On the Complexity of Computing the Planning Width

Jiajia Song[1], **Seeun William Umboh**[1 3], **Nir Lipovetzky**[1 3], **Sebastian Sardina**[2]
[1]The University of Melbourne
[2]RMIT University
[3]ARC Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA)
jiajia.song1@unimelb.edu.au, william.umboh@unimelb.edu.au, nir.lipovetzky@unimelb.edu.au,
sebastian.sardina@rmit.edu.au

Abstract

The *width* of a classical planning instance, among other metrics, indicates the computational difficulty of the instance. However, no result exists on the complexity of computing the width itself. In this paper, we address this by utilising an optimisation complexity framework. We focus on planning instance with polynomially bounded solutions, and prove that computing their width is $\mathsf{OptP}[\mathcal{O}(\log\log L)]$-hard, where L is the size of the instance. In turn, for the upper bound, we show that computing width is in $\mathsf{OptP}[\mathcal{O}(\log L)]^{\mathsf{OptP}[\mathcal{O}(\log L)]}$. Problem set $\mathsf{OptP}[\mathcal{O}(z(L))]$ is the optimisation complexity class with their optimal values' length in binary bounded by $\mathcal{O}(z(L))$. These results contribute to the understanding of width as a proxy measure for the computational difficulty of planning, and suggest that exploiting other structural restrictions beyond bounding solution length, can provide further insights on the complexity of width computation.

Introduction

Width-based algorithms such as SIW (Lipovetzky and Geffner 2012) and BFWS (Lipovetzky and Geffner 2017) have achieved good performance in many planning tasks, even in lifted planning (Corrêa and Seipp 2022). The success of these algorithms is mainly due to the observation that planning instances can often be decomposed into sub-instances featuring single atomic goals with relatively *small widths*, and that low-width planning tasks can be solved in low polynomial time (Lipovetzky and Geffner 2012). The notion of "width" is useful beyond (classical) planning and has been applied to other settings, such as constraints (Dechter 2003), motion planning (Ferrer-Mestres 2018), reasoning over epistemic knowledge (Hu, Miller, and Lipovetzky 2019), optimal control (Ramírez et al. 2018), and reinforcement learning (O'Toole et al. 2021).

The width of a planning instance can indicate its difficulty: *a smaller width implies a relatively easier instance*. For example, Lipovetzky and Geffner (2012) have proved that domains such as Blocks-World with single atomic goals bear a width of no more than *two*, and thus instances within these sub-domains can be solved efficiently, regardless of their initial state and size.

Furthermore, width can help gain insight into the performance of Relational Neural Networks on learning generalisable polices in object-centric domains (Mao et al. 2023), as the circuit complexities of goal-conditioned polices can be derived from width.

However, an open problem remains: *how difficult is it, in general, to compute the width of planning instances?* In this paper, we address this question and formally analyse the computational complexity of finding the width of any planning instance (regardless of its domain).

It is natural to study the width computation problem from the perspective of *optimisation*, given that the *width* metric is defined as the minimum atomic size ranging over all so-called "admissible sets" (Bonet and Geffner 2024). Therefore we utilise the *optimisation complexity framework* (Krentel 1988) to analyse the complexity of the width computation problem.

The optimisation complexity class $\mathsf{OptP}[\mathcal{O}(z(L))]$ is the set of optimisation functions that can be computed in polynomial time by a Metric Turing machine (Krentel 1988), and the length of the computed optimal value in binary is bounded by $\mathcal{O}(z(L))$ with L being the input size. Intuitively, any optimisation problem within this complexity class can be solved by at most $\mathcal{O}(z(L))$ calls to an NP TM to conduct binary search over the thresholds in the decision versions, therefore the quantity $z(L)$ represents the "amount of NP-completeness" (Krentel 1988) in the problem.

One of the advantages to study the width computation problem via the optimisation complexity framework is that it is more desirable to find the exact width value than to decide if the width is above (or below) a given threshold. Moreover, the decision versions of problems may lose important structures of the original optimisation problems: not all NP-complete problems are equivalent when viewing them as optimisation problems (Krentel 1992). For example, while the decision versions of `TSP` and `CLIQUE` are both NP-complete, their optimisation versions are $\mathsf{OptP}[L^{\mathcal{O}(1)}]$-complete and $\mathsf{OptP}[\mathcal{O}(\log L)]$-complete, resp. This indicates that, unless $\mathsf{P} = \mathsf{NP}$, `TSP` is strictly harder than `CLIQUE` (Krentel 1992).

In this paper, we provide lower and upper bounds for width computation on polynomially bounded classical planning instances. The polynomial constraint is motivated by practice as one usually looks for "tractable" plans that can be executed (and possibly stored, verified) in polynomial time and it is customary in classical planning to conduct complexity analysis with polynomial bound on plan lengths such as in (Bylander 1994). For the lower bound, we prove $\mathsf{OptP}[\mathcal{O}(\log\log L)]$ hardness and, for the upper bound we show membership in $\mathsf{OptP}[\mathcal{O}(\log L)]^{\texttt{FCP}}$, where FCP is a problem (defined later) which we prove to be $\mathsf{OptP}[\mathcal{O}(\log L)]$-complete. Given the completeness result for FCP, it is equivalent to state that the upper-bound is $\mathsf{OptP}[\mathcal{O}(\log L)]^{\mathsf{OptP}[\mathcal{O}(\log L)]}$, where L is the size of the planning instance as an input.

The main conceptual insight underlying our complexity bounds is to move from abstract admissible sets to a more concrete notion of admissible sequences, which recovers the essential metaphor of a sequence of "stepping stones", a key element needed to compute width.

While width is an important proxy measure to planning complexity, there is no known efficient mechanistic method to extract the width of a problem. Even worse, there is no current understanding of how difficult that method would be. This paper aims to close this gap in knowledge which can inform the future research agenda on the topic, as discussed at the end of the paper.

The next two sections cover related work in complexity of planning and the necessary background. After that, we formally define the width computation problem and introduce useful new definitions. We then introduce the optimisation complexity framework and use it to prove width complexity membership and hardness. We end by drawing conclusions and possible future directions.

Related Work

The notion of "width" defined in (Freuder 1982) has been applied to factored planning (Amir and Engelhardt 2003; Brafman and Domshlak 2006) as a measurement of domain variables' interaction in the underlying CSP graph and it mainly focuses on tree decomposition. However this notion is different from the notion of "width" in classical planning.

In classical planning, Chen and Giménez (2007) have defined the notion of width as the Hamming distance between the initial state and a goal state. They have shown that instance with its width bounded by a constant can be solved in polynomial time. Lipovetzky and Geffner (2012) have introduced a new version of planning width as the minimum tuple size in tuple graphs which contain optimal paths to goal states. Bonet and Geffner (2024) have defined width equivalently, but via the concept of admissible sets. Specifically, if b is the maximum branching factor and n is the number of atoms, Lipovetzky and Geffner (2012) and Bonet and Geffner (2024) have shown that if a planning instance's width is w, it can be solved optimally in time and space upper bounded by $\mathcal{O}(bn^{2w-1})$ and $\mathcal{O}(bn^{w})$ respectively. Let d be the domain size, Junyent, Gómez, and Jonsson (2021) have also derived a tighter bound of $\sum_{i=0}^{w}[\binom{n-1-i}{w-i}d^i(d-1)^{w-i}]$.

