Viable Algorithmic Options for Problem Solving by State-space Search: A Computational Complexity Perspective

Todd Wareham[0000−0002−2410−7991]

Abstract Over the last 70 years, an increasing amount of research has been done on computational approaches to problem solving. This work is of interest not only in formulating computational theories of human problem solving but also in software systems that automatically solve problems or help human learners acquire problem solving skills. A key issue underlying these efforts is determining what algorithmic options are and are not available for efficient implementations of the various computations invoked during problem solving. In this chapter, we will consider three computational problems arising in the context of a basic model of problem solving as state-space search. Using classical and parameterized complexity analysis, we will assess (1) the efficient algorithmic options for exactly solving these problems relative to all subsets of 9 basic restrictions on problem-solving instances and (2) three popular polynomial-time options for approximately solving these problems. We also discuss the generality and implications of these results and give several promising directions for future research on the computational complexity of problem solving.

Keywords problem solving · state-space search · computational complexity · parameterized complexity

1 Introduction

Problem solving is a long-standing area of research in Psychology and Cognitive Science [11, 22]. Over the last 70 years, starting with the pioneering work of Allen Newell and Herbert Simon on problem solving as state-space search [13, 31, 32, 33], an increasing amount of research has been done on computational approaches to problem solving. This work is of interest not only in formulating computational theories of human problem solving but also in software systems that automatically derive plans for solving

Todd Wareham
Memorial University of Newfoundland, St. John's, NL, Canada, e-mail: harold@mun.ca

specified problems, e.g., STRIPS (STanford Research Institute Problem Solver) [14], or help human learners acquire problem solving skills [5, 26]. A key issue underlying these efforts is determining what algorithmic options are and are not available for efficient implementations of the various computations invoked during problem solving.

In this chapter, we will consider three computational problems arising in the context of a basic model of problem solving as state-space search:

1. Deriving a complete sequence of problem-solving steps that transforms an initial problem state into a goal state;
2. Determining if a specified partial step-sequence augmented by a specified step can be extended to a complete goal-achieving step-sequence; and
3. Deriving a step to augment a specified partial step-sequence such that this augmented partial step-sequence can be extended to a complete goal-achieving step-sequence.

Using classical [19, 35] and parameterized [10, 15] complexity analysis, we will extend existing results [1, 4, 25] and derive new ones to assess (1) the polynomial-time, fixed-parameter, and XP tractability options for exactly solving these problems relative to all subsets of 9 basic restrictions on problem-solving instances and (2) three popular polynomial-time options for approximately solving these problems.

This chapter is organized as follows. In Section 2, we give a brief overview of problem solving as state-space search as well as the basic model of state-space search that we will analyze in later sections. Section 3 gives introductions to basic concepts underlying computational complexity analysis (namely, computational problems (Section 3.1), algorithms and algorithm efficiency (Section 3.2) and reductions and problem classes (Section 3.3)). Sections 4 and 5 give our results, whose generality and implications are discussed in Section 6. Finally, Section 7 gives our conclusions as well as promising directions for future research on the computational complexity of problem solving.

2 Problem Solving as State-space Search

Optimization problems in Computer Science are typically phrased as search over a set of possible solutions to a given input for those solutions with the best value relative to a specified cost function [7]. In AI, this refines to search over a state space, where individual states encode agent knowledge about a given task environment [36]. This conception of state-space search was originally proposed by Allen Newell and Herbert Simon, as the basis for a computational theory of human problem solving [13, 32, 33, 38]. They distinguish three entities underlying such spaces: a set of **objects** encoding various intermediate states of knowledge in the process of an agent solving a given problem, a subset of objects denoting problem solutions (**goals**), and a set of **operators**, each of which transform one of a pair of objects into the other, that implicitly define the links between states in the space and hence possible operator-paths through the state-space from initial states to goal states [32, page 2013].

In this chapter, we will focus on the following basic form of problem solving as state-space search formalized in the Propositional STRIPS with Negation framework

analyzed in [4, 25]. In this framework, states are defined by the truth assignments to a fixed set of variables, a goal is a conjunction of a set of negated and/or unnegated state variables indicating values of those variables that characterize a goal state, and operators are rules of the form `precondition` $\Rightarrow$ `effect`, where `precondition` and `effect` are both conjunctions of (possibly overlapping) sets of negated and/or unnegated state variables. An operator `precondition` $\Rightarrow$ `effect` is **applicable** to a state s if the conjunction `precondition` is true with respect to the state variable assignment in s. The result of applying such an operator to s is a state s' such that the truth assignment of s is modified to have $v = True$ ($v = False$) if v ($\neg v$) is in conjunction `effect`. A **partial plan** is a sequence of operators whose application starting at a specified initial state results in a state s; if s is a goal state, this sequence is a **complete plan**..

Consider the following example encoding the problem of making a hot cup of tea. The states in this problem correspond to truth assignments to the set of Boolean state variables {`faucet, electricity, kettle(empty), kettle(fullCold), kettle(fullHot), teaBag, cup(empty), cup(fullCold), cup(fullHot), cup(teaBag), cup(tea)`}, the initial state has all variables in {`cup(empty), teaBag, kettle(empty), faucet, electricity`} set to $True$ and all others set to $False$, the goal-conjunction is `cup(tea)`, and the eight operators are

1. `kettle(empty) + faucet` $\Rightarrow$ $\neg$`kettle(empty) + kettle(fullCold)`
   [fill kettle with cold water}
2. `cup(empty) + faucet` $\Rightarrow$ $\neg$`cup(empty) + cup(fullCold)`
   [fill cup with cold water(]
3. `kettle(fullCold) + electricity` $\Rightarrow$
   $\neg$`kettle(fullCold) + kettle(fullHot)`
   [boil kettle]
4. `cup(fullCold) + electricity` $\Rightarrow$ $\neg$`cup(fullCold) + cup(fullHot)`
   [boil cup]
5. `cup(empty) + teaBag` $\Rightarrow$ $\neg$`teaBag +`$\neg$`cup(empty) + cup(teaBag)`
   [add tea bag to cup]
6. `cup(empty) + kettle(fullHot)` $\Rightarrow$ $\neg$`cup(empty) + cup(fullHot)`
   [add hot water to cup]
7. `cup(teaBag) + kettle(fullHot)` $\Rightarrow$ $\neg$`cup(teaBag) + cup(tea)`
   [make tea I]
8. `cup(fullHot) + teaBag` $\Rightarrow$ $\neg$`cup(fullHot) +`$\neg$`teaBag + cup(tea)`
   [make tea II]

Let the length of a plan be the number of operators in it. There are multiple complete plans for the example above, including three of length four ([5 1 3 7], [1 5 3 7], [1 3 5 7]) and one of length three ([2 4 8]); many more would be possible if operators were added that reverse the effects of those given above, e.g., empty a cup or kettle filled with cold or hot water. If one is interested in plan efficiency, plan length is a concern. With respect to the efficiency of solving problems, observe that though there are 2^{11} possible states in the example above, only a small subset of these are reachable by the given operators, e.g., states in which any two of `cup(empty), cup(fullCold), cup(fullHot),` or `cup(tea)` are $True$ are not reachable from the initial state.

Given their concern with a computational theory of human problem solving, Newell and Simon used transcripts of human subjects discussing how they solved problems to develop methods mimicking human problem-solving behavior [32, pages 2013–2014]. The most famous of these is means-end analysis, by which operators are selected to reduce the difference between current and goal states. These methods were all implemented in the General Problem Solver (GPS) software system [13, 31]. While these implementations have been very successful in mimicking human behavior [33], the question remains whether there are other methods for problem solving by state-space search with better runtime or goal-achievement guarantees that might be preferable in computer-only or human-computer collaborative problem solving. Such questions are best addressed using computational complexity analysis, which will be introduced in the next section.

