Invited paper to appear in M. Ghallab and A. Milani (eds.), *New Directions in AI Planning: EWSP'95—3rd European Workshop on Planning*, Assisi, Italy, Sep. 27–29, 1995. IOS Press.

Five Years of Tractable Planning

Christer BÄCKSTRÖM*
Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
email: cba@ida.liu.se

Abstract. We summarize the results from the first five years of a project aiming at identifying tractable classes of planning problems and investigate sources of computational difficulties in planning. The paper is a non-formal survey, including also historical remarks on the background of the project as well as discussion and motivation of the underlying assumptions, the methodology and the intended applications.

1 Introduction

Planning has been an important subdiscipline of artificial intelligence for some two to three decades. Although predated by some earlier work, planning research really 'took off' with the two papers on QA3 [30] and STRIPS [27]. Already here, in the infancy of the area, we can see a methodological distinction which has prevailed ever since. In QA3, actions and states were axiomatised in logic (the situation calculus). A theorem prover was used to prove the goal and the plan was extracted from this proof. We refer to this approach as *logic-based planning*. STRIPS, on the other hand, generated a plan by searching through the state of possible operator sequences, until finding a sequence leading to the goal. We refer to this approach (and similar ones, like plan-space search) as *search-based planning*. The area of planning research has been very active ever since, but the two approaches has lived mostly separate lives. The logic-based approach has always been heavily theoretical; it has seriously considered the issue of correctness, but almost entirely ignored efficiency. The search-based approach, on the other hand, was mostly informal, implementation-oriented and experimental—caring little about correctness, but paying some notice to efficiency (although in an informal way). Many novel methods and techniques were developed. Yet it was not until Chapman's paper on TWEAK [20], less than ten years ago, that theoretical research started to slowly make its entrance also into this approach, starting to analyse the methods and systems developed. Some results on correctness and complexity has appeared since then, but only in the past few years have we started to see a trend that these issues are seriously paid attention to by (at least parts of) the planning community.

In many other branches of AI, *e.g.* terminological logics, abduction and non-monotonic reasoning, we have seen very active research into identifying tractable subcases and sources of intractability. For several years now, we have run a similar research project on identifying computationally tractable classes of planning problems, devising polynomial time algorithms for these. We have also tried to identify various sources

*This paper reports research by Inger Klein, Bernhard Nebel, Peter Jonsson and the author.

of computational difficulties and tried to learn more about the borderline between tractable and intractable planning problems. Having published in 1990 what we believe to be the first tractable class of planning problems reported in the literature [9], it seems appropriate now, five years later, to report on the progress so far. Since all our technical results already appear in the literature, this paper is non-technical in nature, summarising and giving references to the results rather than restating them. The paper also discusses many issues underlying the project, *e.g.* motivation and intended applications; the role of correctness and tractability; how to interpret complexity results for planning and more. These are issues often omitted since they can usually be reported only very briefly, if at all, within the page limits of ordinary conference papers reporting a technical result.

2 Motivation and Intended Applications

AI planning has often been intimately connected with robotics. For instance, Charniak and McDermott [21, Chapter 9] introduce their chapter on planning as follows: "This chapter is about robots—intelligent computers that act in the world." Similarly, even though many authors do not explicitly mention robots, they frequently use robotics examples. We believe this tacit assumption of planning = robotics is very unfortunate; while planning is important for robotics, it is also important for many non-robotics applications, a fact which might not always be obvious from reading the literature on AI planning. The typical robotics examples in AI are of a type which leads us to assume that the goal is to make robots that can replicate every-day human behaviour and capabilities. There are applications where this is to some extent desired; we may, for instance, prefer to send robots instead of human firefighters into a burning building. However, what we need in most cases is not such systems, but rather systems which outperform humans in certain narrow fields of expertise. For instance, Hayes and Ford [32] point out that "As many have observed, there is no shortage of humans, and we already have well-proven ways of making more of them." They continue arguing that systems with brittle knowledge in a narrow field can be quite useful. Simon [50] similarly points out that even human expert knowledge is extremely brittle, so why should we require more of machines.

Focussing again on planning, we can find successful attempts at applying knowledge-based planners like NonLin, O-Plan and Sipe to non-robotics applications, in narrow fields of expertise [24]. Our intended applications are likewise narrow fields of expertise, but mainly applications where no human expert has good knowledge of how to plan. Such applications frequently arise in various engineering applications, typically having a large number of simple, specialized operators and being so extensive as to cause computational problems. Many applications of this type arise in the field of *sequential control*, a subarea of *automatic control* dealing with discrete aspects of control applications. Although most industrial processes are continuous systems they almost always also contain a superior discrete level. On this superior level the actions can, hence, often be viewed as discrete actions even if implemented as continuous processes. Many problems on this level can be viewed as action planning. Typical actions can be to open or close a valve or to start or stop a motor. An interesting application for automated planning is the situation where a process breaks down or is stopped in an emergency situation. At such an event the system may end up in any of a very large number of states and it might require a quite complex plan[1] to bring the system back into the normal operation mode. It is not realistic to have pre-compiled plans for how to start up the process again from each such state. Furthermore, the considerable costs involved when a large industrial process, such as a paper mill, is inactive necessitate a prompt response from the planner. There is typically no human expert who knows how to solve the problem or write down rules for this. Typically, the (human) operators have some partial understanding and experience which they combine with experimentation and improvisation—eventually getting the system running again (in most cases). Hence, planning from first principles seem a better approach than knowledge-based planning for this type of applications, even if difficult. In principle, nothing prevents us from also incorporating expert knowledge to the extent it is available.

