Parameterized Complexity Results for Plan Reuse

Ronald de Haan[1*] and **Anna Roubickova**[2] and **Stefan Szeider**[1*]
[1]Institute of Information Systems, Vienna University of Technology, Vienna, Austria
dehaan@kr.tuwien.ac.at stefan@szeider.net
[2]Faculty of Computer Science, Free University of Bozen-Bolzano, Bolzano, Italy
anna.roubickova@stud-inf.unibz.it

Abstract

Planning is a notoriously difficult computational problem of high worst-case complexity. Researchers have been investing significant efforts to develop heuristics or restrictions to make planning practically feasible. Case-based planning is a heuristic approach where one tries to reuse previous experience when solving similar problems in order to avoid some of the planning effort. Plan reuse may offer an interesting alternative to plan generation in some settings.

We provide theoretical results that identify situations in which plan reuse is provably tractable. We perform our analysis in the framework of parameterized complexity, which supports a rigorous worst-case complexity analysis that takes structural properties of the input into account in terms of parameters. A central notion of parameterized complexity is fixed-parameter tractability which extends the classical notion of polynomial-time tractability by utilizing the effect of structural properties of the problem input.

We draw a detailed map of the parameterized complexity landscape of several variants of problems that arise in the context of case-based planning. In particular, we consider the problem of reusing an existing plan, imposing various restrictions in terms of parameters, such as the number of steps that can be added to the existing plan to turn it into a solution of the planning instance at hand.

Introduction

Planning is one of the central problems of AI with a wide range of applications from industry to academics (Ghallab, Nau, and Traverso 2004). Planning gives rise to challenging computational problems. For instance, deciding whether there exists a plan for a given planning instance is PSPACE-complete, and the problem remains at least NP-hard under various restrictions (Bylander 1994). To overcome this high worst-case complexity, various heuristics, restrictions, and relaxations of planning problems have been developed that work surprisingly well in practical settings (Hoffmann 2001; Helmert 2006). Among the heuristic approaches is *case-based planning*, which proceeds from the idea that significant planning efforts may be saved by reusing previous solutions (Kambhampati and Hendler 1992; Veloso 1994). This approach is based on the assumption that planning tasks tend to recur and that if the tasks are similar, then so are their solutions. Empirical evidence suggests that this assumption holds in many settings, and that the case-based approach works particularly well if the planning tasks require complex solutions while the modifications required on the known plans are considerably small.

So far the research on the worst-case complexity of case-based planning did not take into account the essential assumption that similar planning tasks require similar solutions. Indeed, as shown by Liberatore (2005), if none of the known solutions are helpful, then the case-based system needs to invest an effort comparable to generating the solution from scratch, and hence does not benefit from the previous experience. There is no way to benefit from the knowledge of an unrelated solution. However, the result disregards the case-based assumptions which are meant to avoid such worst cases. It seems that the classical complexity framework is not well-suited for taking such assumptions into account.

New Contribution In this paper we provide theoretical results that identify situations in which the plan reuse of the case-based approach is provably tractable. We perform our analysis in the framework of *parameterized complexity*, which supports a rigorous worst-case complexity analysis that takes structural properties of the input into account (Downey, Fellows, and Stege 1999; Niedermeier 2006; Gottlob and Szeider 2006). These structural properties are captured in terms of *parameters*, which are integer numbers that are small compared to the size of the total problem input. The theoretical analysis now considers the impact of these parameters on the worst-case complexity of the considered problems. A central notion of parameterized complexity is *fixed-parameter tractability*, which extends the classical notion of polynomial-time tractability by utilising the impact of parameters. Parameterized complexity also provides a hardness theory that, similar to the theory of NP-completeness, provides strong evidence that certain parameterized problems are not fixed-parameter tractable (fixed-parameter *in*tractable).

In the problems we study we are given a planning task together with a stored solution for a different planning task, where this solution consists of a plan and an initial state the plan is applied to. The question is to modify the existing solution to obtain a solution for the new planning task. By means of various parameters we control the modifications applied to the stored solution. For instance, we can require that the number of additional planning steps added to fit the

*Supported by the European Research Council (ERC), project COMPLEX REASON, 239962.

stored solution to the new planning task is small compared to the length of the stored solution.

In order to evaluate the impact of structural properties on the overall complexity we use parameters based on the following four restrictions, each restriction is associated with one of the four symbols L, A, V, and D.

L: bounds on the number of added planning steps

A: bounds on the size of a specified set of actions from which the added planning steps are built (each action from the set can be added several times)

V: bounds on the size of a specified set of variables that may be mentioned by the added planning steps

D: bounds on the size of a specified set of values that may be mentioned by the added planning steps

We show that parameterized by L the problem is fixed-parameter intractable even if the additional steps may only be added to the beginning and end of the plan. Parameterized by A the problem is fixed-parameter tractable. Parameterized either by V or D, the problem is fixed-parameter intractable; however, if we combine these two parameters, we achieve fixed-parameter tractability. Combining the restriction L with either V or D is not enough to achieve fixed-parameter tractability.

We obtain a full classification as shown in Figure 1. In addition, we show that the same results hold even if we reuse only some "infix" of the stored solution, i.e., if it is allowed to discard any number of actions from the beginning and end of the stored solution before the modification takes place. Finally, we prove that in more general settings where we reuse only the syntactical sequence of actions as represented by the stored solution (disregarding the actual states of the stored solution), all the combinations of parameters considered yield fixed-parameter intractability.

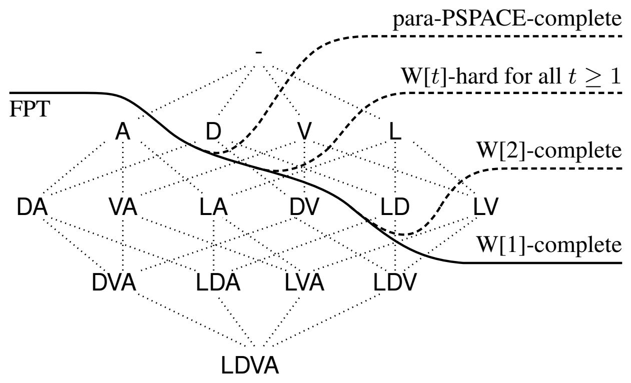

Figure 1: Overview of the fixed-parameter (in)tractability results for all combinations of restrictions L, A, V, D.

