All PSPACE-Complete Planning Problems Are Equal but Some Are More Equal than Others

Christer Bäckström and **Peter Jonsson**
Department of Computer Science, Linköping University
SE-581 83 Linköping, Sweden
christer.backstrom@liu.se peter.jonsson@liu.se

Abstract

Complexity analysis of planning is problematic. Even very simple planning languages are **PSPACE**-complete, yet cannot model many simple problems naturally. Many languages with much more powerful features are also **PSPACE**-complete. It is thus difficult to separate planning languages in a useful way and to get complexity figures that better reflect reality. This paper introduces new methods for complexity analysis of planning and similar combinatorial search problems, in order to achieve more precision and complexity separations than standard methods allow. Padding instances with the solution size yields a complexity measure that is immune to this factor and reveals other causes of hardness, that are otherwise hidden. Further combining this method with limited nondeterminism improves the precision, making even finer separations possible. We demonstrate with examples how these methods can narrow the gap between theory and practice.

1 Introduction

Results on complexity in planning have often been perceived as dissapointing, or even irrelevant, by many AI researchers. This is understandable. Almost everything is **PSPACE**-complete, or worse. Even the simple language of propositional STRIPS is **PSPACE**-complete (Bylander 1994), yet often considered too restricted for modelling interesting problems. On the other hand, most benchmark problems for planning are **NP**-complete or even tractable (Helmert 2006), and many planners work well for those problems. It is hardly surprising if many researchers feel that complexity theory has failed to deliver for planning.

Since STRIPS is **PSPACE**-complete it has the same power as a polynomial-space Turing machine, which is a very powerful device. STRIPS is thus not restricted primarily in computational power, but in expressive power; even if it has the power to solve difficult problems, modelling those problems is not straightforward. Of course, programming a Turing machine is not very straightforward either. It is thus important to distinguish between the computational power of a language and its expressive power. A notable contribution to this is Nebel's (2000) results on using compilation techniques to show separations in expressive power between certain languages that are all **PSPACE**-complete.

A major reason why planning and similar problems are **PSPACE**-complete is that they can be viewed as finding a path in an implicitly represented graph, which is exponentially larger than its representation. This is contrary to the usual assumptions in complexity theory and causes problems, as observed by Balcázar (1996):

> "The standard complexity classes of complexity theory do not allow for direct classification of most of the problems solved by heuristic search algorithms".

Since the graph is of exponential size also the shortest paths are of exponential length in the worst case. This reflects upon the complexity, even if many applications never have instances with such solutions. Planning is thus **PSPACE**-complete already for very simple actions, but also remains so even for quite powerful actions. That is, the class of **PSPACE**-complete problems allows a vast and diverse flora of languages that cannot easily be further separated. We have a spectrum of different reasons why such planning languages are hard, ranging from those where the hard instances have very long plans of simple action to those where the hard instances have short plans of very powerful actions.

The main purpose of this paper is to introduce *new methods* for complexity analysis of planning; methods that give additional information and reveal more details than the standard methods do. While our focus is on planning, it should be obvious that the methods we present are equally useful for many other hard combinatorial problems, for example, model checking. The results in this paper should be viewed primarily as examples of applying these methods. The main contribution is the methods themselves.

The paper is organized as follows. Section 2 is a brief overview of central concepts in complexity theory that are important for the rest of the paper, and Section 3 introduces our framework for planning. In Section 4 we demonstrate why both traditional and parameterized complexity analysis seem inadequate to capture important aspects of planning. In Section 5 we thus introduce the new method of analysing length-padded instances. This results in a hierarchy of language classes with different complexity, which are are all **PSPACE**-complete under traditional analysis. Section 6 gives examples of applying this method, for instance, demonstrating a corresponding hierarchy of fragments of PDDL. We also compare our separation results to Nebel's. In Section 7 we refine the method further, combining it with

limited non-determinism to allow proving even tighter complexity bounds. Finally, in Section 8, we demonstrate the power of this refined method, applying it to partial-order planning as an example. This gives us sufficient precision to conjecture that partial-order planning is harder than total-order planning, which could have implications also for scheduling. Rather than being collected at the end, all discussion and related work appears att appropriate places in the paper, in its proper context.

2 Complexity Theory

This paper is technical and makes heavy use of complexity theory. As a service to the reader we thus start with a very brief reminder of some central concepts, presented in a way consistent with how we use them. First some notation, though; we use $|x|$ to denote the *cardinality* (the number of elements) of x and $||x||$ to denote the *size* (the number of bits) of the *representation* of x.

In complexity theory a *problem* is a general question, having a set of *instances* we may ask it for. An *algorithm* for the problem is a method to answer this question for all instances. An important distinction is the one between analysing the complexity of a particular algorithm (*algorithm complexity*) and analysing the complexity of the problem itself (*problem complexity*). In the latter case we must take all imaginable algorithms into account, not only the ones we already know.

Problem complexity mainly studies *decision problems*, which do not ask for a solution but only whether one exists or not. This is to simplify the theory, and usually, but not always, there is a tight correspondence between generating a solution and deciding if there is one. The complexity of a problem is measured as a function of the instance size. When the instance is (or contains) an integer n, the complexity should thus be a function of $||n|| = \log n$ rather than of n. This subtle but important detail is central to this paper.

A *non-deterministic Turing Machine (NTM)* is like an ordinary *deterministic Turing machine (DTM)* but with the added capability of *guessing*. An NTM first guesses a string; then continues like a DTM, but with read-only access to the string it guessed. It always guesses a correct string whenever such a string exists.

P (or **PSPACE**) is the class of all decision problems that can be solved by some DTM in polynomial time (or space), while **NP** consists of all decision problems that can be solved in polynomial time by some NTM. Since an NTM always guesses correctly, it can solve an instance by guessing a string and then check whether this is a *witness* for a solution (the witness can itself be a solution). If it runs in polynomial time, then the witnesses must be of polynomial size and verifiable in polynomial time. Hence, we can also say that a problem is in **NP** if and only if its witnesses can be verified in polynomial time.

The NTM is a theoretical construction, but it has a very direct relationship to *search*. Consider some combinatorial problem and some instance with n-bit solutions. Solving this by search may require examining the whole *search space* of 2^n strings. If it takes $f(n)$ time to check a string, then it takes $O(2^n f(n))$ time to solve the instance. In the worst case, this coincides with the lower bound. An NTM can instead guess a witness string and then verify this. A DTM can simulate this, 'guessing' by searching through all n-bit strings, which also takes $O(2^n f(n))$ time. Search and NTMs are thus two different ways to describe the same thing, but with one important difference: the NTM does not prescribe how to guess, thus making *no assumption that search is necessary*. Basing complexity classes like **NP** upon NTMs thus allows for the possibility of more efficient, yet unknown, algorithms.

