On the Compilability of Diagnosis, Planning, Reasoning about Actions, Belief Revision, etc.

Paolo Liberatore
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma - Italy
Email: liberato@dis.uniroma1.it
WWW: http://www.dis.uniroma1.it/∼liberato/

Abstract

In this paper we investigate the usefulness of preprocessing part of the input of a given problem to improve the efficiency. We extend the results of [Cadoli *et al.*, 1996] by giving sufficient conditions to prove the unfeasibility of reducing the on-line complexity via an off-line preprocessing. We analyze the problems of diagnosis [Peng and Reggia, 1986], planning [Bylander, 1991], reasoning about actions [Gelfond and Lifschitz, 1993], and belief revision [Williams, 1994]. We analyze other problems from various fields.

1 INTRODUCTION

Many problems in Artificial Intelligence (and in Knowledge Representation in particular) are computationally hard to solve. An observation that may lead to a reduction of the complexity is that often these problems have a property: the input is divided in two parts, where one of them is known long before the rest of the input. In such cases, it makes sense to do some computation on the first part of the input to speed-up the solution of the problem when the second part arrives. This computation made in advance is said *preprocessing*, or *compilation*. Another case in which a preprocessing is useful is when there are many instances of the problem to be solved that share part of the input (i.e. there are many instances for which part of the input is common). In such cases, the preprocessing of the shared part is useful if it decreases the cost of solving each single instance. We call the part of the input we preprocess *fixed part*, while the rest of the input is called *varying part*. When the complexity of the problem decreases thanks to the compilation of the fixed part, we call the problem *compilable*.

The general idea of preprocessing part of the input data has been widely used in many areas of computer science. For example, in computational geometry the techniques for geometric searching (cf. [Preparata and Shamos, 1985, Chap. 2]) use a costly preprocessing phase to speed up the on-line behavior.

In AI, a clear example of a problem with a fixed/varying part pattern occurs in the diagnosis field. Suppose we have a set of possible faults, and a set of rules that, given a fault, determine which (observable) effects will occur. In diagnosis of physical systems, the set of faults is the set of possible malfunctioning parts, while the effects are the (observable) abnormal behaviors of the system. The problem is to determine a minimal explanation (a minimal set of faults) that explains a given set of effects. It is reasonable to assume that the physical system (and its description in terms of faults-effects rules) is known in advance, and thus a preprocessing of it is in many case affordable (even if it is very expensive), in order to obtain a more efficient algorithm to find a diagnosis when a set of effects occurs.

Recently, several papers [Cadoli *et al.*, 1995, Gogic *et al.*, 1995, Kautz and Selman, 1992] focused on techniques to prove that problems can (or cannot) be compiled. The precise formalization of compilability is given in [Cadoli *et al.*, 1996].

What's new. In this paper, we give new sufficient conditions to prove that problems are not compilable, and apply them to the problems of diagnosis [Peng and Reggia, 1986], planning [Bylander, 1991], and reasoning about actions [Gelfond and Lifschitz, 1993]. Some problems of belief revision [Williams, 1994] will be proved to be compilable.

We also show the generality of these sufficient conditions by applying them to problems from database theory [Chandra and Merlin, 1977], network analysis

and design [Garey and Johnson, 1979], software engineering [Ntafos and Hakimi, 1979], and graph theory [Karp, 1972].

Outline. The paper is organized as follows: in the next section, we report the basic concepts of compilability, as defined in [Cadoli *et al.*, 1996]. In Section 3 we show some new sufficient conditions to prove that a problem is not compilable. The other sections contain the analysis of various problems from the point of view of compilability.

2 DEFINITIONS

In this paper we consider propositional formulas, (directed) graphs, and languages. We assume that the basic definitions of these concepts are known. Problems are represented by languages (as usual) over an alphabet Σ. The *length* of a string $x \in \Sigma^*$ is denoted $||x||$. The problem of diagnosis, which we will use to illustrate the rationale of compilation, is defined as follows.

Definition 1 *The problem Minimal Diagnosis (*md*) is: given a set of faults F, a set of effects M, a function $e : F \to \mathcal{P}(M)$, a set of observed effects $M' \subseteq M$, and an integer k, decide if there exists a set $F' \subseteq F$ of size at most k, such that $\bigcup_{f \in F'} e(f) \supseteq M'$.*

We illustrate this definition with an example that will be also useful for explaining the concepts of compilability.