Preliminaries

In this section we introduce basic notions and notation related to (case-based) planning and parameterized complexity, which are used further in the paper.

Planning In this study, we use the SAS+ planning framework (Bäckström and Nebel 1996) in the notational variant of (Chen and Giménez 2010): An instance of the planning problem, or *a planning instance*, is a tuple $\Pi = (V, I, G, A)$, whose components are described as follows.

- V is a finite set of variables, where each $v \in V$ has an associated finite domain $D(v)$. A *state* s is a mapping defined on a set V of variables such that $s(v) \in D(v)$ for each $v \in V$. A *partial state* p is a mapping defined on a subset vars(p) of V such that for all $v \in \text{vars}(p)$ it holds that $p(v) \in D(v)$. We sometimes denote a partial state p by a set of explicit mappings $\{\, v \mapsto p(v) : v \in \text{vars}(p) \,\}$.
- I is a state called the *initial state*.
- G is a partial state called the *goal*.
- A is a set of actions; each action $a \in A$ is of the form $a = (\text{pre}(a) \Rightarrow \text{post}(a))$, where pre$(a)$ is a partial state called *precondition*, and post(a) is a partial state called *postcondition*.

For a (partial) state s and a subset $W \subseteq V$, we let $(s \upharpoonright W)$ be the (partial) state resulting from restricting s to W. We say that a (partial) state s is a *goal state*, or that s satisfies the goal, if $(s \upharpoonright \text{vars}(G)) = G$. A *plan* (for an instance Π) is a sequence of actions $p = (a_1, \dots, a_n)$. The *application* of a plan p on a state s yields a state $s[p]$, which is defined inductively as follows. The application of an empty plan ($p = \epsilon$) does not change the state ($s[\epsilon] = s$). For a non-empty plan $p = (a_1, \dots, a_n)$, we define $s[p]$ based on the inductively defined state $s[p']$, where $p' = (a_1, \dots, a_{n-1})$.

- If $(s[p'] \upharpoonright \text{vars}(\text{pre}(a_n))) \neq \text{pre}(a_n)$ then $s[p] = s[p']$, i.e., if the precondition of a_n does not hold in $s[p']$, the action a_n is not applicable and does not change the state.
- Otherwise, a_n is applicable and $s[p]$ is the state equal to post(a_n) on variables $v \in \text{vars}(\text{post}(a_n))$, and equal to $s[p']$ on $v \in V \setminus \text{vars}(\text{post}(a_n))$.

A plan p is a *solution plan* if $I[p]$ is a goal state.

If $|D(v)| \leq 2$ for each $v \in V$, then we have a *Boolean* (or *binary*) instance, which in fact gives us a notational variant of the STRIPS planning framework (Bylander 1994).

Consider the following example instance, that we will use as a running example in the remainder of this paper.

Example 1. *We let* $\Pi = (V, I, G, A)$ *be the planning instance defined below. A solution plan for* Π *would be* $p = (a_3, a_1, a_2)$.

$$
\begin{array}{ll}
V = \{v_1, v_2, v_3\} & I = \{v_1 \mapsto 0, v_2 \mapsto 0, v_3 \mapsto 0\} \\
D(v_1) = \{0,1,2\} & D(v_2) = D(v_3) = \{0,1\} \\
G = \{v_1 \mapsto 2\} & A = \{a_1, a_2, a_3, a_4\} \\
a_1 = (\{v_1 \mapsto 0, v_2 \mapsto 1\}, \{v_1 \mapsto 1\}) & \\
a_2 = (\{v_1 \mapsto 1, v_2 \mapsto 1\}, \{v_1 \mapsto 2\}) & \\
a_3 = (\emptyset, \{v_2 \mapsto 1, v_3 \mapsto 1\}) & \\
a_4 = (\emptyset, \{v_3 \mapsto 0\}) &
\end{array}
$$

Case-Based Planning Case-based planning (CBP) is a type of case-based reasoning that involves the use of stored experiences (called *cases*) of solving analogous problems. Often, a case is composed of a planning instance $\Pi' = (V, J, H, A)$ and a solution plan c of Π'. The plan c can be replaced by some other information related to the search for a solution to Π' (e.g., a set of justifications) (Kambhampati and Hendler 1992; Veloso 1994; Hanks and Weld 1995). A plan library, or a *case base*, is a collection of such cases, constituting the experience of the planner. For more detailed explanation of implementation choices of specific planners we refer to the survey of Spalazzi (2001).

Example 2. *Consider the case* (Π', c)*, where the planning instance* Π' *coincides with the instance defined in Example 1 on* V, A*, and where* $J = \{v_1 \mapsto 0, v_2 \mapsto 1, v_3 \mapsto 0\}$ *and* $H = \{v_1 \mapsto 2\}$*. The solution plan* $c = (a_1, a_2)$ *can be reused to find the solution* p *given in Example 1.*

When faced with a new problem, the case-based planner follows a sequence of steps common in case-based reasoning (Aamodt and Plaza 1994). First, it queries the library to *retrieve* cases suitable for reuse. The *reuse* step modifies the retrieved solution(s) to solve the new problem and such a new solution is validated in the *revision* phase by execution, simulated execution, etc. The verified solution may be eventually stored in the case base during a *retention* process.

In this paper, we focus on theoretical properties of plan reuse and therefore we skip the retrieval, we simply assume that together with the problem to solve we are also given a case which contains a suitable solution for reuse. In other words, we assume that the instance Π to be solved coincides on the sets V, A with the instance Π' provided in the case.

Parameterized Complexity Here we introduce the relevant concepts of parameterized complexity theory. For more details, we refer to the works of Downey and Fellows (1999), Downey, Fellows and Stege (1999), Flum and Grohe (2006), Niedermeier (2006), and Gottlob and Szeider (2006).

