Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)

Computing Planning Width: How Hard Is It, Really?

Jiajia Song[1], Seeun William Umboh[1, 4], Malte Helmert[2], Nir Lipovetzky[1, 4], Sebastian Sardiña[3]

[1]The University of Melbourne, Australia
[2]University of Basel, Switzerland
[3]RMIT University, Australia
[4]ARC Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA)
{jiajia.song1, william.umboh, nir.lipovetzky}@unimelb.edu.au, malte.helmert@unibas.ch, sebastian.sardina@rmit.edu.au

Abstract

The *width* of a classical planning instance, among other metrics, indicates the computational difficulty of solving the instance. However, no result exists on the complexity of computing the width itself. In this paper, we address this open problem by considering two versions of the width computational problem: a non-parameterised and a parameterised. In the non-parameterised version, a positive integer is given as an input along a classical planning instance, and it is PSPACE-complete to decide if this number is an upper bound on the instance's width. In the parameterised version, it is coNP-complete to decide if the width is no greater than the fixed parameter. These tight results resolve the long-standing question of how hard it is to compute the width of planning instances, and call for further research on width approximations, the generalisation of width computation over multiple instances and islands of tractability.

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 instances can often be decomposed into sub-instances featuring single atomic goals with relatively *small widths*, and that low-width tasks can be solved in low polynomial time (Lipovetzky and Geffner 2012). Computational notions derived from "width" are useful beyond (classical) planning and have been applied to other settings, such as constraints (Dechter 2003), motion planning (Ferrer-Mestres 2018), reasoning over epistemic knowledge (Hu, Miller, and Lipovetzky 2022), optimal control (Ramírez et al. 2018), and reinforcement learning (O'Toole et al. 2021).

The width of a planning instance indicates its difficulty: *a smaller width implies a relatively easier instance*. Intuitively, the width is defined as the "minimum novelty" that each state is required to exhibit in order to allow the search to continue. The novelty of a state in a search is measured as the number of new atoms that become true in this state but are unseen in previous states. Planning instances with smaller width require exploring smaller search spaces in order to be solved.

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 in quadratic time, regardless of their initial state and size.

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).

We consider both the non-parameterised and the parameterised versions of the width computational problem. For the former one, a positive integer and a classical planning instance are given as inputs and we show that it is PSPACE-complete to decide if this given integer is an upper bound on the instance's width. For the parameterised version, a classical planning instance is given as an input and we prove that it is coNP-complete to decide if the fixed parameter upper-bounds the width of this instance.

While width is an important proxy measure of 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 such a 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 on planning complexity and the necessary background. Then we formally define the width computation problems. We then prove width complexity membership and hardness for both non-parameterised and parameterised versions. We end by drawing conclusions and providing 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 be-

tween the initial state and a goal state. They have shown that an 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 the width for classical planning instances. This is what our work aims to address.

Classical Planning and Width

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 $\Pi = \langle \mathcal{F}, \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$, 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 $\mathcal{G}$ is the set of 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*** (Bonet and Geffner 2001), a transition system where a ***state*** $s \in S = 2^{\mathcal{F}}$ can be any combination of atoms, and transitions occur through an ***action sequence*** $\pi = (a_1, \ldots, a_n)$, which induces a ***trace*** $(s_0, s_1, \ldots, s_n)$, starting from the initial state $s_0 = \mathcal{I}$, following the STRIPS transition semantics $s_i = f(s_{i-1}, a_i) = (s_{i-1} \setminus del(a_i)) \cup add(a_i)$, over applicable actions such that $pre(a_i) \subseteq s_{i-1}$, for all $i \in \{1, \ldots, n\}$. We call an action sequence π a ***plan*** for t, where $t \subseteq s$ is an ***atomic conjunction*** or a (partial) state, if the last state of the trace induced by π reaches $t \subseteq s_n$, and an ***optimal plan*** if π is the shortest plan, i.e. a plan whith the least number of actions. When $t = \mathcal{G}$, the plan π solves the planning instance Π. If $G \subseteq I$, then the optimal plan is the ***empty plan*** $\pi = ()$.

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"[1], i.e. admissible sequences.

Admissible sequence, atomic size and width. We now define (goal) admissible sequences of atomic conjunctions.

Definition 1 (Admissible sequence). Given a planning instance Π, a sequence of atomic conjunctions $\sigma = (t_0, t_1, \ldots, t_m)$ is called an ***atomic sequence***, and is said to be ***admissible*** if it satisfies each of the following:

1. t_0 is true in the initial state of Π (i.e., $t_0 \subseteq \mathcal{I}$);
2. for every $i \in \{0, \ldots, m-1\}$, and every optimal plan for t_i, there exists an action that extends it to an optimal plan for t_{i+1}. ■

If the empty plan is an optimal plan for Π, then (t_0) is an admissible sequence if t_0 is true initially and if Π is unsolvable, its admissible sequence is an empty sequence.

Definition 2 (Goal admissible sequence). Given a planning instance Π, an admissible sequence $(t_0, t_1, \ldots, t_m)$ is ***goal admissible*** if for every optimal plan for t_{m-1} there exists an action that extends it to an optimal plan for $t_m \cup G$.■

Let σ be a sequence of atomic conjunctions (not necessarily admissible), we define its *atomic size* as follows.

Definition 3 (Atomic size). (Bonet and Geffner 2024) Given a planning instance Π, the ***atomic size*** of an atomic conjunction t is the number of atoms in t and is denoted as $|t|$. The ***atomic size*** of an atomic sequence τ is the maximum atomic size of $t \in \sigma$, i.e., $size(\sigma) \triangleq \max_{t \in \sigma} |t|$. ■

We denote the set of all admissible sequences of Π as $Adm(\Pi)$ and the set of all goal admissible sequences as $GAdm(\Pi)$. Specifically, the set of all admissible sequences of size no greater than k is denoted as $Adm(\Pi)_k \triangleq \{\sigma \in Adm(\Pi) \mid size(\sigma) \leq k\}$.

