On the Complexity of Heuristic Synthesis for Satisficing Classical Planning: Potential Heuristics and Beyond

Malte Helmert, Silvan Sievers, Alexander Rovner, Augusto B. Corrêa

University of Basel, Switzerland
{malte.helmert,silvan.sievers,augusto.blaascorrea}@unibas.ch, alex_rovner@hotmail.de

Abstract

Potential functions are a general class of heuristics for classical planning. For satisficing planning, previous work suggested the use of descending and dead-end avoiding (DDA) potential heuristics, which solve planning tasks by backtrack-free search. In this work we study the complexity of devising DDA potential heuristics for classical planning tasks. We show that verifying or synthesizing DDA potential heuristics is **PSPACE**-complete, but suitable modifications of the DDA properties reduce the complexity of these problems to the first and second level of the polynomial hierarchy. We also discuss the implications of our results for other forms of heuristic synthesis in classical planning.

Introduction

In a classical planning task, we want to find a sequence of actions, called a plan, transforming an initial world state to a state where a specific goal is satisfied. One common way to solve classical planning tasks is to use heuristic search (Bonet and Geffner 2001). Potential functions (Pommerening et al. 2015) encode heuristics or other functions in a flexible way as a linear combination of state features. To evaluate the function on a state, we sum the weights of all features that are true in the state.

This simple form makes potential functions a popular choice for many different scenarios. For example, in the context of classical planning, they have been used to synthesize admissible heuristics (Pommerening et al. 2015; Seipp, Pommerening, and Helmert 2015) or to detect unsolvable states (Seipp et al. 2016b). Beyond the classical case, they have been used to compute multi-agent planning heuristics (Štolba, Fišer, and Komenda 2016) and to represent heuristics for generalized planning (Francès et al. 2019).

In this paper, we study the computational complexity of potential heuristic synthesis for satisficing planning, complementing existing results for optimal planning (Pommerening et al. 2015; Pommerening, Helmert, and Bonet 2017). Specifically, we consider the problem of *verifying* whether a potential heuristic is *descending and dead-end-avoiding (DDA)* and the decision problem analog of *synthesizing* a DDA potential heuristic. We show that both problems are **PSPACE**-complete and thus as hard as planning.

The main source of hardness is related to testing whether a state is *alive*, i.e., reachable and not a dead end, which is a **PSPACE**-complete problem. We therefore look at variants of DDA verification and synthesis without aliveness tests and show completeness results for the first two levels of the polynomial hierarchy.

The core arguments in our membership proofs do not rely on the specifics of potential functions. They merely require that a potential heuristic is represented in polynomial space and can be evaluated in polynomial time. Therefore, we can generalize these results to arbitrary heuristics with these properties, with implications for heuristic synthesis for satisficing planning in general.

Background

We use the SAS$^+$ planning formalism (Bäckström and Nebel 1995). A SAS$^+$ planning task (or *task* for short) $\Pi = \langle \mathcal{V}, \mathcal{A}, I, G \rangle$ has the following components. $\mathcal{V}$ is a finite set of *state variables* v, each with a finite-domain $\mathit{dom}(v)$. An *atom* of Π is a pair $\langle v, d \rangle$, also written $v \mapsto d$, with $v \in \mathcal{V}$ and $d \in \mathit{dom}(v)$. A *partial state* s of Π is a set of atoms of Π in which no two atoms refer to the same state variable. We write $\mathit{vars}(s)$ for the set of state variables used in s. A partial state with $\mathit{vars}(s) = \mathcal{V}$ is a *state*. We interchangeably treat partial states as sets of atoms or functions from variables to values, writing $s[v] = d$ to denote $(v \mapsto d) \in s$. $\mathcal{A}$ is a finite set of *actions* a, each with a *precondition* $\mathit{pre}(a)$ and *effect* $\mathit{eff}(a)$, both of which are partial states, and a numerical *cost* value $\mathit{cost}(a) \in \mathbb{R}_0^+$. If $s' \subseteq s$ for partial state s' and state s, we also write this as $s \models s'$. Finally, I is a state called the *initial state*, and G is a partial state called the *goal*.

The *representation size* $\|\Pi\|$ of a task is the length of a reasonable compact encoding of Π. For example, we can set $\|\Pi\|$ to the total number of atoms plus the total number of atom occurrences in all actions. As usual in complexity theory (e.g., Garey and Johnson 1979), our results do not depend on details of the encoding. Any polynomially equivalent encoding works.

The semantics of Π is defined as follows. Action a is *applicable* in state s if $s \models \mathit{pre}(a)$. If a is applicable in s, its application results in the *successor state* $s[\![a]\!]$, defined as $s[\![a]\!][v] = \mathit{eff}(a)[v]$ if $v \in \mathit{vars}(\mathit{eff}(a))$ and $s[\![a]\!][v] = s[v]$ otherwise. If a is not applicable in s, $s[\![a]\!]$ is undefined. We

write $succ(s)$ for the set of successor states of s. Applicability and resulting states are extended to finite action sequences π in the natural way. The *cost* of $\pi = \langle a_1, \ldots, a_n \rangle$ is defined as $cost(a) = \sum_{i=1}^{n} cost(a_i)$.

A *goal state* of Π is a state s with $s \models G$. A *plan* for a state s of Π is an action sequence π that takes s to a goal state: $s[\![\pi]\!] \models G$. A plan for I is called a plan for Π.

A state or task is *solvable* if there exists a plan for it. Otherwise it is called *unsolvable*. A state s is *reachable* if there exists an action sequence π with $I[\![\pi]\!] = s$. A state is *alive* if it is reachable and solvable. A state is *dead* if it is not alive.

A plan for state s is *optimal* if it has minimum cost among all plans for s. The cost of such an optimal plan is denoted by $h^*(s)$, with $h^*(s) = \infty$ if s is unsolvable. *Satisficing planning* is the problem of finding plans for a given task or proving that the task is unsolvable. (Plans of lower cost are considered better solutions in satisficing planning, but this aspect is not important for this paper.) In *optimal planning*, only optimal plans are acceptable solutions.

Heuristic Search

Heuristic search is a common approach for both satisficing and optimal planning. A *heuristic* for a task Π with states S is a function $h : S \to \mathbb{R} \cup \{\infty\}$ whose purpose is to provide an estimate of the distance from a given state to the nearest goal state. In optimal planning, heuristics usually aim to estimate the optimal plan cost $h^*(s)$, and *admissibility* and *consistency* are desirable properties of heuristics in this context (Dechter and Pearl 1985). In satisficing planning, the role of plan costs is deemphasized, and many heuristics designed for satisficing planning ignore action costs altogether (e.g., Richter and Westphal 2010; Wilt and Ruml 2014).

