Computational complexity of planning, diagnosis, and diagnostic planning in the presence of static causal laws

Chitta Baral and **Le Chi Tuan**
Dept. Computer Science & Engineering
Arizona State University
Tempe, AZ 85287-5406, USA
{chitta@asu.edu, lctuan@asu.edu}

Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
vladik@cs.utep.edu

Abstract

Planning is a very important AI problem, and it is also a very time-consuming AI problem. To get an idea of how complex different planning problems are, it is useful to describe the computational complexity of different general planning problems. This complexity has been described for problems in which the result $res(a, s)$ of applying an action a to a system in a state s is uniquely determined by the action a and by the state s. In real-life planning, some consequences of certain actions are non-deterministic. In this paper, we expand the known results about computational complexity of planning (with and without sensing) to this more general class of planning problems.

In addition to analyzing computational complexity of regular planning – in which the goal is to achieve a certain property of the system – we also analyze the computational complexity of a simpler problem – of diagnosing the system.

Introduction

It is important to analyze computational complexity of planning problems

Planning is one of the most important AI problems, but it is also known to be one of the most difficult ones. While often in practical applications, we need the planning problems to be solved within a reasonable time, the actual application of planning algorithms may take an extremely long time. It is therefore desirable to estimate the potential computation time which is necessary to solve different planning problems, i.e., to estimate the *computational complexity* of different classes of planning problems.

Of course, if we manage to show that a problem for which no fast algorithm is known is, actually, feasible, such a result is of great practical value: depending on whether the proof is direct (i.e., contains an explicit feasible algorithm) or is indirect, we either have a good planning algorithm, or a proof that such an algorithm is possible. Often, however, for a reasonable class of planning problems, the complexity result confirms that this problem is difficult to solve by showing that this problem belongs to one of the high-level complexity classes (e.g., that it is **PSPACE**-hard). Such "negative" results do not have direct practical use, but they are potentially useful: first, they prevent researchers from wasting their time on trying to design a general efficient algorithm; second, they enable the researchers to concentrate on either finding a feasible sub-class of the original class of planning problems, or on finding (and/or justifying) an approximate planning algorithm.

Known computational complexity results: in brief

There have been several results on computational complexity of planning problems. These results mainly cover the situations in which we have a (complete or partial) information about the current state of the system, and we must find an appropriate plan (sequence of actions) which would enable us to achieve a certain goal. Such situations are described by the language $\mathcal{A}$ which was proposed in (GL93). In this language, we start with a finite set of properties (fluents) $\mathcal{F} = \{f_1, \ldots, f_n\}$ which describe possible properties of a state. A *state* is then defined as a finite set of fluents; e.g., if $n = 2$, then the state $\{f_1\}$ means a state in which f_1 is true, and f_2 is not. Our knowledge about the initial state is described by statements of the type "initially F", where F is a *fluent literal*, i.e., a fluent or its negation. There is also a finite set $\mathcal{A}$ of possible *actions*. The results of different actions $a \in \mathcal{A}$ are described by rules of the type "a causes F if $F_1, \ldots, F_m$", where $F, F_1, \ldots, F_m$ are fluent literals. The semantics include "inertia": if an action does not lead to f or $\neg f$, then the value of the fluent f does not change.

To formulate a planning problem, we must select an *objective*. In general, our objective can be an arbitrary logical combination of the basic properties (fluents); however, since we can always add this combination as a new fluent, we can, without losing generality, assume that our objective is one of the fluents g from $\mathcal{F}$. A *plan* is a sequence of actions $\alpha = [a_1, \ldots, a_m]$. We say that a plan is *successful* if for every initial state s which is consistent with our knowledge, after we apply the plan α, the desired fluent g holds in the resulting state $res(\alpha, s)$.