Example 1 *A car may overheat for (at least) four causes: the water pump is broken, the sensor is broken, the relay does not work, or there is not enough water in the radiator. Each of these faults generates an overheating. They may have other effects: if the water pump is broken, the radiator is not heat, while a fault in the sensor or the relay causes the fan not to turn on. On the contrary, the only effect of not having enough water in the radiator is the overheating*[1]*.*

The set of faults and manifestations are defined as follows.

$$\begin{aligned} F &= \{\texttt{pump}, \texttt{sensor}, \texttt{relay}, \texttt{water}\} \\ M &= \{\texttt{overheat}, \texttt{fan\_off}, \texttt{no\_heat\_radiator}\} \end{aligned}$$

The function e, expressing the effects of faults, is defined as follows.

$$\begin{aligned} e(\texttt{pump}) &= \{\texttt{overheat}, \texttt{no\_heat\_radiator}\} \\ e(\texttt{sensor}) &= \{\texttt{overheat}, \texttt{fan\_off}\} \\ e(\texttt{relay}) &= \{\texttt{overheat}, \texttt{fan\_off}\} \\ e(\texttt{water}) &= \{\texttt{overheat}\} \end{aligned}$$

If the observed effects are $M' = \{\texttt{overheat}, \texttt{fan\_off}\}$, then the minimal diagnoses are $\{\texttt{sensor}\}$ and $\{\texttt{relay}\}$. Thus, there exist diagnoses of size $k = 1$. On the contrary, if the observed effects are $M' = \{\texttt{overheat}, \texttt{fan\_off}, \texttt{no\_heat\_radiator}\}$, then the minimal diagnoses are $\{\texttt{pump}, \texttt{sensor}\}$ and $\{\texttt{pump}, \texttt{relay}\}$, thus there is no diagnosis of size less or equal than $k = 1$.

In this paper we are concerned with problems whose input can be divided in two parts. One part is accessible off-line, that is, we can preprocess it. This part is called fixed part. The rest of the input is called varying part. In the example of diagnosis, we can spend much time of computation on the description of the physical system, while once a set of effects occurs, we want to obtain a diagnosis very quickly (in real time, if possible).

Formally, problems whose input is composed of two parts can be represented by sets of pairs of strings $S = \{\langle x, y\rangle\}$, where the first element of each pair is the fixed part of the input. In the case of diagnosis, the first part of the input is the description of the system, while the varying part is the set of observed effects. Since the function e represent the description of the system, while M' and k depend on the observed effect, we express Minimal Diagnosis as a problem of pairs of strings as follows.

$$\begin{aligned} \mathsf{md} = \{\langle e, (M', k)\rangle \mid \exists F' \text{ of size at most } k, \\ \text{such that } \bigcup_{f \in F'} e(f) \supseteq M'\} \end{aligned}$$

We do not write F and M for the sake of simplicity, and assume that F and M are simply the domain and the co-domain of the function e.

We remark that "fixed" is different from a more common concept of computational complexity, "constant". In computational complexity, the word "constant", when refers to a string, means that the string will always be of small length. On the contrary, "fixed" means that the part of the input will be known to the preprocessor, while the varying part is not. A fixed part, in our definition, may be arbitrarily large. This holds both in cases in which the fixed part is known in advance and in cases in which many input instances share the same fixed part.

In order to formalize the concept of preprocessing, we recall the definitions of polysize functions, compilable

[1] Some of the facts reported here are true, but since we are not really expert about cars, the reader should not try to repair cars following such diagnostic rules.

classes and reductions. These concepts have been introduced in [Cadoli *et al.*, 1996], where more examples and motivations are provided. A function f is called *polysize* if there exists a polynomial p such that for all x it holds $||f(x)|| \leq p(||x||)$, that is, the *size* of the result of f is bounded by a polynomial in the size of x.

Our intuitive notion of compilability states that a problem S can be reduced into a simpler problem by preprocessing only its fixed part. Given a complexity class C, we introduce the class of problems compilable into C, denoted by $\rightsquigarrow$C. The class we mostly use is $\rightsquigarrow$P, which contains all the problems that become polynomial, once the preprocessing is done.