The set of bounded admissible sequences $Adm(\Pi)_k$ is equivalent to the set of traversals in the ***tuple graph*** $\mathcal{G}^k$ from the source vertex (Lipovetzky and Geffner 2012), the set of atomic conjunctions $t \in \sigma \in GAdm(\Pi)$ that appear in goal admissible sequences is equivalent to the notion of ***admissible tuple sets*** (Bonet and Geffner 2024), and goal admissibility relies on the concept of ***optimal implication*** $t \models^* t'$ (Lipovetzky 2012), where an atomic conjunction t optimally implies another conjunction $t' \neq t$ if all optimal plans for t are also optimal plans for t'.

We now have all the technical machinery required to define the width of a planning instance Π.

If no plan exists for Π, we define its width to be ∞ since its admissible sequence is empty and the size of an empty sequence is ∞. If all goal atoms are true in the initial state, we define the instance's width as 0. Otherwise, the width is defined as the minimum atomic size ranging over all goal admissible sequences.

[1] In (Lipovetzky and Geffner 2012; Bonet and Geffner 2024), t is called an "atomic tuple". As the order of atoms in t is irrelevant, we use "atomic conjunction" instead.

Definition 4 (Width; (Lipovetzky and Geffner 2012)). The ***width*** $w(\Pi)$ of a planning instance Π is defined as:

- ∞, if Π is unsolvable;
- 0, if the length of an optimal plan for Π is 0; and
- $\min_{\sigma \in GAdm(\Pi)} size(\sigma)$, otherwise. ■

Intuitively, if we consider all goal admissible sequences as *stepping stones* in the state model to cross from the intitial state to a goal state, the notion of width defines that there exists a sequence of stepping stones you can follow such that the biggest stone (atomic conjunction) you will step on the way to the goal is bounded by the width of the problem.

We use an example to illustrate the above definitions.

Example 1. Consider an instance in the 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, that is, $\mathcal{I} = \{clear(A), on(A,B), on(B,C), onTable(C)\}$. In turn, the goal is to have C on A directly, that is, and $\mathcal{G} = \{on(C,A)\}$.

One goal admissible sequence is $\sigma = (t_0, t_1, t_2, t_3, t_4, t_5, t_6)$, where:

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

However, if we were to replace atomic conjunction t_5 with the alternative $t'_5 = \{hold(C)\}$, σ would no longer be admissible. This is because there exists optimal action sequences to t'_5 that *cannot* be extended to reach another atomic conjunction in σ. One such optimal action sequence, for example, is $\tau = (unstack(A,B), putdown(A), unstack(B,C), stack(B,A), pickup(C))$, because no single action can be further executed to make block C be directly on block A and, therefore, no single action can be appended to τ to reach atomic conjunction t_6. □

It has been shown that there is an algorithm—called IW(k) (Lipovetzky and Geffner 2012)—that is complete and solves the problem optimally if $k = w(\Pi)$, that is, the width of the problem is known and used as a parameter. IW(k) time complexity is bounded by O(n^k), where n is the size of the planning instance (Lipovetzky and Geffner 2012; Bonet and Geffner 2024). IW can be run with incremental bounds $k = 0, \ldots, |\mathcal{F}|$ until the problem is solved. The smallest bound for which IW(k) finds a solution constitutes a lower bound on the actual width of the problem, yet finding the exact width and its complexity is a problem we address in the following sections.

The Width Computation Problems

While elegant, the definition of width suggests that its computation (for a planning instance) may be challenging. Indeed, already determining admissibility (Definition 2) involves reasoning about optimal plans. However, the computational complexity of determining the width of a planning instance remains open. We consider two decision problems.

The most natural problem involves checking whether a planning instance has a width of k or less:

Definition 5 (`WIDTH`). The `WIDTH` decision problem is defined as follows:

- Input: A planning instance Π and a positive integer $k \leq n$, where n is the number of atoms in Π.[2]
- Question: Does Π have a width of at most k?

The input size L is in $\mathcal{O}(n + m + k)$, where n and m are the number of atoms and actions of Π, respectively. ■

This problem is analogous to the `PLANMIN` (Bylander 1994) problem for optimal planning. Notice that, in the `WIDTH` problem, k is part of the input.

Another problem worth considering arises when $k \in \mathbb{N}$ is fixed (i.e., not part of the input), yielding the parameterised version `WIDTH-k`:

Definition 6 (`WIDTH-k`). For $k \in \mathbb{N}$, the `WIDTH-k` decision problem is defined as follows:

- Input: A planning instance Π.
- Question: Does Π have a width of at most k?

The input size L is in $\mathcal{O}(n + m)$, where n and m are the number of atoms and actions of Π, respectively. ■

In the following, we prove completeness results for both problems. The results are summarised in Table 1. Note that we do not consider $k = 0$ or $k = \infty$ in the above definition, as these cases can be easily decided in linear (check $\mathcal{G} \subseteq \mathcal{I}$) and constant time (answer "Yes"), respectively.

Complexity Analysis of `WIDTH`

In this section, we prove that `WIDTH` $\in$ PSPACE-complete. To do so, we first need to establish some preliminary results.

First, it is well known, from the definition of PSPACE complexity, that providing a PSPACE oracle to a PSPACE machine does not increase its computational power.