Potential Functions and Potential Heuristics

A *potential function* for a task Π with states S is a function $h^{\text{pot}} : S \to \mathbb{R} \cup \{\infty\}$ that is defined in terms of a set of (state) *features* $\mathcal{F}$ and *weights* for these features. A feature $F \in \mathcal{F}$ is a partial state and can be viewed as a test that all atoms in this partial state are true in a given state s ($s \models F$). If this is the case, we say the feature is *present* in s. We use Iverson brackets $[s \models F]$ to denote the value 1 if $s \models F$ (F is present in s) and 0 otherwise. Feature weights are given by a weight function $w : \mathcal{F} \to \mathbb{R} \cup \{\infty\}$. Potential functions add the weights of all features present in a given state:

$$h^{\text{pot}}(s) = \sum_{F \in \mathcal{F}} w(F)[s \models F].$$

We can think of potential functions as linear combinations of feature indicator functions. Potential functions used as heuristics are called *potential heuristics*. Our definition follows Pommerening et al. (2015) except that we permit features with weight ∞. The *dimension* of a potential function with features $\mathcal{F}$ is the size of the largest feature in $\mathcal{F}$, i.e., $\max\{|F| \mid F \in \mathcal{F}\}$.

The *representation size* $\|h^{\text{pot}}\|$ of a potential function denotes the length of an encoding of h^{pot}. For example, for features $\mathcal{F}$ and weights w, one way to define the representation size is $\|h^{\text{pot}}\| = \sum_{F \in \mathcal{F}} (|F| + \lceil \log_2(|w(F)| + 1) \rceil)$, bearing in mind that polynomial differences in representation size do not matter for the complexity results in this paper. In this definition, we assume an explicit representation of h^{pot} as a list of features and their associated weights. This means that every feature F contributes its size $|F|$ to the representation size, and every feature weight contributes its number of bits in a binary representation to the representation size. We assume in the following that numbers are represented as arbitrary-precision integers, but our results apply equally to arbitrary-precision rational numbers and floating-point representations.[1]

In principle, potential functions can represent arbitrary functions $f : S \to \mathbb{R} \cup \{\infty\}$: we can treat each state s as a feature and set its weight to $f(s)$. However, this requires exponentially many features and hence an exponential representation size for the potential function. For practical purposes, we are therefore interested in more restricted potential functions, for example with bounded dimension.

Complexity Theory

We assume familiarity with the complexity classes **NP**, **coNP**, and **PSPACE**, with polynomial reductions between decision problems, and with the notions of hard and complete problems for a complexity class. We also assume familiarity with deterministic Turing machines (DTMs), oracles, and the characterization **PSPACE** = **NPSPACE**. Furthermore, we present complexity results for the first and second levels of the *polynomial hierarchy*, which defines complexity classes $\mathbf{\Sigma}_i^{\text{p}}$ and $\mathbf{\Pi}_i^{\text{p}}$ that generalize the complexity classes **NP** and **coNP**. We only need the $\mathbf{\Sigma}_i^{\text{p}}$ classes in this paper, which can be defined based on oracles as

$$\mathbf{\Sigma}_1^{\text{p}} = \mathbf{NP} \text{ and } \mathbf{\Sigma}_{i+1}^{\text{p}} = \mathbf{NP}^{\mathbf{\Sigma}_i^{\text{p}}} \text{ for all } i \geq 1.$$

In other words, $\mathbf{\Sigma}_{i+1}^{\text{p}}$ is the class of decision problems that can be decided by nondeterministic polynomial-time algorithms with access to an oracle for problems in $\mathbf{\Sigma}_i^{\text{p}}$. Note that $\mathbf{\Sigma}_2^{\text{p}} = \mathbf{NP}^{\mathbf{NP}}$. These complexity classes form a hierarchy between **P** and **PSPACE** in the sense that $\mathbf{P} \subseteq \mathbf{NP} = \mathbf{\Sigma}_1^{\text{p}} \subseteq \mathbf{\Sigma}_2^{\text{p}} \subseteq \cdots \subseteq \mathbf{PSPACE}$. We refer to Garey and Johnson (1979) and Arora and Barak (2009) for more information on these topics.

The decision problem most commonly associated with satisficing planning is PLANEXISTENCE: is a given task solvable or not? PLANEXISTENCE is **PSPACE**-complete (Bylander 1994).

Potential Heuristics for Satisficing Planning

Potential function synthesis is the problem of computing suitable features and weights for a given task such that the resulting potential function h^{pot} has desirable characteristics. So far, theoretical results for this problem are largely limited to optimal planning. In the work that introduced potential heuristics, Pommerening et al. (2015) describe a

[1] Arbitrary-precision floating-point representations require care: sums of numbers must be kept as sums rather than being reduced to a common exponent and combined, as reducing to a common exponent would be an exponential-time operation in general.

polynomial-time algorithm based on linear programming for synthesizing admissible and consistent potential heuristics of dimension 1 that optimize an arbitrary linear objective. For example, their algorithm can find the maximum possible heuristic value that an admissible and consistent potential heuristic of dimension 1 can achieve in a given state.

Pommerening, Helmert, and Bonet (2017) generalize this result to dimension 2. They also prove that testing whether a given potential heuristic of dimension 3 is consistent is **coNP**-complete, suggesting that efficient synthesis beyond dimension 2 is not possible if $\mathbf{P} \neq \mathbf{NP}$.

For satisficing planning, admissibility and consistency of heuristics are not usually considered important. Hoffmann (2005) emphasizes the role of local search topology for the efficiency of heuristic search for satisficing planning. Wilt and Ruml (2016) provide a measure of heuristic quality for greedy search algorithm based on qualitative misclassifications of pairs of states: if $h^*(s) < h^*(s')$, then a heuristic h for greedy search should aim to preserve this order, i.e., satisfy $h(s) < h(s')$, but the exact numerical values do not matter.

Both of these works focus on *how far away* a given heuristic is from making greedy search algorithms backtrack-free. Seipp et al. (2016a) investigate *how complicated* a potential heuristic must be in order to make greedy search algorithms backtrack-free. They define the *correlation complexity* of a planning task, which is (roughly speaking) the minimum dimension d for which a potential heuristic that makes greedy search backtrack-free exists. They show that correlation complexity is upper bounded by 2 in all tasks of several nontrivial IPC benchmark domains and see this as evidence that potential heuristics do not need to be complicated or large for efficient satisficing planning in these domains.

In the rest of this paper we study the computational complexity of synthesizing potential heuristics that give rise to backtrack-free search for satisficing planning.

Verification and Synthesis

Throughout the paper, we study two kinds of problems: in *verification*, we ask if a given heuristic for a given task has a certain property, such as the DDA property mentioned in the introduction. This is a decision problem by nature, i.e., it has a yes/no answer.

Synthesis is about *finding* a heuristic for a given task with a certain property and is not naturally a decision problem. As usual in complexity theory, we study a decision problem analog of synthesis, asking whether a heuristic with the given property for a given task exists. Except for one trivial case, all our algorithms for this decision problem are constructive, and hence our results apply equally to the full (non-decision problem) version of synthesis.

We parameterize verification and synthesis along two dimensions. Firstly, we consider different *properties* $\mathcal{P}$, where a property is a predicate that is either true or false for a given potential function for a given planning task. For example, a potential function may or may not be DDA for a given planning task.