Proving that a problem is hard for a complexity class requires a *reduction* concept. A *polynomial many-one reduction* from a decision problem X to another decision problem Y is denoted $\mathsf{X} \leq_m^p \mathsf{Y}$, and defined as a polynomial-time function f that maps every instance $\boldsymbol{x}$ of X to an instance $f(\boldsymbol{x})$ of Y that has the same answer as $\boldsymbol{x}$. Common classes, like **P**, **NP** and **PSPACE**, are closed under such reductions. That X is *complete* for such a class C means that: (a) X is in C and (b) $\mathsf{Y} \leq_m^p \mathsf{X}$ for some Y already known to be complete for C. Problem X is *hard* for C if it satisfies (b).

An *oracle TM* M^C is a DTM (or NTM) M with an *oracle* C for a certain problem or complexity class. Whenever M queries the oracle it immediately receives a correct answer. Similarly, $\mathbf{P}^C$ (or $\mathbf{NP}^C$) is the class of all problems that can be solved in polynomial time on a DTM (or NTM) with an oracle for C. The *polynomial hierarchy* is a sequence of complexity classes $\mathbf{\Sigma}_k^p$ within **PSPACE** s.t. $\mathbf{\Sigma}_0^p = \mathbf{P}$, $\mathbf{\Sigma}_1^p = \mathbf{NP}$, $\mathbf{\Sigma}_2^p = \mathbf{NP}^{\mathbf{NP}}$, $\mathbf{\Sigma}_3^p = \mathbf{NP}^{\mathbf{\Sigma}_2^p}$ etc.

It is known that $\mathbf{NP} \subseteq \mathbf{PSPACE}$, but not whether this inclusion is proper. However, if $\mathbf{NP} \neq \mathbf{PSPACE}$, then every **PSPACE**-complete problem must have infinitely many instances s.t. for all its witnesses w either (a) $||w||$ is polynomial but w is not verifiable in polynomial time or (b) $||w||$ is not polynomial. The negative results for compressing exponential-length plans in the general case (Bäckström and Jonsson 2011) suggest that case (b) occurs in planning (although case (a) may also occur). This is another view of why STRIPS planning is **PSPACE**-complete. As a consequence, transforming planning to some problem in **NP**, like SAT, does not make it simpler. Either we do not solve the same problem, or the transformation is not a polynomial reduction. Hence, the complexity figures for the original and the transformed problems are not comparable.

For problems with an integer *parameter* k in addition to the instance $\boldsymbol{x}$, traditional complexity theory considers the aggregation $\langle \boldsymbol{x}, k \rangle$ as the instance, measuring the complexity in $||\langle \boldsymbol{x}, k \rangle||$. *Parameterized complexity theory*[1] is an alternative way to analyse such problems which considers k as fixed and separate from $\boldsymbol{x}$ in the following way[2]. A *fixed parameter tractable (fpt)* function is a function h on tuples $\langle \boldsymbol{x}, k \rangle$, where $k \geq 0$ is an integer, that is computable in time $O(g(k)p(||\boldsymbol{x}||))$ for some function g and some polynomial p. **FPT** is the class of all problems that can be solved by an fpt function. Let X and Y be two decision problems with pa-

[1]The name parameterized complexity often causes confusion since also traditional complexity theory can handle problems with parameters. The methods differ only in *how* they do it.

[2]See Downey and Fellows (1999) or Flum and Grohe (2006) for further details.

rameters. A *parametric transformation* from X to Y is an fpt function h that maps every instance $\langle \boldsymbol{x}, k\rangle$ of X to an instance $\langle \boldsymbol{y}, k'\rangle$ of Y with the same answer s.t. k' depends only on k. Since neither of k, g and p can depend on $\boldsymbol{x}$, the parameterized complexity can be viewed as $O(g(k)f(||\boldsymbol{x}||))$. This separates it into two independent parts where, intuitively, $g(k)$ is the hard part, since g is an arbitrary function, and $f(||\boldsymbol{x}||)$ is the easy part, since f is a polynomial. It is tacitly assumed that k is small and that g grows moderately fast (although typically exponentially). This theory must use its own complexity classes, where **FPT** is intended as a parameterized version of **P** (note that **P** $\subseteq$ **FPT**) and examples of harder classes are the **W** hierarchy **W**[1],**W**[2],...,**W**[SAT],**W**[P].

3 Planning Framework

In order to make our results general we use an abstract planning framework, covering most ground planning languages.

Definition 1. An instance of *Finite Functional Planning (FFP)* is a tuple $\boldsymbol{p} = \langle n, D, A, I, G\rangle$, where the integer $n \geq 0$ is the number of *state variables* and the finite set D is the *domain* for these variables. The *state space* S is implicitly defined as $S = D^n$. The state $I \in S$ represents the *initial state* and the total function $G : S \to \{0, 1\}$ represents the *goal*. A is a set of *actions*, each action $a \in A$ defining two total functions $\varphi(a) : S \to \{0, 1\}$ (the *precondition*) and $\psi(a) : S \to S$ (the *postcondition*).

Let $\omega = \langle a_1, \ldots, a_\ell\rangle$ be a sequence of actions from A. Also let $\varphi_i = \varphi(a_i)$ and $\psi_i = \psi(a_i)$ for all i s.t. $1 \leq i \leq \ell$, and let $\psi_0 = I$ and $\varphi_{\ell+1} = G$. The *result sequence* $\sigma = \langle s_0, s_1, \ldots, s_\ell\rangle$ of states for $\boldsymbol{p}$ and ω is defined s.t. $s_0 = I$ and $s_i = \psi_i(s_{i-1})$ for all i s.t. $1 \leq i \leq \ell$. The sequence ω is a solution for $\boldsymbol{p}$ iff both $G(s_\ell) = 1$ and $\varphi_i(s_{i-1}) = 1$ for all i s.t. $1 \leq i \leq \ell$. A solution for $\boldsymbol{p}$ is called a *plan* for $\boldsymbol{p}$.

Note that S is not explicitly represented (and usually exponentially larger than $\boldsymbol{p}$). The functions for the goal and the pre- and postconditions need not be explicitly represented either; planning languages like propositional STRIPS and SAS$^+$ are typical subcases of FFP. We also assume that D is an interval of the integers, or similar, so it can be implicitly represented in $O(\log |D|)$ space. To refer implicitly to elements of $\boldsymbol{p}$ we write $A_{\boldsymbol{p}}$ etc.