3 Computational Complexity Analysis

Computational complexity analysis is the process of determining what types of algorithms do and do not exist for a given computational problem. As such, it is based on three sets of concepts (computational problems, algorithms and algorithm efficiency, reductions and problem classes), each of which will be described in subsections below. Readers wanting more details are encouraged to consult [19, 35] for classical complexity analysis and [10, 15] for parameterized complexity analysis..

Some readers may be puzzled at the length of this section and question its necessity. It is our hope that delving a bit deeper than usual into the details of complexity proofs will lead to a better comprehension and appreciation of the elegance by which both existing results are extended and new ones are derived in Sections 4 and 5.

3.1 Computational Problems

A **computational problem** consists of descriptions of a problem's input and the outputs associated with each input. If the problem has the answers "Yes" or "No" it is a **decision problem**; otherwise, it is a **search problem**. The following are a decision problem and a search problem associated with problem-solving by state-space search as formalized in Section 2.

PROBLEM SOLVING PLAN DERIVATION (DECISION) (PSPlan_{DEC})
Input: An input $I = (V, O, s_0, g)$ consisting of a set V of state variables, a set O of operators, an initial state s_0, and a goal g, and an integer k.
Question: Is there a complete plan of length $\leq k$ for I?

PROBLEM SOLVING PLAN DERIVATION (SEARCH) (PSPlan)
Input: An input $I = (V, O, s_0, g)$ consisting of a set V of state variables, a set O of operators, an initial state s_0, and a goal g, and an integer k.
Output: A complete plan of length $\leq k$ for I, if such a plan exists, and special symbol $\perp$ otherwise.

Though we are typically interested in solving search problems, it is on occasion useful to focus initially on special decision versions of those problems. This will be explained more fully in Section 3.3.

3.2 Algorithms and Algorithm Efficiency

An **algorithm** for a problem Π is a finite sequence of instructions that, when executed relative a computer, creates for an input I of Π an output associated with I by Π. We here consider three types of algorithms:

1. If an algorithm is always guaranteed to produce the correct output for a given input, it is an **exact algorithm**;
2. If an algorithm is not guaranteed to produce the correct output for a given input but there are provable guarantees on its performance, e.g., an upper bound on how often the produced output is incorrect, it is an **approximation algorithm**; and
3. If an algorithm is not guaranteed to produce the correct output for a given input and there are no provable guarantees on its performance, it is a **heuristic algorithm**.

Though we are typically interested in exact algorithms, approximation and heuristic algorithms are useful if they are more efficient than exact algorithms and perform correctly often enough in practice.

The above begs the question of what it means for an algorithm to be efficient. There are many notions of algorithm efficiency based on different resources used by an algorithm during its operation, e.g., runtime, computer memory. We will here focus on a useful abstraction of real-world clock runtime that instead considers the total number of instructions executed by an algorithm. In particular, we will say that an algorithm A is efficient if the maximum number of instructions executed by A on inputs of size n is upper-bounded by a particular type of function $f(n)$ as n goes to infinity. Three popular types of efficiency functions for algorithms for a problem Π are as follows.

1. **Polynomial-time tractability** [12]: Worst-case algorithm runtime is upper-bounded by a function of the form $c_1 n^{c_2}$ for constants $c_1, c_2 > 0$.
2. **Fixed-parameter tractability** [10]: Worst-case algorithm runtime is upper-bounded by a function of the form $f(P)n^{c_2}$ for a set P of parameters of problem Π, some function $f()$, and a constant $c_2 > 0$.
3. **XP tractability** [10]: Worst-case algorithm runtime is upper-bounded by a function of the form $c_1 n^{f(P)}$ for a constant $c_1 > 0$, a set P of parameters of problem Π, and some function $f()$.

Table 1 List of Considered Parameters (adapted from [25, Table 1]). Note that an occurrence of a variable v is an occurrence of the negated ($-v$) or unnegated (v) form of v.

Parameter	Description	Value in Sec$\dot{2}$ Example
k	Max # operators in complete plan	–
p	Max # variables in a precondition	2
e	Max # variables in an effect	3
o	Max # precondition + effect variables in an operator	5
no	# operators	8
ng	Max # variables in the goal	1
nv	# state variables	11
vo	Max # occurrences of a variable in an operator	2
ov	Max # variables in common in an operator's precondition and effect	2

In types (2) and (3), a **parameter** is some aspect of a problem's input or output; a problem Π parameterized with respect to parameter-set P will be written as $\langle P \rangle$-Π. Some parameters for problems PSPlan$_{DEC}$ and PSPlan defined in Section 3.1 are given in Table 1. Observe that fixed-parameter and XP tractability are generalizations of polynomial-time tractability that isolate non-polynomial aspects of algorithm runtime to particular parameters rather than the whole input size (which is the case for exponential-time algorithms whose runtimes are upper-bounded by functions of the form $c_1 c_2{}^{p(n)}$ for some polynomial function $p()$). Though we are typically interested in polynomial-time tractable algorithms whose worst-case behavior scales well as the input size increases, if such algorithms do not exist, fixed-parameter tractable algorithms may be acceptable if values of the parameters in P are small in inputs encountered in practice and function $f()$ is well-behaved, e.g., $f(\{p_1, p_2\}) = 2^{p_1+p_2}$. Indeed, even XP tractable algorithms may be acceptable if the values of the parameters in P are very small constants and function $f()$ is very well-behaved, e.g., $f()$ is sublinear in P. Note that in both of these situations, fixed-parameter and XP tractability effectively collapse to polynomial-time tractability.

3.3 Reductions and Problem Classes

In order to show that a problem is tractable, one has only to give an algorithm that is tractable relative to the chosen efficiency criterion. However, how does one show that a problem is intractable? This actually turns out to be surprisingly straightforward if the appropriate types of reductions and problem classes are used.

3.3.1 Reductions

A **reduction** from problem Π_1 to problem Π_2 (written $\Pi_1 \leq \Pi_2$) is essentially an algorithm for solving Π_1 that uses one or more calls to algorithms for solving Π_2.

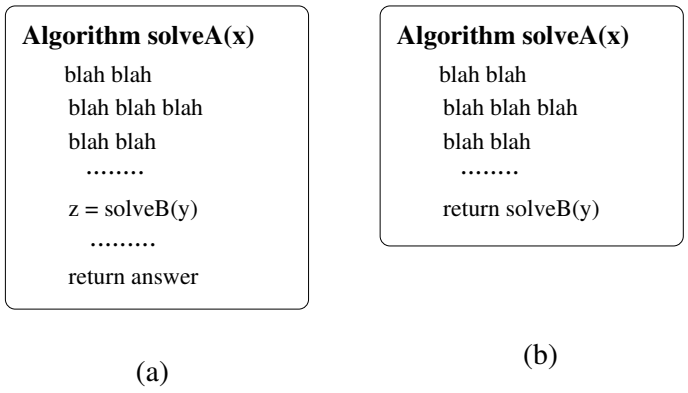

Fig. 1 Types of Reductions Between a Pair of Problems A and B. a) A Turing reduction ($A \leq_T B$). b) A many-one reduction ($A \leq_m B$).

There are many types of reductions, each corresponding to particular ways in which particular numbers of calls are made to the algorithm for solving Π_2. Two popular types (illustrated in Figure 1) are as follows:

1. **Turing reduction** ($\leq_T$): Any number of calls are made to the algorithm for solving Π_2 in any manner.
2. **Many-one reduction** ($\leq_m$): A single call is made to the algorithm for solving Π_2 and the result of this call is the output returned for Π_1.