Secondly, we consider different *restricted classes* of potential functions, for example to show that certain hardness results already hold for potential functions of low dimension. Mathematically, a *potential heuristic class* or *PH class* is represented by a function $\mathcal{C}$ that maps any given planning task Π to a family $\mathcal{C}(\Pi)$ of potential functions to be considered when verifying or synthesizing a potential function for Π. For example, below we introduce $\mathcal{C}^1$ as the class of all potential heuristics of dimension 1, which means that $\mathcal{C}^1(\Pi)$ consists of all such potential heuristics for the specific task Π. To avoid complexity-theoretic oddities, we only allow PH classes $\mathcal{C}$ for which we can test $h^{\text{pot}} \in \mathcal{C}(\Pi)$ in polynomial time in $\|\Pi\|$ and $\|h^{\text{pot}}\|$.

We can now define verification and synthesis.

Definition 1. VERIFICATION$(\mathcal{P}, \mathcal{C})$ *is the following decision problem: given a planning task* Π *and potential heuristic* $h^{\text{pot}} \in \mathcal{C}(\Pi)$*, does* h^{pot} *have property* $\mathcal{P}$ *for* Π*?*

Definition 2. SYNTHESIS$(\mathcal{P}, \mathcal{C})$ *is the following decision problem: given a planning task* Π*, does there exist a potential heuristic* $h^{\text{pot}} \in \mathcal{C}(\Pi)$ *that has property* $\mathcal{P}$ *for* Π*?*

A common requirement for heuristic synthesis is that the synthesized heuristics are efficiently computable. This is usually formalized by requiring that the evaluation time for a given state is bounded by a polynomial in $\|\Pi\|$. For potential functions, this is equivalent to considering a class $\mathcal{C}$ for which there exists a polynomial p with $\|h^{\text{pot}}\| \leq p(\|\Pi\|)$ for all tasks Π and $h^{\text{pot}} \in \mathcal{C}(\Pi)$. We call such classes *compact*. A PH class is compact iff the number of features and the number of digits of each feature weight are polynomially bounded by $\|\Pi\|$.

The most commonly considered restriction of potential functions h^{pot} is to bound their dimension by a constant $k \in \mathbb{N}_0$. This automatically bounds the number of features of h^{pot} by $O(n^k)$, where n is the number of atoms of Π.

Definition 3. *For* $k \in \mathbb{N}_0 \cup \{\infty\}$*,* $\mathcal{C}^k$ *denotes the PH class that allows all potential heuristics of dimension at most* k*. (In particular,* $\mathcal{C}^\infty$ *imposes no restrictions.)*

Synthesizing DDA Potential Heuristics

Ideally, one would prefer a state space topology where the only local minima are at goal states, i.e., every non-goal state has a successor with a lower heuristic value. This implies that greedy search algorithms like greedy best-first search or simple hill-climbing directly reach the goal without backtracking. However, such a state space topology can only exist in state spaces where every state is solvable because hill-climbing from an unsolvable state must necessarily end up in a local minimum without reaching a goal state.

To address this issue, Seipp et al. (2016a) define their desirable heuristic properties as follows: a heuristic is *descending* if every *alive* non-goal state has a successor with a lower heuristic value and it is *dead-end-avoiding* if *dead* successors of alive states s never have a lower heuristic value than s. A *DDA* heuristic is descending and dead-end-avoiding. This is sufficient for greedy best-first search or simple hill-climbing to solve a solvable task without backtracking.

Definition 4. *Let* h *be a heuristic for a planning task* Π *with alive states* S_{A} *and goal states* S_{G}*. We say that* h *is* DDA *for*

Algorithm 1: VERIFICATION(DDA/SDDA, $\mathcal{C}$) decided in polynomial space.

Input: Planning task Π, heuristic h for Π
Output: Accept if h is (S)DDA for Π, reject otherwise

```
1: if not IsAlive(Π, GetInitialState(Π)) then
2:     for the DDA property: accept
3:     for the SDDA property: reject
4: end if
5: for s ∈ GenerateStates(Π) do
6:     if IsAlive(Π, s) and not IsGoalState(Π, s) then
7:         improving_succ ← false
8:         for t ∈ succ(s) do
9:             if IsAlive(Π, t) and h(t) < h(s) then
10:                improving_succ ← true
11:            else if not IsAlive(Π, t) and h(t) < h(s) then
12:                reject
13:            end if
14:        end for
15:        if not improving_succ then
16:            reject
17:        end if
18:    end if
19: end for
20: accept
```

Π *if it satisfies the following two conditions:*

$$\forall s \in (S_A \setminus S_G)\ \exists t \in succ(s) : h(t) < h(s) \quad (1)$$

$$\forall s \in (S_A \setminus S_G)\ \forall t \in (succ(s) \setminus S_A) : h(t) \geq h(s) \quad (2)$$

A subtlety of this definition of DDA is that it only imposes restrictions on h for solvable planning tasks. For unsolvable tasks, there exist no alive states, and as a consequence *every* heuristic is DDA. For example, a constant heuristic that gets stuck immediately without an improving successor nevertheless counts as descending and dead-end-avoiding on unsolvable tasks.

We find this aspect of the definition somewhat unfortunate, as it means that the DDA property combines two disparate properties: being solvable *via local search* and being unsolvable *for any reason*.[2] We nevertheless begin our analysis with the original DDA property, but we will see that the associated complexity results do not provide much insight into the intuitive problem of synthesis. This will be rectified in the following sections, where we consider variants of DDA.

Theorem 1. VERIFICATION(DDA, $\mathcal{C}$) $\in$ **PSPACE** *for all PH classes* $\mathcal{C}$.

Proof. Algorithm 1 decides the verification problem in polynomial space, showing **PSPACE** membership. (Ignore the SDDA property, which is discussed in the following section.) It implements the definition of DDA heuristics in a straightforward way. Lines 1–4 accept (return "yes") for unsolvable tasks (equivalently: tasks where the initial state is not alive), in line with the above discussion. These lines are not strictly necessary, but make the algorithm reusable for the following section. Lines 5–19 verify Eq. 1 and 2.

The algorithm only stores two states and a Boolean variable and only calls functions that can be implemented in polynomial space, so it can itself be implemented in polynomial space. In particular, h is a potential heuristic and can thus be evaluated in polynomial space, and testing whether a state is alive is in **PSPACE**. (A state s is alive if there exists a plan from the initial state to s and a plan from s to the goal, which can be tested with two PLANEXISTENCE queries.)

Note that for this problem we do not need to restrict ourselves to compact PH classes because h is part of the input and hence $\|h\|$ is part of the input size. Every potential function can be computed in polynomial time in its representation size. □

We now turn to hardness results. Verifying the DDA property is already hard in perhaps the most trivial case, potential heuristics of dimension 0. In this restricted class, only the empty feature is allowed, and hence all such potential heuristics are constant functions.

Theorem 2. VERIFICATION(DDA, $\mathcal{C}^k$) *is* **PSPACE***-complete for all* $k \in \mathbb{N}_0 \cup \{\infty\}$.

Proof. Membership in **PSPACE** was shown in the previous theorem.

For hardness, we reduce from plan nonexistence, i.e., the question whether a given planning task is *not* solvable. Because **PSPACE** is closed under complement, this problem is **PSPACE**-complete like PLANEXISTENCE. W.l.o.g., we assume that the initial state of the given planning task Π is not a goal state. Every planning task can easily be transformed to satisfy this requirement without affecting its solvability.