In the traditional setting of considering the complexity of a problem, the input size n of the instance is the only measure available. Parameterized complexity is a two-dimensional framework to classify the complexity of problems based on their input size n and some additional parameter k. An instance of a parameterized problem is a pair (I, k) where I is the main part of the instance, and k is the *parameter*. A parameterized problem is *fixed-parameter tractable* if it can be solved by a fixed-parameter algorithm, i.e., if instances (I, k) can be solved in time $O(f(k)n^c)$, where f is a computable function of k, c is a constant, and n is the size of I. FPT denotes the class of all fixed-parameter tractable decision problems. Many problems that are classified as intractable in the classical setting can be shown to be fixed-parameter tractable.

Parameterized complexity also offers a *completeness theory*, similar to the theory of NP-completeness. This allows the accumulation of strong theoretical evidence that a parameterized problem is not fixed-parameter tractable. Hardness for parameterized complexity classes is based on *fpt-reductions*, which are many-one reductions where the parameter of one problem maps into the parameter for the other. A parameterized problem L is fpt-reducible to another parameterized problem L' if there is a mapping R from instances of L to instances of L' such that (i) $(I, k) \in L$ if and only if $(I', k') = R(I, k) \in L'$, (ii) $k' \leq g(k)$ for a computable function g, and (iii) R can be computed in time $O(f(k)n^c)$ for a computable function f and a constant c, where n is the size of I.

Central to the completeness theory is the hierarchy of parameterized complexity classes FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots \subseteq$ W[P] $\subseteq$ para-PSPACE, where all inclusions are believed to be strict. Each of the classes W[t] for $t \geq 1$ and W[P] contains all parameterized problems that can be reduced to a certain parameterized satisfiability problem under fpt-reductions. For instance, for W[2], the corresponding satisfiability problem asks whether a given CNF formula has a satisfying assignment that sets exactly k variables to true. A sufficient condition for a problem to be hard for the class para-PSPACE is that the problem is PSPACE-hard for a single value of the parameter (Flum and Grohe 2003). There is strong evidence that a parameterized problem that is hard for any of these intractability classes is not in FPT.

We use the following problems to prove some fixed-parameter intractability results.

PARTITIONED-CLIQUE is a W[1]-complete problem (Fellows et al. 2009). The instances are tuples (V, E, k), where V is a finite set of vertices partitioned into k subsets $V_1, \ldots, V_k$, (V, E) is a simple graph, and $1 \leq k$ is a parameter. The question is whether there exists a k-clique in (V, E) that contains a vertex in each V_i.

HITTING-SET is a W[2]-complete problem (Downey and Fellows 1995). The instances are tuples (S, C, k), where S is a finite set of nodes, C is a collection of subsets of S, and $1 \leq k \leq |C|$ is a parameter. The question is whether there exists a hitting set $H \subseteq S$ such that $|H| \leq k$ and $H \cap c \neq \emptyset$ for all $c \in C$.

p-WSAT(CIRC) (weighted circuit satisfiability) is a W[P]-complete problem (Downey and Fellows 1995). The instances are pairs (C, k), where C is a Boolean circuit, and $1 \leq k$ is a parameter. The question is whether there exists a satisfying assignment of C that sets at most k input nodes to true.

LONGEST-COMMON-SUBSEQUENCE-I is a parameterized problem that is W[t]-hard for all $t \geq 1$ (Bodlaender et al. 1995). As input, it takes k strings $X_1, \ldots, X_k$ over an alphabet Σ, and a positive integer m. The parameter is k. The question is whether there is a string $X \in \Sigma^*$ of length at least m that is a subsequence of X_i for all $1 \leq i \leq k$.

Related Work

The first paper providing a complexity-theoretical study of plan reuse (Nebel and Koehler 1995) considered so-called *conservative* plan reuse. Conservative plan reuse maximizes the unchanged part of the known solution. The authors showed that such a plan reuse is not provably more efficient than plan generation. Moreover, they show that identifying what is the maximal reusable part of the stored solution is an additional source of hardness.

Liberatore (2005) studied the problem of plan reuse in a different fashion, interpreting the case (or the case base) as a "hint" that makes the search for the solution plan more informed. The complexity results he provides do not improve over the complexity of uninformed plan generation. He does however give a tractable compilation result for planning instances that differ from the stored instances only in a constant number of valuations from the initial state and goal.

The parameterized complexity of planning was first studied by Downey, Fellows, and Stege (1999) and, more recently, by Bäckström et al. (2012; 2013), using the solution length as the parameter. The analysis by Bäckström et al. reveals that the planning problem is W[2]-complete and there exist fragments that are W[1]-complete and other fragments that are fixed-parameter tractable. More specifically, they provide a full classification of SAS+ planning under all combinations of the P, U, B and S restrictions introduced by Bäckström and Klein (1991), and they provide a full classification of STRIPS planning under the syntactical restrictions studied by Bylander (1994).

Parameterized Complexity of Plan Reuse

In this paper, we study the parameterized complexity of reusing a plan. However, as this work is motivated by plan reuse in the context of case-based planning, we exploit assumptions common in the case-based approaches to ensure that there is a solution at hand that can be reused. Also, we consider a more specific form of a plan reuse (*case reuse*) and generalizations thereof.

Reusing the Case In its general form, the classical complexity of plan reuse is not better than the one of plan generation.Liberatore (2005) has shown it to be PSPACE-complete. However, the case-based approach assumes that similar problems have similar solutions (Leake 1996). This means that cases can either be used to yield a solution by applying only a limited number of modifications or will not be helpful at all in finding a solution. When considering the complexity of plan reuse in the classical setting, we cannot exclude the worst case in which the case provides no guidance and where an uninformed search similar to the traditional plan generation is needed. When using the framework of parameterized complexity instead, we can capture the computational complexity of reusing a case in those settings where it can be used to get a solution plan with only a limited amount of modification.

We consider a case (Π', c) useful for solving an instance Π if c can be modified to a solution plan p for Π by means of limited modification. We define the following template CASEMOD for the decision problems, intended to find such useful cases.