These reductions become useful by the following logic if we assume that the operation of the reduction algorithms are themselves efficient, regardless of possible runtimes of the used algorithms for solving Π_2:

1. *If* Π_1 *efficiently reduces to* Π_2 *and* Π_2 *is efficiently solvable by an algorithm A then* Π_1 *is efficiently solvable* (combine A with the reduction algorithm to get an efficient algorithm for Π_1).
2. *If* Π_1 *efficiently reduces to* Π_2 *and* Π_1 *is not efficiently solvable then* Π_2 *is not efficiently solvable* (if Π_2 were efficiently solvable by an algorithm A, (1) implies that Π_1 is efficiently solvable, which is a contradiction).

We will be interested in polynomial-time reductions ($\leq^{pt}$) between non-parameterized problems and fixed-parameter reductions ($\leq^{fp}$) between parameterized problems. A polynomial-time reduction $\Pi_1 \leq^{pt} \Pi_2$ naturally allows the creation of a polynomial-time algorithm for Π_1 from a polynomial-time algorithm for Π_2 by the logic above. To do the same for a fixed-parameter reduction $\langle P_1 \rangle$-$\Pi_1 \leq^{fp} \langle P_2 \rangle$-$\Pi_2$, we require that each parameter $p_{2i} \in P_2$, $1 \leq i \leq |P_2|$, be such that $p_{2i} \leq f_i(P_1)$ for some function $f_i()$. Given this, the following useful lemmas can be easily derived.

Lemma 1 *If* $\Pi_1 \leq^{pt} \Pi_2$ *and* Π_2 *is polynomial-time tractable then* Π_1 *is polynomial-time tractable.*

Lemma 2 *If* $\Pi_1 \leq^{pt} \Pi_2$ *and* Π_1 *is polynomial-time intractable then* Π_2 *is polynomial-time intractable.*

Lemma 3 *If* $\langle P_1 \rangle$-$\Pi_1 \leq^{fp} \langle P_2 \rangle$-$\Pi_2$ *and* $\langle P_2 \rangle$-Π_2 *is fixed-parameter tractable then* $\langle P_1 \rangle$-Π_1 *is fixed-parameter tractable.*

Lemma 4 *If* $\langle P_1 \rangle$-$\Pi_1 \leq^{fp} \langle P_2 \rangle$-$\Pi_1$ *and* $\langle P_1 \rangle$-Π_2 *is fixed-parameter intractable then* $\langle P_2 \rangle$-Π_2 *is fixed-parameter intractable.*

Though analogous lemmas can be derived with respect to fixed-parameter reductions for XP tractability, they will not be needed in this chapter.

We can now define what was meant by special decision problems associated with search problems in Section 3.1 – namely, **a decision problem Π_{DECSp} is special with respect to a search problem** Π if Π_{DECSp} and Π have the same inputs and (underlying, in the case of Π_{DECSp}) outputs and Π_{DECSp} can be efficiently Turing reduced to Π. Such a relationship holds between problems PSPlan_{DEC} and PSPlan defined in Section 3.1 (consider the Turing reduction in which, given an input I for PSPlan_{DEC}, compute $ans = \text{PSPlan}(I)$; if $ans = \perp$, return "No" and otherwise return "Yes"). As it is frequently easier to demonstrate intractability of decision problems, the strategy above is commonly used to establish the intractability of search problems. If in addition an efficient algorithm A for a special decision problem Π_{DECSp} either (1) internally computes complete solutions for associated search problem Π in the course of computing the final "Yes" / "No" answer, i.e., **algorithm A is search-compliant** or (2) can be called some number of times in an algorithm A' that efficiently solves Π, A can be modified or employed, respectively, to demonstrate the tractability of Π. These strategies for using decision problems to prove the tractability or intractability of associated search problems will be used extensively in Sections 4 and 5 below.

3.3.2 Problem Classes

As was shown in Section 3.3.1, tractability results pass backwards and intractability results pass forwards along efficient reductions. However, where does one get initial intractability results? One can derive these using the concepts of problem classes, class hardness, and class completeness.

A **problem class** is a the set of all problems that share a characteristic such as solvability by a particular type of algorithm, e.g.,

P : The class of all decision problems solvable by deterministic polynomial-time algorithms.

FPT : The class of all parameterized decision problems solvable by deterministic fixed-parameter tractable algorithms.

Given a type of efficient reducibility and a problem class **C**, a problem Π is **C-hard** if every problem in **C** efficiently reduces to Π; if Π is also in **C**, it is **C-complete** (Figure 2). Observe that (at least with respect to the notion of efficiency used) the **C**-complete problems are the computationally hardest problems in **C** and the **C**-hard problems are at

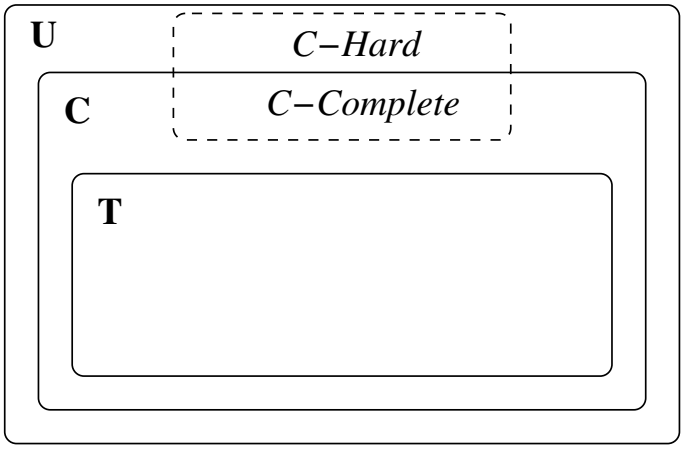

Fig. 2 Hardness and Completeness for Complexity Classes. Classes **T** and **C** of tractable and intractable problems exist within a problem-universe **U**. Note that **T** ⊂ **C** and the **C**-complete problems are the **C**-hard problems that are also in **C**.

least as computationally hard as the hardest problems in **C**. If class **C** properly includes the class **T** of problems that are tractable with respect to the notion of efficiency used, then **C**-hard problems cannot be tractable (Figure 2). Two such classes are:

EXPTIME : The class of all decision problems solvable by deterministic exponential-time algorithms.

XP : The class of all parameterized decision problems solvable by deterministic XP tractable algorithms.

It is known that **P** ⊂ **EXPTIME** and **FPT** ⊂ **XP**, and complete problems for **EXPTIME** and **XP** are hence not polynomial-time and fixed-parameter tractable, respectively.[1]

As many problems of interest are computationally easier than the complete problems in **EXPTIME** and **XP**, we often show hardness relative to easier classes (**NP** and **PSPACE** in the case of non-parameterized problems and the classes {**W[1]**, **W[2]**, ...} in the **W-hierarchy** for parameterized problems; see [19, 35] and [10, 15], respectively, for definitions of these classes). It is known that **P** ⊆ **NP** ⊆ **PSPACE** ⊆ **EXPTIME** and **FPT** ⊆ **W[1]** ⊆ **W[2]** ⊆ **XP**; moreover, the bottom of these containments are widely believed to be proper, i.e., **P** ≠ **NP** [17] and **FPT** ≠ **W[1]** [10]. This allows the following easily-proved lemmas.

Lemma 5 *Given decision problems* Π_1 *and* Π_2 *such that* $\Pi_1 \leq^{pt} \Pi_2$ *and* Π_1 *is NP-hard, if* Π_2 *is polynomial-time tractable then* **P** = **NP**.

Lemma 6 *Given parameterized decision problem* $\langle P_1 \rangle$-Π_1 *and* $\langle P_2 \rangle$-Π_2 *such that* $\langle P_1 \rangle$-$\Pi_1 \leq^{fp} \langle P_2 \rangle$-$\Pi_2$ *and* $\langle P_1 \rangle$-Π_1 *is* **W***-hard for some class* **W** *in the W-hierarchy, if* $\langle P_2 \rangle$-Π_2 *is fixed-parameter tractable then* **FPT** = **W**.