For a given planning task Π, we map to the verification question with task Π and potential heuristic $h = 0$, i.e., we claim that Π is *unsolvable* iff the constant-0 heuristic is DDA for Π. This is easy to see: if Π is unsolvable, then every heuristic is DDA, including h. If Π is not unsolvable, it is solvable. It follows that its initial state is alive and not a goal state. Because it does not have an improving successor, h is not DDA, concluding the proof. □

Next we consider synthesis. While intuitively SYNTHESIS appears harder than VERIFICATION, the following result shows us that this is not always the case.

Theorem 3. SYNTHESIS(DDA, $\mathcal{C}^\infty$) $\in$ **P**. *More precisely, it can be decided in constant time.*

Proof. The question we need to decide is whether *any* DDA potential heuristic exists for a given task Π. The answer is always "yes", and hence the problem can be solved in constant time.

If Π is unsolvable, every potential heuristic is DDA, so the answer must be "yes". If Π is solvable, consider the function that maps each state s to the number of steps needed to reach a goal state from s (∞ for unsolvable states). This heuristic is clearly DDA, and without restrictions, potential functions

[2] The authors that introduced the DDA property (Seipp et al. 2016a) agree with this assessment in personal communications. It should be pointed out that the focus of their work is exclusively on solvable tasks, in which case no problems arise.

can represent arbitrary functions including this one. So in this case the answer is also "yes". □

The lesson we can draw from this trivial result is that for synthesis, we must be careful regarding the classes of heuristics we consider. In the following, we therefore focus on *compact* PH classes for synthesis where necessary. Like for verification, we first show membership in **PSPACE**.

Theorem 4. SYNTHESIS$(\text{DDA}, \mathcal{C}) \in$ **PSPACE** *for all compact PH classes* $\mathcal{C}$.

Proof. Guess a potential heuristic for the given task and apply Theorem 2 to verify in polynomial space that it is DDA.

Because **PSPACE** $=$ **NPSPACE**, we can use guesses. Because $\mathcal{C}$ is compact, allowed heuristics have polynomial size and hence we can make the guess in polynomial time. □

With suitable restrictions in place, we can again show that synthesis is hard even under very severe restrictions.

Theorem 5. SYNTHESIS$(\text{DDA}, \mathcal{C}^0)$ *is* **PSPACE**-*complete.*

Proof. In $\mathcal{C}^0$, the *only* potential heuristics available are constant functions, and hence there exists a DDA heuristic in $\mathcal{C}^0$ iff a constant function is DDA. Since all constant functions are equivalent regarding the DDA property, there is no meaningful difference to VERIFICATION$(\text{DDA}, \mathcal{C}^0)$, shown **PSPACE**-complete in Theorem 2. □

In all hardness results so far, the difficulty exploited in the proofs is not really related to the concept of synthesizing a backtrack-free heuristic. Rather, we obtain hardness because VERIFICATION and SYNTHESIS are required to recognize unsolvable tasks and treat them specially, and it is thus only natural that they are as hard as testing unsolvability.

The results we have seen can be extended to more general PH classes. For example, SYNTHESIS$(\text{DDA}, \mathcal{C}^1)$ and SYNTHESIS$(\text{DDA}, \mathcal{C}^2)$ are also **PSPACE**-hard, with or without restrictions to compact PH classes. The proof idea is a reduction from plan nonexistence: for a given planning task Π, we construct a planning task Π' with two disconnected parts that must both be solved, one consisting of Π and one consisting of a fixed solvable task Π^1 that is known *not* to have DDA heuristics in $\mathcal{C}^1$ (analogously for $\mathcal{C}^2$). We can then prove that Π' has a DDA heuristic in $\mathcal{C}^1$ iff it is unsolvable: if it is unsolvable, it trivially has a DDA heuristic, and it if is solvable, it cannot have a DDA heuristic in $\mathcal{C}^1$ because this would imply that Π^1 also has a DDA heuristic in $\mathcal{C}^1$, which we know not to be the case.

Synthesizing SDDA Potential Heuristics

The DDA property studied in the previous section can be summarized as "Is the task either unsolvable or can a solution be found without backtracking?". The disjunctiveness of this question affects all complexity results, and hence we did not gain much insight on the practical problem of synthesis. Therefore, we now move to the non-disjunctive property "Can a solution be found without backtracking?". This requires considering a modified DDA property that ensures that the answer for unsolvable tasks is "no" rather than "yes". We call this the *solvable DDA* (SDDA) property.

Definition 5. *A heuristic* h *for a planning task* Π *is SDDA if* Π *is solvable and* h *is DDA for* Π.

We begin with bad news. Similar to DDA, verifying SDDA is already **PSPACE**-complete in very restricted cases. However, this time the hardness argument is much less trivial, and the reduction actually needs to look inside solvable tasks and consider state space topology.

Theorem 6. VERIFICATION$(\text{SDDA}, \mathcal{C}^k)$ *is polynomial for* $k = 0$ *and* **PSPACE**-*complete for all* $k \in \mathbb{N}_1 \cup \{\infty\}$.

The hardness result already holds for potential heuristics of dimension 1 and non-branching *tasks, i.e., tasks where every reachable state has at most one successor.*

Proof. For the $k = 0$ result, note that potential heuristics of dimension 0 are constant. Constant heuristics are SDDA iff the initial state is a goal state. This is easily checked in polynomial time. In the following, assume $k \geq 1$.

PSPACE membership is established by Algorithm 1 as in Theorem 1. Note the DDA/SDDA case distinction in the algorithm.

For **PSPACE**-hardness, we polynomially reduce from the problem of determining whether a DTM M with polynomial space bound p halts on a given input of length n.

We first construct a planning task Π that mimics the Turing machine computation. The details of this conversion do not matter. What is important is that Π can be constructed in polynomial time, that there is a 1:1 correspondence between the initial and goal configurations of M and the initial and goal states of Π and that computation steps of M are in 1:1 correspondence to action applications in Π in all reachable configurations/states. See Theorem 3.1 by Bylander (1994) for an example of such a construction.[3]

We augment Π with a binary counter as follows. Let N be a number at least as large as the number of DTM configurations of M. We add $K = \lceil \log(N+1) \rceil$ state variables $C_0, \ldots, C_{K-1}$ with domain $\{0, 1\}$ to Π, which we interpret as representing the K-digit binary number $C_{K-1} \ldots C_0$. In the initial state, the values of the C_i form a binary representation of N. Note that N grows at most exponentially in $\|M\|$ and n, and hence K is polynomial in the input size.

We now modify Π so that every simulated DTM transition also decrements the number encoded by the C_i variables. This requires replacing every action of Π with K different versions, depending on how many digits carry in the decrement. For example, for the case of two carries we add the preconditions $\{C_2 \mapsto 1, C_1 \mapsto 0, C_0 \mapsto 0\}$ and effects $\{C_2 \mapsto 0, C_1 \mapsto 1, C_0 \mapsto 1\}$. The counter cannot decrement below 0, but this is not necessary: an accepting computation for M cannot include more than N computation steps (or else it would repeat a configuration and loop forever). Therefore, after this transformation it is still the case that M halts on its input iff the planning task has a solution.