Lemma 1. $PSPACE^{PSPACE} \subseteq PSPACE$.

Proof. We need to simulate an $f(n)$ space deterministic Turing machine $\mathcal{M}$ which can have access to PSPACE oracles by a $g(n)$ space two-taped deterministic Turing machine $\mathcal{M}'$ with no oracle, where $f(n)$ and $g(n)$ are both polynomials.

Let the input be x with $|x| = n$. When computed by $\mathcal{M}$, let $q_1, q_2, \ldots,$ be the queries and each query can be computed using $h(n)$ space where h is a polynomial.

$\mathcal{M}'$ simulates $\mathcal{M}$ on input x using at most $f(n)$ space in its first tape. Whenever a query is made, $\mathcal{M}'$ pauses the simulation of $\mathcal{M}$, erases any content on the second tape and uses $h(n)$ space to simulate the run of the query and get the "yes" or "no" answer. It resumes the simulation of $\mathcal{M}$ on the first tape. Since the space of the second tape is reused, $\mathcal{M}'$ uses in total $g(n) = h(n) + f(n)$ space, which is a polynomial in terms of the input size. □

[2]Based on the definition of width, the maximum width for any solvable planning instance will be n where n is the number of atoms of the instance. This is because each atomic conjunction can only contain at most n atoms, therefore the size of any admissible atomic set will be no more than n.

Decision Problem	**Computational Complexity**
WIDTH (non-parameterised)	PSPACE-complete (Propositions 1 and 2 and Theorem 1)
WIDTH-k (parameterised)	coNP-complete (Propositions 3 and 4 and Theorem 2)

Table 1: Summary of Complexity Results.

To prove membership, we utilise Savitch (1970)'s result on PSPACE and Bylander (1994)'s result that PLANMIN (does a planning instance have a plan of length at most k?) is PSPACE-complete.

Proposition 1. *WIDTH is in PSPACE.*

Proof. We can decide WIDTH by running the following non-deterministic Algorithm 1, which uses PLANMIN (in PSPACE) as an oracle and $\Pi[\phi]$ to denote the planning problem that is the same as the input planning problem Π with the goal substituted by a formula ϕ over the set of atoms $\mathcal{F}$.

Algorithm 1: Decide whether $w(\Pi) \leq k$

Require: $\Pi = \langle \mathcal{F}, \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$ and $1 \leq k \leq |\mathcal{F}|$
1: ▷ ***Phase 1: find plan length*** ◁
2: Query oracles in PLANMIN to find the optimal plan length ℓ for Π (note that $\ell = |\mathcal{A}||\mathcal{F}|^{2k-1} + 1$ if no valid plan for Π exists).
3: **if** $\ell \leq 1$ **then ACCEPT** ▷ *width 0 or 1*
4: **if** $\ell = |\mathcal{A}||\mathcal{F}|^{2k-1} + 1$ **then REJECT** ▷ *No plan*
5: ▷ ***Phase 2: find goal admissible atomic sequence*** ◁
6: Guess an atomic sequence $\tau = (t_0, \ldots, t_\ell)$ such that $t_i \subseteq \mathcal{F}$ and $|t_i| \leq k$ for each $i = 0, 1, \ldots, \ell$.
7: **if** $s_0 \not\models t_0$ **then REJECT** ▷ *check initial atomic conj.*
8: **for** $1 \leq i \leq \ell$ **do** ▷ *check optimality of* t_i
9: Query PLANMIN oracles to find the optimal plan length ℓ_i for $\Pi[t_i]$
10: **if** $\ell_i \neq i$ **then REJECT** ▷ *wrong* t_i *in* τ
11: **for** $1 \leq i < \ell$ **do** ▷ *check if* t_i *extends to* t_{i+1}
12: Query a non-deterministic (polynomial space) oracle which does the followings in each branch:
1. Guess a state s which is reachable by i executable actions in Π (i.e., simulate i steps)
2. **if** $s \not\models t_i$ **then REJECT**
3. **if** $s \models t_i$ and there exists an applicable action in s that yields a state s' such that $s' \models t_{i+1}$ **then REJECT**
4. **else ACCEPT**
13: **if** any branch of the oracle accepts **then REJECT**
14: **ACCEPT**

While most of the algorithm is self explanatory, lines 12-13 are used to check that all optimal plans to t_i can be extended to t_{i+1}, by rejecting (in line 13) if an optimal plan exists to t_i that cannot be further extended to reach t_{i+1} by one single action (as shown in step 4 in line 12). Note that step 1 in line 12 only requires polynomial space as it does not store the sequence of actions and instead it simulates the actions one by one.

Algorithm 1 can be run in $\mathsf{NPSPACE}^{\mathsf{NPSPACE}}$ since PLANMIN (used in lines 2 and 9) is PSPACE-complete (Bylander 1994) and Line 12 is in NPSPACE as argued above. Since NPSPACE = PSPACE (Savitch 1970), we have $\mathsf{NPSPACE}^{\mathsf{NPSPACE}} = \mathsf{PSPACE}^{\mathsf{PSPACE}}$ and from Lemma 1, it follows that WIDTH $\in$ PSPACE. □

Next, we prove hardness by a reduction from PLANSAT.

Proposition 2. *WIDTH is PSPACE-hard.*

Proof. We can check whether any planning instance Π has a valid plan by simply checking whether $w(\Pi) \leq n$, where n is the number of atoms in Π—a polynomial reduction from PLANSAT to WIDTH. If $w(\Pi) \leq n$, Π has a plan as per Definition 4; otherwise, Π is unsolvable. Since PLANSAT is PSPACE-complete, then the thesis follows. □

From Propositions 1 and 2, we get a tight bound for WIDTH.