CASEMOD

Instance: a planning instance $\Pi = (V, I, G, A)$; a case (Π', c) consisting of an instance $\Pi' = (V, J, H, A)$[1] and its solution plan $c = (c_1, \dots, c_l)$; a subset of actions $A' \subseteq A$; and an integer M.

Question: Does there exist a sequence of actions $(g_1, \dots, g_m) \in (A')^m$ for some $m \leq M$, and does there exist some $0 \leq i \leq m$, such that $(g_1, \dots, g_i, c_1, \dots, c_l, g_{i+1}, \dots, g_m)$ is a solution plan for Π and $I[(g_1, \dots, g_i)] = J$?

[1]In the remainder of the paper, we will often specify an instance Π' in the definition above only by its value of J. Since Π and Π' coincide on V and A, and the choice of H is not relevant for answering the question, this will suffice for most purposes.

R	R-CASEMOD	
{A}	FPT	(Thm 1)
{V, D}	FPT	(Cor 1)
{L, V}	W[1]-complete	(Thm 4)
{L}	W[2]-complete	(Prop 1)
{L, D}	W[2]-complete	(Cor 2)
{V}	W[t]-hard for all $t \geq 1$	(Thm 2)
{D}	para-PSPACE-complete	(Thm 3)

Table 1: Map of parameterized complexity results.

The sequence $g = (g_1, \dots, g_m)$ in the definition above can be thought of as the "glue" that enables the reuse of the plan c by connecting the new initial state I to the beginning of the case (Π', c), using the plan c to reach its goal and connecting it to the goal required by instance Π. In the following, we will often refer to these action occurrences (or steps) as *glue steps*. Though such a reuse may seem naive, CASEMOD is in fact implemented and used by CBP system FAROFF (Tonidandel and Rillo 2002).

Example 3. *Let* Π *be the planning instance from Example 1 and* (Π', c) *the case from Example 2. Consider the instance for* CASEMOD*, given by* $(\Pi, (\Pi', c), A', M)$*, where* $A' = \{a_3, a_4\}$ *and* $M = 3$*. This is a positive instance, since the solution plan* $p = (a_3, a_4, a_1, a_2)$ *can be constructed from* c *by adding the sequence of actions* (a_3, a_4) *from* A'*,* $I[(a_3, a_4)] = J$ *and* $|(a_3, a_4)| \leq M$*.*

We will consider a number of different parameterizations for CASEMOD, where in each case the parameter is intended to capture the assumption that the plan given in the case is similar to the solution we are looking for. In order to define these variants, we define the problems R-CASEMOD, for any subset R of {L, V, D, A}. The choice of the parameterization depends on this set R of restrictions.

- If R includes L, we add to the parameterization the allowed maximum length of the glue sequence g.
- If R includes V, we add to the parameterization the number of variables mentioned in the actions in A'.
- If R includes D, we add to the parameterization the number of values mentioned in the actions in A'.
- If R includes A, we add to the parameterization the number of actions in A'.

For instance, the parameter in the problem {L, A}-CASEMOD is $k + l$, where k is the maximum length allowed for the sequence of glue steps and $l = |A'|$.

In order to establish the parameterized complexity landscape for various combinations of these restrictions as sketched in Figure 1, we need to prove the following results. We show that R-CASEMOD is fixed-parameter tractable for $R \in \{\{A\}, \{V, D\}\}$, that it is W[1]-complete for $R = \{L, V\}$, that it is W[2]-complete for $R \in \{\{L\}, \{L, D\}\}$, that it is W[$t$]-hard for all $t \geq 1$, for $R = \{V\}$, and that it is para-PSPACE-complete for $R = \{D\}$. These results are summarized in Table 1.

The modification of a plan (or a case) concerns addition of actions to the plan stored in the case. One intuitive way to restrict the amount of modification that is allowed in order to reuse the case is to restrict the number of allowed additional steps, resulting in the L restriction. It is believed (Kambhampati and Hendler 1992) that the presence of a similar solution, or rather the fact that only k actions need to be added to the stored plan c in order to find the plan p, will make the decision problem of existence of p (and also its generation) easier than if no suitable solution c is available. Unfortunately, the following result shows that the corresponding problem $\{\mathsf{L}\}$-CASEMOD remains hard.

Proposition 1. $\{\mathsf{L}\}$-CASEMOD *is* W[2]*-complete.*

Proof. The result follows from the W[2]-completeness proof of finding a solution plan of at most k action occurrences (the k-step planning problem) given by Bäckström et al. (2012). They proved that W[2]-hardness already holds for complete goal states. Now, by letting $(c, J) = (\epsilon, G)$, the k-step planning problem directly reduces to $\{\mathsf{L}\}$-CASEMOD.

To show W[2]-membership, we sketch the following reduction to the k-step planning problem. We introduce an additional operator $(J \cup \{\star \mapsto 0\} \Rightarrow J[c] \cup \{\star \mapsto 1\})$, where $\star$ is a fresh variable. Furthermore, we let $\{\star \mapsto 0\} \in I$ and $\{\star \mapsto 1\} \in G$, and we let $k' = k + 1$. It is straightforward to verify that this reduces $\{\mathsf{L}\}$-CASEMOD to the k'-step planning problem. $\square$

Intuitively, the reason of such a result is that the large number of different actions to choose from is a source of hardness. Bäckström et al. showed that for the k-step planning problem, complexity results can be improved by considering only planning instances whose actions satisfy the condition of post-uniqueness. Similarly, we can require the set of actions A', from which glue steps can be taken, to be post-unique (Bäckström et al. 2012). This parameterized problem is in fact fixed-parameter tractable (this result follows from Theorem 5 in (Bäckström et al. 2012)).

In a similar way, parameterizing directly by the cardinality of A' also provides fixed-parameter tractability:

Theorem 1. $\{\mathsf{A}\}$-CASEMOD *is in* FPT.

Proof. We have that $k = |A'|$. Then the number of states s' reachable from any state s by actions from A' is bounded by a function of k and can be enumerated in fixed-parameter tractable time. Similar bounds hold for all the states s'' reachable from $s'[c]$ for each such s'. Overall, checking if any of these states s'' satisfies the goal state can thus be done in fixed-parameter tractable time. $\square$

As a consequence of Theorem 1, we get another fixed-parameter tractability result.