It remains to define a potential heuristic h^{pot} for Π. We define h^{pot} in such a way that $h^{\text{pot}}(s)$ is the value of the counter

[3] In Bylander's reduction, a single DTM transition actually corresponds to a sequence of three action applications because he wants to convert to a planning task of a particularly simple form. But it is easy to combine these three actions into a single action.

represented by the C_i variables. This can be done with a 1-dimensional potential heuristic assigning the weight 2^i to the feature $\{C_i \mapsto 1\}$ for all $0 \leq i < K$.

We have to show that h^{pot} is SDDA for Π iff M halts, or equivalently, iff Π is solvable. If Π is unsolvable, no heuristic can be SDDA, so this case is easy. If Π is solvable, the alive states are exactly the states that simulate the DTM computation. Apart from the goal state at the end of the sequence, each of these states has exactly one successor, in which the counter (= heuristic value) is decremented by 1, and hence h^{pot} is indeed SDDA. □

It is now easy to extend this result to synthesis.

Theorem 7. SYNTHESIS(SDDA, $\mathcal{C}$) $\in$ **PSPACE** *for all compact PH classes* $\mathcal{C}$.

SYNTHESIS(SDDA, $\mathcal{C}^k$) *is polynomial for* $k = 0$ *and* **PSPACE***-hard for all* $k \in \mathbb{N}_1 \cup \{\infty\}$.

SYNTHESIS(SDDA, $\mathcal{C}$) *is* **PSPACE***-complete for all compact PH classes* $\mathcal{C}$ *that permit 1-dimensional features and singly-exponential feature weights. The hardness results already hold for non-branching tasks.*

Proof. The membership argument is the same as for DDA (Theorem 4). For the $k = 0$ case, using the same argument as in the proof of Theorem 5, there is no difference between verification and synthesis, and hence the result follows from Theorem 6.

For **PSPACE**-hardness, we use the same reduction as in Theorem 6, but in this case generating only the planning task and not the potential function. If the given DTM does not halt on its input, the planning task is unsolvable and hence does not have an SDDA heuristic. If it halts, it does have an SDDA heuristic, in particular the 1-dimensional SDDA heuristic constructed in the proof of Theorem 6, which is permitted in the PH classes considered for the hardness results of this theorem. The completeness result follows from the membership and hardness results. □

Theorems 6 and 7 show that verifying or synthesizing SDDA potential heuristics is still as hard as planning itself. Even worse, it is already hard in the very restricted setting of 1-dimensional heuristics and planning tasks that involve no actual choice. These results are in contrast to the tractability results in the 1- and 2-dimensional case for admissible and consistent potential heuristics by Pommerening et al. (2015; 2017).

Fortunately, this is not the final word on synthesizing potential heuristics for satisficing planning. In the following, we consider further variants of the DDA property that retain its useful aspects while having more favorable complexity.

DDA Variants Without Aliveness Tests

Taking a closer look at the definition of the *descending* (Eq. 1) and *dead-end-avoiding* (Eq. 2) properties and the proofs in the preceding section, it becomes apparent that deciding whether a state is alive is the main culprit for the **PSPACE**-completeness of SDDA verification and synthesis. In this section, we consider three further variants of the DDA property that do not depend on aliveness. All variants retain the key characteristic of DDA and SDDA: for any solvable task, greedy search algorithms using a heuristic satisfying any of the variant properties are led towards a goal state without encountering local minima and hence without backtracking.

Our first and simplest variant removes the alive/dead distinction in the DDA definition altogether, treating every state as if it were alive. If we replace S_{A} by the set of all states S in Eq. 1–2, then Eq. 2 becomes tautological, leaving only a simplified variant of Eq. 1. We call a heuristic with this property *unrestricted DDA (UDDA)* because the descending property is no longer restricted to alive states.

Definition 6. *A heuristic* h *for a planning task* Π *is UDDA if it satisfies the* unrestricted descending *property:*

$$\forall s \in (S \setminus S_{\text{G}})\ \exists t \in \mathit{succ}(s) : h(t) < h(s) \tag{3}$$

UDDA is a simpler property than DDA because it does not require checking aliveness, but less powerful because tasks with any unsolvable states cannot have a UDDA heuristic. As a consequence, unlike SDDA, we do not need to require the task to be solvable: unrestricted descending already implies solvability.

Not being applicable to any tasks with unsolvable states is a severe limitation. We therefore consider two further variants of DDA, which do not get rid of the aliveness tests in DDA/SDDA altogether. Instead, starting from SDDA, they replace the "omniscient" tests whether a state is dead or unsolvable with a decision *by the heuristic* that a state should be pruned, i.e., that it has an infinite heuristic estimate.

In the *infinity-based pruning DDA (*∞*DDA)* property, the set of alive states S_{A} in the definition of DDA is replaced by the set of states S_{fin} with finite heuristic value, and the requirement that the task is solvable in the definition of SDDA is replaced by the test $I \in S_{\text{fin}}$. Note that with this change, as in UDDA, Eq. 2 becomes tautological and can be dropped. This leads to the following definition.

Definition 7. *Let* h *be a heuristic for a planning task* Π *with finite-heuristic states* S_{fin} *under* h*, initial state* I *and goal states* S_{G}*. We say that* h *is* ∞DDA *for* Π *if it satisfies the following two conditions:*

$$\forall s \in (S_{\text{fin}} \setminus S_{\text{G}})\ \exists t \in \mathit{succ}(s) : h(t) < h(s) \tag{4}$$

$$I \in S_{\text{fin}} \tag{5}$$

The final and most general variant follows the same idea as ∞DDA of "pruning" states with infinite heuristic value, but uses a second potential function to determine which states receive an infinite heuristic value. This was first suggested by Corrêa and Pommerening (2019) for the problem of representing the perfect heuristic h^* as a potential function. They define heuristics h based on two potential functions h_1^{pot} and h_2^{pot} as follows:

$$h(s) = \begin{cases} \infty & \text{if } h_2^{\text{pot}}(s) > 0 \\ h_1^{\text{pot}}(s) & \text{otherwise.} \end{cases}$$

We call heuristics of this form *nested potential heuristics*. Essentially, the test $[h_2^{\text{pot}}(s) > 0]$ acts as a *predicate* that determines which states should be pruned. Decoupling

the predicate from the numerical heuristic value in this way increases the set of functions that can be compactly represented.[4] Our third problem variant uses the same definition as ∞DDA, but considers nested potential heuristics. We call it *predicate-based pruning DDA (PDDA)*.

We now study the complexity of verification and synthesis for these DDA variants. It turns out that all three variants have the same complexity, and therefore we collectively refer to all of them as VDDA for *variant DDA* in the following.

The outline of our chain of results is as follows: we first show **coNP**-membership of verification for VDDA and, based on this, $\mathbf{\Sigma}_2^{\mathrm{p}}$-membership of synthesis for VDDA. Then we show $\mathbf{\Sigma}_2^{\mathrm{p}}$-hardness (and hence completeness) of synthesis for VDDA using a suitable polynomial reduction, which we then reuse in a simplified form to show **coNP**-hardness (and hence completeness) of verification for VDDA.

Theorem 8. VERIFICATION(VDDA, $\mathcal{C}$) $\in$ **coNP** *for all PH classes* $\mathcal{C}$.