[1] The manner in which initial hardness and completeness results are proved is elegant but need not concern us here. Readers interested in details are encouraged to consult [10, 15, 19, 35].

Lemma 7 *Given a decision problem* Π *that is* **NP***-hard when every parameter in a set* P *is of constant value, if* $\langle P \rangle$-Π *is XP tractable then* **P** = **NP**.

Additional conjecture-driven types of intractability will be derived from **NP**-hardness results and other known and conjectured problem class inclusions in Sections 4 and 5 below.

While the above is well and good for assessing the computational complexity of a decision problem, what does it imply for a search problem, in particular a search problem Π associated with a special decision problem Π_{DECSp}? A lot, as it turns out. For starters, consider the following lemma, which follows from the decision class inclusions and conjectured class inequalities noted above.

Lemma 8 *If* Π_{DECSp} *is* **NP***- or* **PSPACE***-complete then* Π *is not polynomial-time tractable but is solvable by a brute-force exponential-time algorithm that generates and evaluates all possible solutions to* Π.

Such brute-force exponential-time algorithms typically have worst case runtime-bound functions that are exponential in some polynomial $p()$ of the input size, e.g., $2^{p(n)}$. However, as we saw in Section 3.2, fixed-parameter and XP tractability can offer preferable options if algorithms of these types are available. Consider the following additional lemmas derived as above.

Lemma 9 *If* $\langle P \rangle$-Π_{DECSp} *is fixed-parameter tractable courtesy of an algorithm A that is search-compliant then* $\langle P \rangle$-Π *is fixed-parameter tractable.*

Lemma 10 *If* $\langle P \rangle$-Π_{DECSp} *is* **W***-complete for some class in the W-hierarchy then* $\langle P \rangle$-Π *is XP tractable.*

Lemma 11 *If* Π_{DECSp} *is* **NP***- or* **PSPACE***-complete when every parameter in a set* P *has constant value then* $\langle P \rangle$-Π *is not XP tractable but is solvable by a brute-force exponential-time algorithm that generates and evaluates all possible solutions to* Π.

These four lemmas will be used extensively in Sections 4 and 5 below.

4 Full Problem Solving: Deriving Complete Plans

We can now commence our computational complexity analysis of problems PSPlan_{DEC} and PSPlan defined in Section 3.1. We shall first consider the classical and parameterized complexity of these problems as regards exact algorithms, and then look at what these results imply for approximation algorithms for these problems.

4.1 Results for Exact Algorithms

Our analyses with respect to exact algorithms for PSPlan_{DEC} and PSPlan will be simplified greatly by the fact that PSPlan_{DEC} is identical to the decision problem k-PSN analyzed in [25], which is in turn a generalization of decision problem PLANSAT

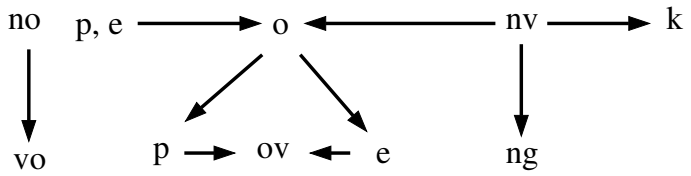

Fig. 3 The parameter dependency graph for the problems PSPlan$_{DEC}$ and PSPlan relative to the parameters in Table 1 (adapted from Figure 1 in [25]).

analyzed in. [4] in which preconditions and goals cannot contain negated state variables. Given this, we know the following.

Theorem 1 *(Adapted from [25, Table 1]) The following results hold:*

(a) *PSPlan$_{DEC}$ is* **PSPACE***-complete (modified from [4, Theorem 3.1]).*
(b) *$\langle k, o, ng, p, e, ov\rangle$-PSPlan$_{DEC}$ is* **W[1]***-hard [25, Theorems 9 and 2(c)].*
(c) *$\langle k, p, ov\rangle$-PSPlan$_{DEC}$ is* **W[2]***-hard [1, Theorems 1 and 2(c)].*
(d) *$\langle k, ng, ov\rangle$-PSPlan$_{DEC}$ is* **W[2]***-hard [25, Theorems 10 and 2(c)].*
(e) *if $\langle o, ng, vo, p, e, ov\rangle$-PSPlan$_{DEC}$ is in* **XP** *then* **P** = **NP** *[25, Theorem 11 and page 956].*
(f) *$\langle nv\rangle$-PSPlan$_{DEC}$ is in* **FPT** *[25, Proposition 6].*
(g) *$\langle k, vo\rangle$-PSPlan$_{DEC}$ is in* **FPT** *[25, Theorem 5].*
(h) *$\langle no\rangle$-PSPlan$_{DEC}$ is in* **FPT** *[25, Theorem 4].*
(i) *$\langle k, e\rangle$-PSPlan$_{DEC}$ is in* **W[1]** *[25, Theorem 8].*
(j) *$\langle k, p, ng\rangle$-PSPlan$_{DEC}$ is in* **W[1]** *[25, Theorem 7].*
(k) *$\langle k\rangle$-PSPlan$_{DEC}$ is in* **W[2]** *[1, Theorem 3].*

The results above imply other results by the following two easily-proven lemmas.

Lemma 12 *If $\langle P_1\rangle$-Π is fixed-parameter tractable and $P_1 \subset P_2$ then $\langle P_2\rangle$-Π is fixed-parameter tractable.*

Lemma 13 *If $\langle P_1\rangle$-Π is* **W***-hard for some class* **W** *in the W-hierarchy and $P_2 \subset P_1$ then $\langle P_2\rangle$-Π is* **W***-hard.*

Many other results follow if we exploit the fact that certain parameters are upper-bounded by functions of other parameters, e.g., $o \leq p + e$ and $ng \leq nv$ for PSPlan$_{DEC}$ and PSPlan. We can summarize all such parameter dependencies for a problem Π in a **parameter dependency graph**, which is a directed graph where nodes are labelled with parameter-sets and there is an arc $P \rightarrow x$ between vertices labeled P and x if $x \leq f(P)$ for some function $f()$. The parameter dependency graph for PSPlan$_{DEC}$ and PSPlan relative to the parameters in Table 1 is shown in Figure 3. Given such a graph and a parameter-set P. let the **closure of** P (written P^*) be the set of all parameters reachable from nodes labelled with elements of P in the graph. This yields the following lemmas.

Lemma 14 *(Adapted from [25, Lemma 2(b)]) If $\langle P_1\rangle$-Π is in class* **C** *and $P_1 \subseteq P_2^*$ then $\langle P_2\rangle$-Π is in class* **C***.*

Lemma 15 *(Adapted from [25, Lemma 2(d)]) If* $\langle P_1 \rangle$*-*Π *is* **C***-hard for some class* **C** *and* $P_2 \subseteq P_1^*$ *then* $\langle P_2 \rangle$*-*Π *is* **C***-hard.*

Theorem 2 *(Adapted from [25, Theorem 3]) The parameterized results in Theorem 1 together with Lemmas 14 and 15 give a complete parameterized complexity classification for* $PSPlan_{DEC}$ *relative to the parameters in Table 1, i.e., for each parameter-set* P *in the* $2^9 - 1$ *parameter-sets of the parameters in Table 1 we can show that* $\langle P \rangle$*-*$PSPlan_{DEC}$ *is either (1) in* **FPT***, (2)* **W[1]***- or* **W[2]***-complete, or (3) not in* **XP***.*

Inspection of the fixed-parameter tractable algorithms in the proofs of Theorems 4 and 5 and Proposition 6 in [25] demonstrates that these algorithms are search-compliant. Given this and the class inclusions and inequalities noted above, we can now draw the following conclusions about viable classical and parameterized algorithmic options for PSPlan.