Ideally, we want to find cases in which the planning problem can be solved by a *feasible* algorithm, i.e., by an algorithm $\mathcal{U}$ whose computational time $t_{\mathcal{U}}(w)$ on each input w is bounded by a polynomial $p(|w|)$ of the length $|w|$ of the input w: $t_{\mathcal{U}}(x) \le p(|w|)$ (this length can be measured bit-wise or symbol-wise). Since, in practice, we are operating in a time-bounded environment, we should worry not only about the time for *computing* the plan, but we should also worry about the time that it takes to actually *implement* the plan. If an action plan consists of a sequence of 2^{2^n} actions, then this plan is not feasible. It is therefore reasonable to restrict ourselves to *feasible* plans, i.e., by plans u whose length m (= number of actions in it) is bounded by a given polynomial $p(|w|)$ of the length $|w|$ of the input w. For each such polynomial p, we can formulate the following *planning problem*: given a domain description D (i.e., the description of the initial state and of possible consequences of different actions) and a goal g (i.e., a fluent which we want to be true), determine whether it is possible to feasibly achieve this goal, i.e., whether there exists a feasible plan α (with $m \le p(|D|)$) which achieves this goal.

By solving this problem, we do not yet get the desired plan, we only check whether a plan exists. However, intuitively, the complexity of this problem also represents the complexity of actually finding a plan, in the following sense: if we have an algorithm which solves the above planning problem in reasonable time, then we can also find this plan. Indeed, suppose that we are looking for a plan of length $m \le P_0$, and an algorithm has told us that such a plan exists. Then, to find the first action of the desired plan, we check (by applying the same algorithm), for each action $a \in \mathcal{A}$, whether from the corresponding state $res(a, s)$ the desired goal g can be achieved in $\le P_0 - 1$ steps. Since a plan of length $\le P_0$ does exist, there is such an action, and we can take this action as a_1. After this, we repeat the same procedure to find a_2, etc. As a result, we will be able to find a plan of length $\le P_0$ by applying the algorithm which checks the existence of the plan $\le P_0 = p(|D|)$ times; so, if the existence-checking algorithm is feasible, the resulting plan-construction algorithm is feasible as well.

General results on computational complexity of planning are given, e.g., in (Byl94; ENS95; Lit97). For the language $\mathcal{A}$, computational complexity of planning was first studied in (Lib97); the results about the computational complexity of different planning problems in $\mathcal{A}$ are overviewed in (BKT99; Rin99).

If the initial information is incomplete, then, in addition to normal actions which form the plan, it is reasonable to consider *sensing* actions, i.e., actions which may not change the state of the system, but which enable us to find the missing information. To describe such actions, the language $\mathcal{A}$ was enriched by rules of the type "a determines f", meaning that after the action a is performed, we know whether f is true or not. At any given moment of time, we have the actual state s of the system (which may be not completely known to the agent), plus a set Σ of all possible states which are consistent with the agent's knowledge; the pair $\langle s, \Sigma \rangle$ is called a *k-state*. A sensing action does not change the actual state s, but it does decrease the set Σ.

In planning, the main purpose of sensing actions is to make a planning decision depending on the actual value of the sensed fluent. Thus, when sensing is allowed, a plan is not a sequence, but rather a *tree*: every sensing action means that we branch into two possible branches (depending on whether the sensed fluent is true or false), and we execute different actions on different branches. Similarly to the case of the linear plan, we are only interested in plans whose execution time is (guaranteed to be) bounded by a given polynomial $p(|D|)$ of the length of the input. (In other words, we require that for every possible branch, the total number of actions on this branch is bounded by $p(|D|)$.)

For such planning situations, the computational complexity was also surveyed in (BKT99).

Towards a more realistic formulation of planning problems

The planning problem, as formulated in the language $\mathcal{A}$, is based on the assumption that the results of each action are uniquely determined by the state. In real life, this assumption is not always valid. Actions are often non-deterministic: the values of some fluents are uniquely determined by the corresponding action, but some other fluents may experience unpredictable changes.

Such non-determinism is often manifested in action languages with static causal laws, such as the one in (Tur97) (IJCAI 95 has other proposals by Lin, Baral, and others that reason with static causal laws), where the language $\mathcal{A}$ was extended by allowing so-called *static causal laws*, i.e., statements of the type "F if $F_1, \ldots, F_m$" meaning that if $F_1, \ldots, F_m$ hold, then F should also be true. Due to these laws, the result $res(a, s)$ of applying the action a to the state s is not uniquely determined; of course, all fluent literals d generated by the action-related (dynamic) causal laws should be in $res(a, s)$; the "inertia" requirement is that $res(a, s)$ should coincide with the deductive closure (with respect to static causal laws) of the union of the new fluent literals d and the remaining fluent literals (i.e., of the intersection between s and $res(a, s)$).