Proof. We are given a task Π and a potential heuristic h. We show that the complement of VERIFICATION(VDDA, $\mathcal{C}$) is in **NP** with a guess-and-check algorithm: guess a state s of Π and verify that it violates one of the conditions required for h to be VDDA.

For UDDA, this entails guessing a state and verifying that it is not a goal state and does not have a successor of lower heuristic value. This can clearly be done in nondeterministic polynomial time.

For ∞DDA or PDDA, either show that the initial state has an infinite heuristic value or guess a state and show that it is not a goal state, has a finite heuristic value, and does not have a successor with lower heuristic value. Again, this is clearly possible in nondeterministic polynomial time. □

Theorem 9. SYNTHESIS(VDDA, $\mathcal{C}$) $\in \mathbf{\Sigma}_2^{\mathrm{p}}$ *for all compact PH classes* $\mathcal{C}$.

Proof. Because $\mathbf{\Sigma}_2^{\mathrm{p}} = \mathbf{NP}^{\mathbf{NP}}$, it suffices to demonstrate that SYNTHESIS(VDDA, $\mathcal{C}$) can be solved in nondeterministic polynomial time if we have access to an **NP** oracle.

The following algorithm achieves this: first, guess the parameters of the potential heuristic to be synthesized. (For PDDA, this involves guessing two potential functions.) These guesses can be performed in polynomial time because we are synthesizing a compact heuristic. Then, use an oracle for the complement of VERIFICATION(VDDA, $\mathcal{C}$) (which is in **NP** from the preceding theorem) to test if the guessed heuristic violates VDDA. If it does not, accept. □

Theorem 10. SYNTHESIS(VDDA, $\mathcal{C}^k$) *is polynomial for* $k = 0$ *and* $\mathbf{\Sigma}_2^{\mathrm{p}}$*-hard for all* $k \in \mathbb{N}_1 \cup \{\infty\}$.

[4] For example, for a task with n binary state variables, consider the heuristic f_n with $f_n(s) = 0$ if $s[v] = 1$ for the majority of state variables and $f_n(s) = \infty$ otherwise. The number of features of a regular potential heuristic representing f_n must grow exponentially in n, while nested potential heuristics only require a linear number of features. More generally, the $[h_2^{\mathrm{pot}}(s) > 0]$ predicate can be used to compactly represent forms of resource reasoning, where states are pruned if the demand for a resource exceeds the supply.

SYNTHESIS(VDDA, $\mathcal{C}$) *is* $\mathbf{\Sigma}_2^{\mathrm{p}}$*-complete for all compact PH classes* $\mathcal{C}$ *that permit 1-dimensional features and singly-exponential feature weights.*

Proof. The $k = 0$ case is trivial as in SDDA and involves testing if the initial state is a goal state and (for ∞DDA and PDDA) testing that the initial heuristic value is finite.

Membership in $\mathbf{\Sigma}_2^{\mathrm{p}}$ for compact PH classes was shown in the preceding theorem. It remains to show $\mathbf{\Sigma}_2^{\mathrm{p}}$-hardness for 1-dimensional features and singly-exponential feature weights, which implies the other hardness results.

For hardness, we reduce from a variant of QBF that is complete for $\mathbf{\Sigma}_2^{\mathrm{p}}$, specifically QBF with an $\exists \ldots \forall \ldots$ quantifier prefix and an inner formula in 3DNF (Arora and Barak 2009, Example 5.6). We are given a QBF formula $\psi = \exists X_1 \ldots \exists X_n \forall Y_1 \ldots \forall Y_m \varphi(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$ where φ is in 3DNF:

$$\varphi = \bigvee_{i=1}^{k} (\ell_{i1} \wedge \ell_{i2} \wedge \ell_{i3})$$

where the ℓ_{ij} are literals over the X_i and Y_i variables.

Below, we construct a planning task Π in polynomial time such that ψ is true iff there exists a VDDA potential heuristic of dimension 1 for Π with at most singly-exponential weights. Moreover, we create Π in such a way that it is solvable iff ψ is true.

If the planning task is unsolvable, no VDDA heuristic can exist for any of the variants. If the planning task is solvable, we construct a potential heuristic that is finite for all states. For finite heuristics, there is no difference between VDDA variants, so the heuristic is simultaneously UDDA, ∞DDA and PDDA. (For the latter, use a trivial h_2^{pot} function that prunes nothing, such as $h_2^{\mathrm{pot}}(s) = 0$ for all states s.)

Π has the following state variables:

- $X_1, \ldots, X_n$ are state variables with domain $\{\mathbf{u}, \mathbf{0}, \mathbf{1}\}$, initially set to $\mathbf{u}$ ("unassigned"), with no goal value.
- $Y_1, \ldots, Y_m$ are state variables with domain $\{\mathbf{0}, \mathbf{1}\}$, initially set to $\mathbf{1}$, with goal $\mathbf{0}$.
- *Confirmed* is a state variable with domain $\{\mathbf{0}, \mathbf{1}\}$, initially set to $\mathbf{0}$, with goal $\mathbf{1}$.

The state variables X_i and Y_i directly correspond to the logic variables of the same name in ψ. Hence, states in Π encode assignments to these variables in ψ and vice versa. The role of the *Confirmed* variable will become clear after introducing the actions of Π, which are:

- the action *reset*, which has no preconditions and sets all state variables to their initial state value
- for all $i = 1, \ldots, n$ and $v \in \{\mathbf{0}, \mathbf{1}\}$ the action *assign*(i, v) with precondition $X_i \mapsto \mathbf{u}$ and effect $X_i \mapsto v$
- m *decrement* actions that implement a binary counter formed by the Y_i variables, interpreted as the binary number $Y_m \ldots Y_1$ (cf. the proof of Theorem 6). All *decrement* actions have *Confirmed* $\mapsto \mathbf{1}$ as an additional precondition and *Confirmed* $\mapsto \mathbf{0}$ as an additional effect.

- for each conjunction $\ell_{i1} \wedge \ell_{i2} \wedge \ell_{i3}$ in φ, the action *confirm*(i) with preconditions corresponding to the literals ℓ_{ij}, additional precondition *Confirmed* $\mapsto \mathbf{0}$ and effect *Confirmed* $\mapsto \mathbf{1}$.

We can clearly construct Π in polynomial time.

The role of *reset* is to ensure that if the planning task is solvable, all states are alive because the *reset* action allows transitioning from any state to the initial state. So either all states or no states are alive, depending on whether the task is solvable, as desired. The role of the *assign*(i, v) actions is to allow choosing suitable truth values for the X_i variables in Π, in the same way that we need to identify suitable values for the (existentially quantified) X_i variables in ψ in order to show that ψ is true.

The *decrement* actions add additional meaning to the Y_i variables, which now also encode a binary counter, as explained in the proof of Theorem 6. The counter is decremented by 1 by each application of *decrement*. Additionally, *decrement* actions can only be used after verifying that the current assignment to the Y_i variables satisfies φ, which is done using the *confirm* actions.

We now show that the task is solvable iff ψ is true.