As the basis for complexity analysis we use the concept of validating a plan step (that is, computing $\varphi(a)$ and $\psi(a)$ for an action a and a state s), which implicitly defines plan validation. The reason for this is to make not only plan length, but also action modelling, visible in complexity analysis later on. To make the rest of the theory clearer, step complexity is defined using decision problems only.

Definition 2. *Step validation* is a collective name for the following three decision problems: (i) given an action a and a state s, decide if $\varphi(a)(s) = 1$; (ii) given an action a, a state s and an integer k, s.t. $1 \leq k \leq ||s||$, decide if bit k in the representation of $\psi(a)(s)$ is 1; and (iii) given a state s, decide if $G(s) = 1$. Step validation is in a complexity class C if all three problems are in C. The decision problem *plan validation* for FFP is: given an FFP instance $\boldsymbol{p}$ and an action sequence ω, decide if ω is a plan for $\boldsymbol{p}$. The complexity of plan validation is measured in $||\boldsymbol{p}|| + ||\omega||$.

We make no further assumptions about when a plan step is valid, but only about the complexity of deciding it.

The restriction of FFP to some subset X of FFP instances is denoted $\mathsf{FFP}(X)$ and the restriction of FFP to subsets where step validation is in some complexity class C is denoted $\mathsf{FFP}([C])$. This notation will be used also for restricting corresponding problems.

Lemma 1. *Let C be an arbitrary complexity class, then (a) for all instances $\boldsymbol{p}$ of $\mathsf{FFP}([C])$, all states $s \in S_{\boldsymbol{p}}$ and all actions $a \in A_{\boldsymbol{p}}$, the resulting state $t = \psi(a)(s)$ can be computed in polynomial time by a DTM with an oracle for C and (b) plan validation for $\mathsf{FFP}([C])$ is in $\mathbf{P}^C$.*

Proof. (a) A state requires $n \log |D| < ||\boldsymbol{p}||$ bits, so M can call the oracle once for each bit in t to find its value. (b) is immediate from (a) and Definitions 1 and 2. □

Note that it does not matter for plan validation whether step validation is in C or in coC, since $\mathbf{P}^C = \mathbf{P}^{coC}$.

The two most common decision problems for planning are the following.

Definition 3. The *plan existence problem (PE)* for FFP is: given an instance $\boldsymbol{p}$ of FFP, decide if $\boldsymbol{p}$ has a plan. The *bounded plan existence problem (BPE)* for FFP is: given an instance $\langle \boldsymbol{p}, \ell\rangle$ s.t. $\boldsymbol{p}$ is an FFP instance and $\ell \geq 0$ an integer, decide if $\boldsymbol{p}$ has a plan with ℓ or fewer actions.

The following proposition is straightforward.

Proposition 1. *There is a polynomial reduction from plan existence for propositional* STRIPS *to $\mathsf{PE}([\mathbf{P}])$.*

4 Inadequacy of the Standard Methods

We first show that traditional complexity analysis is often too coarse to capture differences in action modelling.

Theorem 1. *$\mathsf{PE}([C])$ and $\mathsf{BPE}([C])$ are* **PSPACE***-complete for all complexity classes C s.t. $\mathbf{P} \subseteq C \subseteq$* **PSPACE**.

Proof. (Sketch) Let C be such a class. For membership in **PSPACE** we use the same technique as Bylander (1994, Theorem 3.1), but since we also allow oracle calls we get membership in **NPSPACE**^{C} rather than in **NPSPACE**. However, **NPSPACE**^{C} = **NPSPACE** = **PSPACE**, so $\mathsf{BPE}([C])$ is in **PSPACE**. For completeness, note that $\mathsf{PE}([\mathbf{P}]) \leq_m^p \mathsf{PE}([C]) \leq_m^p \mathsf{BPE}([C])$. Hence, **PSPACE**-completeness follows from Proposition 1 and that propositional STRIPS planning is **PSPACE**-complete (Bylander 1994, Theorem 3.1). □

An immediate consequence of this theorem is that $\mathsf{PE}([\mathbf{PSPACE}])$ and $\mathsf{BPE}([\mathbf{PSPACE}])$ are in **PSPACE**, that is, a planning language no more powerful than STRIPS can have actions as powerful as the language itself. In other words, a planner for a **PSPACE**-complete language can, in principle, plan for actions that themselves solve **PSPACE**-complete planning problems. It is hardly a far-reaching conclusion that traditional complexity analysis is too coarse to improve much on existing results.

Parameterized complexity has proven very useful for many **NP**-complete problems, but has not had much impact on planning. The only general relevant result we know of is

that STRIPS planning is **W**[1]-hard with plan length as parameter (Downey, Fellows, and Stege 1999, Theorem 6.5). However, since the parameter is generally assumed small in parameterized analysis, plan length is not an obvious choice. A closer look at this forces us to question the relevance of their result. Under traditional analysis, planning is more difficult for long plans, which coincides with how search algorithms behave both in theory and in practice. For parameterized analysis, though, we prove the opposite and counter-intuitive result that planning is hard only for finding very short plans and otherwise tractable!

Theorem 2. *Let $h : \mathbb{N} \to \mathbb{N}$ be an increasing function. Then the restiction of BPE([P]) to instances $\boldsymbol{p}' = \langle \boldsymbol{p}, \ell \rangle$ s.t. $|A_{\boldsymbol{p}}| \leq h(\ell)$ is in **FPT**.*

Proof. Let h and $\boldsymbol{p}'$ be as specified, and $x = ||\boldsymbol{p}||$. We ask for a plan ω s.t. $|\omega| \leq \ell$, so $||\omega|| \leq \ell \log |A_{\boldsymbol{p}}| \leq \ell \log h(\ell)$. There are $O(2^{||\omega||})$ possible plans, each of which is verifiable in $O(|\omega| p(x))$ time, for some polynomial p. Finding a plan thus takes $O(2^{||\omega||}|\omega| p(x))$ time, but this is in $O((2^{\ell \log h(\ell)} \ell) p(x))$ which is in **FPT**. □

Since h may grow arbitrarily fast, the hard instances are rare exceptions that probably only occur when plans are very short but there is a very large number of actions to choose from. The reason is that the fpt definition does not enforce any restrictions on the function g of the parameter, and it is well known that pathological cases like this may occur (cf. Proposition 1.7 in Flum and Grohe (2006)).

We conclude that neither traditional complexity analysis nor parameterized complexity analysis seem sufficient to improve much on the current complexity results for planning, at least not in any way yet demonstrated. This calls for new and better methods, and attempts have been made in that direction. One example is Balcázar (1996), who approached the general problem of searching implicit graphs represented by circuits. While this seems to work also for planning, it gives no new separations, but it is an open question if changing the already succinct graph representations of planning to circuits would give any new insights. We will choose another approach, which we find simpler and more immediate, and which gives interesting separation results.