Theorem 1. *WIDTH is PSPACE-complete.*

Thus, determining whether the (exact) width of a planning instance is, in general, as hard as solving the planning instance itself. This result justifies the use of approximations to width in practical planning algorithms (Singh et al. 2021; Rosa and Lipovetzky 2024) and an analysis of width for a fixed k, which we do next.

Complexity Analysis of WIDTH-k

In this section, we focus on the parameterised version WIDTH-k of the width computation problem (Definition 6).

Proposition 3. *WIDTH-k is in coNP, for* $k \in \mathbb{N}$.

Proof. In order to show the result, we prove that the complement of WIDTH-k is in NP. This complement is the problem coWIDTH-k defined as follows:

- Input: A planning instance Π.
- Question: Does Π have a width of at least $k + 1$?

Let Π be the given planning instance. Given an admissible sequence σ, we call (t, t') an *adjacent pair* if t immediately precedes t' in σ. Recall that $Adm(\Pi)_k$ is the set of admissible sequences with size no greater than k.

We now describe a nondeterministic polynomial-time algorithm that decides coWIDTH-k for input Π.

The key idea is the following: on input Π, we non-deterministically compute an over-approximation of $Adm(\Pi)_k$. If none of these over-approximations make the goal true, i.e., none of the sequences in an over-approximation is goal admissible, we know that the width must be greater than k and we accept. Otherwise we reject.

We now describe this nondeterministic algorithm.

First, run the IW(k) algorithm (Lipovetzky and Geffner 2012) on Π, but ignoring the goal. (So we continue the pruned breadth-first search until no further states are generated, without testing if a generated state satisfies the goal

condition.) We can use an arbitrary tie-breaking/action ordering strategy. Let S_k be the set of all states that are generated and not pruned by the algorithm. For all $s \in S_k$, let $d(s)$ denote the depth at which it is generated. For example, $d(\mathcal{I}) = 0$ for the initial state $\mathcal{I}$, and $d(s) = 1$ for all successors of $\mathcal{I}$ that are not pruned by IW(k). This computation requires runtime that is polynomial in the task size and exponential in k (but k is a constant).

Let t be an atomic conjunction of size no greater than k. Define $d(t)$ as the minimum depth in which t occurs among the states generated by IW(k), that is, $d(t) = \min_{s \in S_k, s \models t} d(s)$, where the minimum over the empty set is ∞. Let $d^*(t)$ denote the length of an optimal plan to a state satisfying t, again using ∞ if no such plan exists.[3]

From the optimality proof of IW(k) for Π with width k in the work of (Lipovetzky and Geffner 2012), we know the following: if t is an atomic conjunction that occurs in a sequence of Π, then IW(k) finds an optimal plan on our input instance with the goal set to t, and therefore $d(t) = d^*(t)$. We also know that, unconditionally, $d(t) \geq d^*(t)$ because $d(t)$ is the length of a plan that reaches t (that was found by IW(k)), and such a plan cannot be shorter than an optimal plan for reaching t.

Now we try to build a graph (T, E), where the vertex set T is the set of atomic conjunctions with $d(t) < \infty$ and (t, t') is a directed edge in the edge set E if (t, t') are an adjacent pair in an admissible sequence.

From the preceding discussions, we know the following for all atomic conjunctions of Π:

- If t is in an admissible sequence, $t \in T$ and $d(t) = d^*(t)$.

In particular, this shows that the set T is an over-approximation as follows:

1. Guess a subset $T' \subseteq T$.
2. For each $t \in T'$:
   (a) Guess an action sequence π_t of length strictly less than $d(t)$. (If this is not possible because $d(t) = 0$, skip t.)
   (b) If π_t is applicable in the initial state and results in a state that satisfies t, remove t from T.

These computations can be performed in nondeterministic polynomial time.

No matter how the nondeterminism resolves, after these steps, T still contains an over-approximation of $\mathit{Adm}(\Pi)_k$: the only nodes we remove are ones where we have a witness that t can be reached with plans of length strictly less than $d(t)$, which proves $d(t) \neq d^*(t)$, which implies that t is not part of $\mathit{Adm}(\Pi)_k$.

Moreover, there exists a nondeterministic execution path where all atomic conjunctions in T now satisfy $d(t) = d^*(t)$ (guess T' to contain exactly the atomic conjunctions violating this condition, and for each such atomic conjunction t guess π_t as an optimal plan for goal t).

Next, we over-approximate the adjacent pairs of atomic conjunctions in each admissible sequence of $\mathit{Adm}(\Pi)_k$.

[3]Our algorithm computes function $d(t)$, but does not have access to $d^*(t)$. We just define $d^*(t)$ to prove the correctness of the algorithm.

Initially, we set $E = \{(t, t') \mid d(t') = d(t) + 1\}$. As argued before, for all t included in $\mathit{Adm}(\Pi)_k$, we have $t \in T$ and $d(t) = d^*(t)$. In particular, this means that E includes all pairs (t, t') where t and t' are included in some sequence in $\mathit{Adm}(\Pi)_k$, and $d^*(t') = d^*(t)+1$. It is easy to see from the definition of admissible sequence in Definition 1 that every adjacent atomic conjunctions (t, t') satisfy $d^*(t') = d^*(t) + 1$, and hence it follows that E includes all adjacent pairs (t, t') of $\mathit{Adm}(\Pi)_k$, and that for all such pairs $d(t) = d^*(t)$ and $d(t') = d^*(t')$. (This is important because the algorithm has access to d, but not to d^*.)