Let ψ be true. We show that there exists a plan for Π. Since ψ is true, there exists a suitable assignment α for the X_i variables such that for every assignment β to the Y_i variables, $\alpha \cup \beta \models \varphi$. The plan for Π begins with *assign* actions corresponding to the assignment α. After applying these actions, all X_i state variables have the value $\mathbf{0}$ or $\mathbf{1}$. Afterwards, the plan iteratively considers all assignments to the Y_i variables by alternating between *confirm* actions, which confirm that the current assignment satisfies some conjunction of φ, and *decrement* actions, which decrement the counter and reset *Confirmed* to $\mathbf{0}$. Because $\alpha \cup \beta \models \varphi$ for all β, a suitable *confirm* action can be found for every β. At the end of this, all variables have reached their goal value.

For the converse direction, assume that the planning task is solvable and show that the QBF formula is true. We can ignore the *reset* action: because it leads to the initial state, it never occurs in a non-redundant plan. So let us consider a plan without *reset*.

Note that without the *reset* action, every X_i variable can be assigned at most once in a plan because *assign* actions have $X_i \mapsto \mathbf{u}$ as a precondition and only *reset* can set X_i to $\mathbf{u}$. Let α be an assignment to the X_i logic variables that matches the assignments made by the plan. If the plan leaves some X_i unassigned, we can set $\alpha(X_i)$ arbitrarily. Note that it does not matter at which point in the plan the X_i variables are assigned; the important point is that each variable can only receive one of the values $\mathbf{0}$ and $\mathbf{1}$ in the plan.

The Y_i variables encode a binary counter that starts at the maximum value $1 \dots 1$ (from the initial state $Y_i = 1$ for all i) and must end at the minimum value $0 \dots 0$ (from the goal $Y_i = 0$ for all i). The only way to reach the goal is to keep decrementing the binary counter, and this will traverse all possible assignments β to the Y_i logic variables.

We can only decrement the counter after confirming that φ is true for the current assignment because each *confirm* action checks that at least one of the conjunctions is satisfied. Together, this means that the plan confirms all assignments to the Y_i variables. Therefore ψ is true: there exists a *fixed* assignment α to the X_i variables such that for *all* assignments β to the Y_i variables, $\alpha \cup \beta \models \varphi$.

It remains to show that if the task is solvable, there exists a 1-dimensional VDDA potential heuristic h^{pot} with singly-exponential weights that is finite for every state. Because the task is solvable, ψ is true. Let α be an assignment to the X_i variables that witnesses the truth of ψ. We use the following weights for h^{pot}:

- $\{X_i \mapsto \mathbf{u}\}$ has a weight of 1
- $\{X_i \mapsto \alpha(X_i)\}$ has a weight of 0
- the complementary atom $\{X_i \mapsto 1 - \alpha(X_i)\}$ has a weight of $N + 1$, with $N = n + 2^{m+1} - 1$
- $\{$*Confirmed* $\mapsto \mathbf{0}\}$ has a weight of 1
- $\{$*Confirmed* $\mapsto \mathbf{1}\}$ has a weight of 0
- $\{Y_i \mapsto \mathbf{1}\}$ has a weight of 2^i. We emphasize that the least significant digit in the number $Y_m \dots Y_1$ has index 1, not 0, so the least significant digit has a weight of 2, the next one a weight of 4, and so on.

Note that $N = h^{\text{pot}}(I)$: the n atoms $X_i \mapsto \mathbf{u}$ (all of weight 1) contribute n, the Y_i variables contribute $\sum_{i=1}^{m} 2^i = 2^{m+1} - 2$, and *Confirmed* contributes 1.

We show that the resulting heuristic is indeed VDDA, i.e., every non-goal state has a successor of lower heuristic value. Let us call a state *bad* if it includes an assignment to some X_i that contradicts α, and let us call it *good* otherwise.

Bad states contain at least one bad assignment, contributing $N + 1$ to the heuristic value. Therefore, *reset* is an improving action, leading to the initial state with heuristic value N.

In a good state, if not all X_i variables are assigned yet, assigning X_i to $\alpha(X_i)$ reduces the heuristic value (by 1). Otherwise, if *Confirmed* is $\mathbf{0}$, a *confirm* action must be applicable, reducing the heuristic value (by 1). Otherwise, *Confirmed* is $\mathbf{1}$. If the Y_i binary counter is already at its minimum value, we are in a goal state. Otherwise, we can use a *decrement* action to decrement the counter by 1. This reduces the heuristic value by 1: the effect of the *decrement* action on the Y_i variables reduces the heuristic value by 2, while the effect on *Confirmed* increases it by 1. □

Theorem 11. VERIFICATION(VDDA, $\mathcal{C}^k$) *is polynomial for* $k = 0$ *and* **coNP***-complete for all* $k \in \mathbb{N}_1 \cup \{\infty\}$.

Proof. The case $k = 0$ is again trivial.

Membership in **coNP** was shown in Theorem 8.

For hardness, we can use essentially the same construction as in the previous proof, but without existential variables in the QBF. So we are given a QBF of the form $\psi = \forall Y_1 \dots \forall Y_m \varphi(Y_1, \dots, Y_m)$ with φ in 3DNF. This is the **coNP**-hard 3DNF tautology problem, the complement of the 3CNF satisfiability problem.

We construct the same planning task Π as in the previous reduction for the special case $n = 0$ (no existential variables). We also construct the potential heuristic h^{pot} in the same way as the heuristic described in the previous proof.

Because there are no existential variables, h^{pot} does not depend on finding a "correct" assignment α to these variables and can hence be constructed directly in polynomial time with the weights given in the previous proof.

With the same arguments as in the previous proof, h^{pot} is a VDDA potential heuristic for Π iff the given QBF is true, and the reduction is polynomial-time. □

In summary, the verification and synthesis problems for the three DDA variants that do not involve aliveness tests are complete for the first and second level of the polynomial hierarchy, in contrast to the **PSPACE**-completeness for the original DDA and SDDA properties. This suggests that these variants are more suitable for practical synthesis algorithms than the original DDA property or SDDA and that they could be tackled by compilations to other problems that fall into these levels of the polynomial hierarchy, such as (restricted) QBF solving.

While this is certainly positive news, there are also negatives: our results show that the islands of tractability at dimensions 1 and 2 that exist for synthesizing admissible and consistent potential heuristics do not exist for any of the DDA variants. Therefore, practical synthesis algorithms for this setting remain a formidable challenge.

Beyond Potential Heuristics

Before we conclude, we briefly discuss which of our results are specific to potential heuristics and which ones apply to heuristic synthesis more generally. Broadly construed, every algorithm that selects a specific heuristic from a large family of candidate heuristics is a form of heuristic synthesis. This includes selecting pattern databases or pattern database collections (e.g., Edelkamp 2006; Haslum et al. 2007), merge-and-shrink strategies (Sievers and Helmert 2021), fluent merging strategies (van den Briel, Kambhampati, and Vossen 2007), learning interesting conjunctions for critical-path heuristics (e.g., Keyder, Hoffmann, and Haslum 2012; Steinmetz and Hoffmann 2017), approaches that learn neural networks that represent heuristics (e.g., Ferber, Helmert, and Hoffmann 2020) and many other examples.