Although Bonet and Geffner (2024) have proved that the planning width is unbounded in the general case, Lipovetzky and Geffner (2012) have proved that if the goals only contain a single atom, the instance's width is upper bounded by 2 for problems in some domains, such as Blocks-World, Logistics and n-Puzzles.

While width reveals the hardness of planning problems, to the best of our knowledge, there is no formal analysis of the complexity of finding width for classical planning instances. This is what our work aims to address.

Background and Problem Definitions

We provide a brief overview of classical planning and the notion of planning width, as a proxy metric for the computational difficulty of planning instances.

Classical Planning

A ***classical planning instance*** in the STRIPS language (Fikes and Nilsson 1971) is $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G}\rangle$, where $\mathcal{F}$ is a set of Boolean variables (atoms), $\mathcal{A}$ is a set of actions, $\mathcal{I}$ is the set of atoms that are true initially and $\mathcal{G}$ are the atoms that must be true in the goal. Each action a is $\langle pre(a), post(a)\rangle$: $pre(a) \subseteq \mathcal{F}$ and represents the set of atoms that need to be true for a to be applied; $post(a) = \langle add(a), del(a)\rangle$ where $add(a)$ is the set of atoms that become true after applying a and $del(a)$ is the set of atoms that become false after the action.

A planning instance denotes, succinctly, a ***state model*** $\langle S, \mathcal{A}, s_0, S_G, f\rangle$ (Bonet and Geffner 2001), where S is a set of states, $\mathcal{A}$ is a set of actions, $s_0 \in S$ is the initial state, $S_G \subseteq S$ is the set of goal states and $f : S \times \mathcal{A} \to S$ is the (partial) transition function. We denote the set of actions that are applicable to a state s as $\mathcal{A}_s$, i.e., the system can transition from s to s' by an action a if $a \in \mathcal{A}_s$ and $s' = f(s, a)$.

Each state s can be viewed as the set of atoms that are conjunctively true in it. Therefore, if $a \in \mathcal{A}_s$, we have $pre(a) \subseteq s$ and $f(s, a) = (s \setminus del(a)) \cup add(a)$. We also have $\mathcal{I} = s_0$ and $\mathcal{G} \subseteq s$ for all $s \in S_G$.

A state $s' \in S$ is said to be ***reachable*** from a state $s \in S$ if there exists a, possibly empty, finite sequence of actions $\pi = (a_1, \ldots, a_n)$, that induces a ***trace*** $(s_0, s_1, \ldots, s_n)$, such that $s_0 = s$, $s_i = f(s_{i-1}, a_i)$, for all $i \in \{1, \ldots, n\}$, and $s_n = s'$. We call such a sequence a ***valid action sequence*** from s to s'. If no shorter sequence of actions can reach s' from s, we then say that s' is *reached optimally* from s by plan π. If $s' = s_0$, the ***empty plan*** ( ) optimally reaches s'.

Finally, if $s' \in S_G$ and is reachable by the sequence of actions π from s_0, we say that π is a ***(valid) plan*** for the planning problem P. The length of π is the number of actions in it. If there exists no plan with shorter length than π, we call π an ***optimal plan***. Trivially, given a planning instance P, if all goal atoms are true in the initial state, the empty plan ( ) is optimal with length 0.

Planning Width

The notion of "planning width" analysed in this work first appeared in (Lipovetzky and Geffner 2012) and was defined via the idea of "tuple graphs". However, in this paper, we adopt an equivalent definition through the concept of "admissible sets" as in (Bonet and Geffner 2024) since it is more convenient for our purposes.

Atomic conjunction. Given a planning instance P, we call a set of atoms t an ***atomic conjunction***.[1] If all atoms in t are true in a state s, we denote it as $t \subseteq s$.

Given two atomic conjunctions t, t', we say that t is ***reachable*** from t' if there exist two states s, s' such that $t \subseteq s, t' \subseteq s'$ and s is reachable from s' by a valid action sequence. Consider a valid action sequence π which reaches t from t', if no shorter valid action sequence can reach t from t', we say that t is ***reached optimally*** from t' via π.

We say t is reachable (without mentioning the start) if it can be reached from an atomic conjunction $t' \subseteq \mathcal{I}$. Given a valid action sequence π to t from a $t' \subseteq \mathcal{I}$, if no shorter valid action sequence to t from any atomic conjunction which is true in $\mathcal{I}$, we call π an ***optimal plan*** to t.

Admissible set, atomic size and width. A set of atomic conjunctions is called an *atomic set*. We now define admissible sets.

Definition 1 (Admissible set)**.** *(Bonet and Geffner 2024) Given a planning instance P, an atomic set T is an* ***admissible set*** *if all of the following hold:*

- *Each $t \in T$ is reachable;*
- *There exists a tuple $t \in T$ such that $t \subseteq s_0$;*
- *For any optimal action sequence π to an atomic conjunction $t \in T$, if π is not an optimal plan for the instance P, we can extend it by adding one single action a to reach another atomic conjunction $t' \in T$ optimally.*

Let $Admissible^{Set}(P)$ be the set of all admissible sets of P.

Note that the only atomic conjunctions in an admissible set T that cannot be extended are the ones with their optimal plans being optimal plans for the instance P.

Let T be a set of atomic conjunctions (not necessary admissible), we define its *atomic size* as below.

Definition 2 (Atomic size)**.** *(Bonet and Geffner 2024)*

- *For an atomic conjunction t, its* atomic size $|t|$ *is the number of atoms in t.*
- *For an atomic set T, its* atomic size *is the maximum atomic size of $t \in T$, i.e., $size(T) \triangleq \max_{t \in T} |t|$.*

Next, we define the width of a planning instance P. If no plan exists for P, we define its width as $n + 1$ where n is the number of atoms. If all goal atoms are true in the initial state or all goal atoms can be reached in only one action from the initial state, we define its width as 0. Otherwise, the width is defined as the minimum atomic size ranging over all admissible sets.

Definition 3 (Width)**.** *(Bonet and Geffner 2024) Given a planning instance P with n atoms, its* ***width*** *$w(P)$ is*

- *$n + 1$, if P is unsolvable;*
- *0, if all goal atoms are true initially or P can be solved by one action;*
- *Otherwise, $w(P) \triangleq \min_{T \in Admissible^{Set}(P)} size(T)$.*

We use an example to illustrate the above definitions.