We now non-deterministically tighten E:

1. Guess a subset $E' \subseteq E$.
2. For each $(t, t') \in E'$:
   (a) Guess an action sequence π_t of length exactly $d(t)$.
   (b) If π_t is applicable in the initial state and results in a state that satisfies t, and there does not exist a 1-step extension of π_t that satisfies t', remove (t, t') from E.

Once again, these computations can be performed in nondeterministic polynomial time.

No matter how the nondeterminism resolves, after these steps, E still contains an over-approximation of the adjacent pairs in $\mathit{Adm}(\Pi)_k$: if we select an adjacent pair (t, t') that belongs to $\mathit{Adm}(\Pi)_k$ and hence must not be removed, then we know $d(t) = d^*(t)$ and therefore the guessed action sequence π_t is an optimal plan for t if it is applicable in the initial state and results in a state that satisfies t. But then from the definition of admissible set, there exists a 1-step extension that is an optimal plan for t' (and hence satisfies t'), so this pair is not pruned.

On the other hand, there exists a nondeterministic execution path where after this step, all pairs $(t, t') \in E$ now satisfy the condition that all optimal plans for t can be 1-step extended to optimal plans for t'. To see this, consider a scenario where $d(t) = d^*(t)$ for all $t \in T$ before this step. (As discussed before, there exists a nondeterministic execution path that achieves this.) Then guess E' to contain all pairs (t, t') where not all optimal plans for t can be 1-step extended to optimal plans for t'. Then for each $(t, t') \in E'$, guess π_t as an optimal plan for t (which must have length $d^*(t) = d(t)$) that cannot be 1-step extended to reach t'. Then (t, t') gets removed from E, as desired.

At this point, it may be the case that the graph (T, E) contains atomic conjunctions that are not reachable from a root in the graph (an atomic conjunction satisfied by the initial state), while $\mathit{Adm}(\Pi)_k$ by definition only includes atomic conjunctions that are reachable from a root. So we can further prune (T, E) to the set of atomic conjunctions (and edges between these atomic conjunctions) that can be reached from a root atomic conjunction by the edges in E. Again, this clearly retains the property that (T, E) is an over-approximation of $\mathit{Adm}(\Pi)_k$, and it is easy to see that on an execution path where T was pruned to exactly the atomic conjunctions with $d(t) = d^*(t)$ and E was pruned to exactly the edges that satisfy the 1-step optimal extension property, the resulting graph is exactly $\mathit{Adm}(\Pi)_k$.

As the final step of the algorithm, we check if any atomic conjunction in T covers the goal. If not, we accept: we

have found an over-approximation of $\mathit{Adm}(\Pi)_k$ that does not cover the goal, which shows that the width of the task is strictly larger than k. Otherwise we reject.

If all paths reject, this means that all over-approximations of $\mathit{Adm}(\Pi)_k$ produced on any execution path cover the goal. Because there exists an execution path where the computed over-approximation is $\mathit{Adm}(\Pi)_k$ itself, this implies that $\mathit{Adm}(\Pi)_k$ covers the goal and therefore the width of the planning task is indeed at most k.

Putting the pieces together, at least one execution path accepts iff the width of the task is strictly larger than k, and hence this is indeed a correct nondeterministic algorithm for deciding coWIDTH-k. □

Next, we prove a hardness result for WIDTH-k. Remember that UNSAT, the problem of checking whether a given Boolean formula in CNF is unsatisfiable, is a well-known coNP-complete problem (Arora and Barak 2009).

Proposition 4. *WIDTH-k is coNP-hard, for $k \in \mathbb{N}$.*

Proof. Consider a fixed $k \in \mathbb{N}$. We reduce UNSAT to WIDTH-k as follows. To that end, given a (CNF) Boolean formula, we shall build a planning instance $\Pi_\phi^{\texttt{UNSAT}}$ with the following properties:

P1. $|\Pi_\phi^{\texttt{UNSAT}}| \leq |\phi|^{\mathcal{O}(1)}$—the size of Π_ϕ is upper bounded by a polynomial in the size of ϕ; and
P2. formula ϕ is unsatisfiable *if and only if* $w(\Pi_\phi) \leq k$.

As illustrated in Figure 1, the planning problem $\Pi_\phi^{\texttt{UNSAT}}$ is obtained by putting together two "sub-planning" instances, namely, the instance $\Pi_\phi = \langle \mathcal{F}_\phi, \mathcal{I}_\phi, \mathcal{A}_\phi, \mathcal{G}_\phi \rangle$ and the instance $\Pi_k = \langle \mathcal{F}_k, \mathcal{I}_k, \mathcal{A}_k, \mathcal{G}_k \rangle$.

The sub-planning instance Π_ϕ. Let n and m be the number of variables and m clauses in ϕ, resp. Planning instance Π_ϕ will satisfy the following properties:

1. Π_ϕ can be constructed in polynomial time w.r.t $|\phi|$.
2. The goal of Π_ϕ is a set of atoms $\{d_1, \ldots, d_k\}$.
3. If ϕ is unsatisfiable, then Π_ϕ will be unsolvable.
4. If ϕ is satisfiable, then Π_ϕ will be solvable and will admit an optimal plan of length exactly $n + 2^k$.