All the *membership* results we showed (for **PSPACE**, **coNP** or $\mathbf{\Sigma}_2^{\text{p}}$) are universal in the sense that they only require that the synthesized heuristics can be represented in polynomial space and evaluated in polynomial time. These are very loose requirements that are satisfied by all the above examples. At the extreme end of the spectrum, we can define a compact heuristic family as "all computer programs up to a polynomial length bound". If we add a polynomial timeout to the heuristic evaluation and return an arbitrary value (such as ∞) for states where this timeout is exceeded, this family has all properties required by our membership proofs, and hence VDDA synthesis for this family of heuristics is in $\mathbf{\Sigma}_2^{\text{p}}$. At this point, heuristic synthesis is essentially the synthesis of arbitrary programs, with clear connections to synthesis problems that arise for other flavors of planning, such as generalized planning (e.g., Srivastava, Immerman, and Zilberstein 2011; Bonet and Geffner 2018; Francès et al. 2019; Ståhlberg, Francès, and Seipp 2021). Therefore, one of the consequences of our study is that the new variants of the DDA property, giving rise to a $\mathbf{\Sigma}_2^{\text{p}}$ synthesis problem, have an intrinsic advantage over the original DDA property, for which we only get a **PSPACE** membership result.

In contrast, the hardness results are specific to potential heuristics. In some sense, these hardness results can be viewed as positive results regarding the expressiveness of potential heuristics because they show that potential heuristics are sufficiently expressive for capturing the essence of problems at the second level of the polynomial hierarchy, such as $\exists\forall$-QBF. In particular, the potential heuristics used in these reductions, together with a verification that they have the variant DDA property, are polynomial-sized witnesses for the solvability of these planning tasks even though the shortest plans for these tasks are exponentially long. In this way, our results also contribute to old questions regarding the synthesis of compact plans for tasks requiring exponentially long solutions, as studied for example in the work of Bäckström and Jonsson (2012).

Conclusion

We investigated the computational complexity of synthesizing potential heuristics for satisficing planning. We showed that both verifying and synthesizing *descending and dead-end avoiding* (DDA) heuristics is **PSPACE**-complete even in the restricted case of potential heuristics of dimension 1.

However, we discussed variants of the DDA property that reduce the complexity of verification and synthesis to the first and second levels of the polynomial hierarchy. Again, the membership results are accompanied by matching hardness results that already hold for dimension 1.

The membership results are not limited to potential heuristics. They hold for all heuristics that can be represented in polynomial space and evaluated in polynomial time. In particular, this suggests that the new DDA variant properties are more amenable for heuristic synthesis than the original DDA property.

In future work, we would like to investigate practical algorithms for potential heuristic synthesis. Based on the results in this paper, one idea worth exploring is compilation to problems in the second level of the polynomial hierarchy for which well-developed solvers exist, such as (restricted forms of) QBF. Another potential avenue is to use inductive techniques to "guess" a candidate heuristic as in recent approaches for generalized planning (Francès et al. 2019; Ståhlberg, Francès, and Seipp 2021) and using SAT solvers for the resulting **coNP**-complete verification problem.

Acknowledgments

We have received funding for this work from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 817639). Moreover, this research was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215.

References

Arora, S.; and Barak, B. 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press.

Bäckström, C.; and Jonsson, P. 2012. Algorithms and Limits for Compact Plan Representations. *Journal of Artificial Intelligence Research*, 44: 141–177.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS^+ Planning. *Computational Intelligence*, 11(4): 625–655.

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence*, 129(1): 5–33.

Bonet, B.; and Geffner, H. 2018. Features, Projections, and Representation Change for Generalized Planning. In Lang, J., ed., *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, 4667–4673. IJCAI.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69(1–2): 165–204.

Corrêa, A. B.; and Pommerening, F. 2019. An Empirical Study of Perfect Potential Heuristics. In Lipovetzky, N.; Onaindia, E.; and Smith, D. E., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 114–118. AAAI Press.

Dechter, R.; and Pearl, J. 1985. Generalized Best-First Search Strategies and the Optimality of A^*. *Journal of the ACM*, 32(3): 505–536.

Edelkamp, S. 2006. Automated Creation of Pattern Database Search Heuristics. In Edelkamp, S.; and Lomuscio, A., eds., *Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence (MoChArt 2006)*, 35–50.

Ferber, P.; Helmert, M.; and Hoffmann, J. 2020. Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. In De Giacomo, G., ed., *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, 2346–2353. IOS Press.

Francès, G.; Corrêa, A. B.; Geissmann, C.; and Pommerening, F. 2019. Generalized Potential Heuristics for Classical Planning. In Kraus, S., ed., *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 5554–5561. IJCAI.

Garey, M. R.; and Johnson, D. S. 1979. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012. AAAI Press.

Hoffmann, J. 2005. Where 'Ignoring Delete Lists' Works: Local Search Topology in Planning Benchmarks. *Journal of Artificial Intelligence Research*, 24: 685–758.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-Relaxed Plan Heuristics. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 128–136. AAAI Press.

Pommerening, F.; Helmert, M.; and Bonet, B. 2017. Higher-Dimensional Potential Heuristics for Optimal Classical Planning. In Singh, S.; and Markovitch, S., eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3636–3643. AAAI Press.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From Non-Negative to General Operator Cost Partitioning. In Bonet, B.; and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3335–3341. AAAI Press.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New Optimization Functions for Potential Heuristics. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 193–201. AAAI Press.

Seipp, J.; Pommerening, F.; Röger, G.; and Helmert, M. 2016a. Correlation Complexity of Classical Planning Domains. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3242–3250. AAAI Press.

Seipp, J.; Pommerening, F.; Sievers, S.; Wehrle, M.; Fawcett, C.; and Alkhazraji, Y. 2016b. Fast Downward Aidos. In Muise, C.; and Lipovetzky, N., eds., *Unsolvability International Planning Competition: Planner Abstracts*, 28–38.

Sievers, S.; and Helmert, M. 2021. Merge-and-Shrink: A Compositional Theory of Transformations of Factored Transition Systems. *Journal of Artificial Intelligence Research*, 71: 781–883.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2011. A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 175(2): 393–401.

Ståhlberg, S.; Francès, G.; and Seipp, J. 2021. Learning Generalized Unsolvability Heuristics for Classical Planning. In Zhou, Z.-H., ed., *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*, 4175–4181. IJCAI.

Steinmetz, M.; and Hoffmann, J. 2017. State Space Search Nogood Learning: Online Refinement of Critical-Path Dead-End Detectors in Planning. *Artificial Intelligence*, 245: 1–37.

Štolba, M.; Fišer, D.; and Komenda, A. 2016. Potential Heuristics for Multi-Agent Planning. In Coles, A.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 308–316. AAAI Press.

van den Briel, M.; Kambhampati, S.; and Vossen, T. 2007. Fluent Merging: A General Technique to Improve Reachability Heuristics and Factored Planning. In *ICAPS 2007 Workshop on Heuristics for Domain-Independent Planning: Progress, Ideas, Limitations, Challenges*.

Wilt, C.; and Ruml, W. 2014. Speedy versus Greedy Search. In Edelkamp, S.; and Barták, R., eds., *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014)*, 184–192. AAAI Press.

Wilt, C.; and Ruml, W. 2016. Effective Heuristics for Suboptimal Best-First Search. *Journal of Artificial Intelligence Research*, 57: 273–306.