Definition 2 *A language of pairs of strings*[2] $S \subseteq \Sigma^* \times \Sigma^*$ *belongs to* $\rightsquigarrow$C *if and only if there exist a polysize function* f *and a language of pairs* S' *such that for all* $x, y \in \Sigma^*$ *we have that*

1. $\langle x, y\rangle \in S$ *iff* $\langle f(x), y\rangle \in S'$.
2. $S' \in C$.

Notice that no restriction is imposed on the *time* needed to compute f, but only on the size of the result. This definition can be represented in terms of a computing machine as in Figure 1.

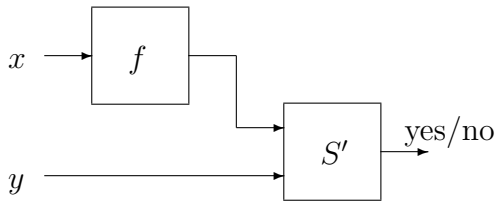

Figure 1: The $\rightsquigarrow$C machine.

This machinery captures our intuitive notion of compilability into C of a problem S with fixed and varying parts. We process off-line the fixed part x, obtaining $f(x)$. Now we can decide $\langle f(x), y\rangle \in S'$ with an algorithm in C. The whole process is convenient if the latter is easier than deciding $\langle x, y\rangle \in S$. In this paper we use mainly the class $\rightsquigarrow$P, that is the class of problems that can be solved in polynomial time after a preprocessing of the fixed part. The condition that f must be polysize is due to the fact that the fixed part x of the input may be arbitrarily large. If x were small, there would be no problem in having $f(x)$ of exponential size. Since x may be large, the condition ensures that the result of the preprocessing $f(x)$ will be of reasonable size.

Let us show how the idea of compilation can be used in the scenario of Example 1.

Example 2 *Consider the diagnosis problem of Example 1. The problem can be solved by considering each possible set* $F' \subseteq F$ *with size at most* k*, and verifying whether the effects of* F' *includes* M'*. In our example, we should check each subset of* $F = \{\texttt{pump}, \texttt{sensor}, \texttt{relay}, \texttt{water}\}$.

The complexity of the problem can be reduced via a compilation of the function e*. Since* e *is known much in advance, we can preprocess it. The result of the compilation is a table, whose rows are pairs (set of manifestations, minimal size of diagnosis). When applied to the problem of the car, the result of the compilation is the following.*

{overheat}	*1*
{fan_off}	*1*
{no_heat_radiator}	*1*
{overheat, fan_off}	*1*
{overheat, no_heat_radiator}	*1*
{fan_off, no_heat_radiator}	*2*
{overheat, fan_off, no_heat_radiator}	*2*

The table can be built by considering each possible $F' \subseteq F$ *and then calculating its effects. This is very inefficient, because we need to consider each possible* F'*. However, this is done during the compilation phase, and thus we are not very concerned about efficiency. When a set of manifestation occurs, the problem can be solved by a single check in the table. If the set of manifestations is for example* {overheat, fan_off}*, the size of the minimal diagnosis can be determined by looking up the table for a row whose first element is the set* {overheat, fan_off}*. Since the second element of the row is* 1*, there is a diagnosis of size* 1*.*

Note that the table has been calculated during the compilation phase, only knowing the function e*. The solution of a specific instance of the problem can then be determined very efficiently. In this case, we were able to build the table because* M *was small. In general, such a table requires* 2^m *rows, where* m *is the number of possible manifestations. While an exponential running time may be considered affordable during the compilation phase, it is always impossible to store an exponential-size table.*

So far, we have defined when a problem is compilable. What is missing is a way to prove that a problem *is not* compilable. Let us consider how this is done for

[2] In the sequel we omit the two words "of strings": in the sequel, languages of pairs are always languages of pairs of strings.

general problems (without a fixed/varying part pattern). The usual way to prove intractability (that is, the non-membership in the class P) of a problem is to give a polynomial reduction from an NP-hard problem to it. Since language of pairs of strings can also been seen as problems of strings, this definition can be specialized for languages of pairs.