Another example of applications in sequential control is large traffic systems, *e.g.* a railway or underground network. In this case a planner might be used more frequently, not only when the system breaks down, but rather to prevent break-downs and maintain a smooth traffic flow. It can be used, for example, as a support system for network operators to decide how to direct the train traffic in a station area.

Our project grew out of a series of joint meetings and discussions between computer scientists and automatic-control scientists at Linköping University. While the discipline of automatic control has a long tradition and huge accumulated knowledge of continuous systems, their research into discrete systems is still in its infancy, while it is the other way around for computer science. It all started with some experiments trying to apply AI planning to a toy problem in automatic control, the *tunnel problem* (see Section 6). Control researchers are used to think in terms of state variables and consider STRIPS style languages as awkward and inappropriate. Hence, a more suitable choice was the *actions structures* formalism [49], a state-variable-based representation for plans with parallel actions. However, since we concentrated mainly on the computational aspects of non-parallel plans to start with, we introduced the *simplified actions structures formalism (SAS)*.

One of our target test applications has been the *LEGO Car Factory*, a miniature assembly line for LEGO cars, used for undergraduate laboratory assignments in sequential control at the department of EE at Linköping University. This application is interesting because it contains many of the problems and properties found in real industrial processes, while being small enough to be used as a test application and to analyse theoretically. We have recently identified a class of tractable planning problems which includes this application and devised a correct, polynomial-time planning algorithm for this [42].

3 Formal Basis and Correctness

Two seemingly reasonable requirements on a planner are that it is *sound* and *complete* wrt. to some notion of correct plans. A planner that is sound never returns an incorrect plan, but it may answer that there is no solution even when there is one. A planner that is complete always returns a correct plan when a solution exists, but it may return an (incorrect) plan when there is no solution. To prove that a planner does or does not have these properties requires that it be based on some formal theory—otherwise no such proof is possible. Many implemented, working planning systems are not based on

[1]Such a plan may also contain parallel structures, of course. The word *sequential* in sequential control should not be interpreted literally.

any formal theory, however, so not much is known about their correctness. Wilkins [53], for instance, says of his SIPE system that a correctness analysis probably cannot be carried out. It is often claimed that provable correctness is neither necessary nor possible for planning systems that are capable of solving real problems efficiently. Provable correctness *is* important, however, especially if we want to apply planning to automatic control problems. Automatic Control has a long tradition of using mathematically well-founded theories with provable properties. Researchers in automatic control see the lack of such theories as one of the major problems with AI planning. For example, Passino and Antsaklis [48, page 1] say:

> The essential ideas in the theory of AI planning have been developed and reported in the literature. There is, however, a need to create a mathematical theory of AI planning systems that operate in dynamic, uncertain, and time-critical environments (real-time environments).
>
> In a control system the controller produces inputs to a dynamical system to change its undesirable behavior to a desirable one. In contrast to AI planning, there exists a relatively well-developed mathematical systems and control theory for the study of properties of systems represented with, for instance, linear ordinary differential equations.

Furthermore, Åström *et. al.* [2, 3] have carried out a study of how computer software is used in large-scale control applications, such as the process industry and underground traffic networks. One finding was that the industry wants more mathematical methods to "tackle the issue of 'software errors' and increase productivity", *e.g.* "new models of dynamical systems for systems that are combinatorial in nature (supervision or information processing systems...)". Another finding was that expert systems are frequently used in large-scale control applications, but *never* in safety-critical parts of the systems.

It may, finally, be worth pointing out that even if a particular application does not demand that the planner always returns correct solutions, having a formal basis for it and knowing what the correct solutions are can still help us characterize when the algorithm is reliable and when it is not.

4 Tractability and the Restricted-Problems Approach

Computational tractability is a fundamental issue in all problem solving. If a problem is not tractable, we cannot hope to solve an arbitrary instance of it within reasonable time. This is obviously important if our goal is to use computers for solving planning problems. Unfortunately, planning is known to be very hard. Even simple propositional planning is PSPACE-complete [18], which means that it is most likely intractable. Planning with variables ranging over an infinite domain is even undecidable [20, 26]. Until recently the planning community has not shown much interest in the formal complexity analysis of planning. Common arguments against such analysis are:

- The hard problem is to formally characterize the right answers. Once we have done that, it is trivial to write algorithms.

- Of course there may be worst cases that are intractable, but these do not occur in practice.

- Planning is undecidable even for problems that are trivial for humans, so classical AI planning will never succeed.

- Complexity results for planning are useless since everything is intractable, yet real planners work in practice.

The first argument is typically advocated by researchers in the logic-based approach to planning. Bylander [17] has critically reviewed three arguments along these lines—arguments applying to AI in general, not only planning. His counter-argument is [17, page 174]:

> In all three cases, the error can be traced to a methodological separation between getting the answer right (epistemology, computational theory level, knowledge level) and getting the answer tractably (heuristics, representation and algorithm level, symbol level). They presuppose, quite correctly, that it is difficult to discover a description that characterizes the right answer. However, they also presuppose, quite incorrectly, that given such a problem, it is relatively easy to implement a tractable program. Unfortunately, computational complexity theory implies that unless a problem is inherently tractable, it is impossible to implement a tractable program.