Corollary 1. $\{\mathsf{V}, \mathsf{D}\}$-CASEMOD *is in* FPT.

Proof. If the set A' of actions refers to at most k variables and at most m values, then the number of different actions that A' can possibly contain is bounded by $(m + 1)^{2k}$. The result then follows from Theorem 1. $\square$

The fact that the above results are the only fixed-parameter tractable results under the considered restrictions suggests that plan reuse is not the answer to the high computational complexity of planning in general. However, we can use these results to identify settings in which plan reuse is likely to perform well. For example, Theorem 1 suggests that replanning in case of an execution failure is tractable to implement as plan reuse, provided that the number of applicable actions is limited due to, e.g., limited resources.

These results for plan reuse as implemented in case-based planning are quite unpleasant as in such settings usually $A = A'$. Additionally, $|A|$ tends to be quite high, as the set of actions is obtained by grounding a set of (few) operators (propositional implication rules) over a set of potentially many objects, giving a rise to a rich set of actions which only very rarely satisfies the condition of post-uniqueness to make $\{\mathsf{L}\}$-CASEMOD fixed-parameter tractable. Nevertheless, these claims suggest that, besides identifying *where* to apply the glue steps, a case-based planning system needs to employ heuristics to identify *which* glue steps may be useful. Even though $\{\mathsf{V}, \mathsf{D}\}$-CASEMOD is in FPT, parameterizing only on the number of variables occurring in actions in A', or only on the number of values occurring in actions in A', yields fixed-parameter intractability.

Theorem 2. $\{\mathsf{V}\}$-CASEMOD *is* W[t]*-hard for all* $t \geq 1$.

Proof. We prove the result by giving an fpt-reduction from LONGEST-COMMON-SUBSEQUENCE-I, which is W[t]-hard for all $t \geq 1$. Let the strings $X_1, \ldots, X_k$ over the alphabet Σ and the integer m constitute an instance of LONGEST-COMMON-SUBSEQUENCE-I. For a string X of length l we write $X[0] \ldots X[l - 1]$. For each X_i we let $l_i = |X_i|$. We construct an instance of $\{\mathsf{V}\}$-CASEMOD specified by $\Pi = (V, I, G, A)$, (c, J), A' and M. We let $(c, J) = (\epsilon, G)$, $A' = A$ and M be a sufficiently large number (that is, $M \geq \sum_{1 \leq i \leq k} |X_i| + (k + 1)m + k$). Also, we define:

$$\begin{array}{rl}
V = & \{v_1, \ldots, v_k, s_1, \ldots, s_k, t_1, \ldots, t_k, w\}; \\
D(v_i) = & \{0, \ldots, l_i\}; \\
D(s_i) = & \Sigma \cup \{\star\}; \\
D(t_i) = & \{\text{none}, \text{read}, \text{used}\}; \\
D(w) = & \{0, \ldots, m\}; \\
A = & A_{\text{skip}} \cup A_{\text{read}} \cup A_{\text{check}} \cup A_{\text{finish}}; \\
A_{\text{skip}} = & \{\, (\{v_i \mapsto u, t_i \mapsto \text{none}\} \Rightarrow \{v_i \mapsto u + 1, \\
& t_i \mapsto \text{none}\}), (\{v_i \mapsto u, t_i \mapsto \text{used}\} \Rightarrow \\
& \{v_i \mapsto u + 1, t_i \mapsto \text{none}\}) : 1 \leq i \leq k, \\
& 0 \leq u < l_i \,\}; \\
A_{\text{read}} = & \{\, (\{v_i \mapsto u, t_i \mapsto \text{none}\} \Rightarrow \{s_i \mapsto X_i[u], \\
& t_i \mapsto \text{read}\}) : 1 \leq i \leq k, 0 \leq u < l_i \,\}; \\
A_{\text{check}} = & \{\, (\{t_1 \mapsto \text{read}, \ldots, t_k \mapsto \text{read}, s_1 \mapsto \sigma, \ldots, \\
& s_k \mapsto \sigma, w = u\} \Rightarrow \{t_1 \mapsto \text{used}, \ldots, \\
& t_k \mapsto \text{used}, w \mapsto u + 1\}) : 0 \leq u < m, \\
& \sigma \in \Sigma \,\}; \\
A_{\text{finish}} = & \{\, (\{v_i \mapsto l_i\} \Rightarrow \{t_i \mapsto \text{none}, s_i \mapsto \star\}) : \\
& 1 \leq i \leq k \,\}; \\
I = & \{\, v_i \mapsto 0, s_i \mapsto \star, t_i \mapsto \text{none} : 1 \leq i \leq k \,\} \cup \\
& \{w \mapsto 0\};\ \text{and} \\
G = & \{\, v_i \mapsto l_i, s_i \mapsto \star, t_i \mapsto \text{none} : 1 \leq i \leq k \,\} \cup \\
& \{w \mapsto m\}.
\end{array}$$

Note that $|V| = 3k + 1$.

The idea behind the reduction is that any solution plan that results in an assignment of variable w to any $d \geq 1$ corresponds to a witness that the strings $X_1, \ldots, X_k$ have a common subsequence of length d. The variables $v_1, \ldots, v_k$ correspond to the position of reading heads on the strings that can only move from left to right, and the variables $s_1, \ldots, s_k$

are used to read symbols in the string on the position of the reading heads. The variables $t_1, \dots, t_k$ are used to ensure that each symbol is read at most once (each symbol is either read by using an action in A_{read} or skipped by using an action in A_{skip}). Then the variable w can only be increased if in all strings the same symbol is read (by using an action in A_{check}). The actions A_{finish} are used to be able to enforce a complete goal state.

It is now straightforward to verify that there exists a common subsequence X for $X_1, \dots, X_k$ of length m if and only if the constructed instance is a yes-instance. ☐

As mentioned above in the preliminaries, if we restrict the planning instances to Boolean values, we get a framework corresponding to the STRIPS planning framework. By the fact that $\{\mathsf{V}, \mathsf{D}\}$-CASEMOD is fixed-parameter tractable, we get that $\{\mathsf{V}\}$-CASEMOD for STRIPS instances is also fixed-parameter tractable.