Theorem 3 *(Adapted from [25, Theorem 3]) Part (a) of Theorem 1 and the parameterized results in Theorem 2 together with Lemmas 8–11 give a complete classification of the classical and parameterized algorithmic options for PSPlan relative to the parameters in Table 1, i.e., PSPlan is not polynomial-time tractable but is solvable by a brute-force exponential-time algorithm and for each parameter-set* P *in the* $2^9 - 1$ *parameter-sets of the parameters in Table 1 we can show that* $\langle P \rangle$*-PSPlan is either (1) fixed-parameter tractable (2) not fixed-parameter tractable but XP tractable, or (3) not XP tractable but solvable by a brute-force exponential-time algorithm.*

4.2 Results for Approximation Algorithms

Let us now consider polynomial-time solvability of $PSPlan_{DEC}$ and PSPlan by approximation algorithms, which are not guaranteed to give the correct outputs for all inputs but nonetheless have provable guaranteed bounds on their performance. Three popular types of polynomial-time approximation algorithms are:

1. algorithms that always run in polynomial time but are frequently correct in that they produce the correct output for a given input in all but a small number of cases (i.e., the number of errors for input size n is bounded by function $err(n)$) [21];
2. algorithms that always run in polynomial time but are frequently correct in that they produce the correct output for a given input with high probability, e.g., $\geq \frac{2}{3}$ [30]; and
3. algorithms that run in polynomial time with high probability, e.g., $> \frac{1}{2}$, but are always correct [20].

Algorithms of types (2) and (3) are characteristic of decision problems in the classes **BPP** and **ZPP**, respectively, where **BPP** is considered the most inclusive class of decision problems that can be efficiently solved using probabilistic methods. These algorithms are of particular interest as they include useful types of evolutionary algorithms. Unfortunately, as it is known that **ZPP** $\subseteq$ **BPP** and it is widely believed that **P** = **BPP** [47, Section 5.2], part (a) in Theorem 1 together with Corollary 2.2 in [21] and the problem class inclusions and inequalities noted above establish the following.

Theorem 4 *(Adapted [45, Results A.4 and A.5] and [44, Result E]) Neither of $PSPlan_{DEC}$ or PSPlan is polynomial-time approximately solvable in any of the senses (1–3) above.*

Note that these inapproximability results also place limits on the senses and situations in which polynomial-time heuristic algorithms such as means-end analysis can produce correct solutions.

5 Assisted Problem Solving: Operator Evaluation and Suggestion

Quite aside from deriving complete plans for a problem, computers can also give assistance during plan creation. Two ways of doing this are (1) to evaluate if a specified partial plan can be extended with a specified operator to a complete plan and (2) to suggest a new operator by which a specified partial plan can be extended to a complete plan. Such hints would be invaluable both in Computer Assisted Learning (CAL) systems [37] that teach humans to develop problem-solving skills [5, 18, 26, 27, 28] and in human-computer collaborative problem-solving systems where, depending on the situation, computers or humans may be called upon to give hints.

Given a partial path pth and an operator op, let $pth \mid op$ be the partial path created by the concatenation of op to the end of pth. The computational problems associated with hints can now be defined as follows.

Problem solving operator evaluation (PSOpEval) *Input*: An input $I = (V, O, s_0, g)$ consisting of a set V of state variables, a set O of operators, an initial state s_0, and a goal g, a partial path pth starting at s_0, an operator $op \in O$, and an integer k. *Question*: Can $pth \mid op$ be extended to a complete plan of length $\leq k$ for I?

Problem solving operator suggestion (PSOpSugg) *Input*: An input $I = (V, O, s_0, g)$ consisting of a set V of state variables, a set O of operators, an initial state s_0, and a goal g, a partial path pth starting at s_0, and an integer k. *Output*: An operator $op \in O$ such that $pth \mid op$ can be extended to a complete plan of length $\leq k$ for I, if such an operator exists, and special symbol $\perp$ otherwise.

We shall below, as in Section 4, first consider the classical and parameterized complexity of these problems with regards to exact algorithms. and then look at what these results imply for approximation algorithms for these problems.

5.1 Results for Exact Algorithms

In order to analyze PSOpSugg, we shall define the following associated special decision problem.

PROBLEM SOLVING OPERATOR SUGGESTION (decision) (PSOpSugg$_{DEC}$) *Input*: An input $I = (V, O, s_0, g)$ consisting of a set V of state variables, a set O of operators, an initial state s_0, and a goal g, a partial path pth starting at s_0, and an integer k. *Question*: Is there an operator $op \in O$ such that $pth \mid op$ can be extended to a complete plan of length $\leq k$ for I?

Observe that the parameters in problems PSOpEval, PSOpSugg$_{DEC}$, and PSOpSugg have the same relationships shown in the parameter dependency graph in Figure 3 as problems PSPlan$_{DEC}$ and PSPlan. Given this, we can now define the following four reductions.

Lemma 16 *PSPlan$_{DEC}$ $\leq_m^{pt}$ PSOpEval.*

Proof. Given an instance $X = \langle I = (V, O, s_0, g), k\rangle$ of PSPlan$_{DEC}$, construct an instance $X' = \langle I' = (V', O', s'_0, g'), pth', op', k'\rangle$ of PSOpEval as follows: let $V' = V \cup \{v_s\}$, $O' = O'' \cup \{o_s\}$ where O'' consists of the operators in O with $\neg v_s$ added to each of their preconditions and $o_s : v_s \Rightarrow \neg v_s$, $s'_0 = s_0 \cup \{v_s\}$, $g' = g \cup \{\neg v_s\}$, pth' is the empty path, $op' = o_s$, and $k' = k + 1$. Note that X' can be constructed from X in time polynomial in the size of X.

We now prove the correctness of the reduction above by establishing that the answer to the given instance X of PSPlan$_{DEC}$ is "Yes"if and only if the answer to the constructed instance X' of PSOpEval is "Yes". This is done in two parts.

$\Rightarrow$ Suppose we have a complete path pth of length $\leq k$ for the given instance X of PSPlan$_{DEC}$ that transforms initial state s_0 into a goal state s_g. Consider the path $pth'' = pth' \mid o_s \mid O'(pth) = o_s \mid O'(pth)$, where $O'(pth)$ is the sequence of operators created by replacing each operator o in pth with its $\neg v_s$-enhanced version from O'. Observe that pth'' transforms s'_0 into s_g and is hence a complete path of length $\leq k' = k + 1$ for instance X' of PSOpEval.

$\Leftarrow$ Suppose we have a complete path pth'' of length $\leq k' = k + 1$ for the given instance X' of PSOpEval that transforms initial state s'_0 into a goal state s_g. As o_s is the operator being evaluated, $pth'' = pth' \mid o_s \mid pth''' = o_s \mid path'''$. Consider the path $pth = O(pth''')$, where $O(pth''')$ is the sequence of operators created by replacing each operator o'' in pth''' with its $\neg v_s$-eliminated version from O. Observe that as o_s transforms s_0 to s_0, pth transforms s_0 into s_g and is hence a complete path of length $\leq k' - 1 = k$ for instance X of PSPlan$_{DEC}$.

This completes the proof of this reduction. □

Lemma 17 *For any set P of the parameters in Table 1, $\langle P\rangle$-PSPlan$_{DEC}$ $\leq_m^{fp}$ $\langle P\rangle$-PSOpEval.*

Proof. Follows from the observations that (1) any polynomial-time reduction also trivially runs in fixed-parameter tractable time, i.e., $f(P)n^{c_2} = c_1 n^{c_2}$ when $f(P) = c_1$, and (2) in the instance X' of PSOpEval constructed from the given instance X of PSPlan$_{DEC}$ in the reduction in the proof of Lemma 16, $k' = k + 1$, $p' = p + 1$, $e' = e$, $o' = o + 1$, $no' = no + 1$, $ng' = ng + 1$, $nv' = nv + 1$, $vo' = vo$", and $ov' = ov$, such that any set of parameters P' for PSOpEval has each member $p' \in P'$ upper-bounded by a trivial function of the parameters P for PSPlan$_{DEC}$. □

Lemma 18 *PSPlan$_{DEC}$ $\leq_m^{pt}$ PSOpSugg$_{DEC}$.*

Proof. Given an instance $X = \langle I = (V, O, s_0, g), k\rangle$ of PSPlan$_{DEC}$, construct an instance $X' = \langle I' = (V', O', s''0, g'), pth', k'\rangle$ of PSOpSugg$_{DEC}$ as in the proof of Lemma 16. Note that X' can be constructed from X in time polynomial in the size of X We prove the correctness of this reduction by establishing that the answer to the given instance X of PSPlan$_{DEC}$ is "Yes"if and only if the answer to the constructed instance X' of PSOpSugg$_{DEC}$ is "Yes". This is done in two parts.