5 Method I: Padded Complexity

While PE/BPE(**[P]**) becomes **NP**-complete when restricting plans to polynomial length, this is not as straightforward as it may sound. There are cases of restricted languages where all optimal plans are of polynomial length, but they are too restricted for most applications. On the other hand, while many application problems for planning are **NP**-complete, or simpler, nobody has yet found a general way to capture this by language restrictions. Alternatively, we could try to capture the concept of polynomial solutions with the problem definition rather than the language, for instance, defining a variant of BPE that asks not for a plan of length ℓ but of length $||\boldsymbol{p}||^{\ell}$. Just as for BPE, however, there is no reason for ℓ to be bounded in the general case, so it is questionable what can be gained from this. We take a different approach to this and introduce the method of *padded complexity analysis* for planning.

Padding refers to the technique of representing one or more integers in *unary* notation (representing an integer n with the string 1^n of n 1's), rather than in binary. Padding is commonly used in structural complexity theory, to transfer results between complexity classes or to construct complete languages for classes. In problem complexity, however, padding is usually considered taboo, since it can make a problem artificially easier by blowing up the instance size exponentially. It is used, though, to distinguish between complete problems for **NP** (and other classes) that remain complete when representing integers in unary (referred to as *strongly complete* problems) and those that do not. For example, reachability for single-path Petri nets is strongly **PSPACE**-complete (Howell, Jančar, and Rosier 1993). Although planning has many similarities with this problem, it does not share the property of being strongly **PSPACE**-complete—which is exactly what we will exploit.

We will boldly break against the padding taboo, and pad instances with the deliberate intention of actually blowing up the instance size. When asking for a plan of length ℓ we represent ℓ in unary. Since shortest plans can be of exponential length also ℓ may be exponential, making the new padded instance exponentially larger. The point of this is to get a complexity measure that does not primarily depend on plan length and that can reveal details that are otherwise swamped by the complexity caused by the plan length. (This technique has previously been used by Bonet (2010) in a different context; defining a complexity measure for finding contingent plans that depends on the number of branch points in such plans). This is somewhat in line with the idea of Impagliazzo et al. (2001) that it may be more robust to measure complexity in the size of the solution than in the instance. Another view is that if we accept exponential-size plans, then we must commit to that in advance and pay the price for it with the size of the instance. In other words, we must allocate enough memory in advance for the solution we are actually asking for. Furthermore, it is not uncommon to measure the complexity in the size of the output (so called *output-sensitive* analysis) when analysing algorithms, with examples also in planning (Jonsson and Bäckström 1998).

Definition 4. The *length-padded plan existence problem (LPPE)* for FFP is: given an instance $\langle \boldsymbol{p}, 1^{\ell} \rangle$, where $\boldsymbol{p}$ is an FFP instance and $\ell \geq 0$ an integer, decide if $\boldsymbol{p}$ has a plan with ℓ or fewer actions.

Planning is obviously **NP**-complete under padded analysis if step validation is polynomial. We prove a more general version of this statement, demonstrating a hierarchy of increasing hardness that depends on the complexity of step validation but not on plan length.

Theorem 3. *LPPE([C]) is $\mathbf{NP}^C$-complete for arbitrary complexity class C (s.t. $\mathbf{NP}^C$ has complete problems).*

Proof. Membership: Let M be an NTM with an oracle for C. Given an LPPE([C]) instance $\boldsymbol{p}_L = \langle \boldsymbol{p}, 1^{\ell} \rangle$ as input, M first guesses a plan ω with $\leq \ell$ actions and then validates

it. It only needs to guess a polynomial number of bits since $||\omega|| \leq |\omega| \log |A_{\boldsymbol{p}}| \leq \ell ||\boldsymbol{p}|| < ||\boldsymbol{p}_L||^2$. Using Lemma 1, M can thus validate ω in polynomial time since $||\boldsymbol{p}|| + ||\omega||$ is polynomial in $||\boldsymbol{p}_L||$.

Hardness: Let X be an arbitrary problem in $\mathbf{NP}^C$. Then there is a polynomial p and an NTM M with an oracle for C s.t. for every instance $\boldsymbol{x}$ of X, (1) M halts in at most $p(||\boldsymbol{x}||)$ steps and (2) $M(x)$ accepts iff $\boldsymbol{x}$ has a solution. We can thus consider M as equivalent with X. Wlog. we assume M has only one accepting state and one bidirectional tape, where cells number $-1, -2, \ldots$ are used for oracle queries only. We also assume it has tape alphabet $\Gamma = \{0, \ldots, k\}$, where $k > 0$ and 0 is the blank symbol. The input alphabet is implicitly assumed to be $\{1, \ldots, k\}$. Let $M = \langle \Gamma, Q, \Delta, q_0, q_a \rangle$ be the description of the machine, where Γ is the tape alphabet, Q the set of states, Δ the set of transitions, $q_0 \in Q$ the initial state and $q_a \in Q$ the accepting state. We assume that Q contains the usual three query states q_q, q_y, q_n s.t. in state q_q the only possible transition is the one that queries the oracle, resulting in either the yes state q_y or the no state q_n.

For every valid input $\boldsymbol{x}$ for M, we can construct an LPPE instance $\boldsymbol{p}_L = \langle \boldsymbol{p}, 1^\ell \rangle$, where $\boldsymbol{p} = \langle n, D, A, I, G \rangle$ and $\ell = p(||\boldsymbol{x}||)$, as follows. Let $n = 4\ell + 3$ and $D = \{0, \ldots, \max(k, |Q|)\}$. We name the variables as follows: $t_{-\ell}, \ldots, t_\ell$ represent the tape cells, $h_{-\ell}, \ldots, h_\ell$ represent the position of the head and s holds the current state. (Since M halts in ℓ steps, it can never reach further on the tape than $\pm\ell$ cells.) A transition $\langle q, x, r, y, L \rangle$ is encoded as 2ℓ actions

$$s = q, h_i = 1, t_i = x \Rightarrow \; s = r, h_i = 0, h_{i-1} = 1, t_i = y$$

for $-\ell < i \leq \ell$ (in notation $\varphi(a) \Rightarrow \psi(a)$). Transitions moving right are encoded analogously. The action for the query transition has precondition that $s = q_q$ and postcondition that either $s = q_a$ or $s = q_n$, depending on what the oracle answers. All variables have value 0 in I, except that $t_0, \ldots, t_{||x||-1}$ contain the input x, $h_0 = 1$ and $s = q_0$. The goal function G has value 1 iff $s = q_a$.