Static causal laws not only take care of the non-determinism of actions, they also describe the natural restrictions on the values of different fluents. For example, in a variant of the Yale Shooting problem, a pilgrim is hunting a turkey that is initially alive and trotting. After shooting, the turkey is no longer alive, but to make a common-sense conclusion that it is also not trotting, we need a static causal law of the type $\neg$*trotting* if $\neg$*alive* (Tur97).

In this more realistic situation, we can also ask about the existence of a plan, i.e., a sequence (or tree) of actions with a feasible execution time which guarantees that after this plan, the objective $g \in \mathcal{F}$ will be satisfied.

In this paper, we answer the following natural question: *How does the addition of static causal laws change the computational complexity of different planning problems?*

Comments.

- For planning problems with non-deterministic actions *without* sensing, computational complexity of planning is analyzed (in probabilistic setting) in (Lit97). In this paper, we analyze new classes of planning problems, in which non-determinism (in the form of static causal laws) is combined with the possibility of sensing actions.
- In this paper, we consider planning situations in which the only information we have is the information about the current state. In real life, in addition to the information about the *current* state, we often have some information about the *previous* behavior of the system. The corresponding extension of the language $\mathcal{A}$ was developed in (BGP97) (see also (BGP98; BMS00)). In the future we plan to analyze how this additional possibility changes the computational complexity of different planning problems.

Useful complexity notions

Most papers on computational complexity of planning problems classifies these problems to different levels of polynomial hierarchy. For precise definitions of the polynomial hierarchy, see, e.g., (Pap94). Crudely speaking, a decision problem is a problem of deciding whether a given input w satisfies a certain property P (i.e., in set-theoretic terms, whether it belongs to the corresponding set $S = \{w \,|\, P(w)\}$).

- A decision problem belongs to the class $\mathbf{P}$ if there is a feasible (polynomial-time) algorithm for solving this problem.
- A problem belongs to the class $\mathbf{NP}$ if the checked formula $w \in S$ (equivalently, $P(w)$) can be represented as $\exists u P(u, w)$, where $P(u, w)$ is a feasible property, and the quantifier runs over words of feasible length (i.e., of length limited by some given polynomial of the length of the input). The class $\mathbf{NP}$ is also denoted by $\Sigma_1\mathbf{P}$ to indicate that formulas from this class can be defined by adding 1 existential quantifier (hence Σ and 1) to a polynomial predicate ($\mathbf{P}$).
- A problem belongs to the class $\mathbf{coNP}$ if the checked formula $w \in S$ (equivalently, $P(w)$) can be represented as $\forall u P(u, w)$, where $P(u, w)$ is a feasible property, and the quantifier runs over words of feasible length (i.e., of length limited by some given polynomial of the length of the input). The class $\mathbf{coNP}$ is also denoted by $\Pi_1\mathbf{P}$ to indicate that formulas from this class can be defined by adding 1 universal quantifier (hence Π and 1) to a polynomial predicate (hence $\mathbf{P}$).
- For every positive integer k, a problem belongs to the class $\Sigma_k\mathbf{P}$ if the checked formula $w \in S$ (equivalently, $P(w)$) can be represented as $\exists u_1 \forall u_2 \ldots P(u_1, u_2, \ldots, u_k, w)$, where $P(u_1, \ldots, u_k, w)$ is a feasible property, and all k quantifiers run over words of feasible length (i.e., of length limited by some given polynomial of the length of the input).
- Similarly, for every positive integer k, a problem belongs to the class $\Pi_k\mathbf{P}$ if the checked formula $w \in S$ (equivalently, $P(w)$) can be represented as $\forall u_1 \exists u_2 \ldots P(u_1, u_2, \ldots, u_k, w)$, where $P(u_1, \ldots, u_k, w)$ is a feasible property, and all k quantifiers run over words of feasible length (i.e., of length limited by some given polynomial of the length of the input).
- All these classes $\Sigma_k\mathbf{P}$ and $\Pi_k\mathbf{P}$ are subclasses of a larger class $\mathbf{PSPACE}$ formed by problems which can be solved by a polynomial-*space* algorithm. It is known (see, e.g., (Pap94)) that this class can be equivalently reformulated as a class of problems for which the checked formula $w \in S$ (equivalently, $P(w)$) can be represented as $\forall u_1 \exists u_2 \ldots P(u_1, u_2, \ldots, u_k, w)$, where the number of quantifiers k is bounded by a polynomial of the length of the input, $P(u_1, \ldots, u_k, w)$ is a feasible property, and all k quantifiers run over words of feasible length (i.e., of length limited by some given polynomial of the length of the input).

