain a good overall performance, no matter the quality. And in fact, the results in Figure 5.7 show that the best configurations from the single trial experiment do not outperform the most efficient calculations .
A direct comparison between propagation and conformant heuristic on the different domains (Figure 5.6) shows that, on every domain, the conformant heuristic only performs a small fraction of the trials the propagation heuristic can achieve. With increasing search depths, it can keep a good or decent percentage of the propagation trials on the easier domains with limited action spaces, such as crossing-traffic-2011, elevators-2011,navigation-2011,skill-teaching-2011, and triangle-tireworld-2014. On all other domains, with a larger number of possible actions, it sinks under 10% of the trials performed by the propagation heuristic after the smalles search depth.
This is partly due to the fact that the propagation heuristic is very fast to compute and

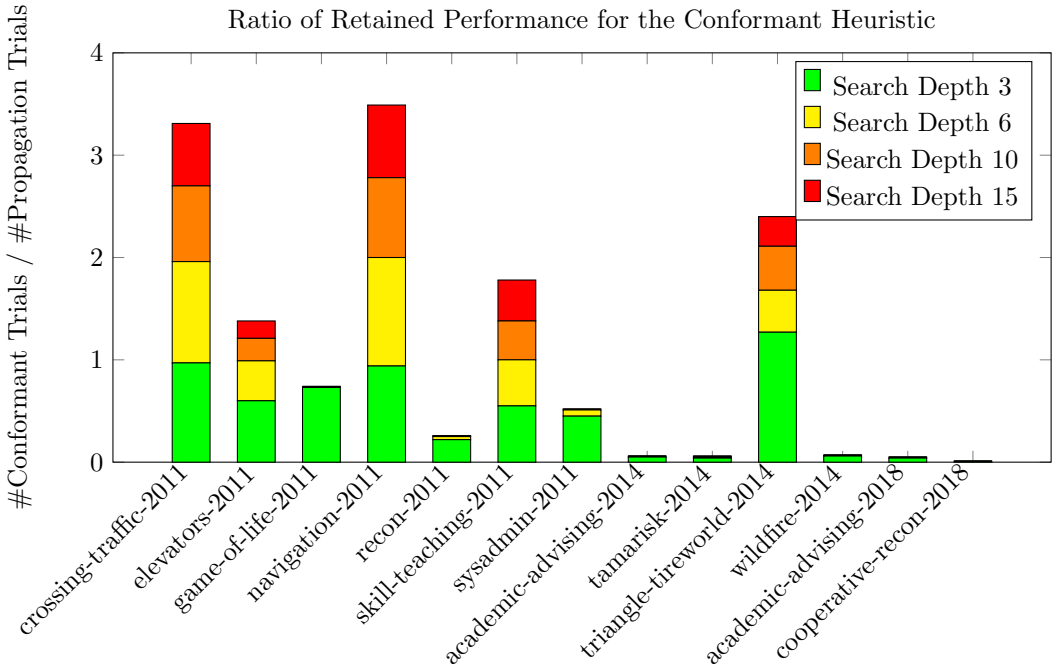


Figure 5.6: The ratio of performed trials for the conformant heuristic to the performed trials of the propagation heuristic over different search depths.

performs a very large amount of trials. It is to be expected that the gradient calculation is a core bottleneck of this procedure, leading to notable decline in performed trials. Still, the decrease in performed trials at higher search depths in particular is significantly lower from what we would expect with only one gradient step more than the forward propagation. As a result, the total number of trials becomes unusably small in larger problems. These results indicate that the gradient calculation is simply too slow to produce good results in this heuristic setting.

As to why this is, we can identify two main problems in the concrete implementation of the presented concepts which can explain a slow gradient calculation, both related to the autodiff library used. The first problem has to do with the calculation mode of the gradients: As we discussed in Section 2.2, theory suggests that reverse mode is ideal for such large, scalar valued functions, due to it being able to calculate gradients for all nodes in only one sweep of the graph. Yet, in practice, reverse mode proved unusable for these gradients, failing to calculate the gradients even for small search depths on simple instances in a reasonable time. While reverse mode does have a certain memory overhead, in this case, it appears that this due to an unoptimized implementation of the library.[6] As a custom implementation of reverse mode without any guarantees of leading to an improvement was outside the scope of this work and as forward mode seemed to be very optimised in the library, functioning reasonably well in tests, we decided to stick to forward mode. This does mean, however, that gradient calculation has to suffer to some extent if the action state, and with that the number