This is a polynomial-time construction and $\boldsymbol{p}_L$ is an $\mathsf{LPPE}([C])$ instance, since an oracle for C is sufficient for step validation. It is straightforward that $\boldsymbol{p}_L$ has a plan iff $M(x)$ accepts. Hence, $\mathsf{X} \leq_m^p \mathsf{LPPE}([C])$, so we conclude that $\mathsf{LPPE}([C])$ is $\mathbf{NP}^C$-complete. □

This implies an immediate relationship to the polyonomial hierarchy.

Corollary 1. *(a) $\mathsf{LPPE}([P])$ is **NP**-complete.*
(b) $\mathsf{LPPE}(\boldsymbol{\Sigma}_k^p)$ is $\boldsymbol{\Sigma}_{k+1}^p$-complete for all integers $k \geq 1$.

This provides a complexity hierarchy of FFP fragments which is immune to plan length and reflects the complexity of validating plan steps rather than the whole plan. By combining the complexity for PE/BPE with the complexity for LPPE, we get a more refined picture that tells us something both about complexity due to plan length and complexity due to action models. One might say that LPPE gives the most relevant figure for moderately long plans and PE/BPE for very long plans.

SAT encodings of planning can also be considered as padded, but in contrast to our LPPE concept, SAT planning seems not to give any further insight into different sources of complexity or allow for any separation results.

The method we have just introduced can obviously capture distinctions of the kind that Theorem 1 proved impossible with traditional analysis. Yet, we actually still use traditional complexity analysis and have only abused the standard for instance encodings. The obvious question, whether such a change of encodings could also make parameterized analysis more relevant, must be answered negatively. It adds no new separations.

Theorem 4. *Let X be a set of LPPE instances and define $Y = \{\langle \boldsymbol{p}, \ell \rangle \mid \langle \boldsymbol{p}, 1^\ell \rangle \in X\}$). If $\mathsf{LPPE}(X)$ is in **FPT**, then also $\mathsf{BPE}(Y)$ is in **FPT**.*

Proof. Assume $\mathsf{LPPE}(X)$ is in **FPT**. Given an instance $\langle \boldsymbol{p}, \ell \rangle$ of $\mathsf{BPE}(Y)$, compute a corresponding LPPE instance $\langle \boldsymbol{p}, 1^\ell \rangle$ in $O(2^\ell \cdot ||\boldsymbol{p}||)$ time. This is a parametric transformation and **FPT** is closed under such transformations. □

6 Examples and Connections

One obvious application of padded analysis is planning languages where step validation must solve constraint problems (for instance temporal constraint problems (Tsamardinos, Vidal, and Pollack 2003)). Temporal constraint problems are typically **NP**-complete so LPPE is complete for $\mathbf{NP}^{\mathbf{NP}} = \boldsymbol{\Sigma}_2^p$ for such languages. This separates them as harder than many other planning languages, which is not possible to show with traditional complexity analysis.

Such languages move only one step up in the hierachy, however. To climb further up, we first note that according to Corollary 1 each level $\mathsf{LPPE}([\boldsymbol{\Sigma}_k^p])$ of this hierarchy corresponds to level $\boldsymbol{\Sigma}_{k+1}^p$ of the polynomial hierarchy. Furthermore, $\boldsymbol{\Sigma}_{k+1}^p$ corresponds to satisfiability of quantified boolean formulae with at most k alternations of quantifiers. Hence, a language with step complexity in **NP** can be viewed as having preconditions with existentially quantified variables, while languages higher up in the hierarchy can have alternating existential and universal quantifiers, with the number of alternations depending on the level of the hierarchy. The planning language PDDL allows for arbitrary alternation of quantifiers in preconditions. It thus contains an infinite hierarchy of succesively harder fragments corresponding to the levels of the $\mathsf{LPPE}([\boldsymbol{\Sigma}_k^p])$ hierachy, even when the variable domains have only two values. This also means that we cannot cheat with padded complexity and transform an instance inte one with quantifier-free actions, since this is not a polynomial reduction.

Yet another example is planning languages with domain axioms. Although FFP does not allow axioms this would be trivial to include. The axioms would be an additional part of the instance and checking them would go into the pre- and postcondition functions, thus immediately reflecting upon the step complexity. If we know the complexity of checking the axioms, then we can place the language at the proper level in the $\mathsf{LPPE}([\boldsymbol{\Sigma}_k^p])$ hierachy.

These examples demonstrate how padded complexity analysis can be used to show separations in complexity for various planning languages, separations that would not have

been possible to show in a simple and straightforward way with traditional methods. Padded complexity can thus provide further insight into, and a separation between, problem classes that are otherwise **PSPACE**-complete.

While the $\mathsf{LPPE}([\Sigma_k^p])$ hierarchy shows a separation between languages at different levels, it says little about languages within the same level. For example, each level $\mathsf{LPPE}([\Sigma_k^p])$ is closed under $\leq_m^p$, yet it can contain subsets X and Y s.t. $\mathsf{LPPE}(X) \not\leq_m^p \mathsf{LPPE}(Y)$. Similarly, $\mathsf{BPE}(X) \leq_m^p \mathsf{BPE}(Y)$ does not imply $\mathsf{LPPE}(X) \leq_m^p \mathsf{LPPE}(Y)$ or vice versa. However, with a more restricted reduction, that additionally promises to increase the size of solutions at most polynomially, we get the following relationship between BPE and LPPE.

Theorem 5. *Let X and Y be sets of FFP instances and function f a polynomial many-one reduction from $\mathsf{BPE}(X)$ to $\mathsf{BPE}(Y)$. If there is a polynomial p s.t. for all $\langle \boldsymbol{p}, \ell \rangle$ in X, the corresponding $\langle \boldsymbol{p}', \ell' \rangle = f(\langle \boldsymbol{p}, \ell \rangle)$ in Y satisfies that $\ell' \leq p(||\boldsymbol{p}|| + \ell)$, then $\mathsf{LPPE}(X) \leq_m^p \mathsf{LPPE}(Y)$.*

Proof. Assume that f is as claimed, $\langle \boldsymbol{p}, \ell \rangle$ is in $\mathsf{BPE}(X)$ and $\langle \boldsymbol{p}', \ell' \rangle = f(\langle \boldsymbol{p}, \ell \rangle)$. Then $\langle \boldsymbol{p}, 1^\ell \rangle$ is in $\mathsf{LPPE}(X)$ and $\langle \boldsymbol{p}', 1^{\ell'} \rangle$ is in $\mathsf{LPPE}(Y)$. By definition of f there are polynomials p and q s.t. $\ell' \leq p(||\boldsymbol{p}|| + \ell)$ and $||\boldsymbol{p}'|| \leq q(||\boldsymbol{p}||)$. Hence, $||\langle \boldsymbol{p}', 1^{\ell'} \rangle|| \leq q(||\boldsymbol{p}||) + p(||\boldsymbol{p}|| + \ell)$, which is polynomial in $||\langle \boldsymbol{p}, 1^\ell \rangle||$. □