Similarly, Ullman [51, p. 43] comments that in the early days of of computer science the success on finding efficient algorithms for certain problems misled the community to believe there were more efficient algorithms for all problems. He continues saying:

> The theory community was not alone in its naivite. AI was similarly taken in by the early successes ... This view has led to periodic overly optimistic predictions of success that continue to trouble AI.

The answer to the second argument is that if the hard cases do not occur in the real problem, then we have not modelled the problem correctly. That is, the real problem is a tractable, proper subproblem of the formal problem we have characterized, or to quote Levesque [44, page 386]:

> If the only problematic cases are the ones that do not seem to occur in practice ... we can simply decide to eliminate them from consideration ... But this does not eliminate our concern with extreme cases; it merely changes what cases we consider to be extreme.

This leads us to the *restricted-problems* approach, to be discussed shortly.

An archetypical example of the third argument is given by Chapman, who writes about his TWEAK planner [20, page 350]:

> The restrictions on action representation make TWEAK almost useless as a real-world planner. It is barely possible to formalize the cubical blocks world in this representation; HACKER's blocks world with different-sized blocks, can not be represented.

It seems as if Chapman wants us to conclude that classical AI planning cannot be used to solve real-world problems. However, he also writes [20, page 344]:

> Any Turing machine with its input can be encoded as a planning problem in the TWEAK representation. Therefore, planning is undecidable, and no upper bound can be put on the amount of time required to solve a problem.

This leads to an interesting consequence: if we can encode a Turing machine in the TWEAK representation, then TWEAK can solve any problem that a computer can solve (if accepting Church's hypothesis). As a consequence, the real-world planning problems considered cannot be solved by any computer, whatever program we use. Furthermore, if taking seriously the claims that only tractable theories are candidates for explaining human intelligence [17, 44] the above reasoning would imply that not even humans can solve the real-world problems considered. This is most likely not the conclusion Chapman intended. A more reasonable conclusion from Chapman's observations is that TWEAK is too restricted in some aspects, but at the same time it has too much expressive power in other aspects, the latter being the cause of the undecidability. A similar case arises when encoding the blocks world in propositional STRIPS, quite possible despite Chapman's claims. Planning in propositional STRIPS is PSPACE-complete [18], but blocks-world planning *per se* is solvable in low-order polynomial time [4, 5, 31]. This is no contradiction; the PSPACE-completeness figure corresponds to the hardest problems encodable in propositional STRIPS, not to every application encodable in it. The answer to the fourth argument above is, thus, that complexity figures are useful, but we must know how to interpret them and we must analyse the right thing. We have to distinguish the inherent complexity of an application problem from the complexity of planning in a formalism. While the latter can give an upper bound for the former, a tractable application problem does not guarantee that the problem is solved in polynomial time by a general problem solver for an intractable formalism. Hence, it seems worthwhile to look for restricted subcases of planning that wrap up interesting application problems more tightly and are tractable (or at least of reasonable complexity), which we refer to as the *restricted-problems approach.*

Furthermore, we may even have to modify, rather than restrict, formalisms. Consider, *e.g.*, the blocks-world problem where several blocks may have the same label, thus being interchangeable. This problem is NP-complete [22], thus probably simpler than the PSPACE-complete problems, yet it cannot be naturally modelled in propositional STRIPS, further supporting what we said in the case of TWEAK above. That is, although we could wish for a number of more advanced constructs in the standard planning formalisms, we could, at the same time, most likely limit the use of many other constructs in order to reduce the complexity—still being able to model most application problems.

Similar observations have motivated research into identifying restrictions that make hard problems tractable in other areas of AI than planning, *eg.*, *concept languages* [16], *abduction* [19] and *non-monotonic reasoning* [39].

5 The SAS and SAS$^+$ formalisms

As the basic vehicle for our research we have used the SAS$^+$ formalism and its predecessor, the SAS formalism. The SAS$^+$ formalism models states by multi-valued state variables, rather than propositional atoms, and also allows a state variable to have an unknown value in a state, allowing a rudimentary form of uncertainty. Operators are modelled using three conditions, the *pre-*, *post-* and *prevail*-conditions. The pre- and post-condition must only define variables which are affected, *i.e.* changed, by the operator; the pre-condition specifies the required initial value for such a variable and the post-condition specifies its value after executing the operator. The pre-condition may be left unspecified for an affected variable if its initial value does not matter. The prevail-condition specifies those variables which must have some particular initial value in order to execute the operator and which are not affected by the operator. Obviously, such a variable has the same value after executing the operator so its value prevails, the reason for the name of this condition.[2] It is easy to see that the pre- and prevail-conditions taken together correspond to the pre-condition of a STRIPS operator and the post-condition corresponds to the STRIPS post-condition, *i.e.* the combination of its add- and delete-lists. Making a distinction between the affected and the non-affected variables in the initial condition has been very important for identifying restrictions. There are alternative ways of making this distinction than having two separate conditions, but we have used this method, partly for historical reasons (heritage from the actions structures formalism) and partly because we have found it the simplest and clearest method for definitions and proofs.