Definition 3 *A polynomial reduction from a language of strings R to a language of pairs (of strings) S is a pair of polynomial functions $\langle r, h\rangle$ such that, for any $x \in \Sigma^*$ it holds $x \in R$ iff $\langle r(x), h(x)\rangle \in S$.*

This is the obvious specialization of the usual definition of polynomial reduction to the case in which the target language is a language of pairs. The following is the specialization of the definition of hardness. We use these definitions in the next section.

Definition 4 *A language of pairs S is C hard if any language R in C can be polynomially reduced to it.*

The NP hardness of a problem proves that the problem is not polynomial. However, the NP hardness of a problem of pairs S does not suffice to show its incompilability: an NP hard problem may be very well compilable into P. On the converse, the main aim of compilation is to solve NP hard problems in polynomial time, using a preprocessing of part of the input.

In order to obtain a class of reductions that preserve the compilability property and are powerful enough to allow the definition of complete problems, we introduce the notion of *comp reduction*:

Definition 5 *A* comp reduction *between two problems of pairs A and B is a triple $\langle f_1, f_2, g\rangle$ such that f_1 and f_2 are polysize functions, g is a polynomial function, and for every pair of strings $\langle x, y\rangle$ it holds*

$$\langle x, y\rangle \in A \quad \textit{iff} \quad \langle f_1(x), g(f_2(x), y)\rangle \in B$$

If there exists a comp reduction between two problems of pairs A and B we say that A is *comp reducible* to B.

Intuitively, A is comp reducible to B if 1) the fixed part of B can be obtained from the fixed part of A using a polysize function (f_1), and 2) the variable part of B can be constructed using both a polysize function (f_2) applied to the fixed part of A, and a polynomial-time function (g) applied to the variable part of A. These reductions satisfy all the basic properties of reductions. Indeed, if C is a class of the polynomial hierarchy (e.g. P, NP), we have that:

Theorem 1 *The comp reductions are transitive and compatible*[3] *with the class $\rightsquigarrow$C.*

Therefore, it is possible to define a notion of *completeness* for $\rightsquigarrow$C.

Definition 6 *Let B be a language of pairs. B is $\rightsquigarrow$C-hard iff all problems $A \in \rightsquigarrow$C are comp reducible to B.*

Using the above definition we can find hard problems. Nevertheless, for many natural problems the non-compilability proofs appeared in the literature [Kautz and Selman, 1992, Cadoli *et al.*, 1997, Cadoli *et al.*, 1995, Gogic *et al.*, 1995] cannot be rephrased as proofs of compNP hardness. For example, the problem of diagnosis introduced in the last section is not compNP hard. However, we can prove its incompilability (its non-membership in $\rightsquigarrow$P), using the non-uniform classes and reductions. For languages of strings, the non-uniform classes were introduced by Karp and Lipton [1980]. The extension for languages of pairs is given in [Cadoli *et al.*, 1996], where the non-uniform compilability classes are also compared with Karp and Lipton's classes.

Definition 7 *A language of pairs S belongs to $\|\!\rightsquigarrow$C (non-uniform $\rightsquigarrow$C) iff there exists a polysize function f and a language of pairs S' such that for all $\langle x, y\rangle$ we have that:*

1. *$\langle x, y\rangle \in S$ iff $\langle f(x, ||y||), y\rangle \in S'$;*
2. *$S' \in C$.*

In order to prove the non-compilability of problems, the concepts of nucomp reduction and hardness are defined.

Definition 8 *A nucomp reduction (non-uniform comp reduction) between two problems of pairs A and B is a triple $\langle f_1, f_2, g\rangle$ such that f_1 and f_2 are polysize functions, g is a polynomial function, and for each pair $\langle x, y\rangle$ it holds*

$$\langle x, y\rangle \in A \quad \textit{iff} \quad \langle f_1(x, ||y||), g(f_2(x, ||y||), y)\rangle \in B$$

We say that A is *nucomp reducible* to B if there exists a nucomp reduction from A to B. The definition of hardness follows.

Definition 9 *A problem of pairs B is said $\|\!\rightsquigarrow$C hard (non-uniform $\rightsquigarrow$C hard) if any problem in $\|\!\rightsquigarrow$C is nucomp reducible to it B.*

[3] For the definition of compatibility see [Johnson, 1990, pg. 79].