[1]In (Lipovetzky and Geffner 2012; Bonet and Geffner 2024), t is called an "atomic tuple". As the order of atoms in t is not used in deriving any results, we use "atomic conjunction" instead

Example 1. *Consider an instance in Blocks-World domain with 3 blocks: A, B and C. Initially, A is directly on B, B is on C and C is on table. The goal is to have C on A directly, i.e., $\mathcal{I} = \{clear(A), on(A, B), on(B, C), onTable(C)\}$ and $\mathcal{G} = \{on(C, A)\}$.*

One admissible set is $T = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6\}$, where

$$t_0 = \{clear(A)\},\ t_1 = \{hold(A)\},\ t_2 = \{onTable(A)\},$$
$$t_3 = \{hold(B)\},\ t_4 = \{onTable(B)\},$$
$$t_5 = \{hold(c), clear(A)\},\ t_6 = \{on(C, A)\}$$

However, if we replace t_5 with $t'_5 = \{hold(C)\}$, T is no longer admissible. This is because an optimal action sequence to t'_5 may not be extended to reach another atomic conjunction in T. One such optimal action sequence is $(unstack(A, B)$, $putdown(A)$, $unstack(B, C)$, $stack(B, A)$, $pickup(C))$.

The Width Computation Problem

In this paper, we aim to analyse the computational complexity of finding the width of a classical planning instance which is polynomially bounded, i.e., the lengths of its optimal plans (if there exist plans) are bounded polynomials of a fixed degree in terms of the number of atoms. We denote the set of such planning instances as $\mathcal{P}_{\text{poly}}$.

The width computation problem is defined below.

Problem: `Find-Width`
Input: a planning instance $P \in \mathcal{P}_{\text{poly}}$ with size L
Output: the width $w(P)$

For the width computation problems, the input is P, which includes the set of atoms $\mathcal{F}$, the set of actions $\mathcal{A}$ and some other elements such as the initial and goal sets. Let L be the ***size of a planning instance***, which is in $\mathcal{O}(n + m)$, where n and m are the number of atoms and actions, resp.

Challenges and Key Insight

The definition of width makes its difficult to design an efficient algorithm for computing width. In fact, it is unclear how to even efficiently determine if a given atomic set T is admissible or not. This is because T is defined to be admissible if *every* optimal sequence of actions (not an optimal plan for the instance) to every atomic conjunction $t \in T$ is extendable by one action to form an *optimal* sequence of actions to another $t' \in T$. Since there are potentially an exponential number of optimal plans and possible extensions to atomic conjunctions, efficiently checking admissiblity is already challenging.

Our key insight is that not all atomic conjunctions in an admissible set are equally "useful". Intuitively, the useful ones are those that serve as intermediate "stepping stones" which can be extended optimally towards the goal. Moreover, we only need one such sequence of stepping stones in each admissible set. We will formalise this intuition and show that every admissible set induces a sequence of stepping stones for the instance, which in turn induces an optimal plan for the problem (Proposition 2). Thus, in order to build a single sequence of stepping stones, we only consider the *useful* atomic conjunctions in an admissible set that are part of the same sequence of well-arranged stepping stones.

We define these well-arranged sequences as *admissible sequences* in Definition 5. It can be seen that an admissible sequence contains only "useful" atomic conjunctions compatible with optimal plans as shown in Definition 5.

In order to use admissible sequences to compute width, we need to ensure that every admissible set T can induce an admissible sequence whose atomic size is no greater than the atomic size of T. This claim is formally proved in Proposition 2. Then we show that width can be computed via admissible sequences as shown in Equation (1).

Properties of Width

We begin by defining the *conjunction optimal position* of an atomic conjunction.

Definition 4 (Conjunction optimal position)**.** *Given a planning instance $P \in \mathcal{P}_{poly}$, the* ***conjunction optimal position*** *$pos(t)$ of an atomic conjunction t is the length of an optimal plan to t. If t is unreachable, then $pos(t) = p(n) + 1$, where n is the number of atoms and $p(\cdot)$ is a* ***polynomial bound on plan length****.*

Definition 5 (Admissible sequence)**.** *Given a planning instance P, a sequence of atomic conjunctions $(t_0, t_1, \ldots, t_m)$ is called an* ***atomic sequence****, and is said to be* ***admissible*** *if it satisfies each of the following:*

- *t_0 is true in the initial state;*
- *for each $i \in \{0, 1, \ldots, m-2\}$,*
  - *For each optimal plan for t_i, there exists an action a that extends it to an optimal plan for t_{i+1};*
- *for each optimal plan for t_{m-1} there exists an action that extends it to an optimal plan for $t_m \cup G$.*

Let $Admissible^{Seq}(P)$ denote the set of all admissible sequences of P.

Note that if ( ) is an optimal plan for P, then (t_0) is an admissible sequence if t_0 is true initially.

We define the *atomic size* of an atomic sequence (not necessary admissible) $\tau = (t_1, t_2, \ldots, t_m)$ as the atomic size of the atomic set $\{t_1, t_2, \ldots, t_m\}$ and denote it as *size*(τ).

Definition 5 implies that the set of atomic conjunctions in an admissible sequence is an admissible set and each t_i in this sequence can be reached optimally by exactly i actions.

Proposition 1. *Given a planning instance P and an admissible sequence $(t_0, t_1, \ldots, t_m)$, the atomic set $T = \{t_0, t_1, \ldots, t_m\}$ is admissible.*

From Definition 5 and Proposition 1, we know that if viewing admissible sequences as sets of atomic conjunctions, the set of all admissible sequences is a subset of the set of all admissible sets, i.e., $Admissible^{Seq} \subseteq Admissible^{Set}$.

As explained above, admissible sequences are easier to verify than admissible sets as they are "ordered" by an optimal plan and include no unnecessary atomic conjunctions (i.e., no unnecessary "non-stepping stones"). To rely on those sequences, though, we need to ensure that each admissible set induces an admissible sequence.

The following proposition shows that every admissible set can induce an admissible sequence, which is the key to compute width from admissible sequences.

Proposition 2. *Given a planning instance P and an admissible set T, the following claims hold.*

1. *There exists an atomic conjunction $t \in T$ such that any optimal plan to t is also an optimal plan for P.*
2. *There exists an admissible sequence $\tau = (t_0, t_1, \ldots, t_m)$ with $m \geq 0$ and $t_i \in T$ for all $0 \leq i \leq m$.*

In order to prove the first claim of Proposition 2, we construct an acyclic sequence of atomic conjunctions (no atomic conjunction appears more than once in this sequence). From Definition 1 and the finity of the given admissible set, we show that the optimal plan to the last atomic conjunction via each conjunction in this sequence is an optimal plan for the planning instance. We then prove that any optimal plan to this last atomic conjunction is an optimal plan for the planning instance by contradiction. In order to prove the second claim, we adopt similar idea as above and construct a sequence of atomic conjunction which satisfy the requirements of admissible sequence. We leave the detailed proof in the supplementary material.

It is easy to see that the size of an induced sequence τ cannot be larger than that of the admissible set T, that is, *size*$(\tau) \leq \max_{t \in T} |t| =$ *size*(T). In addition, since width is defined as the *minimum* atomic size among all admissible sets, we have the following equation: (Recall that $Admissible^{Set}(P)$ and $Admissible^{Seq}(P)$ are the sets of all admissible sets and sequences for P, respectively.)

$$w(P) \triangleq \min_{T \in Admissible^{Set}(P)} size(T) = \min_{\tau \in Admissible^{Seq}(P)} size(\tau). \quad (1)$$

We define `Find-ConjunctionPos` (`FCP`), an auxiliary problem which decides if a pair of conjunctions is part of an admissible sequence. The input to `FCP` is a tuple $\langle P, t, t' \rangle$, where P is a planning instance and t, t' are two atomic conjunctions. The output is $\langle pos(t), d \rangle$ where $pos(t)$ is the conjunction optimal position of t and $d = 1$ represents that *all* optimal plans to t can be extended by one action to be an optimal plan to t', otherwise $d = 0$.