Let $V = \{X_1, \ldots, X_n\}$ and $C = \{c_1, \ldots, c_m\}$, with $n, m \geq 1$, be the sets of variables and clauses of ϕ, resp. Let $L = V \cup \{\neg X \mid X \in V\}$ be the set of literals over V. We define planning instance $\Pi_\phi = \langle \mathcal{F}_\phi, \mathcal{I}_\phi, \mathcal{A}_\phi, \mathcal{G}_\phi \rangle$, where:

$$\begin{aligned}
\mathcal{F}_\phi &= \{\mathit{assigned}_1, \mathit{unassigned}_i \mid 1 \leq i \leq n\} \cup \\
&\quad \{\mathit{satisfied}_i \mid 1 \leq i \leq m\} \cup \{d_i, nd_i \mid 1 \leq i \leq k\}; \\
\mathcal{I}_\phi &= \{\mathit{unassigned}_i \mid 1 \leq i \leq n\}; \\
\mathcal{A}_\phi &= \{\mathit{assign}_\ell \mid \ell \in L\} \cup \{\mathit{start}_\phi\} \cup \\
&\quad \{\mathit{enum}_j \mid 1 \leq j \leq 2^k - 1\}; \text{and} \\
\mathcal{G}_\phi &= \{d_i \mid 1 \leq i \leq k\}.
\end{aligned}$$

Intuitively, the problem is solvable if we can first make ϕ satisfiable via actions assign_ℓ, which would enable action start_ϕ so as to run a binary enumeration process of length $2^k - 1$ that ends with all atoms d_i in the goal $\mathcal{G}_\phi$ being true.

Let us now describe the operators in Π_ϕ. Action assign_ℓ, where $\ell \in \{X_i, \neg X_i\}$ for some $i \leq n$, sets the truth value of ℓ to true. Its precondition is $\{\mathit{unassigned}_i\}$ and its post-condition is $\{\neg\mathit{unassigned}_i, \mathit{assigned}_i\} \cup \{\mathit{satisfied}_j \mid \ell \in c_j\}$, making true all clauses c_j containing literal ℓ.

The action start_ϕ signals the start of a binary enumeration process once all clauses are satisfied. Its precondition is $\bigcup_{i=1}^{n}\{\mathit{assigned}_i\} \cup \bigcup_{i=1}^{m}\{\mathit{satisfied}_i\}$ and its post-condition is $\{nd_i \mid 1 \leq i \leq k\}$.

Actions enum_j, for $j \in \{1, \ldots, 2^k - 1\}$, together with atoms d_i and nd_i, are used to implement a binary enumeration process $1, 10, 11, 100, 101, \ldots, \underbrace{11\ldots1}_{k \text{ copies of } 1}$ as a sequence of true atoms $\{d_1\}, \{d_2\}, \{d_2, d_1\}, \{d_3\}, \{d_3, d_1\}, \ldots, \{d_k, d_{k-1}, \ldots, d_1\}$. Intuitively, this enumeration is used to yield a particular width for the problem. Atom nd_i (or d_i) indicates that the i-th least significant digit is 0 (or 1). The truth value of *all* atoms d_i and nd_i together denote the current number in the enumeration process in binary format—e.g., if $k = 4$, state $\{d_1, d_3, nd_2, nd_4\}$ denotes number 0101 (or 5 in decimal). For legibility, let $\gamma(j)$ be the set of d_i and nd_i atoms representing the binary representation of decimal number j.

Action enum_j enumerates decimal number j in binary format through atoms d_i: its precondition is $\gamma(j-1)$ and its post-condition is $\gamma(j)$.

There are in total $2n + m + 2k$ atoms and $2n + 2^k$ actions in Π_ϕ. Since k is a fixed number, the size of Π_ϕ is upper bounded by a polynomial in terms of n and m—the first property is satisfied.

Now, to achieve the goal $\mathcal{G}_\phi$, the enumeration process must be realised in full through the action sequence:

$$(\mathit{enum}_1, \mathit{enum}_2, \ldots, \mathit{enum}_{2^k-1}).$$

This sequence has $2^k - 1$ enum_j actions. The key is that, in order to start this enumeration sequence, the action start_ϕ must be first applied (to enable enum_1), and for it to be enabled, all clauses in ϕ must have been satisfied (through appropriate assign_ℓ actions). So, if ϕ is unsatisfiable, start_ϕ can never become applicable and the third requirement is therefore fulfilled.

If ϕ is satisfiable, then there is an optimal plan solving the instance with $n + 2^k$ actions: n for a satisfying assignment for ϕ, one for action start_ϕ, and $2^k - 1$ enum_j actions. Therefore, the the fourth property holds.

The sub-planning instance Π_k. We require the constructed Π_k to satisfy the following properties:

1. The number of atoms and actions in Π_k is upper bounded by a polynomial of n, the number of variables in ϕ.
2. The goal of Π_k is a set of atoms $\{d_1, \ldots, d_k\}$.
3. Π_k admits an optimal plan of length $n + 2^k$.
4. The width of Π_k is k, that is, $w(\Pi_k) = k$.

We define planning instance $\Pi_k = \langle \mathcal{F}_k, \mathcal{I}_k, \mathcal{A}_k, \mathcal{G}_k \rangle$,