By naively keeping track of all states reachable from the initial state (which are at most n^k many, for $n = |D|$ and $k = |V|$) we get that $\{\mathsf{V}\}$-CASEMOD can be solved in polynomial time for each constant value of k. As a consequence, the following theorem shows that $\{\mathsf{D}\}$-CASEMOD is of higher complexity than $\{\mathsf{V}\}$-CASEMOD (unless P = PSPACE).

Theorem 3. $\{\mathsf{D}\}$-CASEMOD *is* para-PSPACE-*complete.*

Proof. The para-PSPACE-membership result follows from the fact that $\{\mathsf{D}\}$-CASEMOD, when unparameterized, is in PSPACE (Bäckström and Nebel 1996).

For the hardness result, consider the case where the number k of values allowed in the set of actions A' is 2. The problem then reduces to the problem of finding a solution plan for the Boolean planning instance Π, in case we let $(c, J) = (\epsilon, G)$. Bäckström and Nebel (1996) showed that finding a solution plan for Boolean planning instances (even for complete goal states) is PSPACE-hard. Since this hardness result holds already for a single value of k, the para-PSPACE-hardness result follows (Flum and Grohe 2003). ☐

Parameterizing on the combination of the number of allowed additional steps together with either the number of variables or the number of values occurring in actions in A' is not enough to ensure fixed-parameter tractability.

Theorem 4. $\{\mathsf{L}, \mathsf{V}\}$-CASEMOD *is* W[1]-*complete.*

Proof. W[1]-membership can be proven analogously to the W[1]-membership proof given by Bäckström et al. (2012, Theorem 4) for the k-step planning problem restricted to actions with one postcondition. In this proof the problem is reduced to a certain first-order model checking problem.

For the hardness result, we reduce from the W[1]-complete problem PARTITIONED-CLIQUE. Let (V, E, k) be an instance of PARTITIONED-CLIQUE, where V is partitioned into $V_1, \dots, V_k$. We define the instance $(\Pi, (\Pi', c), A', k')$ of $\{\mathsf{L}, \mathsf{V}\}$-CASEMOD as follows: $\Pi = (W, I, G, A)$, Π' is specified by its initial state J, $(c, J) = (\epsilon, G)$, $k' = 2k + \binom{k}{2}$ and $A' = A$. We define:

$$\begin{aligned}
W &= \{x_1, \dots, x_k\} \cup \{\, y_{i,j} : 1 \le i < j \le k \,\};\\
D(x_i) &= V_i \cup \{\star\} \text{ for all } x_i \text{ (and arbitrary } \star \notin V);\\
D(y_{i,j}) &= \{0, 1\} \text{ for all } y_{i,j};\\
A &= \{\, \text{guess}^i_d, \text{clear}^i_d : 1 \le i \le k, d \in V_i \,\} \cup\\
&\quad \{\, \text{check}^{v,w}_{i,j} : 1 \le i < j \le k, v \in V_i,\\
&\quad w \in V_j, \{v, w\} \in E \,\};\\
\text{guess}^i_d &= (\emptyset \Rightarrow \{x_i \mapsto d\}), \text{ for each guess}^i_d;\\
\text{clear}^i_d &= (\emptyset \Rightarrow \{x_i \mapsto \star\}), \text{ for each clear}^i_d;\\
\text{check}^{v,w}_{i,j} &= (\{x_i \mapsto v, x_j \mapsto w\} \Rightarrow \{y_{i,j} \mapsto 1\}),\\
&\quad \text{for each check}^{v,w}_{i,j};\\
I &= \{\, x_i \mapsto \star : 1 \le i \le k \,\} \cup\\
&\quad \{\, y_{i,j} \mapsto 0 : 1 \le i < j \le k \,\}; \text{ and}\\
G &= \{\, x_i \mapsto \star : 1 \le i \le k \,\} \cup\\
&\quad \{\, y_{i,j} \mapsto 1 : 1 \le i < j \le k \,\}.
\end{aligned}$$

The intuition behind the reduction is as follows. The budget of k' actions allows for k guessing steps, to set the variables x_i using actions guess^i_d; $\binom{k}{2}$ verification steps, to set the variables $y_{i,j}$ to 1 using actions $\text{check}^{v,w}_{i,j}$; and k cleanup steps, to reset the variables x_i using actions clear^i_d. The only way to achieve the goal state is by guessing a k-clique.

It is now straightforward to verify that the graph (V, E) has a k-clique if and only if there exist plans p, p' of total length k' such that $I[p] = J$ and $J[c][p']$ satisfies G. ☐

Corollary 2. $\{\mathsf{L}, \mathsf{D}\}$-CASEMOD *is* W[2]-*complete.*

Proof. The claim follows directly from the proof of Proposition 1, since k-step planning is W[2]-complete already for Boolean planning instances (Bäckström et al. 2012). ☐

The above results together give us the complete parameterized complexity characterization as depicted in Figure 1.

Reusing an infix of the case As a slight generalization of the CASEMOD problem, we consider the problem CASEMOD*. In this problem, we require not that the full plan c from the case is being reused together with its initial state J, but that any infix c' of the plan (i.e., any subplan c' resulting from removing any prefix and postfix from c) is reused with its corresponding initial state J'. Formally, the question becomes whether there exists a sequence of actions $(g_1, \dots, g_m) \in (A')^m$ for some $m \le M$, and whether there exists some $0 \le i \le m$ and some $1 \le i_1 \le i_2 \le l$ such that $(g_1, \dots, g_i, c_{i_1}, \dots, c_{i_2}, g_{i+1}, \dots, g_m)$ solves the new planning instance Π and $I[(g_1, \dots, g_i)] = J[(c_1, \dots, c_{i_1 - 1})]$, where $c = (c_1, \dots, c_l)$.

The following results show that this generalization does not change the parameterized complexity results that we obtained in the previous section.