[6] From personal communication with the author of the library, it appears that it is much more optimized for forward mode than it is for reverse mode.

of derivatives that have to be calculated, increases. And especially using the conformant variant, we have to pay a high price for this, as we have many actions to be optimized over all layers, which normally would have almost no overhead. The second problem has to do with the strucure of our problem: We have a potentially very large and complex function we build in the Q function. A large part of gradient calculation will be spent calculating this function and building the associated graph. But, if we want to calculate multiple gradient steps for one state-action pair (which we do pretty much always), the structure of this graph does not change at all. We only have different values at the action level which we want to propagate through the graph. As such, it would be possible to exploit this and save time by not recalculating the whole graph but reusing the previously calculated one. Unfortunately, this is not implemented in the autodiff library. Using a proper reverse mode and reusing the Q function graph would presumably lead to an improvement of gradient calculation speed and, as it appears, to the overall quality of the procedure.
In any case, as is, it is very hard to justify the use of the conformant version when the falloff in trials is so steep compared to the non-conformant version. This also means that it does not make sense to attempt a more rigorous gradient calculation with more steps at this time, as already one step provides a struggle for the calculation speed.

Performance as a heuristic: We showed above that, again, search depth is the key parameter, with a large one leading to better guidance but also a much slower calculation. The remaining question is where the best performing tradeoff for the search depth is.
Unfortunately, it appears that the increased informativeness of a higher search depth does not do enough to compensate the significantly lower amount of trials, and the best configuration is the one with a search depth of 3.

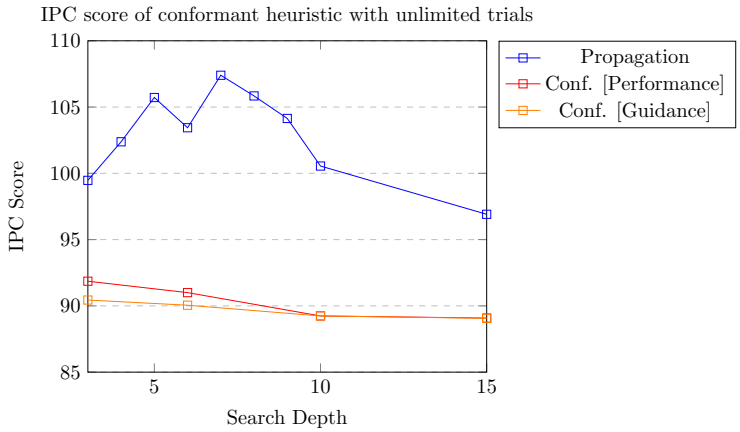


Figure 5.7: IPC Score of the conformant heuristic at varying search depths, compared to the propagation heuristic. Conformant [Performance] stands for the configuration providing fastest calculation (with a single gradient step, Conformant [Guidance] stands for the configuration providing the best guidance).

But, as Figure 5.7 shows, not by a huge margin. There are also some domains where a higher search depth leads to better results, specifically academic-advising-2018, sysadmin-2011, and wildfire-2014 (as shown in Table 5.13).

Table 5.13: IPC scores for some domains showing an improvement of the conformant heuristic at a higher search depth.

Domain	SD 3	SD 6	SD 10	SD 15
sysadmin-2011	5.83	7.11	8.35	8.84
wildfire-2014	8.72	9.26	9.31	9.43
academic-advising-2018	3.61	3.47	3.48	4.73

These are all domains where the calculation was very expensive, but also among the domains with the best guidance and a notable benefit trough increased search depth. This shows that, given a particularly informative configuration, it can be useful for the conformant heuristic as well to invest more time into its calculation, indicating that the guidance is solid. It is very likely that, with a fix of the discussed performance problems of the gradient calculation and the resulting increase in performed trials, the overall performance of the heuristic would be much better. Because that would also allow the use of the much more informed configurations at a higher search depth than 3 and more than one gradient step.

5.6 Comparison to State-of-the-Art

With good configurations for the standalone planner and the heuristic found for the non-conformant and the conformant version, we can now evaluate how they compare to the state-of-the-art.

Table 5.14: IPC Scores for the the best performing configuration of each category. Notable results in bold.