It should be quite obvious to the reader now that the SAS$^+$ formalism is very similar to the propositional STRIPS language, except that we allow not only binary state variables. It may seem that multi-valued state variables add to the expressive power, but this turns out not to be the case. We have proven [8] that the propositional variants of the STRIPS and TWEAK formalisms are equally expressive as the SAS$^+$ formalism, under a very strong form of polynomial reduction. This means that of negative pre-conditions and goals; undefined propositions in the initial state and multi-valued state variables neither add to the expressiveness. However, naturalness and easiness of modelling applications can differ, of course. Furthermore, most of the restrictions we have identified are difficult to translate into restrictions for the other formalisms and become awkward and unintelligible when translated [47].

While SAS$^+$ is our 'standard' formalism, we actually started with defining the SAS formalism, which does not allow partial initial states or goals and does not allow operators changing a variable from *any* value to some specific value. The SAS$^+$ formalism appeared as an extension to the SAS formalism.

6 The First Phase—Syntactical Restrictions

The first phase of our project started with an attempt to apply AI planning to a toy problem in sequential control—the tunnel problem. This problem assumes a tunnel (see Figure 1) divided into n sections such that the light can be switched on and off independently in each of these. Furthermore, the only light switches for a section are located at each end of that section. It is also assumed that one can only pass through a section if the light is on in that section. As a typical instance of this problem assume that all lights are off, except in section n, the innermost section. The task is to make sure that all lights are switched off. This can be achieved by going into the tunnel, repeatedly switching on the light in each new section encountered until reaching the innermost section, then leaving the tunnel again, repeatedly switching off the light in each section, including the innermost one, when leaving it. Although this problem might seem like a trivial toy problem it has a structure similar to many realistic problems, and it is hardly more 'toyish' than the blocks-world problem, which has followed us in AI planning ever since the beginning. Furthermore, although any programmer could easily come up with a program specifically designed to solve the tunnel problem, there are no general standard methods in automatic control for coping with such problems.

[2]It was, perhaps, even more obvious in the action structures formalism [49] where actions are modelled to have a duration, why the value actually prevails *during* the action execution.

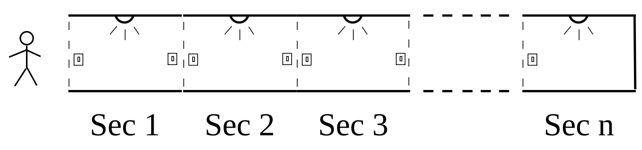

Figure 1: The tunnel example.

We started by looking for inherent restrictions in the tunnel problem and we identified four such restrictions (out of many more possible ones, of course).

P *Post-uniqueness:* For each variable and each possible value for that variable, there must be at most one operator changing the variable to this particular value. That is, there are essentially no alternative ways to achieve an effect.

U *Unariness:* Every operator changes exactly one state variable. That is, an operator cannot have multiple effects.

B *Binariness:* All state variables have two possible values, in addition to the undefined value. In principle, this simulates STRIPS or TWEAK.

S *Single-valuedness:* It is not allowed for two operators to both define the same variable in their prevail-conditions if they specify different values. For instance, consider the two operators of showing some slides and of reading a book, respectively. Showing slides require that the lights are off in this room, but also leaves the lights off, *i.e.* it has as a prevail-condition that the lights are off. To read a book, on the other hand, has as prevail-condition that the lights are on, a value conflicting with the value off. Hence, a set of operators containing both the slide-showing operator and the book-reading operator cannot be single-valued.

These restrictions appear as quite natural and straightforward in the SAS formalism, but they seem much harder to express in STRIPS. In fact, it is most questionable whether we would have been able to identify them if using a STRIPS formalism instead. Hence, it seems useful to also try non-standard formalisms when looking for restrictions—thus getting another point of view. We refer to the subclass of planning problems in the SAS formalism satisfying all four restrictions above as the *SAS-PUBS* class. While the planning problem in the SAS formalism *per se* is not tractable, we proved the SAS-PUBS planning problem tractable by devising a sound and complete polynomial-time algorithm for it [9, 11, 41].

Starting from the SAS-PUBS problem, we have then incrementally removed restrictions and modified the SAS formalism into SAS$^+$, resulting in a number of new tractable subclasses. In parallel with incrementally lower-bounding the tractability borderline in this way, we have also tried to upper-bound it, by proving intractability results for certain subcases (see Section 9).

7 The Second Phase—Structural Restrictions

Until recently, we worked only with the syntactical restrictions in the previous section. Similarly, other results in the literature on planning complexity also considered syntactical restrictions only, *e.g.* restricting the number and/or polarity of atoms in the precondition [18, 25].

Syntactical restrictions are very appealing to study since, typically, they are easy to define and not very costly to test. However, to gain any deeper insight into what makes planning problems hard and easy respectively probably require that we study the structure of the problem, in particular the *state-transition graph* induced by the operators. To some extent, syntactic restrictions allow us to do this since they undoubtedly have implications for what this graph looks like. However, their value for this purpose seems somewhat limited since many properties that are easy to express as structural restrictions would require horrendous syntactical equivalents. Putting explicit restrictions on the state-transition graph must be done with great care, however, since this graph is typically of size exponential in the size of the planning problem instance, making it extremely costly to test arbitrary properties.