A problem is called *complete* in a certain class if, crudely speaking, this is the toughest problem in this class (so that any other general problem from this class can be reduced to it by a feasible-time reduction).

It is still not known (2000) whether we can solve any problem from the class $\mathbf{NP}$ in polynomial time (i.e., in precise terms, whether $\mathbf{NP}$=$\mathbf{P}$). However, it is widely believed that we cannot, i.e., that $\mathbf{NP} \neq \mathbf{P}$. It is also believed that to solve a $\mathbf{NP}$-complete or a $\mathbf{coNP}$-complete problem, we need exponential time $\approx 2^n$, and that solving a complete problem from one of the second-level classes $\Sigma_2\mathbf{P}$ or $\Pi_2\mathbf{P}$ requires more computation time than solving $\mathbf{NP}$-complete problems (and solving complete problems from the class $\mathbf{PSPACE}$ takes even longer).

Results

General description of possible planning problems

In accordance with the above text and with (BKT99), we will consider the following four main groups of planning situations:

- complete information about the initial state, no sensing actions allowed;
- possibly incomplete information about the initial state, no sensing actions allowed;
- possibly incomplete information about the initial state, sensing actions allowed;
- possibly incomplete information about the initial state, full sensing (i.e., every fluent can be sensed).

For comparison, we will also mention the results corresponding to the language $\mathcal{A}$, when no static causal laws are allowed.

Complexity of plan checking

Before we describe the computational complexity of checking the existence of a plan, let us consider a simpler problem: if, through some heuristic method, we have a plan, how can we check that this plan works?

This plan checking problem makes perfect sense only for the case of no sensing: indeed, if sensing actions are possible, then we can have a branching at every step; as a result, the size of the tree can grow exponentially with the plan's execution time, and even if we can check this tree plan in time polynomial in its size, it will be still take un-realistically long.

For the language $\mathcal{A}$, the complexity of this problem depends on whether we have complete information of the initial state or not:

Theorem 1. (language $\mathcal{A}$, no sensing)

- *For situations with complete information, the plan checking problem is feasible.*
- *For situations with incomplete information, the plan checking problem is* **coNP**-*complete.*

Comment about the proofs. Similarly to (BKT99), all the proofs consist of two parts: First, we prove that the corresponding problem *belongs* to a given complexity class by explicitly representing this problem in a form defining this class. Second, we prove that the problem is *complete* in the class by reduction to a known complete propositional problem from the corresponding complexity class. For example, for **NP**, the known complete problem is propositional satisfiability, i.e., checking whether $\exists x_1 \ldots \exists x_n F$, where x_i are propositional variables and F is a propositional formula. For other classes, we have similar complete problems, with several quantifier groups instead of one. The reductions are similar to the ones performed in (BKT99).

Theorem 2. (static causal laws, no sensing) *In the presence of static causal laws:*

- *for situations with complete information, the plan checking problem is* **coNP**-*complete.*
- *For situations with incomplete information, the plan checking problem is* **coNP**-*complete.*

Idea of the proof. Both in the situation with complete information and in the situation with incomplete information, checking whether a plan w is successful is equivalent to checking that $\forall u P(u, w)$, where u is a history (sequence of consecutive states), and $P(u, w)$ is a feasible-to-check property meaning that u is consistent with the domain description (i.e., with the information about the initial state and with the dynamic causal laws), and that at the final state, the goal g is satisfied. Thus, for both situations, the planning problem belongs to the class **coNP** ($= \Pi_1\mathbf{P}$).