Problem:	`FCP`
Input:	a planning instance $P \in \mathcal{P}_{\text{poly}}$ with size L, two atomic conjunctions t and t'
Output:	$\langle pos(t), d \rangle$ where $d = 0$ or 1.

The Optimisation Complexity Framework

As the width computation problem can be viewed as an optimisation problem, we analyse it using the framework of optimisation complexity due to (Krentel 1988).

We begin by formally defining minimisation problems. Intuitively, an instance of a minimisation problem is associated with a set of feasible solutions and each solution has a value. The computational problem is to find the smallest value among the feasible solutions.

Formally, Let $\mathbb{N}_0^k$ denote the set of k-tuples with each element being a non-negative integer and let $(\mathcal{R}, \preceq)$ be a totally ordered subset of $\mathbb{N}_0^k$.[2] A ***minimisation problem*** consists of a function $f : \Sigma^* \rightarrow 2^{\mathcal{R}}$ that maps each string

[2]Every element in $\mathcal{R}$ is a k-tuple $a = \langle a_1, a_2, \ldots, a_k \rangle$ with a_i being a non-negative integer for all i. For any $a, b \in \mathcal{R}$, either $a \preceq b$ or $b \preceq a$. The binary relation $\preceq$ is reflexive ($a \preceq a$), transitive

	Optimisation Complexity	**Number of calls to a TM**
`Find-Width` upper bound	in $\mathsf{OptP}[\mathcal{O}(\log L)]^{\mathsf{A}}$ (Theorem 3)	$\mathcal{O}(\log L)$-$\mathsf{NP}^{\texttt{FCP}}$
`Find-Width` lower bound	$\mathsf{OptP}[\mathcal{O}(\log\log L)]$-hard (Theorem 4)	$\Omega(\log\log L)$-NP
`FCP`	$\mathsf{OptP}[\mathcal{O}(\log L)]$-complete (Theorem 7)	$\Theta(\log L)$-NP

Table 1: Summary of Complexity Results: L is the input size. Recall that an optimisation problem in $\mathsf{OptP}[\mathcal{O}(z(L))]$ can be solved by at most $\mathcal{O}(z(L))$ calls to an NP Turing machine to conduct binary search over the thresholds in the decision version.

x over an alphabet Σ to $f(x)$, a subset of $\mathcal{R}$.[3] We denote the minimum value (based on the order $\preceq$) of $\min f(x)$ as $\min f(x) \triangleq \min\{r : r \in f(x)\}$. The corresponding computational problem $\mathsf{f}_{\min}$ is to compute the value of $\min f(x)$.

The above motivated (Krentel 1988) to define a variant of non-deterministic Turing machines called Metric Turing Machines. We use the same definition and also allow the machine to be equipped with an oracle.

Definition 6 (Optimal Turing Machine)**.** *An* ***Optimal Turing Machine (OTM)****, $\mathcal{M}$, is a non-deterministic polynomially time bounded Turing machine such that each non-deterministic computation path writes a value in binary and accepts. The minimum of all these values is $opt_{\mathcal{M}}(x)$.*

Let A be a function on Σ^*. We denote by $\mathcal{M}^{\mathsf{A}}$ the resulting machine obtained by equipping an OTM $\mathcal{M}$ with oracle access to A, i.e., the machine can compute $\mathsf{A}(y)$ for any $y \in \Sigma^*$ in time equal to the length of the binary representation of y and $\mathsf{A}(y)$.

A minimisation problem f is then said to be *computable* by OTM $\mathcal{M}$ if $opt_{\mathcal{M}}(x) = \min f(x)$ for every input x.

The power of an OTM is characterised by the number of output bits it uses. Observe that an OTM that, for some function z, is restricted to $z(L)$ bits of output on inputs of length L can be simulated using $z(L)$ queries to an NP oracle by conducting binary search over the thresholds. (Krentel 1988) showed that the reverse is also true. Consequently, the difficulty of an optimisation problem can also be characterised by the number of output bits required by any OTM that computes it. This motivates the following family of complexity classes parameterised by the number of output bits.

Definition 7 ($\mathsf{OptP}[z(L)]$)**.** *Let $z : \mathbb{N}_0 \to \mathbb{N}_0$ be a polytime computable function. A minimisation problem $\mathsf{f}_{\min}$ is in the class of $\mathsf{OptP}[z(L)]$ if there exists an OTM $\mathcal{M}$ with $opt_{\mathcal{M}}(x) = \min f(x)$ and the length of $\min f(x)$ in binary is upper bounded by $z(L)$ for every input x of size L.*

If the problem is computable by $\mathcal{M}^{\mathsf{A}}$, the problem is in the complexity class $\mathsf{OptP}[z(L)]^{\mathsf{A}}$.

In our paper, we focus on $\mathsf{OptP}[\mathcal{O}(\log L)]$ and $\mathsf{OptP}[\mathcal{O}(\log\log L)]$.

Finally, we need to define reductions to prove hardness.

Definition 8 (Metric reduction)**.** *A* ***metric reduction*** *from a minimisation problem $\mathsf{f}_{\min}$ to another minimisation problem $\mathsf{g}_{\min}$ is a pair of functions (T_1, T_2) that can be computed in polynomial time such that $T_1 : \Sigma^* \to \Sigma^*$, $T_2 : \Sigma^* \times \mathbb{N}_0^k \to \mathbb{N}_0^l$, and $\min f(x) = T_2(x, \min g(T_1(x)))$ for all $x \in \Sigma^*$.*

Definition 9 (Hardness and Completeness)**.** *If every optimisation problem in $\mathsf{OptP}[z(L)]$ can be metrically reduced to an optimisation problem $\mathsf{f}_{\min}$, then $\mathsf{f}_{\min}$ is said to be* $\mathsf{OptP}[z(L)]$-hard*. Additionally, if $\mathsf{f}_{\min}$ is in $\mathsf{OptP}[z(L)]$, it is* $\mathsf{OptP}[z(L)]$-complete.

`Max-SAT` and `BIN-Packing` are examples of problems that are $\mathsf{OptP}[\mathcal{O}(\log(L))]$-complete, and belong to $\mathsf{OptP}[\mathcal{O}(\log\log L)]$, respectively (Krentel 1988). In `Max-SAT`, the input is a Boolean formula in conjunctive normal form, and the goal is to find the maximum number of clauses that can be satisfied by an assignment of truth values to the variables in the formula. In `BIN-Packing`, the input is a set of items and their sizes, and bins with equal capacity, and the goal is to find the fewest number of bins that can hold all items.