The converse of this theorem does not hold, but its contrapositive yields some insight: if there is no polynomial reduction from $\mathsf{LPPE}(X)$ to $\mathsf{LPPE}(Y)$, then there is no such solution-constrained reduction from $\mathsf{BPE}(X)$ to $\mathsf{BPE}(Y)$. This suggests a connection to the separation results for planning languages by Nebel (2000). He used compilation techniques to separate a number of variants of STRIPS into classes with different expressive power, but did not show any separations in complexity. For all cases he considered the compilations could as well be viewed as polynomial reductions with the additional condition that the plan size increases at most polynomially[3]. That is, what Nebel used is essentially a reduction of the type described in Theorem 5. Since he only considered planning languages with step complexity in **P**, all his separation results are between languages that all belong to $\mathsf{LPPE}(\mathbf{[P]})$, that is, they cannot be separated by our padded analysis. It thus seems that Nebels separation results and ours are orthogonal to each other, capturing different aspects of planning languages.

7 Method II: Limited Nondeterminism

Knowing that a problem is in **NP** guarantees that there is some polynomial p and some NTM M that solves any instance $\boldsymbol{x}$ of size x in time $O(p(x))$. Let $g(x)$ be the number of bits M guesses and $f(x)$ the time it takes to verify the guess. Then $g(x) \leq f(x)$ since M must at least read the guess to verify it. Further than that we can only conclude that both $g(x)$ and $f(x)$ are in $O(p(x))$. Hence, all we know is that M guesses $O(p(x))$ bits and takes $O(p(x))$ time to verify the guess, and that the problem thus can be solved by search in time $O(2^{p(x)}p(x))$. This clearly overestimates the number of bits to guess. For example, to solve a SAT instance we need to guess at most one bit per variable. Even if overestimating this figure by x, we can solve the instance by search in $O(2^x p(x))$ time, which is a factor $2^{p(x)}$ better than the previous figure $O(2^{p(x)}p(x))$. Unfortunately, classes like **NP** cannot capture such 'subtle' differences.

Kintala and Fischer (1977) suggested to treat also non-determinism as a quantifiable resource, and they invented what is known as *limited nondeterminism*. For our purposes, the following definition of a restricted family of complexity classes based on limited non-determinism is sufficient.

Definition 5. Let $g : \mathbb{N} \to \mathbb{N}$ be a time-constructible function. The complexity class $g(x)$-**P** consists of all decision problems that can be solved in polynomial time on some NTM that guesses at most $O(g(x))$ bits, where x is the size of the instance.

This means that a problem in $g(x)$-**P** can be solved by search in time $2^{O(g(x))}p(x)$, for some polynomial p. Also note that $\log x$-**P** $=$ **P** and $\cup_{c>1}(x^c\text{-}\mathbf{P}) = \mathbf{NP}$. Kintala and Fischer (1977) studied the hierarchy $\mathbf{P} = \log x\text{-}\mathbf{P} \subseteq (\log x)^2\text{-}\mathbf{P} \subseteq \ldots \subseteq x\text{-}\mathbf{P} \subseteq x^2\text{-}\mathbf{P} \subseteq \ldots \subseteq \mathbf{NP}$ of classes between **P** and **NP**. It is still not known if any of these inclusions are strict, and proving that one of them is would imply that $\mathbf{P} \neq \mathbf{NP}$. However, the classes are also useful due to the connection between non-determinism and search.

Limited non-determinism has mostly been used in structural complexity (see Goldsmith et al. (1996) for an overview and Williams (2010) for recent result using this concept). It has also been used to achieve more precise results for certain problems. Examples are that computing V-C dimension (used in learning, for instance) is in $(\log x)^2$-**P** (Papadimitriou and Yannakakis 1996) and more precise complexity bounds for satisfiability of restricted CNF formulas (Gottlob, Pichler, and Wei 2008).

We now proceed to combine our padding method with limited non-determinism to achieve even more precision. We still use LPPE and it is only how we analyse it that differs. Our first result is a tighter upper bound for $\mathsf{LPPE}([\mathrm{P}])$.

Theorem 6. *$\mathsf{LPPE}([\mathrm{P}])$ is in $x \log x$-**P**.*

Proof. Let $\boldsymbol{p}_L = \langle \boldsymbol{p}, 1^\ell \rangle$ be an $\mathsf{LPPE}(\mathbf{[P]})$ instance and ω a plan for $\boldsymbol{p}_L$. We note that (1) $||\boldsymbol{p}|| + \ell \leq ||\boldsymbol{p}_L||$, (2) $|A_{\boldsymbol{p}}| < ||A_{\boldsymbol{p}}|| < ||\boldsymbol{p}||$ and (3) $||\omega|| \leq \ell \log |A_{\boldsymbol{p}}|$. Hence, it is sufficient to guess a plan of size $||\omega|| < ||\boldsymbol{p}_L|| \log ||\boldsymbol{p}_L||$ and then verify it, which is in $\mathbf{P}^{\mathbf{P}} = \mathbf{P}$. Hence, $\mathsf{LPPE}(\mathbf{[P]})$ is in $x \log x$-**P**. □

While this result might seem trivial, it could hardly have been proved by standard methods. Proving completeness is less straightforward, though. While polynomial reductions are sufficient to prove completeness for classes of the type $(\log x)^k$-**P**, much studied in the literature, they are, somewhat counterintuitively, not useful for classes of the type x^k-**P** or $x^{1/k}$-**P**, which are not even closed under logspace

[3] A compilation leaves some parts of the instance to be filled in later (like our reformulation concept (Bäckström and Jonsson 2011)). Since this is assumed simpler than the compilation, the overall process is essentially a reduction.

reductions. It is an open question if completeness can be proved using som other reduction, like quasilinear reduction.

As an alternative option we use the *Exponential Time Hypothesis (ETH)* to establish a reasonably tight lower bound. ETH was suggested by Impagliazzo and Paturi (2001) and says, in one of several forms it can be stated (Impagliazzo, Paturi, and Zane 2001), that 3-SAT cannot be solved in subexponential time, that is, in time $2^{o(m)}$ where m is the number of clauses. While ETH is not obviously as strong as the hypothesis that $\mathbf{P} \neq \mathbf{NP}$, it still has a similar flavour: many **NP**-complete problems are related s.t. if ETH is true, then none of them can be solved in subexponential time, and if ETH is false, then all of them can.