To complete the proof of the theorem, it is sufficient to prove that both problems are complete in this class. Since the first problem (planning with complete information) is a particular case of the second one (planning with possibly incomplete information), it is sufficient to prove **coNP**-completeness for planning with complete information. For this, as we have mentioned in our general comment about the proofs, we will reduce the known **coNP**-complete problem of checking the validity of $\forall x_1 \ldots \forall x_n F$, where x_i are propositional variables and F is a propositional formula.

In (BKT99), it is shown that, if we can describe x_i as initial fluents, then we can represent a computation of F in terms of appropriate action laws. Here, however, we have a complete knowledge of the initial state, so we cannot directly use the reduction from (BKT99). We must, therefore, use the potential non-determinism of static causal laws to go from a single state to an arbitrary Boolean combination of certain fluents $x_1, \ldots, x_n$.

Namely, in addition to fluents $x_1, \ldots, x_n$, let us introduce $n+1$ auxiliary fluents $y_1, \ldots, y_n, z$. In the initial state, all the fluents x_i, y_i, and z are true. To all the dynamic causal laws describing F, we add an extra condition $\neg z$. Also, for every action $a \in \mathcal{A}$, we add a dynamic control rule "a causes $\neg z$ if z". We also add static causal laws "$\neg x_i$ if $y_i, \neg z$" and "$\neg y_i$ if $x_i, \neg z$" for all $i = 1, \ldots, n$. In the initial state, we have x_i and y_i. Due to these static causal laws, after the first action (when z becomes false), for each i, we cannot any longer have both x_i and y_i, so we will have either $x_i \& \neg y_i$, or $\neg x_i \& y_i$. As a result, after the first action, each of the fluents x_i can be either true or false, and so we can have all 2^n possible combinations of truth values for the variables $x_1, \ldots, x_n$. Thus, the successfulness of the plan means that this plan should be successful for all possible combinations. Now, we can follow (BKT99) and reformulate the problem of checking $\forall x_1 \ldots \forall x_n F$ as a planning problem.

Complexity of planning

Theorem 3. (no sensing) *In the presence of static causal laws, for situations with complete information about the initial state and with no sensing, the planning problem is* $\Sigma_2\mathbf{P}$-*complete.*

For the language $\mathcal{A}$ (without static causal laws), this problem is **NP**-complete.

Theorem 4. (no sensing) *In the presence of static causal laws, for situations with incomplete information about the initial state and with no sensing, the planning problem is* $\Sigma_2\mathbf{P}$-*complete.*

For the language $\mathcal{A}$ (without static causal laws), this problem is also $\Sigma_2\mathbf{P}$-complete.

Theorem 5. (with sensing) *In the presence of static causal laws, for situations with incomplete information about the initial state and with sensing, the planning problem is* **PSPACE**-*complete.*

For the language $\mathcal{A}$ (without static causal laws), this problem is also **PSPACE**-complete.

Theorem 6. (full sensing) *In the presence of static causal laws, for situations with incomplete information about the initial state and with full sensing, the planning problem is* $\Pi_2\mathbf{P}$*-complete.*

For the language $\mathcal{A}$ (without static causal laws), this problem is also $\Pi_2\mathbf{P}$-complete.

Computational complexity of diagnosis and diagnostic planning

In the above text, the main objective was to *change* the system, namely, to change its state in such a way that a certain goal fluent becomes true. For example, in maintenance, the goal would be to repair the system. A related problem that is recently discussed in (BMS00) is the problem of planning so as to reach a unique diagnosis. In this we first present diagnosis(Rei87)[1] and diagnostic planning and then analyze their complexity.

What is diagnosis and what is diagnostic planning

Consider the following two steps often used in real life.

- first, we are trying to use our knowledge of the system's present and past behavior to find out what exactly is wrong;
- if from the existing information, we cannot uniquely determine what is wrong, then we plan and perform appropriate tests, and use the results of these tests to find out what exactly in wrong with the system.

It is natural to call the first step *diagnosis*, and the second step *diagnostic planning*.