⇒ Suppose we have a complete path pth of length $\leq k$ for the given instance X of PSPlan$_{DEC}$ that transforms initial state s_0 into a goal state s_g. Consider the path $pth'' = pth' \mid o_s \mid O'(pth) = o_s \mid O'(pth)$, where $O'(pth)$ is the sequence of operators created by replacing each operator o in pth with its $\neg v_s$-enhanced version from O'. Observe that pth'' transforms s'_0 into s_g and is hence a complete path of length $\leq k + 1 = k'$ for instance X' of PSOpSugg; therefore, we can set $op = o_s$.

⇐ Suppose we have a complete path pth'' of length $\leq k' = k + 1$ for the given instance X' of PSOpEval that transforms initial state s'_0 into a goal state s_g. As pth' is empty and all operators in O' can only apply if $v_s = False$, operator o_s (that flips the $True$ value of v_s in s'_0 to $False$ to create s_0) must be the first operator in pth'', such that $pth'' = o_s \mid pth'''$. Consider the path $pth = O(pth''')$, where $O(pth'')$ is the sequence of operators created by replacing each operator o'' in pth'' with its $\neg v_s$-eliminated version from O. Observe that pth transforms s_0 into s_g and is hence a complete path for instance X of PSPlan$_{DEC}$.

This completes the proof of this reduction. □

Lemma 19 *For any set P of the parameters in Table 1, $\langle P\rangle$-PSPlan$_{DEC}$ $\leq_m^{fp}$ $\langle P\rangle$-PSOpSugg$_{DEC}$.*

Proof. Same as the proof of Lemma 17, modified to be for PSOpSugg$_{DEC}$ rather than PSOpEval. □

These reductions allow us to prove the following theorems. Note that though the fixed-parameter tractable algorithms in parts (f)–(h) of Theorem 1 can be easily modified to solve PSOpEval and PSOpSugg$_{DEC}$ (and remain search-compliant in the bargain in the case of PSOpSugg$_{DEC}$), it is not known if the **W**-membership proofs for parts (i)–(k) can also be modified; hence, the original induced **W**-completeness results are for now **W**-hardness results.

Theorem 5 *(Adapted from Theorem 1) The following results hold:*

(a) *PSOpEval is* **PSPACE***-complete (Theorem 1(a), Lemma 16, and a modification of [4, Theorem 3.1]).*
(b) $\langle k, o, ng, p, e, ov \rangle$*-PSOpEval is* **W[1]***-hard (Theorem 1(b) and Lemma 17).*
(c) $\langle k, p, ov \rangle$*-PSOpEval is* **W[2]***-hard (Theorem 1(c) and Lemma 17).*
(d) $\langle k, ng, ov \rangle$*-PSOpEval is* **W[2]***-hard (Theorem 1(d) and Lemma 17).*
(e) *if* $\langle o, ng, vo, p, e, ov \rangle$*-PSOpEval is in* **XP** *then* **P** = **NP** *(Theorem 1(e) and Lemma 16).*
(f) $\langle nv \rangle$*-PSOpEval is in* **FPT** *(modified from Theorem 1(f)).*
(g) $\langle k, vo \rangle$*-PSOpEval is in* **FPT** *(modified from Theorem 1(g)).*
(h) $\langle no \rangle$*-PSOpEval is in* **FPT** *(modified from Theorem 1(h)).*

Theorem 6 *(Adapted from Theorem 2) The parameterized results in Theorem 5 together with Lemmas 14 and 15 give a mostly complete parameterized complexity classification for PSOpEval relative to the parameters in Table 1, i.e., for each parameter-set P in the* $2^9 - 1$ *parameter-sets of the parameters in Table 1 we can show that* $\langle P \rangle$*-PSOpEval is either (1) in* **FPT***, (2)* **W[1]***- or* **W[2]***-hard and possibly in* **XP***, or (3) not in* **XP***.*

Theorem 7 *(Adapted from Theorem 1) The following results hold:*

(a) $PSOpSugg_{DEC}$ *is* **PSPACE***-complete (Theorem 1(a), Lemma 18, and a modification of [4, Theorem 3.1]).*
(b) $\langle k, o, ng, p, e, ov \rangle$*-*$PSOpSugg_{DEC}$ *is* **W[1]***-hard (Theorem 1(b) and Lemma 19).*
(c) $\langle k, p, ov \rangle$*-*$PSOpSugg_{DEC}$ *is* **W[2]***-hard (Theorem 1(c) and Lemma 19).*
(d) $\langle k, ng, ov \rangle$*-*$PSOpSugg_{DEC}$ *is* **W[2]***-hard (Theorem 1(d) and Lemma 19).*
(e) *if* $\langle o, ng, vo, p, e, ov \rangle$*-*$PSOpSugg_{DEC}$ *is in* **XP** *then* **P** = **NP** *(Theorem 1(e) and Lemma 18).*
(f) $\langle nv \rangle$*-*$PSOpSugg_{DEC}$ *is in* **FPT** *(modified from Theorem 1(f)).*
(g) $\langle k, vo \rangle$*-*$PSOpSugg_{DEC}$ *is in* **FPT** *(modified from Theorem 1(g)).*
(h) $\langle no \rangle$*-*$PSOpSugg_{DEC}$ *is in* **FPT** *(modified from Theorem 1(h)).*

Theorem 8 *(Adapted from Theorem 2) The parameterized results in Theorem 7 together with Lemmas 14 and 15 give a mostly complete parameterized complexity classification for* $PSOpSugg_{DEC}$ *relative to the parameters in Table 1, i.e., for each parameter-set P in the* $2^9 - 1$ *parameter-sets of the parameters in Table 1 we can show that* $\langle P \rangle$*-*$PSOpSugg_{DEC}$ *is either (1) in* **FPT***, (2)* **W[1]***- or* **W[2]***-hard and possibly in* **XP***, or (3) not in* **XP***.*

Theorem 9 *(Adapted from Theorem 3) Part (a) of Theorem 7 and the parameterized results in Theorem 8 together with Lemmas 8–11 give a mostly complete classification of the classical and parameterized algorithmic options for PSOpSugg relative to the parameters in Table 1, i.e., PSOpSugg is not polynomial-time tractable but is solvable by a brute-force exponential-time algorithm and for each parameter-set P in the* $2^9 - 1$ *parameter-sets of the parameters in Table 1 we can show that* $\langle P \rangle$*-PSOpSugg is either (1) fixed-parameter tractable (2) not fixed-parameter tractable but possibly XP tractable, or (3) not XP tractable but solvable by a brute-force exponential-time algorithm.*

5.2 Results for Approximation Algorithms

It is tempting to believe that the simplicity of our operator evaluation and suggestion problems compared to their complete plan derivation counterparts would allow a wider range of polynomial-time approximate and heuristic solvability options. Unfortunately, the logic given in Section 4.2 and parts (a) of Theorems 5 and 7 establishes the following.