Theorem 7. *If LPPE(**P**) is in $g(x)$-**P** for some function g s.t. $g(x) \in o(x/\log x)$, then ETH is false.*

Proof. Suppose LPPE(**P**) is in $g(x)$-**P**, where $g(x) \in o(x/\log x)$. Let $\boldsymbol{s}$ be an arbitrary 3-SAT instance (a 3-CNF formula) with n variables and m clauses. Then $||\boldsymbol{s}|| \in \Theta(m \log n)$, but $n \leq 3m$ and $m < 8n^3$, so $||\boldsymbol{s}|| \in \Theta(m \log m)$. Construct an FFP([P]) instance $\boldsymbol{p}$ with n binary variables, one action for each (setting it true) and the goal G implementing the CNF of $\boldsymbol{s}$. Then, $\boldsymbol{s}$ is satisfiable iff $\boldsymbol{p}$ has a plan of length n or shorter and $\boldsymbol{s}$ can be encoded s.t. $||\boldsymbol{p}|| \in \Theta(m \log m)$. Construct the padded instance $\boldsymbol{p}_L = \langle \boldsymbol{p}, 1^{3m} \rangle$, also of size $\Theta(m \log m)$. Since $n \leq 3m$, also $\boldsymbol{p}_L$ has a solution iff $\boldsymbol{s}$ is satisfiable. Let $x = ||\boldsymbol{p}_L||$. By assumption, it suffices to guess $O(g(x))$ bits to solve $\boldsymbol{p}_L$, which is equivalent to guessing $O(g(m \log m))$ bits since $x \in \Theta(m \log m)$. Also by assumption, $g(m \log m)$ is in $o(\frac{(m \log m)}{\log(m \log m)}) = o(\frac{m \log m}{\log m}) = o(m)$, so for some polynomial p, we can solve $\boldsymbol{p}_L$, and thus also $\boldsymbol{s}$, in $2^{o(m)}p(m) = 2^{o(m)}$ time, which contradicts ETH. □

We can thus bound the complexity of LPPE(**P**) to between $x/\log x$-**P** and $x \log x$-**P** (assuming ETH is true).

We can further define a more precise hierarchy of upper bounds, similar to Theorem 3.

Theorem 8. *LPPE([$f(x)$-**P**]) is in $x^2 f(x)$-**P**, for arbitrary time constructible function $f : \mathbb{N} \to \mathbb{N}$.*

Proof. Let $\boldsymbol{p}_L = \langle \boldsymbol{p}, 1^\ell \rangle$ be such an instance and $x = ||\boldsymbol{p}_L||$. Guess a plan ω for $\boldsymbol{p}_L$. Since $|\omega| < x$ and $||\omega|| < x \log x$, there are $\leq x$ steps to validate, each requiring $< x$ calls to an $f(x)$-**P** oracle. Hence, plan validation requires guessing in total $\leq x^2 f(x)$ bits. This dominates the plan size, so LPPE($f(x)$-**P**) is in $x^2 f(x)$-**P**. □

8 An Example: Partial-order Plans

So far plans have been *total-order plans* (sequences of actions), but *partial-order plans* are also important and frequently used. We now turn our attention to such plans as an example of applying the method from the previous section. A partial-order plan is a partially ordered set of *action occurences* (we must now explicitly distinguish different occurences of the same action). That two actions are mutually unordered can either mean that they can be scheduled concurrently, or only that they can be scheduled in either order, depending on the application. A partial-order plan is thus more flexible, allowing for a scheduler to post-process and optimise the action ordering according to further criteria.

Definition 6. Let $\boldsymbol{p}$ be an FFP instance. A *partial-order plan* for $\boldsymbol{p}$ is a tuple $\rho = \langle B, \prec \rangle$ s.t. B is a set of *action occurences* over $A_{\boldsymbol{p}}$ and $\prec$ is a partial order on B. If $\prec$ is a total order, then ρ is also a *total-order plan*.

Note that it is sufficient to encode as much of the ordering as is necessary to reconstruct its transitive closure, so we should not assume an explicit partial order represented.

Validation of partial-order plans is not as straightforward as for total-order plans (Nebel and Bäckström 1994). Step validation is no longer obviously useful since there is no well-defined state to validate an action against. Hence, we switch to using plan validation as the primary concept.

Definition 7. The *partial-order length-padded plan existence problem (PO-LPPE)* for FFP is: given an instance $\langle \boldsymbol{p}, 1^\ell \rangle$, where $\boldsymbol{p}$ is an FFP instance and $\ell \geq 0$ an integer, decide if $\boldsymbol{p}$ has a partial-order plan $\rho = \langle B, \prec \rangle$ s.t. $|B| \leq \ell$. Given a complexity class C, PO-LPPE([C]) refers to the restriction where plan validation is in $\mathbf{P}^C$.

An obvious definition of validity for partial-order plans is that all topological sortings of it must be valid. That is pointless, however; since every total-order plan is also a partial-order plan, an algorithm could always return a total-order plan. To make a partial-order plan interesting at all, we must have some additional criteria that makes it more useful than any of its topological sortings. Just as for step complexity, we make no assumptions about when a plan is valid, but only about the complexity of deciding this. Intuitively, the reader should bear in mind that, in this context, the concept of plan validation can include, for instance, optimization criteria on the plan order. Examples of such criteria are the ones suggested by Bckstrm (1998), intended to maximize the flexibility of the plan to allow for better scheduling and other post processing. However, PO-LPPE([C]) is still a richer class of instances than LPPE([C]) even without such explicit requirements, since plan validation in P^C is a weaker criterion than step validation in C. The following result is thus not exactly comparable to Theorem 6, but similarly provides an upper bound.

Theorem 9. *PO-LPPE([**P**]) is in $x^2 \log x$-**P**.*

Proof. Let $\boldsymbol{p}_L = \langle \boldsymbol{p}, 1^\ell \rangle$ be an PO-LPPE([**P**]) instance and $x = ||\boldsymbol{p}_L||$. Guess a plan $\rho = \langle B, \prec \rangle$ (B assumed encoded as a sequence of actions from $A_{\boldsymbol{p}}$ and $\prec$ as pairs of sequence indices). By requirement, $|B| \leq \ell$, so $||B|| \leq \ell \log |A_{\boldsymbol{p}}| \leq x \log x$. Obviously, $|\prec| \leq \ell^2$, so $||\prec|| \leq \ell^2(2 \log \ell) \leq 2x^2 \log x$. Hence, it is sufficient to guess $O(x \log x + 2x^2 \log x) = O(x^2 \log x)$ bits. Since validation of ρ is in $\mathbf{P}^{\mathbf{P}} = \mathbf{P}$, it follows that PO-LPPE([**P**]) is in $x^2 \log x$-**P**. □

Padding alone is not sufficient to even suggest a difference between LPPE([**P**]) and PO-LPPE([**P**]), since these are both **NP**-complete; limited nondeterminism is necessary to get different results. Proving completeness is as difficult

as before, though, and ETH seems not immediately applicable here. However, we note that there are plans where the partial order must be of quadratic size if we require any of Bckstrm's optimality criteria, even if not representing the transitive closure explicitly, In fact, Kleitman and Rotschild (1970) showed that the number of different partial orders over a set of n elements is $2^{n^2/4+o(n^2)}$. Hence, we conjecture that partial-order planning is actually harder than total-order planning.