Recently we took an intermediate approach [36, 37]. Instead of analysing the whole state-transition graph, we concentrate on its projections onto the variables, that is, we study the possible transitions for each variable in isolation. Since these *domain-transition graphs* are only of polynomial size, it possible to do some analysis in polynomial time. Naturally, much information is lost here compared to using the state-transition graph, but interesting structural properties can still be revealed, especially if paying some attention to dependencies between variables. Although not being a substitute for restrictions on the whole state-transition graph, many interesting and useful properties of this graph can be indirectly exploited. In particular, we identified three structural restrictions (I, A and O, presented below) which, taken together, make planning tractable and properly generalize the tractable problems we have previously defined using syntactical restrictions. These restrictions are more difficult to explain informally than the syntactic ones and will, thus, be presented in a much simplified form here, focussing on how to exploit them for planning. To understand the restrictions, however, it helps to realize that the algorithm used to plan for the SAS$^+$-IAO class works by generating subplans separately for each variable and then merging these. Obviously, it cannot do so without taking interference effects between the subplans into account, so the process is somewhat more complex, iteratively recreating subplans until converging on a number of subplans that can be trivially merged to a valid plan. The restrictions are as follows:

- I *Interference-safeness:* If an operator changes more than one variable, then for each such variable it splits the domain into two partitions and is the only operator which can move between these partitions. This means that if we have to move from one partition to the other for a variable, then we know that we must also do so for all other variables the operator affects, *i.e.* we get an interference between subplans. If this side-effect is unacceptable, then we can know immediately that there is no solution. Otherwise, this helps by splitting the problem into smaller subproblems.

- A *Acyclicity:* For each variable, certain distinguished values must be achieved in a particular order, if achieved at all. Since these are precisely those values that may cause interference between subplans, they can be used as synchronization points in the planning process. In fact, these values drive the iterative process and define its termination condition. There are also the variants A$^+$ and A$^-$ which consider fewer and more distinguished values respectively.

- O *Prevail-order-preservation:* For each variable, the sequence of prevail-conditions along a shortest operator sequence between two values must be the same or a relaxation of the values for any other sequence between the same values. Hence, it can never be easier to generate a plan by generating non-shortest subplans for a variable since this would cause more interference between the subplans.

Despite being structural, these restrictions can be tested in polynomial time. Further, this approach would not be very useful for a planning formalism based on propositional atoms, since the resulting two-vertex domain-transition graphs would not allow for much structure to exploit.

In total we have so far worked with 9 syntactic and structural restrictions. These are not all orthogonal to each other, however; for instance, I subsumes U, O subsumes P and IA subsumes US.

8 Issues in Planning Complexity

Before summarizing our complexity results in the next section, it seems appropriate to clear out some subtle details in planning complexity. When turning to the issue of complexity, we have to make clear what problem we are discussing. In simple terms, the *planning problem* is to find a plan, *i.e.* a seuquence of operators, from an initial state to a goal state.[3] However, in complexity theory one distinguishes between *decision problems* and *search problems*. A decision problem only asks whether a solution exists or not, while the corresponding search problem also asks for a solution. We may also restrict our problems to consider optimal solutions only. In the case of a search problem we can ask for an optimal solution, but in the case of decision problems this makes no sense; if there is any solution, then there must also be an optimal solution. Hence, one usually adds an extra parameter instead, asking for/whether there is a solution having a certain measure less than/greater than this parameter.

We thus distinguish the following variants of the planning problem, all taking a set of operators, an initial state and a goal state as parameters: The *plan existence problem (PE)* asks whether there exists a sequence of operators (a plan) leading from the initial state to the goal state, *i.e.* it asks for a YES/NO answer. The *plan generation problem (PG)* asks for such a plan, or the answer NO. If adding also a positive integer K as parameter we further have: The *bounded plan existence problem (BPE)* asks whether there exists a sequence of at most K operators leading from the initial state to the goal state, while the *bounded plan generation problem (BPG)* asks for such a sequence or the answer NO.

Normally, one only considers the decision variants of problems since these are simpler to handle and often give a good indication of the complexity for the corresponding search problems. If a decision problem is polynomial, then the search problem is typically polynomial and if the decision problem is NP-complete, then the search problem is typically NP-equivalent.[4] In the case of planning problems, however, this turns out to be a dangerous assumption. Formalisms like propositional STRIPS and SAS$^+$ let us encode problem instances having only solutions with exponentially many operators, which makes the plan generation problem inherently intractable—we could not even output the solution in polynomial time. We know that the corresponding plan existence problems is PSPACE-complete, but this does not guarantee intractability—nobody

[3]In principle, it is sufficient to study total-order plans since every total-order plan is a partial-order plan. However, the complexity results for generating total-order and partial-order plans respectively may differ if we demand certain optimality criteria of the partial-order plans [6, 7].

[4]Loosely speaking, NP-equivalent means NP-complete, but for search problems.

knows yet whether P$\neq$PSPACE. Even worse (or even more interesting, depending on one's point of view), we have recently identified a restricted class of planning problems, 3S, where minimal solutions may be exponentially long, hence the plan generation problem is intractable, but where we can decide in polynomial time whether a solution exists or not [38]. Since we are mostly interested in a plan to execute, not only knowing whether it exists or not, this result shows that it is not sufficient to prove tractability for the plan existence problem—we must rather show that plan generation, our ultimate goal, is tractable. The situation is somewhat different for bounded plans, because of the length parameter. The bounded plan existence problem can actually be used to solve the bounded plan generation problem using a method referred to as *prefix search* [15, 28]. However, unless we bound the value of K polynomially in the instance size, this only yields a pseudopolynomial reduction and, thus, still does not allow us to relate the complexity of the existence and generation problems. Although exponentially sized plans are, in a sense, pathological and unrealistic [4, 12, 13], it is non-trivial to rule them out formally, thus plaguing our complexity figures.