Now, from [Cadoli *et al.*, 1996] it follows that a $\|\not\rightsquigarrow$NP hard problem cannot be compiled into P, unless $\Pi_2^p = \Sigma_2^p$ (that is, unless the polynomial hierarchy collapses).

The concepts defined in this section might look complex. However, what is really needed for understanding the sequel of the paper can be summarized in the following small box.

S is in $\rightsquigarrow$P $\Rightarrow$ S can be compiled into P
S is $\|\not\rightsquigarrow$NP hard $\Rightarrow$ S cannot be compiled into P

A problem S can be compiled into P if it can be solved with a polynomial algorithm, after preprocessing part of the input.

The $\|\not\rightsquigarrow$NP hardness of a problem is the way to prove that a problem cannot be compiled into P. However, this requires to prove that there is a nucomp reduction from a previously proved $\|\not\rightsquigarrow$NP hard problem. This is not simple. The proofs of $\|\not\rightsquigarrow$NP hardness (the nucomp reductions) are usually complex and hard to find. In the next section we give some sufficient conditions to prove the $\|\not\rightsquigarrow$NP hardness of problems.

3 MONOTONIC POLYNOMIAL REDUCTIONS AND DIAGNOSIS

In this section we give new sufficient conditions for proving the $\|\not\rightsquigarrow$NP hardness of problems. These conditions are based on the concept of monotonic polynomial reductions, which are particular polynomial reductions from languages of strings to languages of pairs. The use of monotonic reductions greatly simplifies the proofs of incompilability. The proofs of $\|\not\rightsquigarrow$NP hardness that can be found in the literature [Cadoli *et al.*, 1995, Cadoli *et al.*, 1997] are often long and complex. We show in the sequel how the proofs based on monotonic reductions are instead very simple and intuitive.

A proof of NP hardness of a language S is usually a polynomial reduction from a (previously) proved NP hard problem R to S. Suppose, for example, that we have proven the NP hardness of a language of pairs S by means of a polynomial reduction from the problem 3sat. We state the following definition.

Definition 10 *A polynomial reduction $\langle r, h\rangle$ from* 3sat *to a language of pairs S is said to be monotonic if, for any two sets of clauses Π_1 and Π_2 over the same alphabet, with $\Pi_1 \subseteq \Pi_2$, it holds*

$$\langle r(\Pi_1), h(\Pi_1)\rangle \in S \quad \Leftrightarrow \quad \langle r(\Pi_2), h(\Pi_1)\rangle \in S \qquad (1)$$

It can be proved that, given a monotonic polynomial reduction, one can build a proof of $\|\not\rightsquigarrow$NP hardness for the problem S.

Theorem 2 *If there exists a monotonic polynomial reduction from* 3sat *to a problem of pairs S, then S is $\|\not\rightsquigarrow$NP hard.*

As a result, in order to prove that a problem S is not compilable, it suffices to find a polynomial reduction from 3sat to S, and then to prove that this reduction has the property of monotonicity (1). This is useful, because often there already exists a proof of hardness of the problem S that uses a reduction that can be easily proved to be monotonic.

The following theorem, and its proof, shows how this result can be applied to the problem of diagnosis. The inputs of the problem are the set of all the possible faults and effects F and M, the function e, the set of the observed effects M' and the number k. The set of possible effects, faults, and the function e are the fixed part (since it is determined by a static analysis of the system to be diagnosed), while the set of the current effect (that is, the abnormal behavior to be analyzed) and k are the varying part.

Theorem 3 *The problem* md *is $\|\not\rightsquigarrow$NP hard (and thus it is not in $\rightsquigarrow$P).*

Proof. The problem md has been proved to be NP hard via the following reduction from 3sat [Allemang *et al.*, 1987]. Without loss of generality, assume that the given set of clauses Π_1 contains all the clauses $x_i \vee \neg x_i$ for each x_i in the given alphabet. The set of faults F is equal to the set of literals of the alphabet, and the set of effects M is equal to the set of clauses. The function e is defined as

$$e(f) = \{\ m \mid \text{the clause } m \text{ contains the literal } f\ \}$$

Finally, the set of current effects M' is equal to M, and the number k is equal to the number of atoms in the considered alphabet.