Complexity of Width Computation

In this section, we prove upper and lower bounds on the computational complexity of `Find-Width` (Theorems 3 and 4, respectively). Theorem 4 is arguably the most important result of this paper. The main results in this paper are summarised in Table 1.

We now describe the intuition behind our upper bound proof. At a high level, given a planning problem P, we construct an OTM that guesses atomic sequences and relies on the returned results from the `FCP` problem. In particular, each computation path is an atomic sequence τ. The results from `FCP` problem help verify if τ is admissible and writes $size(\tau)$ if so. The computational complexity of `FCP` is analysed in the next section.

Let L represent the size of the planning instance $P \in \mathcal{P}_{\text{poly}}$ which is the input to the `Find-Width` problem, m be the number of actions and n be the number of atoms of P. We also let $p(n)$ be the polynomial which bounds the optimal plan length if a plan exists. For meaningful representations of P, we have $n + m \leq L$.

Theorem 3. *`Find-Width` is in* $\mathsf{OptP}[\mathcal{O}(\log L)]^{\mathsf{A}}$.

Proof. Given a planning instance $P \in \mathcal{P}_{\text{poly}}$, let $t_{\mathcal{G}}$ denote the atomic conjunction formed by all atoms in $\mathcal{G}$.

We execute the following steps:

```
1: Step 1: obtain $\langle K, - \rangle$ from the FCP problem on input $\langle P, t_{\mathcal{G}}, t_{\mathcal{G}} \rangle$;
2: if $K > p(n)$ then
3:     write $n + 1$ and end computation;
4: if $K \leq 1$ then
5:     write 0 and end computation;
```

$((a \preceq b) \wedge (b \preceq c) \implies a \preceq c)$ and anti-symmetric $((a \preceq b) \wedge (b \preceq a) \implies a = b)$.

[3]Think of x as an instance of the problem and $f(x)$ as the set of values of its feasible solutions.

```
6: Otherwise, $K$ is the length of the optimal plans;
7: Step 2: guess an atomic sequence $\tau = (t_0, t_1, \ldots, t_K)$
   and a sequence of actions $(a_1, a_2, \ldots, a_K)$;
8: if $t_0$ is not contained in the initial state $s_0$ then
9:     write $n+1$ and end computation;
10: for $0 \leq i \leq K-1$ do
11:     obtain $\langle pos(t_i), d\rangle$ from FCP with input
        $\langle P, t_i, t_{i+1}\rangle$;
12:     if $pos(t_i) \neq i$ or $d = 0$ then
13:         write $n+1$ and end computation;
14:     record the state $s_{i+1}$ reached from $s_i$ by $a_{i+1}$;
15:     if $t_{i+1}$ is not contained in $s_{i+1}$ then
16:         write $n+1$ and end computation;
17: end for
18: if $\mathcal{G} \nsubseteq s_K$ then
19:     write $n+1$ and end computation;
20: compute $size(\tau)$, write it and end computation;
```

First of all, the above computation takes no more than polynomial-time of L to execute since there are K actions and K atomic conjunctions in τ with $K \leq p(n)$, and each state and atomic conjunction contain no more than n atoms.

Second, based on Definition 5 and the computation problem FCP, we know that if $size(\tau) \leq n$, it is an admissible atomic sequence. More specifically, we know that for all i, all optimal plans to t_i can be extended by one action to t_{i+1} optimally.

Third, let r be the minimum value among all written numbers, which can be computed by the OTM. If $r = n+1$, we have $w(P) = n+1$ as there is no plan for P. If $r < n+1$, we know that it is the minimum size among all admissible sequences of P and thus $r = w(P)$ as shown in Equation 1.

Finally, the minimum length of r in binary is no greater than $\log(n+1) < \log(L)$. From Definition 7, Find-Width is in the complexity class of $\mathsf{OptP}[\mathcal{O}(\log L)]^{\mathsf{A}}$. □

What follows is arguably the main result of the paper: a lower-bound complexity for width computation.

Theorem 4. *Find-Width is* $\mathsf{OptP}[\mathcal{O}(\log\log L)]$*-hard.*

Proof. The proof idea is illustrated in Figure 1.

We reduce any Optimal Turing machine and its input to an instance of Find-Width. Our reduction is an adaptation and extension of that by (Bylander 1994) for non-deterministic Turing machines.

Let the OTM $\mathcal{M} = \langle Q, \Gamma, \Sigma, \delta, q_0, Q_F, \Box\rangle$, where Q is a set of states, Γ is a tape alphabet which contains all symbols that can be written on the tape, $\Sigma \subseteq \Gamma$ is the input alphabet, $\Box \notin \Sigma$ is the blank symbol and $\Box \in \Gamma$, q_0 is the initial state, $Q_F \subseteq Q$ is the set of final accepting states, $\delta : Q \times \Gamma \to 2^{Q\times\Gamma\times\{\text{Left,Right}\}}$ is the transition function. Finally, if n is the size of the input, let $g_1(n)$ be the polynomial bounding the execution time of $\mathcal{M}$, and $g_2(n) \in \mathcal{O}(\log\log n)$ be the strict bound (non-equal) of the number of binary digits of the optimal value computed by $\mathcal{M}$. W.L.O.G., we assume no transition is enabled in final accepting states.

Now, given an input x on the tape of $\mathcal{M}$, we shall build below a planning instance $P_{(\mathcal{M},x)}$ such that its width is closely related to the minimum value computed from $\mathcal{M}$ when run on x, or more specifically, from the width of $P_{(\mathcal{M},x)}$, we can obtain the minimum value in polynomially bounded time. A trace of $P_{(\mathcal{M},x)}$ will first encode an execution branch of $\mathcal{M}$ on x, and then perform a kind of "enumeration" of states that yields an admissible atomic sequence whose width is related to the number written by $\mathcal{M}$ in the corresponding branch.

Let $G - 1 \triangleq g_1(|x|)$ be the maximum time $\mathcal{M}$ may run on input x, and of course, G will be the the maximum number of tape positions used. Let $H = 2\lceil\log G\rceil$ and we denote $[n] \triangleq \{1, 2, \ldots, n\}$.

We let $K = (2^{g_2(|x|)} - 1)$ denote the decimal value of the binary number $\underbrace{111\ldots11}_{g_2(|x|)\text{ copies of 1's}}$. If a computational branch intends to write a number greater than K, then our planning instance will just "write" K instead and K is upper bounded by $\log(|x|)$.

We now define the instance $P_{(\mathcal{M},x)} = \langle\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G}\rangle$.

Atoms. $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2 \cup \bigcup_{k=1}^{K}\{output_k\} \cup \{goal\}$ where:

- $\mathcal{F}_1 = \{in(j,x), at(j,q)\}_{q\in Q, x\in\Gamma, j\in[G]}$, as in (Bylander 1994) to capture all $\mathcal{M}$'s workings: $in(i,x)$ means that the symbol x is in position i. $at(i,q)$ means that $\mathcal{M}$ is currently in state q, reading the content in position i, and is ready to perform the transition according to δ.
- $\mathcal{F}_2 = \{d_i, p_i\}_{i\in[HK]}$, atoms used to represent and implement counting of binary numbers in order to construct an atomic sequence with a given atomic size. Atom d_i represents the i-th least significant digit of a binary number (1 if true), while p_i states that we are adding one to the i-th least significant digit.
- Atom $output_k$ (with $k \in [K]$) signals that machine $\mathcal{M}$ branch left k (in binary) on the tape, and that the counting process should commence.
- Atom *goal* will be used as dummy goal.