9 Summary of Complexity Results for SAS$^+$ Planning

A complete map over the complexities of both the bounded and unbounded versions of the plan existence and generation problems for all combinations of the syntactic restrictions P, U, B and S appear in Bäckström and Nebel [12, 13]. Since results for plan existence seem less relevant, we restrict ourselves to summarizing the results for plan generation here. SAS$^+$-PUS is the maximal tractable subclass for bounded plan generation wrt. to the four syntactical restrictions P, U, B and S. That is, if we want to generate optimal plans in polynomial time, then all three restrictions P, U and S must hold. If we drop the U restriction, then BPG becomes NP-equivalent. However, if we do not require the plans to be optimal, then it is sufficient to have the U and S restrictions to find a plan in polynomial time. If either of these two conditions does not hold, then planning is inherently intractable, due to exponentially sized solutions—even if all other conditions hold, *i.e.* SAS$^+$-PUB and SAS$^+$-PBS are intractable.

Similar complexity maps for the bounded and unbounded plan generation problem when taking both the syntactic restrictions P, U, B and S and the structural restrictions I, A and O into account appear in Jonsson and Bäckström [34, 35]. The maximal tractable class for bounded plan generation then turns out to be SAS$^+$-IA$^-$O, which properly includes the SAS$^+$-PUS class. Dropping either I or A$^-$ leads to inherent intractability. Dropping O, however, results in NP-equivalence, since the optimal solutions remain polynomially bounded in this case. Furthermore, it is also worth noting that the algorithm for the SAS$^+$-IA$^-$O class remains sound and runs in polynomial time also for the SAS$^+$-IA$^-$ class, but completeness is lost.[5] In the case of non-bounded plan generation, the SAS$^+$-US formalism remains maximal.

Recently, we have stepped outside also these restrictions, identifying a tractable class of planning problems which allows modelling the LEGO car factory [42] and plan for it using an extension of the algorithm for the SAS$^+$-IAO class. Another recent result is the 3S class [38], mentioned above, which is based on analysing variable dependencies in a propositional STRIPS formalism.

[5]Similarly, the SAS-PUBS algorithm is sound but incomplete for the SAS-PUB class [40].

10 Discussion

Having read this far, the reader may rightfully ask whether we believe it possible to eventually find tractable classes covering most application problems. The answer, however, is not as simple as the question. While we believe that many application problems in structured environments, like the industry, are inherently tractable, this fact may not be easily exploited. Finding and exploiting the underlying structure causing tractability may be non-trivial. In other cases, tractability may only hold for certain values of parameters which cannot be easily bounded, but can be assumed from experience to have reasonable values.[6] While this latter case does not guarantee formal tractability, it may guarantee tractability under a certain hypothesis, which may be quite reasonable in many cases. It is also worth pointing out that in many cases of NP-complete planning problems, the NP-completeness is likely to stem from a scheduling and/or resource allocation subproblem, while planning, *i.e.* finding the actions, is simple. In this case, a deliberate separation of the problems may enable the use of an efficient scheduler/resource allocator and a simple planner in combination.

Furthermore, we should also not be to obsessed by tractability only. The search for tractable subclasses and the complexity analysis of restrictions can provide useful information about how to find efficient, although not polynomial, algorithms for other restricted classes (cf. results in temporal reasoning [52]). For instance, in some of our algorithms it may be possible to relax some restriction and introduce a limited form of search. Another possibility is to build up a library of algorithms for tractable subclasses and then use these as subroutines in a more general search-based planner, or to use a classifier and invoke one of these algorithms whenever possible and otherwise use a general search-based planner.

It may be worth pointing out that the restricted-problem approach is somewhat similar to knowledge-based planning as used in HTN planners like O-PLAN [24]. In an HTN planner, expert knowledge is encoded as task reduction schemata having the effect of allowing only a small portion of the whole search-space of plans to be explored. Tate [pers. comm.] argues that search should be avoided entirely whenever possible. In principle, the main difference between this approach and ours is whether the information used to avoid or reducing search comes from expert knowledge or from a formal analysis of the problem. The HTN approach is more general, but not formally verifiable, while our approach requires special algorithms tailored to subclasses, but provides formal guarantees.

Similarly, using a general search-based planner like TWEAK or SNLP equipped with heuristics to prune or reorder the search tree is also a complementary approach. In principle, it may be possible to tailor such a planner to a tractable class, but the heuristic needed for this may be quite complex and non-trivial to find, and a tailored algorithm is most likely more efficient.

Another issue is whether our approach scales up to handle more realistic applications requiring reasoning about uncertainty, resources and metric time. This question remains to be satisfactorily answered also for the other approaches we just discussed. For our approach, the truth is that we have hope but no guarantee. However, even if our approach does not scale up, our research is hardly wasted; we strongly believe that we have to thoroughly understand these simpler formalisms before we can formally attack and understand more expressive ones, and that many lessons learned will carry

[6]An example of a *constant-by-experience* parameter limiting the plan length appears in manufacturing planning [45].

over. An example supporting this is the paper on temporal projection by Lin and Dean [33]. Dean [pers. comm.] says that the problem they are trying to solve is temporal projection with uncertainty, but after having spent some years trying this they found the problem so difficult they had to switch back, analysing the basic case first.