These two problems can be naturally formalized as follows. We assume that in the set $\mathcal{F}$ of all the fluents, there is a subset $Ab \subseteq \mathcal{F}$ formed by fluents of the type $ab(c_i)$ whose meaning is that i-th component of the system is faulty. If all our observations are consistent with the assumption that all these fluents are false, this means that there is no indication of anything going wrong with the system. So, the first questions is: *To check whether the system needs a diagnosis at all*, i.e., whether it is consistent with the assumption $\neg ab(c_i)$ for all components c_i.

If the system does need a diagnosis, then the natural next question is: can we make this diagnosis based on the existing information? In system terms, diagnosing a system means finding which exactly components are faulty. In other words, a diagnosis means selecting a subset $C_f \subseteq C$ in the set C of all the components in such a way that the existing information is consistent with the statements $ab(c_i)$ for $c_i \in C_f$ and $\neg ab(c_i)$ for $c_i \notin C_f$. We will call such subsets *consistent*.

This definition is not yet complete: Every diagnosis can be described by a consistent subset $C_f \subseteq C$, but not every consistent subset corresponds to a meaningful diagnosis. Indeed, most rules describe what happens if a component is functioning properly and predict nothing definite about the system's abnormal behavior. Therefore, if we can explain the system's abnormal behavior by assuming that, say, component c_1 is faulty while all other components are OK (i.e., that $C_f = \{c_1\}$), then we can also explain the same behavior by assuming that all the system's components are faulty (i.e., that $C_f = C$). So, together with the actual set of faulty components, usually, every superset of this set is also consistent with the given information. Thus, a meaningful diagnosis is not simply a consistent subset C_f of the set of components C, but a *minimal* consistent subset, i.e., a subset which is itself consistent but for which no proper subset is consistent.

For a given information, there may be several different diagnoses. So, the next question is: *Is the given information sufficient to select a single diagnosis?*

If not, then we have the third question: *Is it possible to find a feasible plan which would lead to a unique diagnosis?*

Let us describe the computational complexity of these three questions.

These problems cannot be formulated in the language $\mathcal{A}$

In the language $\mathcal{A}$, the only information we have is the information about the present state; there are no static causal laws, so for every fluent, every value is possible irrespective of the values of all other fluents. Thus, in the simplified situations described by this language, the only way to conclude that one of the components is faulty (i.e., that the fluent $ab(c_i)$ is true), is to observe this fluent to be true. Hence, if our observations do not include any fluents of this type, we can safely assume that the system is not faulty.

In other words, in the language $\mathcal{A}$, the diagnosis problem is trivial. In view of this triviality, in the following text, we will only consider the situations with static causal laws.

Computational complexity of checking whether the system is functioning properly

Theorem 7. *Checking whether the system is functioning properly is* $\mathbf{NP}$*-complete.*

In some situations, we already know that some components are malfunctioning, the question is: whether other components are functioning OK. The corresponding checking problem has the same computational complexity:

Theorem 8. *In situations when some components are known to malfunction, checking whether the remaining components are functioning properly is* $\mathbf{NP}$*-complete.*

Computational complexity of checking whether the existing information is sufficient for a diagnosis

We have already mentioned that in realistic maintenance situations, if a set C_f is consistent, then every superset of this set is also consistent, and the set $C_f = C$ is always consistent. We can call a planning problem

[1]For lack of space we skip the other references to model based diagnosis.

realistic if it satisfies these two properties, and consider only realistic diagnostic problems.

For a realistic problem, checking whether we can have a unique diagnosis – and even computing this diagnosis – is as simple as checking consistency. To be more precise, we can reduce the problem of computing the diagnosis and checking that it is unique to a feasible (polynomial) number of calls to consistency-checking. (This reduction is similar to the above-described reduction of plan construction to the algorithm which checks the existence of a plan.) This reduction can be performed as follows:

- First, we find a minimal consistent set. According to our assumption, the set $C_1 = C = \{c_1, \ldots, c_m\}$ is consistent. This set C_1 will be our first approximation; we will now find sets $C_2 \subset C_1$, $C_3 \subset C_2$, etc., until we get a minimal consistent set. If we get C_k, then, to find C_{k+1}, we check consistency of the sets $C_k - \{c\}$ for all $c \in C_k$;
  - if none of these smaller sets is consistent, then C_k is the desired minimal consistent set and we stop;
  - if one of these smaller sets is consistent, then we take this smaller set as C_{k+1}.