Since K is upper bounded by $\mathcal{O}(\log|x|)$ and $H = \lceil\log G\rceil$, there are at most $(|Q|G + |\Gamma|G + 2HK + K + 1)$ atoms, which is polynomial in terms of the size of the description of $\mathcal{M}$ and its input x.

Initial and goal states. The initial state describes the input $x = x_1, \ldots, x_{|x|}$ in the tape with the rest of the cells on the tape blank, and the OTM $\mathcal{M}$'s initial configuration.

$$\begin{aligned}\mathcal{I} = &\{in(1,x_1), in(2,x_2), \ldots, in(|x|, x_{|x|})\} \cup \\ &\{in(i,\Box) \mid i \in \{0, |x|+1, \ldots, G\}\} \cup \{at(1,q_0)\}.\end{aligned}$$

The goal of the planning task is to reach a state where distinguished atom *goal* holds true, that is, $\mathcal{G} = \{goal\}$.

Actions. The set of actions is built from three different "components".

First, we have actions dedicated to modelling the dynamics of the OTM $\mathcal{M}$. This is basically as in (Bylander 1994) except that we use one single action to model the Turing machine transition rather than three. Concretely, for every $i \in [G]$ (representing the position in the tape),

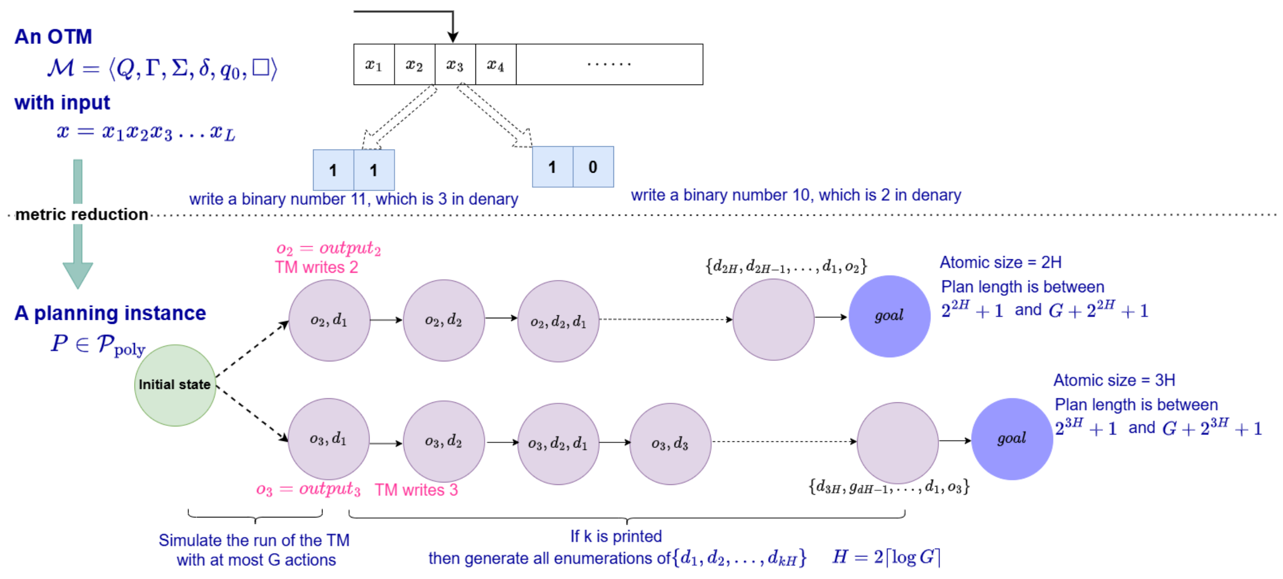
An OTM
$\mathcal{M} = \langle Q, \Gamma, \Sigma, \delta, q_0, \Box \rangle$
with input
$x = x_1 x_2 x_3 \dots x_L$
x_1 x_2 x_3 x_4 ......
1 1
write a binary number 11, which is 3 in denary
1 0
write a binary number 10, which is 2 in denary
metric reduction
A planning instance
$P \in \mathcal{P}_{\text{poly}}$
Initial state
$o_2 = output_2$
TM writes 2
o_2, d_1
o_2, d_2
o_2, d_2, d_1
$\{d_{2H}, d_{2H-1}, \dots, d_1, o_2\}$
goal
Atomic size = 2H
Plan length is between
$2^{2H} + 1$ and $G + 2^{2H} + 1$
o_3, d_1
o_3, d_2
o_3, d_2, d_1
o_3, d_3
$\{d_{3H}, g_{dH-1}, \dots, d_1, o_3\}$
goal
Atomic size = 3H
Plan length is between
$2^{3H} + 1$ and $G + 2^{3H} + 1$
$o_3 = output_3$ TM writes 3
Simulate the run of the TM
with at most G actions
If k is printed
then generate all enumerations of $\{d_1, d_2, \dots, d_{kH}\}$
$H = 2\lceil \log G \rceil$

Figure 1: Metric reduction to a planning problem for hardness proof in Theorem 4.

and every transition $(q, x, q', y, d) \in \delta$, the planning instance includes the action $\alpha(i, q, x, q', y, d)$ with precondition $\{at(i, q), in(i, x)\}$ and post-conditions:

- $\{at(i?1, q'), \neg at(i, q), in(i, y), \neg in(i, x)\}$, if $x \neq y$; or
- $\{at(i?1, q'), \neg at(i, q)\}$, if $x = y$.

where $? = +$ if $d =$ Right and $? = -$ if $d =$ Left.

Once the OTM $\mathcal{M}$ reaches an accepting state, and thus leaves a binary number (say, k in decimal) on the tape—the minimum across all execution branches—a special action will "read" such binary number and set corresponding atom $output_k$ representing it (in decimal). Concretely, for every final accepting state $q \in Q_F$, tape position $i \in [G]$, and potential decimal number left on the tape $k \leq K$ (encoded via atoms $in(\cdot, \cdot)$), the planning instance includes an action $read_k^i$ with precondition $at(i, q)$ plus the corresponding set of atoms $in(\cdot, \cdot)$ encoding the binary representation of number k; its post-condition is $\{output_k, p_1\}$.

Now, when an atom $output_k$ becomes true, the next phase is enabled and a set of ("counting") actions implementing a sequential "counting" of binary numbers from 1 up to the binary representation of $2^{Hk} - 1$ (i.e., $\underbrace{111\dots11}_{Hk \text{ copies of 1's}}$) become applicable. Intuitively, this counting process will force the planning trace to have an atomic sequence whose size is directly related to k.

Technically, the counting process is realised with the following actions, for each (digit index) $i \in [HK]$: (Recall, atom d_i is true if the i-th least significant digit is 1 and p_i states we are processing the i-th least significant digit.)