Appendix. Chronology and References for Planning Algorithms

Below is listed in chronological order the tractable subcases we have devised planning algorithms for, together with the year and references to the literature.

SAS-PUBS 1990 [9, 11, 41]

SAS-PUS 1991 [10]

SAS$^+$-PUS 1992 [4, 5]

SAS$^+$-US 1993 [12, 13] (only for the PG problem)

SAS$^+$-IAO 1994 [36, 37]

SAS$^+$-IA$^-$O 1994 [34, 35]

SAS$^+$-IAO w. interweaving 1995 [42] (can plan for the LEGO car factory)

3S 1995 [38] (solves PE in polynomial time and PG in solution-polynomial time)

Acknowledgements

Inger Klein and Peter Jonsson have provided helpful comments on this paper. This paper and the research reported is also influenced by many comments and discussions over the years with various people including, but not limited to, Tom Bylander, Marco Cadoli, Tom Dean, Mark Drummond, John Hallam, Jim Hendler, Lennart Ljung, Bernhard Nebel, Christos Papadimitriou, Erik Sandewall, Bart Selman, Austin Tate, Wolfgang Wahlster, Qiang Yang and lots of anonymous referees. The research reported in this paper has been sponsored by various sources inlcuding the Swedish National Board for Technology Development (STU), the Swedish Research Council for Engineering Sciences (TFR), the German Ministry for Research and Technology and the European Commission.

References

[1] J. Allen, J. Hendler, and A. Tate, editors. *Readings in Planning.* San Mateo, CA, 1990.

[2] K.-J. Åström and A. Benveniste. Meeting the challenge of computer science in the industrial applications of control: An introductury discussion to the special issue. *IEEE Trans. Automatic Control*, 38(7):1004–1010, July 1993.

[3] K.-J. Åström, A. Benveniste, et al. Facing the challenge of computer science in the industrial applications of control: a joint IEEE CSS–IFAC project. Progress Report, Mar. 1991.

[4] C. Bäckström. *Computational Complexity of Reasoning about Plans.* Doctoral dissertation, Linköping University, Linköping, Sweden, June 1992.

[5] C. Bäckström. Equivalence and tractability results for SAS$^+$ planning. In *Proc. 3rd Int'l Conf. on Principles of Knowledge Repr. and Reasoning (KR-92)*, pages 126–137, Cambridge, MA, USA, 1992.

[6] C. Bäckström. Finding least constrained plans and optimal parallel executions is harder than we thought. In Bäckström and Sandewall [14], pages 46–59.

[7] C. Bäckström. Executing parallel plans faster by adding actions. In Cohn [23], pages 615–619.

[8] C. Bäckström. Expressive equivalence of planning formalisms. *Artif. Intell., Special Issue on Planning and Scheduling*, 76(1–2):17–34, 1995. ARTINT 1234.

[9] C. Bäckström and I. Klein. Planning in polynomial time. In *Expert Systems in Eng.: Principles and Appl. Int'l WS.*, volume 462 of *Lecture Notes in Artificial Intelligence*, pages 103–118, Vienna, Austria, 1990.

[10] C. Bäckström and I. Klein. Parallel non-binary planning in polynomial time. In *Proc 12th Int'l Joint Conf. on Artif. Intell. (IJCAI-91)*, pages 268–273, Sydney, Australia, 1991.

[11] C. Bäckström and I. Klein. Planning in polynomial time: The SAS-PUBS class. *Comput. Intell.*, 7(3):181–197, Aug. 1991.

[12] C. Bäckström and B. Nebel. Complexity results for SAS$^+$ planning. In *Proc 13th Int'l Joint Conf. on Artif. Intell. (IJCAI-93)*, pages 1430–1435, Chamberý, France, 1993.

[13] C. Bäckström and B. Nebel. Complexity results for SAS$^+$ planning. *Comput. Intell.*, 11(4), 1995. In Press.

[14] C. Bäckström and E. Sandewall, editors. *Current Trends in AI Planning: Proc. 2nd Eur. WS. Planning (EWSP'93)*, Vadstena, Sweden, Dec. 1993. IOS Press.

[15] J. L. Balcázar, J. Diaz, and J. Gabarró. *Structural Complexity I.* Springer, 1988.

[16] R. J. Brachman and H. J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. 4th (US) Nat'l Conf. on Artif. Intell. (AAAI-84)*, pages 34–37, Austin, TX, USA, 1984.

[17] T. Bylander. Tractability and artificial intelligence. *J. Expt. Theoret. Artif. Intell.*, 3:171–178, 1991.

[18] T. Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69:165–204, 1994.

[19] T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson. Some results concerning the computational complexity of abduction. In KR [43], pages 44–54.

[20] D. Chapman. Planning for conjunctive goals. *Artif. Intell.*, 32:333–377, 1987.

[21] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence.* Addison Wesley, Reading, MA, 1985.

[22] S. V. Chenoweth. On the NP-hardness of blocks world. In *Proc. 9th (US) Nat'l Conf. on Artif. Intell. (AAAI-91)*, pages 623–628, Anaheim, CA, USA, 1991.

[23] A. G. Cohn, editor. *Proc. 11th Eur. Conf. on Artif. Intell. (ECAI-94)*, Amsterdam, Netherlands, Aug. 1994. Wiley.