Theorem 10 *(Adapted from Theorem 4) Neither of PSOpEval, PSOpSugg$_{DEC}$, or PSOpSugg is polynomial-time approximately solvable in any of the senses (1–3) described in Section 4.2.*

6 Discussion

6.1 Implications for Stated Problems

Our results show that, provided several widely-believed conjectures are true, none of PSPlan, PSOpEval, or PSOpSugg are polynomial-time solvable by either exact algorithms (Theorems 3, 5, and 9, respectively) or three popular types of approximation algorithms (Theorems 4 and 10). There are, however, more options as regards fixed-parameter and XP tractability with respect to the parameters in Table 1. There appear to be three core sets of parameters for fixed-parameter tractability ($\{nv\}$, $\{k, vo\}$, and $\{no\}$) which, courtesy of Lemmas 12 and 14, imply a number of other fixed parameter tractability options. Moreover, as the given parameterized complexity classifications are complete, there can be no other such core sets with respect to the parameters in Table 1.

This paucity of fixed-parameter tractability cores may initially seem problematic given the frankly ludicrous runtimes of the algorithms in the proofs of fixed-parameter tractability. However, this despair is unwarranted; once fixed parameter tractability is initially proved, algorithms with far better and frequently very practical runtimes can be derived using specialized techniques [8, 16, 34]. The results of such "FPT races" [24] can be spectacular — for example, once it was realized in 1992 that the polynomial-time intractable problem Vertex Cover is fixed-parameter tractable relative to the parameter k encoding the size of the wanted vertex cover, the $2^k k^{2k+2} + k|V|$ runtime of the initial algorithm was bettered over the next 20 years to $1.2738^k + k|V|$. It seems reasonable to conjecture that if there is sufficient interest from the problem solving community, improved algorithms will be derived for PSPlan, PSOpEval, and PSOpSugg.

6.2 Generality of Derived Results

The results in this chapter were derived relative to a basic model of problem states and operators that is perhaps too simple for instances of problem solving that arise

in practice. This simplicity is, however, key to the generality of our results. Instances phrased in a framework such as ours can be readily transformed into instances in more complex frameworks, e.g., states composed of conjunctions of Boolean variables may be recoded as conjunctions of integer-valued variables or (as in [32]) logic or mathematical expressions. Provided the transformations are computationally efficient, algorithms for solving these complex instances can be used to efficiently solve our simple instances. Such transformations can be viewed as reductions. In this light, the resulting efficient solvability of our simple instances is an application of our first rule of the logic of reducibility given in Section 3.3.1. Moreover, the second rule of this logic also holds – namely, *that both classical and parameterized class intractability and inapproximability results proven in our framework pass along the transformation to problem solving in more complex frameworks*. As many of the parameters in Table 1 probably have analogues in more complex frameworks and we have parameterized results for all subsets of parameters in that table, this should yield a fine bounty of intractability and inapproximability results for these frameworks.

This generality, unfortunately, is only guaranteed to hold for our intractability and inapproximability results — the algorithms underlying our tractability results frequently rely on the details of our problem solving framework, and hence need not (and probably do not) readily translate to more complex frameworks. Indeed, it may be the case that parameter combinations which yielded parameterized tractability in our framework may be intractable with respect to more complex frameworks. That being said, our algorithms may yet suggest ways of structuring tractable algorithms within more complex frameworks (particularly if new parameters are added to our analysis along the lines sketched in Section 7).

6.3 Implications for Problem Solving by Computers and Humans

Our given results and their generalizations along the lines described in Section 6.2 were derived within a theoretical framework designed to analyze real-world computation and hence readily apply to problem solving by computers, both full and assisted. Knowledge that there are relatively few islands of fixed-parameter tractability relative to a large set of parameters should be useful in helping to structure both general problem solving systems and more powerful educational software that generates evaluations and suggestions for human learners relative to arbitrary real-world examples rather than (as is the case now [5, 26]; see also [27, Section 4.2.1]) selecting pre-computed evaluations and suggestions for fixed examples. It is worth noting that results for problems PSOpEval and PSOpSugg, though initially defined to model processes in Computer Assisted Learning, also have implications for operator selection when deriving complete plans. In particular, our results for these problems not only place limits on how well heuristic operator selection methods like means-end analysis can work in general but also suggest (via combinations of parameters underlying fixed-parameter tractability) restrictions on problem environments that may be being exploited by these heuristic methods when they do work well.

If one in addition accepts that computational complexity results apply to the computations underlying human cognition (as exemplified in van Rooij's Tractable Cognition Thesis [39, 40]), then our results also have implications for human problem solving. In particular, results for problem PSOpSugg may (along the lines sketched in the previous paragraph) help explain both the range of human problem solving strategies discovered by researchers such as Newell and Simon [33] as well as the situations in which these strategies do and do not work well. In combination with the computer problem solving implications mentioned above, this should also help establish in human-computer collaborative problem solving at which points and in what situations the human and computer collaborators can best be used to ensure the most efficient and successful solving of problems.

7 Conclusions and Future Work

In this paper, we have used computational complexity analysis to determine viable classical and parameterized algorithmic options for three computational problems associated with problem solving by state-space search (deriving complete plans (PSPlan), determining if a specified partial path augmented by a specified operator can be extended to a complete path (PSOpEval), and determining if there is an operator that can augment a specified partial path can be extended to a complete path (PSOpSugg)). As regards exact solvability, assuming certain widely-believed conjectures are true, all three problems are polynomial-time intractable in general and remain fixed-parameter intractable for a large number of combinations of 9 basic parameters. That being said, all three problems are fixed-parameter tractable relative to the same three core parameter-sets. As regards approximate solvability, none of our problems allow any of three popular deterministic (few-error polynomial-time) or probabilistic (probably-correct always polynomial-time / always-correct probably polynomial-time) approximation algorithms. All of these results are derived relative to a basic state-space based on conjunctions of Boolean variables but nonetheless generalize readily to problem solving in more complex state-spaces.

Two counter-intuitive surprises emerged during our analysis — first, that the apparently much simpler problems PSOpEval and PSOpSugg are as computationally difficult as the apparently much more complex problem PSPlan, and second, that all three problems share the same pattern of classical and parameterized tractability, intractability, and inapproximability results. Such surprises, oddly enough, are not unexpected. It often happens that human "armchair complexity" intuitions about problem intractability as well as what problem aspects are and are not responsible for this intractability are utterly wrong ([41, 42]; see also [40, Chapter 11]). To reliably get at the truth about problem intractability, there is no substitute for formal computational complexity analyses such as that given in this chapter.

Though extensive, the work presented in this chapter is but a beginning. There are a variety of directions that future investigations on the computational complexity of problem solving can take. Several of the most promising are as follows.

- **Exploring other parameters**: There are many other parameters whose restriction (either by themselves or with parameters considered here) may grant XP, fixed-parameter, or even polynomial-time tractability. A valuable source of such parameters comes from restrictions that break reductions and prevent class hardness proofs. Two immediate candidates are the number of negations allowed in goals and/or operator preconditions and effects [4] and the number of variable-value changes in a complete plan [25].
- **Exploring other types of problem tractability**: If instances of problem solving occurring in practice are characterized by parameters with very small constant values, it may be worth doing a detailed analysis of constant-value parameter computational complexity. Such an analysis for PLANSAT in [4] uncovered three exact polynomial-time tractable cases (arbitrary-length preconditions and length-one effects; length-one preconditions, arbitrary effects, and fixed-length goals; length-zero preconditions and arbitrary effects); given that PLANSAT is a special case of PS-Plan, a similar analysis should be done for our problems. It may also be of interest to apply advanced proof techniques to our problems to assess viable options for parameterized probabilistic tractability [9, 29], sublinear-exponent XP tractability [6], and subexponential-time tractability [23].
- **Exploring other types of problem solving**: More complex frameworks for problem solving by state-space search than that considered here are an obvious target; an excellent starting point for such investigations would be complexity analyses in the AI planning literature [1, 2, 3, 4, 25] (note the inclusion of [4, 25] in this list, as they examine other types of planning than those discussed here). Researchers wishing to look further afield may find the complexity analyses of insight-enabled and analogical problem solving given in [43, 46] of interest.