  Since on every step, we decrease the number of elements in C_k by one, we will stop in $\leq m$ steps and get a minimal consistent set.
- As a result of the first part of the reduction, we get a minimal consistent set $C_f = \{c_{i_1}, \ldots, c_{i_p}\}$. Let us check whether this minimal consistent set is unique. For that, for all $k = 1, \ldots, p$, we check whether a set $C - \{c_{i_k}\}$ is consistent.
  - If one of the sets $C - \{c_{i_k}\}$ is consistent, this means that by applying a procedure similar to our first part, we will get a new minimal set which does not contain c_{i_k} and which is, therefore, different from C_f. Thus, in this case, we conclude that C_f is not a unique minimal consistent set.
  - If none of the sets $C - \{c_{i_k}\}$ is consistent, this means that every consistent set must contain all the elements $c_{i_1}, \ldots, c_{i_p}$, i.e., every consistent set must contain C_f. Thus, in this case, C_f is indeed the unique minimal consistent set.

The reduction is complete.

Due to this reduction, the computational complexity of checking whether the existing information is sufficient for a diagnosis is the same as the computational complexity of checking consistency (i.e., of checking whether the diagnosis is needed).

Computational complexity of checking whether diagnostic planning is possible

The complexity of diagnosis is discussed to some extent in some model based diagnosis papers (such as (Rei87)) when they discuss about algorithms. In that sense our complexity analysis of diagnosis is not completely novel. But the complexity analysis of diagnostic planning (the term that has been only recently coined in (BMS00)) that we present now is novel.

Theorem 9. *The computational complexity of diagnostic planning is* **PSPACE**-*complete.*

References

C. Baral, A. Gabaldon, and A. Provetti, "Formalizing Narratives using nested circumscription", *Artificial Intelligence*, 1998, Vol. 104, No. 1/2, pp. 107–164.

C. Baral, M. Gelfond, and A. Provetti, "Representing actions: Laws, Observations and Hypotheses", *Journal of Logic Programming*, 1997, Vol. 31, No. 1–3, pp. 201–243.

C. Baral, V. Kreinovich, and R. Trejo, "Computational Complexity of Planning and Approximate Planning in Presence of Incompleteness", *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence IJCAI'99*, Stockholm, Sweden, July 31 – August 6, 1999, Vol. 2, pp. 948–953 (full paper to appear in *Artificial Intelligence*).

C. Baral, S. McIlraith, and T. Son, "Formulating diagnostic problem solving using an action language with narratives and sensing", *Proceedings of KR'2000* (to appear).

T. Bylander, "The computational complexity of propositional STRIPS planning", *Artificial Intelligence*, 1994, Vol. 69, pp. 161–204.

K. Erol, D. Nau, and V. S. Subrahmanian, "Complexity, decidability and undecidability results for domain-independent planning", *Artificial Intelligence*, 1995, Vol. 76, No. 1/2, pp. 75–88.

M. Gelfond and V. Lifschitz, "Representing actions and change by logic programs", *Journal of Logic Programming*, 1993, Vol. 17, No. 2–4, pp. 301–323.

P. Liberatore, "The complexity of the language $\mathcal{A}$", *Electronic Transactions on Artificial Intelligence*, 1997, Vol. 1, pp. 13–28.

M. Littman, "Probabilistic propositional planning: representations and complexity", *Proc. AAAI'97*, 1997, pp. 748–754.

C. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.

R. Reiter, "A theory of diagnosis from first principles", *Artificial Intelligence*, 1987, Vol. 32, No. 1, pp. 57–95.

J. Rintanen, "Constructing conditional plans by a theorem prover", *Journal of AI Research*, 1999, Vol. 10, pp. 323–352.

H. Turner, "Representing actions in logic programs and default theories", *Journal of Logic Programming*, 1997, Vol. 31, No. 1–3, pp. 245–298.