Domain	Prost	Planner	C. Planner	Propagation	Conformant
crossing-traffic-2011	8.66	4.19	4.19	**9.72**	8.07
elevators-2011	9.38	0.04	0.04	9.28	**9.55**
game-of-life-2011	9.60	4.86	4.79	9.02	8.57
navigation-2011	8.88	0.24	0.24	9.31	9.28
recon-2011	9.52	0.00	0.00	9.57	**9.61**
skill-teaching-2011	9.07	8.39	8.02	9.09	**9.30**
sysadmin-2011	6.76	**9.70**	**9.75**	7.45	5.76
academic-advising-2014	2.99	1.18	0.00	3.61	3.06
tamarisk-2014	7.64	6.37	6.08	**9.65**	7.52
triangle-tireworld-2014	7.61	1.08	1.09	6.37	4.92
wildfire-2014	5.52	**9.68**	**9.70**	8.99	8.59
academic-advising-2018	3.23	**6.68**	**4.76**	4.72	3.62
cooperative-recon-2018	9.58	1.79	0.94	**10.23**	3.96
Sum	98.44	54.17	49.58	**107.00**	91.81

Table 5.14 shows the IPC scores for the best configurations of the non-conformant standalone planner, the conformant standalone planner, the Prost planner using the propagation heuristic, and the Prost planner using the conformant heuristc in comparison against the state-of-the-art Prost IPC2014 configuration.
We discuss these results in the following.

5.6.1 Comparison of Standalone Planners

Overall, the non-conformant version of the standalone planner produces better results than the conformant version. This is in contrast to our expectations, but can be explained by the somewhat ill-suited implementation of gradient calculation that was explained in Section 5.5. This also affects both versions of the standalone planner, as it limits the amount of random restarts that can be performed, especially in large domains. But in particular, the conformant SOGBOFA originally was such a large improvement to SOGBOFA due to not being slower to calculate thanks to the reverse mode gradient calculation [5]. If the time increases for the gradient calculation, there is a tradeoff where it is not necessarily worth it any more to perform the conformant procedure.
When compared to the state-of-the-art benchmark of the PROST IPC2014, neither versions can reach results that strong.
As discussed in Section 5.1, it unfortunately does not make sense to compare the results directly to the original SOGBOFA planner used at the IPC 2018 due to necessary restrictions on the benchmark domains leading to mutually exclusive benchmarks for the planners.
This restriction also impacts our comparison with the IPC2014 planner, as it means that we do not have most of the 2018 domains to compare on. The SOGBOFA procedure was shown and designed to perform much better than PROST specifically on the 2018 domains, which means that this benchmark set includes a bias towards good results from IPC2014 compared to the original SOGBOFA results.
With this in mind, together with the performance issues, it is understandable that the standalone planner can not produce results as good as PROST IPC2014.
Yet, on the domains sysadmin-2011, wildfire-2014, and academic-advising-2018, the standalone planners manage to achieve the best results, indicating that the procedure can even under these limitations lead to an advantage on large and very stochastic domains.

5.6.2 Comparison of Heuristics

We have seen above that both heuristics show similar trends regarding the search depth as main component controlling guidance and calculation speed.
When compared against each other, we can see in Figure 5.6 and Figure 5.5 that the theoretically more informed and slower conformant heuristic also behaves like that in practice. Yet, when it comes to the overall performance, Figure 5.7 shows that the propagation heuristic strikes a better balance between speed and quality.
The last step is now to compare them against the PROST heuristic IDS used in the IPC2014 configuration, representing our state-of-the-art benchmark.
Most importantly, of course, we are interested in the overall score, but we also want to know about their strengths and weaknesses regarding quality and speed.

Heuristic guidance: We have seen in Figure 5.6 that, in accordance with the theory, the conformant heuristic provides better guidance.
However, the IDS beats the guidance of both our heuristics in informativeness over all domains. Still, there is a number of domains where the both heuristics are better than IDS:

Table 5.15: Heuristic guidance through IPC scores of the different heuristics as search engine on interesting domains.

Domain	IDS	Propagation	Conformant
skill-teaching-2011	8.09	9.49	9.26
sysadmin-2011	5.11	9.21	9.24
tamarisk-2014	5.00	9.30	9.75
wildfire-2014	6.38	9.42	5.04
academic-advising-2018	0.77	4.49	3.32
Sum (all domains)	89.13	61.89	54.29

academic-advising-2018, tamarisk-2014, wildfire-2014, skill-teaching-2011, and sysadmin-2011, as shown in Table 5.15. Notice that the conformant wildfire-2014 score is lower because it uses a low search depth in the best configuration. Again, these are generally larger and more stochastic domains where SOGBOFA gives a good guidance.