- Action inc_i increments the i-th least significant digit when it is 0, thus completing the current increment. Its precondition is $\{\neg d_i, p_i\}$ and its post-condition is $\{d_i, \neg p_i, p_1\}$ (p_1 signals the start of a new increment).
- Action $carry_i$ propagates the "carry" at the i-th least significant digit to the next digit. Its precondition is $\{d_i, p_i\}$, while its post-condition is $\{\neg d_i, \neg p_i, p_{i+1}\}$ if $i < HK$; and $\{\neg d_i, \neg p_i\}$ if $i = HK$ (i.e., the last digit).

Finally, $P_{(\mathcal{M},x)}$ includes a set of *final actions* that complete the counting process when number $2^{Hk} - 1$ number (in binary) has been enumerated (that is, we have "counted" to k). Concretely, for every $k \in [K]$, there is an action $finish_k$ whose precondition is $\bigcup_{i=1}^{Hk} \{d_i\} \cup \{output_k\}$ and post-condition is $\{goal\}$.

Size of encoding. There are at most $2G|Q|^2|\Gamma|^2$ actions $\alpha(i, q, x, q', y, d)$, $KG|Q|$ actions $read_k^i$, HK actions inc_i, HK actions $carry_i$ and K actions $finish_k$, so the total number of actions is polynomial in terms of the size of the description of $\mathcal{M}$ and its input x.

With the above (polynomial) encoding at hand, we first prove that the reduced instance $P_{(\mathcal{M},x)}$ is a planning instance in $\mathcal{P}_{\text{poly}}$, i.e., the lengths of its valid plans are bounded polynomially in terms of the number of atoms.

From above analysis, it can be seen that the lengths of valid plans for this planning instance $P_{(\mathcal{M},x)}$ are upper bounded by $G + 2^{HK}$ as it takes at most G steps to reach the atom $output_k$ and at most 2^{HK} counting steps to reach the goal from there. Recall that $H = 2\lceil \log G \rceil$ and $K = (2^{g_2(|x|)} - 1)$, so we have that

$$G + 2^{HK} \leq 2^{\log G^2 + 1} \cdot 2^{2^{g_2(|x|)}} = 2G^2 \cdot 2^{2^{g_2(|x|)}}.$$

Since $g_2(n) \in \mathcal{O}(\log \log n)$, we know that the lengths of valid plans are upper bounded polynomially in terms of the size of the description of $\mathcal{M}$ and its input x. Moreover, the number of atoms are upper bounded polynomially in terms of the size of the description of $\mathcal{M}$ and its input x, therefore, there must exist a fixed degree polynomial in terms of the number of atoms which upper bounds the lengths of valid plans, and thus, $P_{(\mathcal{M},x)} \in \mathcal{P}_{\text{poly}}$.

Finally, we are to prove that from the width of the planning instance, we are able to easily reconstruct the minimum value computed by $\mathcal{M}$ on x. To that end, note that for every number k written in a computational branch of $\mathcal{M}$, there exists an *atomic sequence* with size $Hk + 1$ reaching the goal. More concretely, the latter part of this sequence is

$$\{output_k, d_1\}, \{output_k, d_2\}, \{output_k, d_2, d_1\}, \\ \{output_k, d_3\}, \{output_k, d_3, d_1\}, \ldots, \\ \{output_k, d_{Hk}, d_{Hk-1}, \ldots, d_1\}, \{goal\}.$$

Thus k can be obtained by subtracting 1 from its atomic size and then dividing H.

Now consider two atomic sequences of the planning instance $P_{(\mathcal{M},x)}$: one has atomic size i and the other has atomic size j, where $1 \leq i < j$. We assume that it takes len_i and len_j actions to reach $output_i$ and $output_j$ respectively, thus the lengths of the two sequences will be $len_i + 2^{Hi}$ and $len_j + 2^{Hj}$. Since $0 \leq len_i, len_j \leq G$ and $H = 2\lceil \log G \rceil$, we have

$$len_i + 2^{Hi} \leq G + 2^{Hi} \leq 2^{\frac{H}{2}} + 2^{Hi} < 2^{Hi} + 2^{Hi} = 2^{Hi+1} \\ \leq 2^{Hj} \leq len_j + 2^{Hj}.$$

Therefore, an atomic sequence which has the smallest atomic size Hi will be the trace of the optimal plan for $P_{(\mathcal{M},x)}$, and according to the definition of width, we therefore conclude that the width of $P_{(\mathcal{M},x)}$ is indeed Hi. We can then derive the minimum value computed by $\mathcal{M}$ on x by just subtracting 1 from the width and then dividing it by H.

Putting it all together, we have found a polynomial metric reduction from $\mathcal{M}$ on x to the `Find-Width` problem. According to Definition 9, we know that `Find-Width` is $\mathsf{OptP}[\log\log L]$-hard with L as the input size. □

Complexity of the FCP Problem

We consider the computational complexity of the problem `FCP`. More specifically, given a planning instance $P \in \mathcal{P}_{\text{poly}}$ and two atomic conjunctions t, t', it computes the conjunction optimal position of t and checks if all optimal plans to t can be extended to t' optimally by one action.

First we show the membership (upper bound) of the problem `FCP`.

Lemma 5. *The problem `FCP` is in the complexity class of* $\mathsf{OptP}[\mathcal{O}(\log L)]$.

We show the proof sketch below and include the detailed proof in supplementary material.

Proof sketch. In order to prove the upper bound, we construct an algorithm which can be run by an OTM $\mathcal{M}$ in polynomial time and then we show that on input $\langle t, t', d \rangle$, it can return the desired result $\langle pos(t), d \rangle$, where $pos(t)$ is the conjunction optimal position of t and d is an indicator. The value d being 1 represents that every optimal plan to t can be extended by one single action to reach t' optimally.

In this algorithm, an action sequence π_1 for t and an action sequence π_2 for t' are guessed, then we check if they can reach t and t' respectively. If yes, we denote the shortest lengths to reach t and t' by these two sequences as l_1 and l_2. If π_1 can reach t, we then check if it can be extended by one action to t' (if yes, let $r = 1$), otherwise $r = 0$. For each guess, the algorithm writes a 3-tuple $\langle l_1, l_2, r \rangle$.

The OTM $\mathcal{M}$ computes the minimum value of the 3-tuple based on lexicographical order and obtains $\langle l, l', d \rangle$. It can be shown that $l = pos(t)$ and $l' = pos(t')$, which are the lengths of optimal plans to t and t' respectively. If $l+1 = l'$, we consider the value d. Since d is the minimum of the third element in the 3-tuple under the condition that $l_1 = l$ and $l + 1 = l'$, we know that if $d = 0$, there exists an optimal plan to t which cannot be extended to t' optimally by one action.

The final step is to inspect $\langle l, l', d \rangle$. If $l+1 = l'$ and $d = 1$, the algorithm returns $\langle l, 1 \rangle$, otherwise it returns $\langle l, 0 \rangle$.