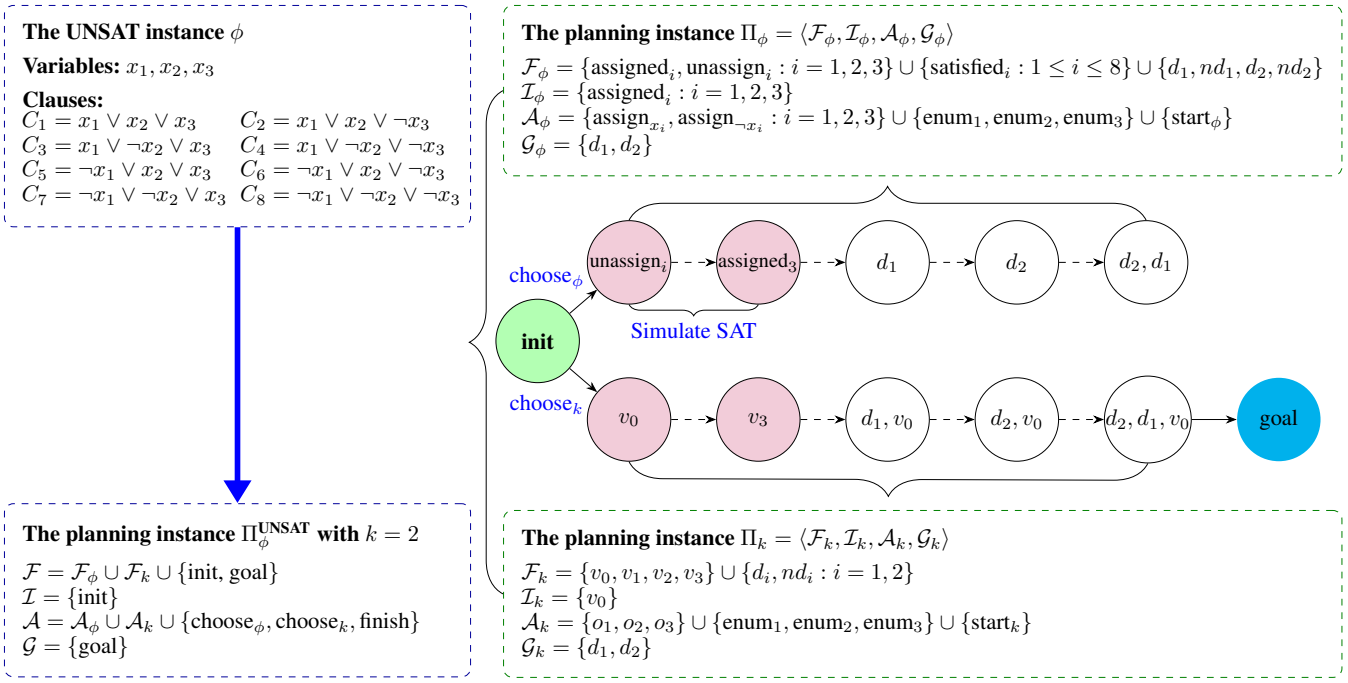

Figure 1: An illustrative example of the reduction of UNSAT to WIDTH-2 for the hardness proof in Proposition 4. The Π_ϕ^{UNSAT} (left hand side) is built from sub-planning instances Π_ϕ and Π_k (right hand side).

where:

$$\begin{aligned}
\mathcal{F}_k =& \{v_i \mid 0 \le i \le n\} \cup \{d_i, nd_i \mid 1 \le i \le k\};\\
\mathcal{I}_k =& \{v_0\};\\
\mathcal{A}_k =& \{o_i \mid 1 \le i \le n\} \cup \{\mathit{start}_k\} \cup\\
& \{\mathit{enum}_j \mid 1 \le j \le 2^k - 1\}; \text{and}\\
\mathcal{G}_k =& \{d_i \mid 1 \le i \le k\}.
\end{aligned}$$

The action o_i has precondition $\{v_{i-1}\}$ and post-condition $\{v_i\}$. The action start_k has precondition $\{v_n\}$ and post-condition $\{nd_i \mid 1 \le i \le k\}$. Like action start_ϕ, it signals the start of the binary enumeration process, through the actions enum_j which are defined exactly as in Π_ϕ above.

There are in total $n + 1 + 2k$ atoms and $n + 2^k$ actions. Since k is a constant, the number of atoms and actions is upper bounded by a polynomial in n, and the first property holds. The second property holds by construction of Π_k.

We next verify the third property. There are n actions to make v_n true from the initial state, one action start_k to start the binary enumeration process, and $2^k - 1$ actions enum_j as before. So, the optimal plan is of length $n + 2^k$.

Finally, it is easy to check that a goal admissible atomic sequence with the smallest size is as following.

$$\begin{aligned}
\sigma = \Big(& t_1 = \{v_0\}, t_2 = \{v_1\}, \ldots, t_{n+1} = \{v_n\}, t_{n+2} = \{nd_1\},\\
& t_{n+3} = \{d_1\}, t_{n+4} = \{d_2\}, t_{n+5} = \{d_2, d_1\},\\
& t_{n+6} = \{d_3\}, \ldots, t_{n+2^k+1} = \{d_k, d_{k-1}, \ldots, d_1\}\Big).
\end{aligned}$$

From Definition 4, we know the width of Π_k is k and thus Property 4 holds.

The planning instance Π_ϕ^{UNSAT}. Now we combine the above two sub-planning instances to construct the classical planning instance $\Pi = \langle \mathcal{F}, \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$, where:

$$\begin{aligned}
\mathcal{F} =& \mathcal{F}_\phi \cup \mathcal{F}_k \cup \{\mathit{init}, \mathit{goal}\};\\
\mathcal{I} =& \{\mathit{init}\};\\
\mathcal{A} =& \mathcal{A}_\phi \cup \mathcal{A}_k \cup \{\mathit{choose}_\phi, \mathit{choose}_k, \mathit{finish}\}; \text{ and}\\
\mathcal{G} =& \{\mathit{goal}\}.
\end{aligned}$$

Both actions choose_ϕ and choose_k have precondition $\{\mathit{init}\}$. The post-condition of choose_ϕ is $\{\neg\mathit{init}\} \cup \mathcal{I}_\phi$ and the post-condition of choose_k is $\{\neg\mathit{init}\} \cup \mathcal{I}_k$. Informally, these two actions select which sub-instance needs to be solved, namely, Π_ϕ or Π_k, resp.