Table 5.16: Performed trials of the different heuristics.

Domain	IDS	Propagation	Conformant
sysadmin-2011	232'050	249'611	139'629
Sum (all domains)	1'490'326	2'948'572	1'649'386

Calculation time: Figure Table 5.16 shows that the propagation heuristic can perform notably more trials than IDS over all domains. It can calculate about twice the trials IDS can, which is a very large increase. But, there are some domains where it struggles and the IDS is faster, especially at the higher search depths. For example, in the sysadmin-2011 domain, it only performs slightly more trials than IDS. The conformant heuristic can compete on many domains with the number of performed trials of the IDS, but not on the more complex domains such as sysadmin.

Performance as a heuristic: Regarding overall performance, it appears that the propagation heuristic strikes a very strong balance between guidance and calculation time. This leads to it outperforming the IDS by quite a bit, as shown in Table 5.14, only being beaten narrowly in the domains elevators-2011 and game-of-life-2011, where its heuristic guidance was particularly bad. This is very interesting, as it indicates that there were cases where its fast calculation made up for mediocre guidance, but also vice versa, meaning it combines both aspects very well.
The conformant heuristic can, unfortunately, not boast with such results. It cannot perform enough trials, which means it cannot make up for cases where the heuristic guidance is insufficient and prevents it from using more insightful heuristic configurations. As such, it is generally worse than the IDS, except for the domains where its heuristic guidance was better.
Still, the very strong performance of the propagation heuristic demonstrates that the incorporation of the SOGBOFA graph into a THTS heuristic can lead to state-of-the-art results.

6 Conclusion

In this thesis, we apply the method of Symbolic Online Gradient-Based Optimization for Factored Action MDPs to guide Trial-based Heuristic Tree Search. SOGBOFA, created by Cui and Khardon [4], builds a symbolic approximation of the Q value function for probabilistic planning tasks. On this differentiable function, it performs gradient ascent to optimize random starting actions based on the expected Q value. This is, in concept, a symbolic extension of Monte-Carlo Tree Search through independence assumptions. As such, the information provided by this approach can also be used to guide THTS, a generalized MCTS framework. In a first step of our efforts to evaluate the potential guidance SOGBOFA can bring to THTS, we implemented two search engines in THTS as standalone planners who are as close as possible to the original SOGBOFA planner and its conformant improvement [5]. Based on this, we then created two heuristics in THTS following the underlying concepts of SOGBOFA. The first propagation heuristic abuses a very fast and minimalistic evaluation of the Q value function graph with a simple forward propagation of the current state to profit from a big part of the contained information while retaining a very small calculation time. The second conformant heuristic pushes for more information from the Q value function graph by optimizing the representation of future actions in the graph making it a more accurate approximation of the actual Q value function. This is done following a modified conformant SOGBOFA procedure with gradient ascent steps towards future actions promising better rewards. Hence, the increased guidance comes at the cost of a notably harder calculation.

6.1 Results

Our practical evaluation of the standalone planners generally confirmed the trends of the original SOGBOFA, showing that the procedure works and can be adapted to the THTS framework. While we were unable to reproduce state-of-the-art performance with the planners by beating IPC2014, we demonstrated that the SOGBOFA procedure can lead to better results on some domains. We also saw that a conformant approach can indeed lead to a better performance through the increased guidance, but this was not the case here due to limitations of the implementation. Furthermore, we suggested a generalized way of handling

constraints on actions in the form of action preconditions by including them in the SOGBOFA Q value function graph. The results showed this approach in its current form to be promising on domains with many action preconditions, but not worth it in general.
For the main evaluation of the heuristics based on SOGBOFA components, we showed that the search depth is important to balance guidance and speed for the heuristics. This is because the guidance gained from increasing the search depth can be essential at low values for recognizing key connections in problems, but can also yield diminishing returns at higher values if not much new information is gained; while the calculation time is always increasing. Using a well suited search depth, we managed to outperform the state-of-the-art benchmark of the Prost configuration IPC2014 with our propagation heuristic by a reasonable margin. Investigating the reasons for this success, we showed that it has a worse heuristic guidance than the Iterative Deepening Search used by the IPC2014 configuration overall, but is very strong in some domains. It is, however, much more efficient to compute, and strikes a very strong and balanced combination of efficiency and guidance. Excellent results in one aspect can compensate mediocre results in the other aspect: this happens here in both directions. The conformant heuristic, on the other hand, encountered performance problems, at least partially due to an unoptimized method of gradient calculation. This meant that, while theoretically yielding a better guidance, the insufficient number of trials prevents it from scoring good results in practice.