It can be seen that the size of the returned value is upper bounded by $\mathcal{O}(\log L)$ and the computation time is polynomially bounded. □

Lemma 6. *The problem of `FCP` is* $\mathsf{OptP}[\mathcal{O}(\log L)]$*-hard.*

The proof of the hardness (lower bound) of `FCP` bears the same idea as in the proof for Theorem 4. The detailed proof is in supplementary material due to the limitation of space.

From Lemma 5 and Lemma 6, we have the following completeness result for `FCP`.

Theorem 7. *`FCP` is* $\mathsf{OptP}[\mathcal{O}(\log L)]$*-complete.*

Given that we have just shown `FCP` to be complete in $\mathsf{OptP}[\mathcal{O}(\log L)]$, it follows from Theorem 3, that width computation is in $\mathsf{OptP}[\mathcal{O}(\log L)]^{\mathsf{OptP}[\mathcal{O}(\log L)]}$.

Conclusion

In this paper, we have applied the optimisation complexity framework to analyse the computational complexity of finding the width of classical planning problems. We focused on polynomially-bounded planning—instances that admit polynomially long solution plans—and shown that the problem of width computation is in $\mathsf{OptP}[\mathcal{O}(\log L)]^{\mathsf{OptP}[\mathcal{O}(\log L)]}$ and in $\mathsf{OptP}[\mathcal{O}(\log\log L)]$-hard ($L$ being the size of the planning instance). The `FCP` problem used in the upper bound computes the conjunction-optimal position of a pair of tuples, and checks if they can be part of an admissible sequence. This problem is $\mathsf{OptP}[\mathcal{O}(\log L)]$-complete.

The polynomial bound constraint allows us to derive a stronger result on the lower bound (hardness) than the general case. As the lower bound is the most significant result in this paper, we think it is justifiable to impose this constraint. We conjecture that the complexity class will be "OptPSPACE" for the problems without constraints. However, since this complexity class has not been defined yet, it may require additional work on the computational complexity itself and we leave it as our future work.

There are planning algorithms that are able to exploit the width of problems. However, no technique to compute the *exact* width of planning instances exist and, until now, it is not even known how difficult that calculation is. The results obtained in this work, particularly the hardness one, indicate that indeed finding the width of a planning task is computationally difficult (e.g., as hard as Bin Packing (Krentel 1988)). This is significant in that it can inform future research agenda on *how to exploit the width notion*.

One possibility is to develop algorithms that can *approximate* the width. As with delete-relaxation (Bylander 1994), which is also hard in its optimal version, approximate width values would turn out to be very useful *in practice*, for example, to indicate what versions of width-based planners one needs to (not) use. According to (Krentel 1988), the analysis of optimisation complexity can give insights on the problem's approximation properties. For example, if an optimisation problem is $\mathsf{OptP}[\mathcal{O}(\log\log n)]$-complete, there is a lower bound on how closely it can be approximated. We would like to further develop this insight and gain a better understanding on the possible approximation of width computation in classical planning.

Another avenue worth exploring is the development of a framework that can allow us to *generalise* the width value across planning instances within the same domain. Thus, even if we pay the high cost of computing the width once, it can later be exploited in subsequent planning tasks with different goals or initial states.

Finally, as done by Bylander (1994), it is worth exploring syntactic restrictions that can lead to islands of tractability in the width computation.

Acknowledgments

This research was partially funded by the Australian Government through the Australian Research Council DP240101353, Australian Research Council Industrial Transformation Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA), Project ID IC200100009.

The first author is supported by the Australian Government Research Training Program Scholarship and by the University of Melbourne Faculty of Engineering and Information Technology Ingenium Scholarship Award.

References

Amir, E.; and Engelhardt, B. 2003. Factored Planning. In *Proc. of IJCAI*, 929–935.

Bäckström, C.; and Klein, I. 1991. Parallel Non-Binary Planning in Polynomial Time. In *Proc. of IJCAI*, 268–273.

Bäckström, C.; and Nebel, B. 1993. Complexity Results for SAS+ Planning. In *Proc. of AAAI*, 1430–1435.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1-2): 5–33.

Bonet, B.; and Geffner, H. 2024. General Policies, Subgoal Structure, and Planning Width. *Journal of Artificial Intelligence Research (JAIR)*, 80: 475–516.

Brafman, R. I.; and Domshlak, C. 2006. Factored planning: how, when, and when not. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, 809–814. AAAI Press.

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

Chen, H.; and Giménez, O. 2007. Act Local, Think Global: Width Notions for Tractable Planning. In *Proc. of ICAPS*, 73–80.

Corrêa, A. B.; and Seipp, J. 2022. Best-First Width Search for Lifted Classical Planning. In *Proc. of ICAPS*, 11–15.

Dechter, R. 2003. *Constraint processing*. Elsevier Morgan Kaufmann.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, Decidability and Undecidability Results for Domain-Independent Planning. *Artificial Intelligence*, 76(1-2): 75–88.

Ferrer-Mestres, J. 2018. *Combined task and motion planning as classical AI planning*. Ph.D. thesis, Pompeu Fabra University, Spain.

Fikes, R.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Proc. of IJCAI*, 608–620.

Freuder, E. 1982. A Sufficient Condition for Backtrack Free Search. *Journal of the ACM*, 29(1): 24–32.

Hu, G.; Miller, T.; and Lipovetzky, N. 2019. What you get is what you see: Decomposing Epistemic Planning using Functional STRIPS. *CoRR*, abs/1903.11777.

Junyent, M.; Gómez, V.; and Jonsson, A. 2021. Hierarchical Width-Based Planning and Learning. In *Proc. of ICAPS*, 519–527.

Krentel, M. W. 1988. The Complexity of Optimization Problems. *Journal of Computer and System Sciences*, 36(3): 490–509.

Krentel, M. W. 1992. Generalizations of Opt P to the Polynomial Hierarchy. *Theoretical Computer Science*, 97(2): 183–198.

Lipovetzky, N.; and Geffner, H. 2012. Width and Serialization of Classical Planning Problems. In *Proc. of ECAI*, 540–545.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In *Proc. of AAAI*, 3590–3596.

Mao, J.; Lozano-Pérez, T.; Tenenbaum, J. B.; and Kaelbling, L. P. 2023. What Planning Problems Can A Relational Neural Network Solve? In *Proc. NeurIPS*.

O'Toole, S.; Lipovetzky, N.; Ramírez, M.; and Pearce, A. R. 2021. Width-based Lookaheads with Learnt Base Policies and Heuristics Over the Atari-2600 Benchmark. In *Proc. NeurIPS*, volume 34, 26536–26547.

Ramírez, M.; Papasimeon, M.; Lipovetzky, N.; Benke, L.; Miller, T.; Pearce, A. R.; Scala, E.; and Zamani, M. 2018. Integrated Hybrid Planning and Programmed Control for Real Time UAV Maneuvering. In *Proc. of AAMAS*, 1318–1326.

Ray, K.; and Ginsberg, M. L. 2008. The Complexity of Optimal Planning and a More Efficient Method for Finding Solutions. In *Proc. of ICAPS*, 280–287.