Observation 1. *Whenever R-*CASEMOD *is in* FPT*, then also R-*CASEMOD* *is in* FPT.

Proof. Let $c = (c_1, \dots, c_n) \in A^n$. There are only n^2 different ways of selecting subplans $(c_d, \dots, c_e)$ to consider, for $1 \le d \le e \le n$. For each of these, we can compute the initial state $J[(c_1, \dots, c_{d-1})]$ in linear time. Simply trying all these n^2 possibilities using the algorithm for R-CASEMOD results in an fixed-parameter tractable algorithm for R-CASEMOD*. ☐

Theorem 5. *The completeness results in Proposition 1, Theorems 2, 3 and 4 and Corollary 2 also hold for the corresponding variants for R-*CASEMOD*.

Proof (sketch). For the hardness results, it suffices to note that the hardness proofs of these theorems use a case containing the empty plan ϵ.

For the membership results, we note that the R-CASEMOD* problem can be solved by answering the disjunction of polynomially many (independent) R-CASEMOD instances. The W[t]-membership results can then be proved by encoding the R-CASEMOD instances as instances of certain first-order model checking problems (Bäckström et al. 2012), and combining these into one model checking problem instance that is equivalent to the disjunction of the separate R-CASEMOD instances. For the para-PSPACE-membership result, we can straightforwardly evaluate the disjunction of the R-CASEMOD instances in polynomial space. ☐

Generalized infix reuse The problem of CASEMOD can be generalized even further by reusing an infix of the stored solution plan c from *any state* that satisfies the preconditions of the plan infix. For this problem, the instances coincide with those of CASEMOD, but the question is:

Question: Does there exist a sequence of actions $(g_1, \dots, g_m) \in (A')^m$ for some $m \leq M$, and does there exist $0 \leq i \leq m$ and $0 \leq i_1 \leq i_2 \leq l$ such that $(g_1, \dots, g_i, c_{i_1}, \dots, c_{i_2}, g_{i+1}, \dots, g_m)$ is a solution plan for Π and for all $i_1 \leq j \leq i_2$ we have that action c_j is applicable in $I[(g_1, \dots, g_i, c_{i_1}, \dots, c_{j-1})]$?

This generalization does not change the parameterized complexity results stated in Table 1. In all cases where R-CASEMOD is fixed-parameter tractable, we can obtain a fixed-parameter tractable algorithm to solve the above problem, since in those cases we can enumerate all states reachable from a given state in fixed-parameter tractable time. For the fixed-parameter intractability results, the hardness follows straightforwardly from the hardness proofs for the corresponding R-CASEMOD problems.

Reusing a sequence of actions In principle, there is no need to require anything from the state to which the stored plan is applied. Therefore we will consider the following generalization of the CASEMOD problems discussed above. In this problem, denoted by PLANMOD, we remove the requirement that the additional steps added before the plan c result in the initial state J or some other state that satisfies the preconditions of (the infix of) the plan c. Also, we allow the insertion of additional steps in the middle of the plan c. Formally, the question then becomes whether there exists some $m \leq k$, a sequence of actions $g = (g_1, \dots, g_m) \in A^m$, and some sequence of actions $p = (p_1, \dots, p_{l+m})$ such that p is a solution plan of Π and p can be divided into two subsequences c and g, i.e., interleaving c and g yields p. In other words, the additional steps g can be used anywhere before, after or in the middle of the plan c. Since we do not restrict ourselves to any particular state being visited in our solution plan, the actions in the glue sequence g can be used anywhere before, after or in the middle of the plan c. Similarly to the case for CASEMOD, we define the variants PLANMOD*, R-PLANMOD and R-PLANMOD*. In the following, we show that all variants of PLANMOD are fixed-parameter intractable.

Theorem 6. $\{\mathsf{L}, \mathsf{V}, \mathsf{D}, \mathsf{A}\}$-PLANMOD *is* W[P]*-complete.*

Proof (sketch). For W[P]-membership, we sketch how to reduce $\{\mathsf{L}, \mathsf{V}, \mathsf{D}, \mathsf{A}\}$-PLANMOD to the problem of determining whether a nondeterministic Turing machine T accepts the empty string within a bounded number of steps using at most k' nondeterministic steps (parameterized by k'). Since this parameterized halting problem is in W[P] (Cesati 2003), this suffices to show W[P]-membership. First T guesses k pairs (m_i, a_i), for $0 \leq m_i \leq |c|$ and $a_i \in A'$. Pair (m_1, a_1) corresponds to the application of the first m_1 actions from the given plan c, followed by the application of action a_i. Similarly, for each $i > 1$, pair (m_i, a_i) corresponds to the application of the next m_i actions from the given plan c, followed by the application of action a_i. Then T (deterministically) verifies whether applying the plan corresponding to $(m_1, a_i), \dots, (m_k, a_k)$ is a solution plan.

To prove W[P]-hardness, we reduce from p-WSAT(CIRC). Let C be a circuit for which we want to check whether there exists a satisfying assignment of weight at most k. Let $x_1, \dots, x_n$ be the input nodes, $y_1, \dots, y_m$ the internal nodes, and z the output node of C, together denoted nodes(C). We assume without loss of generality that C contains only AND and NEGATION nodes. Since C is acyclic, we let the sequence $(g_1, \dots, g_l)$ denote the nodes of C in any order such that for each g_i we have that $j < i$ for all input nodes g_j of g_i. We construct an instance of $\{\mathsf{L}, \mathsf{V}, \mathsf{D}, \mathsf{A}\}$-PLANMOD consisting of a planning instance $\Pi = (V, I, G, A)$, a plan c, and an integer k'. We let $k' = k$, and we define:

$$\begin{aligned}
V &= \text{nodes}(C) \cup \{\sigma\};\\
D(v) &= \{0,1\} \text{ for all } v \in V;\\
I &= \{\, v \mapsto 0 : v \in V \,\};\\
G &= \{z \mapsto 1\};\\
A &= \{a^{x_1}, \dots, a^{x_n}\} \cup \{a^{\text{on}}, a^{\text{off}}\} \cup \{a^{g_1}, \dots, a^{g_l}\};\\
A' &= \{a^{\text{on}}\};\\
a^{x_i} &= (\{\sigma \mapsto 1\} \Rightarrow \{x_i \mapsto 1\}) \text{ for each } x_i;\\
a^{\text{on}} &= (\emptyset \Rightarrow \{\sigma \mapsto 1\});\\
a^{\text{off}} &= (\emptyset \Rightarrow \{\sigma \mapsto 0\}); \text{ and}\\
c &= (a^{\text{off}}, a^{x_1}, a^{\text{off}}, a^{x_2}, \dots, a^{\text{off}}, a^{x_n}, a^{g_1}, \dots, a^{g_l}).
\end{aligned}$$