Acknowledgements The author would like to thank the two anonymous reviewers, whose comments helped to improve both the technical content and presentation of this chapter. Work underlying that reported here was carried out with the support of NSERC Discovery Grants 228104-2005, 228104-2010, and 228104-2015.

References

1. Bäckström, C., Chen, Y., Jonsson, P., Ordyniak, S., Szeider, S.: The complexity of planning revisited —- A parameterized analysis. In: Proceedings of the 26th AAAI Conference on Artificial Intelligence, 1, pp. 1735–1741 (2012)
2. Bäckström, C., Jonsson, P., Ordyniak, S., Szeider, S.: A complete parameterized complexity analysis of bounded planning. Journal of Computer and System Sciences **81**(7), 1311–1332 (2015)
3. Bäckström, C., Nebel, B.: Complexity results for SAS+ planning. Computational Intelligence **11**(4), 625–655 (1995)
4. Bylander, T.: The computational complexity of propositional STRIPS planning. Artificial Intelligence **69**(1-2), 165–204 (1994)
5. Chang, K.E., Sung, Y.T., Lin, S.F.: Computer-assisted learning for mathematical problem solving. Computers & Education **46**(2), 140–151 (2006)
6. Chen, J., Huang, X., Kanj, I.A., Xia, G.: Strong computational lower bounds via parameterized complexity. Journal of Computer and System Sciences **72**, 1346–1367 (2006)

7. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, third edn. The MIT Press, Boston, MA (2009)
8. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer (2015)
9. Donselaar, N.: Probabilistic parameterized polynomial time. In: International Conference on Current Trends in Theory and Practice of Informatics, pp. 179–191. Springer (2019)
10. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Berlin (1999)
11. Dunbar, K.: Problem solving. In: W. Bechtel, G. Graham (eds.) A Companion to Cognitive Science, pp. 289–298. Wiley Online Library (2017)
12. Edmonds, J.: Paths, trees, and flowers. Canadian Journal of Mathematics **17**, 449–467 (1965)
13. Ernst, G.W., Newell, A.: GPS: A Case Study in Generality and Problem Solving. Academic Press, New York, NY (1969)
14. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial intelligence **2**(3-4), 189–208 (1971)
15. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Berlin (2006)
16. Fomin, F.V., Lokshantov, D., Saurabh, S., Zehavi, M.: Kernalization: Theory of Parameterized Preprocessing. Cambridge University Press, Cambridge, UK (2019)
17. Fortnow, L.: The Status of the P Versus NP Problem. Communications of the ACM **52**(9), 78–86 (2009)
18. Gable, A., Page, C.V.: The use of artificial intelligence techniques in computer-assisted instruction: An overview. International Journal of Man-Machine Studies **12**(3), 259–282 (1980)
19. Garey, M.R., Johnson, D.S.: Computers and Intractability. W.H. Freeman (1979)
20. Gill, J.: Computational complexity of probabilistic Turing machines. SIAM Journal on Computing **6**(4), 675–695 (1977)
21. Hemaspaandra, L.A., Williams, R.: Complexity Theory Column 76: An atypical survey of typical-case heuristic algorithms. ACM SIGACT News **43**(4), 70–89 (2012)
22. Holyoak, K.J.: Problem solving. In: E.E. Smith, D.N. Osherson (eds.) Thinking: An Invitation to Cognitive Science, vol. 3, pp. 267–296. MIT Press, Cambridge, MA (1995)
23. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? Journal of Computer and System Science **63**(4), 512–530 (2001)
24. Komusiewicz, C., Niedermeier, R.: New races in parameterized algorithmics. In: B. Rovan, V. Sassone, P. Widmayer (eds.) International Symposium on Mathematical Foundations of Computer Science, *Lecture Notes in Computer Science*, vol. 7464, pp. 19–30. Springer (2012)
25. Kronegger, M., Pfandler, A., Pichler, R.: Parameterized complexity of optimal planning: A detailed map. In: International Joint Conference on Arttificial Intelligence (IJCAI), pp. 954–961 (2013)
26. Lee, C.I.: An appropriate prompts system based on the Polya method for mathematical problem-solving. Eurasia Journal of Mathematics, Science and Technology Education **13**(3), 893–910 (2017)
27. Lu, D., Xie, Y.N.: The application of educational technology to develop problem-solving skills: A systematic review. Thinking Skills and Creativity **51**, 101454 (2024)
28. Mensah, F.S., Ampadu, E.: Benefits, challenges and opportunities of using computer-assisted instruction in mathematics education. In: IoT, AI, and ICT for Educational Applications: Technologies to Enable Education for All, pp. 31–49. Springer (2024)
29. Montoya, J.A., Müller, M.: Parameterized random complexity. Theory of Computing Systems **52**(2), 221–270 (2013)
30. Motwani, R., Raghavan, P.: Randomized Algorithms. Chapman & Hall/CRC (2010)
31. Newell, A., Shaw, J.C., Simon, H.A.: Report on a general problem-solving program. Tech. Rep. P-1584, The RAND Corporation (1959)
32. Newell, A., Simon, H.A.: Computer simulation of human thinking: A theory of problem solving expressed as a computer program permits simulation of thinking processes. Science **134**(3495), 2011–2017 (1961)
33. Newell, A., Simon, H.A.: Human Problem Solving. Prentice-Hall, Englewood Cliffs, NJ (1972)
34. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
35. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
36. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, fourth edn. Prentice Hall, Englewood Cliffs, NJ (2022)

37. Sharma, R.: Computer assisted learning – A study. Computer **4**(2), 102–105 (2017)
38. Simon, H.A., Newell, A.: Human problem solving: The state of the theory in 1970. In: Structural Learning (Volume 2), pp. 131–151. Routledge (2017)
39. van Rooij, I.: The Tractable Cognition Thesis. Cognitive Science **32**(6), 939–984 (2008)
40. van Rooij, I., Blokpoel, M., Kwisthout, J., Wareham, T.: Cognition and Intractability: A Guide to Classical and Parameterized Complexity Analysis. Cambridge University Press, Cambridge, UK (2019)
41. van Rooij, I., Evans, P.A., Müller, M., Gedge, J., Wareham, T.: Identifying Sources of Intractability in Cognitive Models: An Illustration using Analogical Structure Mapping. In: Proceedings of the 30th Annual Conference of the Cognitive Science Society, pp. 915–920. Cognitive Science Society, Austin, TX (2008)
42. Wareham, T.: Systematic Parameterized Complexity Analysis in 'Computational Phonology. Ph.D. thesis, University of Victoria, Canada (1999)
43. Wareham, T.: The roles of internal representation and processing in problem solving involving insight: A computational complexity perspective. The Journal of Problem Solving **10**(1), 3 (2017)
44. Wareham, T.: Creating Teams of Simple Agents for Specified Tasks: A Computational Complexity Perspective. CoRR abs/2205.02061 (20 pages) (2022)
45. Wareham, T., de Haan, R., Vardy, A., van Rooij, I.: Swarm control for distributed construction: A computational complexity perspective. ACM Transactions on Human-Robot Interaction **12**(1), 6:1–6:45 (2022)
46. Wareham, T., Evans, P., van Rooij, I.: What does (and doesn't) make analogical problem solving easy? A complexity-theoretic perspective. The Journal of Problem Solving **3**(2), 30–71 (2011)
47. Wigderson, A.: P, NP and mathematics — A computational complexity perspective. In: Proceedings of ICM 2006: Volume I, pp. 665–712. EMS Publishing House, Zurich (2007)