To further consider also parallel execution of actions, we follow Bckstrm's (1998) concepts and definitions.

Definition 8. Let $\boldsymbol{p}$ be an FFP instance. A *non-concurrency relation* over $A_{\boldsymbol{p}}$ is a reflexive and symmetric relation $\# \subseteq A^2$, that implicitly transfers to action occurences. Let $\rho = \langle B, \prec \rangle$ be a partial-order plan for $\boldsymbol{p}$ and $\#$ a non-concurrency relation over $A_{\boldsymbol{p}}$. A *parallel execution* of *length* m for ρ is a partitioning $B_1, \ldots, B_m$ of B s.t. for all $b_i \in B_i$ and $b_j \in B_j$, it holds that (1) $b_i \prec b_j$ implies $i < j$ and (2) $b_i \# b_j$ implies $i \neq j$.

Definition 9. The *parallel length-padded plan existence problem (PAR-LPPE)* is: given a instance $\boldsymbol{p}' = \langle \boldsymbol{p}, \#, 1^\ell, k \rangle$, where $\boldsymbol{p}$ is an FFP instance, $\#$ is a non-concurrency relation over $A_{\boldsymbol{p}}$ and $\ell, k > 0$ integers, decide if $\boldsymbol{p}$ has a partial-order plan $\rho = \langle B, \prec \rangle$ s.t. $|B| \leq \ell$ and ρ has a parallel execution of length k or shorter.

Theorem 10. *PAR-LPPE([P]) is in $x \log x$-**P**.*

Proof. Let $\boldsymbol{p}' = \langle \boldsymbol{p}, \#, 1^\ell, k \rangle$ be such an instance and $x = ||\boldsymbol{p}'||$. Guess a set B of at most ℓ action occurences over $A_{\boldsymbol{p}}$ and a partitioning $B_1, \ldots, B_k$ of B. Encode the partitions in order as sequences of actions, which allows mulitple occurences of an action in a partition and requires $O(\ell \log |A_{\boldsymbol{p}}|) \subseteq O(x \log x)$ bits. Since plan validation is in $\mathbf{P}^{\mathbf{P}} = \mathbf{P}$, the problem is in $x \log x$-**P**. □

This indicates that finding an optimally scheduled parallel plan is no harder than finding a total-order plan. This does not contradict our conjecture, however. PAR-LPPE asks for a plan and one optimal schedule, while PO-LPPE allows asking for a plan that can be scheduled in several different optimal ways. (Although PAR-LPPE(**[P]**) logspace reduces to PO-LPPE(**[P]**), this is not relevant since the classes $x \log x$-**P** and $x^2 \log x$-**P** are not closed under logspace reductions). A practical consequence of this is that if we know the scheduling constraints already at planning time, then it could actually be exponentially more efficient to combine planning and scheduling, than to first generate a plan and then schedule it separately. We finally note that the lower bound of Theorem 7 holds also for PAR-LPPE(**[P]**).

References

Bäckström, C., and Jonsson, P. 2011. Limits for compact representations of plans. In *21st Int'l Conf. Automated Planning and Scheduling, (ICAPS'11), Freiburg, Germany*. 18-25.

Bäckström, C. 1998. Computational aspects of reordering plans. *J. Artif. Intell. Res.* 9:99–137.

Balcázar, J. 1996. The complexity of searching implicit graphs. *Artif. Intell.* 86(1):171–188.

Bonet, B. 2010. Conformant plans and beyond: Principles and complexity. *Artif. Intell.* 174(3-4):245–269.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artif. Intell.* 69(1-2):165–204.

Downey, R., and Fellows, M. 1999. *Parameterized Complexity*. Monographs in Computer Science. Springer.

Downey, R.; Fellows, M.; and Stege, U. 1999. *Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability*, volume 49 of *DIMACS Series in Disc. Math. Theor. Comput. Sci.* 49–99.

Flum, J., and Grohe, M. 2006. *Parameterized Complexity Theory*. Springer.

Goldsmith, J.; Levy, M.; and Mundhenk, M. 1996. Limited nondeterminism. *SIGACT News* 27(2):20–29.

Gottlob, G.; Pichler, R.; and Wei, F. 2008. Monadic datalog over finite structures with bounded treewidth. arXiv:0809.3140v1.

Helmert, M. 2006. New complexity results for classical planning benchmarks. In *6'th Int'l Conf. Automated Planning and Scheduling, (ICAPS'06), Cumbria, UK*, 52–62.

Howell, R.; Jančar, P.; and Rosier, L. 1993. Completeness results for single-path Petri nets. *Inf. Comput.* 106(2):253–265.

Impagliazzo, R., and Paturi, R. 2001. On the complexity of k-SAT. *J. Comput. Syst. Sci.* 62(2):367–375.

Impagliazzo, R.; Paturi, R.; and Zane, F. 2001. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.* 63(4):512–530.

Jonsson, P., and Bäckström, C. 1998. Tractable plan existence does not imply tractable plan generation. *Ann. Math. Artif. Intell.* 22(3-4):281–296.

Kintala, C., and Fischer, P. 1977. Computations with a restricted number of nondeterministic steps. In *9th ACM Symp. Theory Comput. (STOC'77), Boulder, CO, USA*, 178–185.

Kleitman, D., and Rotschild, B. 1970. The number of finite topologies. *Proc. Amer. Math. Soc.* 25:276–282.

Nebel, B., and Bäckström, C. 1994. On the computational complexity of temporal projection, planning, and plan validation. *Artif. Intell.* 66(1):125–160.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *J. Artif. Intell. Res.* 12:271–315.

Papadimitriou, C., and Yannakakis, M. 1996. On limited nondeterminism and the complexity of the V-C dimension. *J. Comput. Syst. Sci.* 53(2):161–170.

Tsamardinos, I.; Vidal, T.; and Pollack, M. 2003. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints* 8(4):365–388.

Williams, R. 2010. Improving exhaustive search implies superpolynomial lower bounds. In *42nd ACM Symp. Theory Comput. (STOC'10), Cambridge, MA, USA*, 231–240.