[24] K. Currie and A. Tate. O-Plan: The open planning architecture. *Artif. Intell.*, 52:49–86, 1991.

[25] K. Erol, D. S. Nau, and V. S. Subrahmanian. On the complexity of domain-independent planning. In *Proc. 10th (US) Nat'l Conf. on Artif Intell. (AAAI-92)*, pages 381–386, San José, CA, USA, 1992.

[26] K. Erol, D. S. Nau, and V. S. Subrahmanian. When is planning decidable? In *Proc. 1st Int'l Conf. on Artif. Intell. Planning Sys. (AIPS-92)*, pages 222–227, College Park, MD, USA, 1992.

[27] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2:189–208, 1971.

[28] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[29] M. Ghallab and A. Milani, editors. *New Trends in AI Planning: Proc. 3rd Eur. WS. Planning (EWSP'95)*, Assissi, Italy, 1995. IOS Press. (This volume.)

[30] C. Green. Application of theorem proving to planning. In *Proc. 1st Int'l Joint Conf. on Artif. Intell. (IJCAI-69)*, pages 219–239, Washington, DC, USA, 1969. Reprinted in Allen *et al* [1], pages 67–87.

[31] N. Gupta and D. S. Nau. On the complexity of blocks-world planning. *Artif. Intell.*, 56:223–254, 1992.

[32] P. Hayes and K. Ford. Turing test considered harmful. In Mellish [46], pages 972–977.

[33] S. Hong Lin and T. Dean. Exploiting locality in temporal reasoning. In Bäckström and Sandewall [14], pages 199–212.

[34] P. Jonsson and C. Bäckström. Complexity results for state-variable planning under mixed syntactical and structural restrictions. In *Proc. 6th Int'l Conf. on Artif. Intell.: Methodology, Systems, Applications (AIMSA-94)*, Sofia, Bulgaria, 1994. World Scientific.

[35] P. Jonsson and C. Bäckström. Complexity results for state-variable planning under mixed syntactical and structural restrictions. Research Report R-95-17, Department of Computer and Information Science, Linköping University, 1994.

[36] P. Jonsson and C. Bäckström. Tractable planning with state variables by exploiting structural restrictions. In *Proc. 12th (US) Nat'l Conf. on Artif. Intell. (AAAI-94)*, pages 998–1003, Seattle, WA, USA, 1994.

[37] P. Jonsson and C. Bäckström. Tractable planning with state variables by exploiting structural restrictions. Research Report R-95-16, Department of Computer and Information Science, Linköping University, 1994.

[38] P. Jonsson and C. Bäckström. Incremental planning. In Ghallab and Milani [29].

[39] H. A. Kautz and B. Selman. Hard problems for simple default logics. In KR [43], pages 189–197.

[40] I. Klein. *Automatic Synthesis of Sequential Control Schemes.* Doctoral dissertation, Linköping University, Linköping, Sweden, Apr. 1993.

[41] I. Klein and C. Bäckström. On the planning problem in sequential control. In *Proc. 30th IEEE Conf. on Decision and Control (CDC-91)*, pages 1819–1823, Brighton, UK, 1991.

[42] I. Klein, P. Jonsson, and C. Bäckström. Tractable planning for an assembly line. In Ghallab and Milani [29].

[43] *Proc. 1st Int'l Conf. on Principles of Knowledge Repr. and Reasoning (KR-89)*, Toronto, Canada, 1989.

[44] H. J. Levesque. Logic and the complexity of reasoning. *J. Phil. Logic*, 17:355–389, 1988.

[45] A. Márkus and J. Váncza. Inference and optimization methods for manufacturing process planning. In Cohn [23], pages 595–599.

[46] C. Mellish, editor. *Proc. 14th Int'l Joint Conf. on Artif. Intell. (IJCAI-95)*, Montréal, PQ, Canada, 1995. Morgan Kaufmann.

[47] B. Nebel and C. Bäckström. On the computational complexity of temporal projection, planning and plan validation. *Artif. Intell.*, 66(1):125–160, 1994. ARTINT 1063.

[48] K. M. Passino and P. J. Antsaklis. A system and control theoretic perspective on artificial intelligence planning systems. *Appl. Artif. Intell.*, 3(1):1–32, 1989.

[49] E. Sandewall and R. Rönnquist. A representation of action structures. In *Proc. 5th (US) Nat'l Conf. on Artif. Intell. (AAAI-86)*, pages 89–97, Philadelphia, PA, USA, 1986.

[50] H. A. Simon. Explaining the ineffable: AI on the topics of intuition, insight and inspiration. In Mellish [46], pages 939–948. Research Excellence Award paper.

[51] J. D. Ullman. The role of theory today. *ACM Comput. Surveys*, 27(1):43–44, 1995.

[52] P. van Beek. Reasoning about qualitative temporal information. In *Proc. 8th (US) Nat'l Conf. on Artif. Intell. (AAAI-90)*, pages 728–734, Boston, MA, USA, 1990.

[53] D. E. Wilkins. *Practical Planning.* Morgan Kaufmann, San Mateo, CA, 1988.

Most of the author's publications are available by anonymous ftp at:
`ftp.ida.liu.se@pub/labs/rkllab/people/chrba`
and by WWW at:
`http://www.ida.liu.se/labs/rkllab/people/chrba/index.html`