6.2 Future Work

We have shown that it is possible to achieve state-of-the-art heuristic performance through the use of the SOGBOFA concepts. The main failing of the conformant heuristic was the slow calculation, for which we already discussed potential ways for improvement: We have seen the theoretically better suited reverse mode calculation of gradients for scalar valued functions. This would likely lead to better results overall with a significant improvement on domains with large action states. The other improvement would be to save time by reusing the Q value function graph and simply changing the necessary values, as the structure remains exactly the same over multiple gradient steps. As both these improvements can not be done with the autodiff library used, a natural way to improve the results further would be find a way to solve these issues, ideally by implementing a custom version of automatic differentiation. This could allow for an even more efficient implementation by integrating the Prost formula evaluation calls.
While these fixes are not new work itself, they would allow testing of many interesting setups that are currently not possible to realize in practice. With a faster calculation, it would be possible to use and realistically test configurations of the conformant heuristic giving a much better guidance than the ones we had to limit ourselves to. It would also benefit the standalone planners, who would profit by being able to do more random restarts. Calculating the gradients for the conformant actions would also mean no time increase with reverse mode, which would massively boost the performance of the conformant verstion of the standalone compared to the non-conformant one.
We have also seen that the standalone planners have a lot of components where the current

solutions work reasonably well. But it is entirely possible that with a bigger focus on them, better solutions could be found. For example, further investigation of the outlined generalized handling of action constraints could likely increase its general viability.
On the other hand, there are also possibilities for interesting future work in our successes: The Q value function graph of SOGBOFA has proven to be very successful when applied to the traditional THTS framework of planners, even when isolated from the gradient ascent. While it was not optimal on every domain, it proved its worth through strong performance on several domains and a very efficient calculation. As this was a simple and very direct way to use it, there are probably other, more sophisticated ways to successfully integrate it into a traditional procedure.

Bibliography

[1] Craig Boutilier, Thomas Dean, and Steve Hanks. Planning under uncertainty: Structural assumptions and computational leverage. In *Proceedings of the Second European Workshop on Planning*, pages 157–171. IOS Press, 1996.

[2] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[3] Hao Cui. *Aggregate Simulation for Planning and Inference*. PhD thesis, Tufts University, Medford and Somerville, MA, USA, 2019.

[4] Hao Cui and Roni Khardon. Online symbolic gradient-based optimization for factored action MDPs. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 3075–3081, 2016.

[5] Hao Cui, Thomas Keller, and Roni Khardon. Stochastic planning with lifted symbolic trajectory optimization. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling*, ICAPS 2018, pages 119–127, 2019.

[6] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2008.

[7] Thomas Keller. *Anytime optimal MDP planning with trial-based heuristic tree search*. PhD thesis, University of Freiburg, Freiburg im Breisgau, Germany, 2015.

[8] Thomas Keller and Patrick Eyerich. PROST: probabilistic planning based on UCT. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*, ICAPS 2012, pages 119–127, 2012.

[9] Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling*, ICAPS 2013, pages 135–143, 2013.

[10] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006, 17th European Conference on Machine Learning*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2006.

[11] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1980.

[12] Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf, 2010.

[13] Scott Sanner, Sungwook Yoon, and Thomas Walsh. IPC2011-Probabilistic. http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/, 2011. [Online; accessed 29-April-2020].

[14] Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2012.

[15] Gerald Tesauro and Gregory R. Galperin. On-line policy improvement using monte-carlo search. In *NIPS*, 1996.

[16] Thomas Keller, Scott Sanner, and Buser Say. IPC2018-Probabilistic. https://ipc2018-probabilistic.bitbucket.io/, 2018. [Online; accessed 29-April-2020].

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

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

Author — Autor
Ferdinand Badenberg

Matriculation number — Matrikelnummer
2014-055-206

Title of work — Titel der Arbeit
SOGBOFA as heuristic guidance for THTS

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, 29.4.2020

Signature — Unterschrift