We define a^{g_i} for each g_i as follows. If g_i is a NEGATION node with input y, we let $a^{g_i} = (\{y \mapsto 0\} \Rightarrow \{g_i \mapsto 1\})$. If g_i is an AND node with inputs $y_1, \dots, y_u$, we let $a^{g_i} = (\{y_1 \mapsto 1, \dots, y_u \mapsto 1\} \Rightarrow \{g_i \mapsto 1\})$.

It is now straightforward to verify that (Π, c, k') is a yes-instance of $\{\mathsf{L}, \mathsf{V}, \mathsf{D}, \mathsf{A}\}$-PLANMOD if and only if the circuit C has a satisfying assignment of weight k. ☐

Note that the proof of the above theorem suffices to show fixed-parameter intractability of all variants of the PLANMOD and PLANMOD* problems. We also point out that this fixed-parameter intractability result holds even when the problem is restricted to instances for which the entire set A of actions satisfies post-uniqueness.

Conclusion

We provided theoretical results, using the framework of parameterized complexity, to identify situations in which plan reuse is provably tractable. We drew a detailed map of the parameterized complexity landscape of several variants of problems that arise in the context of case-based planning. In particular, we considered the problem of reusing an existing plan, imposing various restrictions in terms of parameters, such as the number of steps that can be added to the existing plan to turn it into a solution of the planning instance at hand.

The results show that contrary to the common belief, the fact that the number of modifying actions is small does not guarantee tractability on its own. We additionally need to restrict the set of actions that can participate in the modifications. This indicates the need for a good heuristic function that identifies a limited set of actions used for modifications.

In the future, these results may be extended to richer planning formalisms, e.g., considering variables of different types or using predicates to express certain properties related to a planning domain rather than planning instance.

References

Aamodt, A., and Plaza, E. 1994. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications* 7(1):39–59.

Bäckström, C., and Klein, I. 1991. Planning in polynomial time: the SAS-PUBS class. *Computational Intelligence* 7:181–197.

Bäckström, C., and Nebel, B. 1996. Complexity results for SAS+ planning. *Computational Intelligence* 11:625–655.

Bäckström, C.; Chen, Y.; Jonsson, P.; Ordyniak, S.; and Szeider, S. 2012. The complexity of planning revisited - a parameterized analysis. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Bäckström, C.; Jonsson, P.; Ordyniak, S.; and Szeider, S. 2013. Parameterized complexity and kernel bounds for hard planning problems. In Spirakis, P., and Serna, M., eds., *Algorithms and Complexity (CIAC 2013)*, volume 7878 of *Lecture Notes in Computer Science*.

Bodlaender, H. L.; Downey, R. G.; Fellows, M. R.; and Wareham, H. T. 1995. The parameterized complexity of sequence alignment and consensus. *Theoretical Computer Science* 147:31–54.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69:165–204.

Cesati, M. 2003. The Turing way to parameterized complexity. *Journal of Computer and System Sciences* 67:654–685.

Chen, H., and Giménez, O. 2010. Causal graphs and structurally restricted planning. *Journal of Computer and System Sciences* 76(7):579–592.

Downey, R. G., and Fellows, M. R. 1995. Fixed-parameter tractability and completeness I: Basic results. *SIAM J. Comput.* 24(4):873–921.

Downey, R. G., and Fellows, M. R. 1999. *Parameterized Complexity*. Monographs in Computer Science. New York: Springer Verlag.

Downey, R.; Fellows, M. R.; and Stege, U. 1999. Parameterized complexity: A framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, volume 49 of *AMS-DIMACS*, 49–99. American Mathematical Society.

Fellows, M. R.; Hermelin, D.; Rosamond, F. A.; and Vialette, S. 2009. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science* 410(1):53–61.

Flum, J., and Grohe, M. 2003. Describing parameterized complexity classes. *Information and Computation* 187(2):291–319.

Flum, J., and Grohe, M. 2006. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Berlin: Springer Verlag.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. Morgan Kaufmann.

Gottlob, G., and Szeider, S. 2006. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction, and database problems. *The Computer Journal* 51(3):303–325. Survey paper.

Hanks, S., and Weld, D. 1995. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research (JAIR)* 2:319–360.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26(1):191–246.

Hoffmann, J. 2001. Ff: The fast-forward planning system. *AI magazine* 22(3):57.

Kambhampati, S., and Hendler, J. A. 1992. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence* 55:193–258.

Leake, D. B., ed. 1996. *Case-Based Reasoning*. Cambridge, Massachusetts: The MIT Press.

Liberatore, P. 2005. On the complexity of case-based planning. *Journal of Experimental & Theoretical Artificial Intelligence* 17(3):283–295.

Nebel, B., and Koehler, J. 1995. Plan reuse versus plan generation: A complexity-theoretic perspective. *Artificial Intelligence- Special Issue on Planning and Scheduling* 76:427–454.

Niedermeier, R. 2006. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and its Applications. Oxford: Oxford University Press.

Spalazzi, L. 2001. A survey on case-based planning. *Artificial Intelligence Review* 16(1):3–36.

Tonidandel, F., and Rillo, M. 2002. The FAR-OFF system: A heuristic search case-based planning. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *AIPS*, 302–311. AAAI.

Veloso, M. 1994. *Planning and Learning by Analogical Reasoning*, volume 886 of *Lecture Notes in Artificial Intelligence and Lecture Notes in Computer Science*. New York, USA: Springer-Verlag Inc.