The action finish has precondition $\{d_1, \ldots, d_k, v_0\}$ and post-condition $\{\mathit{goal}\}$—the only one achieving the goal.

First of all, there are in total $(3n + m + 3 + 4k)$ atoms and $(3n + 2^{k+1} + 3)$ actions in Π_ϕ^{UNSAT}. Since k is a constant, the size of Π_ϕ^{UNSAT} is upper bounded by a polynomial in the size of ϕ and thus Property P1 holds.

Since action finish is the only one producing the goal, all atoms $d_1, \ldots, d_k$ must be achieved before executing finish. We can potentially achieve all atoms $d_1, \ldots, d_k$ in two different ways:

1. execute choose_ϕ and then solve sub-instance Π_ϕ; or
2. execute choose_k and then solve sub-instance Π_k.

The key here is that *only* in the second case, we can extend the plan that achieves all $d_1, \ldots, d_k$ to a plan for goal because finish has v_0 as an additional precondition, and v_0 is only true if we solved Π_k and not Π_ϕ.

Next, we show that Property P2 holds. More specifically,

- if ϕ is unsatisfiable, $w(\Pi) \leq k$; and
- if ϕ is satisfiable, $w(\Pi) > k$.

First, we consider the case where ϕ is unsatisfiable. Then there exists no plan that achieves all atoms $d_1, \ldots, d_k$ after starting with $choose_\phi$. From the construction of Π_k, we know that σ can be extended to a goal admissible sequence σ' of $\Pi_\phi^{\texttt{UNSAT}}$ by adding an atomic conjunction $t_{n+2^k+2} = \{goal\}$ and the size of this goal admissible sequence is still k. Therefore, the width of $\Pi_\phi^{\texttt{UNSAT}}$ is no greater than k.

Next, it is not difficult to see that, due to the construction of $\Pi_\phi^{\texttt{UNSAT}}$, the second last atomic conjunction t in any admissible sequence must contain set $\{d_1, \ldots, d_k\}$. This is because very optimal plan to such second last tuple, must be extandable with one action to reach $\mathcal{G} = \{goal\}$ optimally. If any d_i is missing in t, then optimal plans to t will not enable action *finish* as such action requires all d_i to be true.

Now, consider the case when ϕ is satisfiable. We know that in Π_ϕ there exists an optimal plan for the atomic conjunction $\{d_1, \ldots, d_k\}$ that will not make *any* atom v_i true and thus will never be able to enable action *finish*, the only one achieving the goal $\{goal\}$. Therefore, in any goal admissible sequence, there must be a tuple that contains all atoms d_i (as argued above), but with size *greater* than k, that is, it must contain additional atoms beyond $d_1, \ldots, d_k$. Based on the definition of width, the width of $\Pi_\phi^{\texttt{UNSAT}}$ must be strictly greater than k.

From above, we know that $\Pi_\phi^{\texttt{UNSAT}}$ satisfies both properties P1 and P2, and the constructed instance of `WIDTH-k` is a valid polynomial-reduction from the instance in `UNSAT`. Since `UNSAT` is coNP-complete, we conclude that `WIDTH-k` is a coNP-hard problem. □

From Propositions 3 and 4, we get the following completeness result.

Theorem 2. *`WIDTH-k` is coNP-complete.*

Conclusion

In this paper, we have analysed the computational complexity of finding the width of classical planning problems. We focused on both the non-parameterised and parameterised versions and showed that the non-parameterised width computation is PSPACE-complete and the parameterised version is coNP-complete.

There are planning algorithms that are able to exploit the width of problems. However, no technique to compute the *exact* width of planning instances exists and, until now, it was not even known how difficult that computation 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., no easier than classical planning itself (Bylander 1994) in the worst case). This is significant because it can inform future research agendas on *how to compute and exploit the notion of width*.

One possibility is to develop algorithms that can *approximate* the width of a problem. As with delete-relaxation (Bylander 1994), which is also hard in its optimal version but useful practical approximations have been proposed, approximate width values can be very useful *in practice*, for example, to indicate what versions of width-based planners one needs to (not) use. We would like to further develop this insight in our complexity results and gain a better understanding on the possible approximation of width computation in classical planning.

Another avenue worth exploring is the development of a computational framework that allows us to *generalise* the width value across different planning instances within the same domain. Such width results have been reported over several domains with analytical proofs and no computational framework for special cases where goals were expressed using single atoms (Lipovetzky 2012). 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 supported by the Commonwealth through an Australian Government Research Training Program Scholarship [DOI: https://doi.org/10.82133/C42F-K220] and also by the University of Melbourne Faculty of Engineering and Information Technology Ingenium Scholarship Award.

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, Applications (OPTIMA), Project ID IC200100009, and by the Swiss National Science Foundation (SNSF) as part of the project "Unifying the Theory and Algorithms of Factored State-Space Search" (UTA).

References

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

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

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 *Proc. of AAAI*, 809–814.

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.

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. 2022. Planning with Perspectives - Decomposing Epistemic Planning using Functional STRIPS. *Journal of Artificial Intelligence Research (JAIR)*, 75: 489–539.

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

Lipovetzky, N. 2012. *Structure and inference in classical planning*. Ph.D. thesis, Pompeu Fabra University, Spain.

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.

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.

Rosa, G.; and Lipovetzky, N. 2024. Count-Based Novelty Exploration in Classical Planning. In *Proc. of ECAI*, volume 392, 4181–4189.

Savitch, W. J. 1970. Relationships Between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences*, 4(2): 177–192.

Singh, A.; Lipovetzky, N.; Ramírez, M.; and Segovia-Aguas, J. 2021. Approximate Novelty Search. In *Proc. of ICAPS*, 349–357.