Now, consider two sets of clauses Π_1 and Π_2 over the same alphabet, with $\Pi_1 \subseteq \Pi_2$. The instance $\langle r(\Pi_1), h(\Pi_1)\rangle$ is built as in the previous construction. The instance $\langle r(\Pi_2), h(\Pi_1)\rangle$ has new effects (the clauses in $\Pi_2 \backslash \Pi_1$), and the function e is modified accordingly. However, these new effects need not to be explained (since the set of the current effects is the same of the previous case), thus a diagnosis of size k exists for the second instance if and only if it exists for the first one. □

Note an interesting feature of this kind of proofs:

we do not need to prove that the given reduction works, that is, we do not need to prove that a set of clauses is satisfiable if and only if the corresponding diagnosis problem has a solution of size less or equal than k. This can be proved elsewhere (in this case, in [Allemang *et al.*, 1987]). All we have to do is to prove that this reduction is monotonic. This is proved just by comparing two similar instances of the problem of diagnosis (completely disregarding the instances of 3sat).

Of course, not all the proofs of NP hardness use a reduction from 3sat. Giving an appropriate definition of monotonicity of a reduction from the problems *node cover, clique, exact cover, three-exact cover*, one can prove that the existence of a monotonic reduction to a problem implies the $\|\!\rightsquigarrow$NP hardness of it.

Definition 11 *A reduction $\langle r, h\rangle$ from Node Cover (*NC*) or* clique *to a problem of pairs S is said to be monotonic if, for any two sets of edges $E_1 \subseteq E_2$ over the same set of nodes N, it holds*

$$\langle r((N, E_1), k), h((N, E_1), k)\rangle \in S \Leftrightarrow$$
$$\langle r((N, E_2), k), h((N, E_1), k)\rangle \in S$$

The monotonic reductions from NC and clique have the same property of the monotonic reductions from 3sat. Namely, they prove the $\|\!\rightsquigarrow$NP hardness of problems.

Theorem 4 *If there exists a monotonic polynomial reduction from in* NC *or* clique *to a problem of pairs B, then B is $\|\!\rightsquigarrow$NP hard (and thus it does not belong to $\rightsquigarrow$P, unless the polynomial hierarchy collapses).*

The above theorem is useful for reductions whose starting point is a problem on graphs. For problems of sets, we give the definition of monotonic reduction using the problem Exact Cover (ec).

Definition 12 *A reduction $\langle r, h\rangle$ from the problem Exact Cover (*ec*) or Three-Exact Cover (*x3c*) to a problem of pairs B is said to be monotonic if, for any two subsets S_1, S_2 of W, with $S_1 \subseteq S_2$, it holds*

$$\langle r(S_1), h(S_1)\rangle \in B \quad \Leftrightarrow \quad \langle r(S_2), h(S_1)\rangle \in B$$

These monotonic reductions have the same property of the other monotonic reductions.

Theorem 5 *If there exists a monotonic polynomial reduction from* ec *or* x3c *to a problem of pairs B, then B is $\|\!\rightsquigarrow$NP hard (and thus it does not belong to $\rightsquigarrow$P, unless the polynomial hierarchy collapses).*

4 PLANNING

STRIPS [Fikes and Nilsson, 1971] is a formalism for expressing planning problems. In the last years, it has been mainly used for studying the theoretical complexity of planning.

A STRIPS planning problem is a 4-tuple $\langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G}\rangle$, where $\mathcal{P}$ is a set of conditions, $\mathcal{O}$ is the set of operators (actions that can be performed to accomplish the goal), $\mathcal{I}$ is the initial state, and $\mathcal{G}$ is the goal.

The *conditions* are facts that can be true or false in the world of interest. A *state* S is a set of conditions, and represents the state of the world in a certain time point. The conditions in S are those representing facts that are true in the world, while those not in S represent facts currently false.

The *initial state* is a state, thus a set of conditions. The *goal* is specified by giving a set of conditions that should be achieved, and another set specifying which conditions should not be made true. Thus, a goal $\mathcal{G}$ is a pair $\langle \mathcal{M}, \mathcal{N}\rangle$, where $\mathcal{M}$ is the set of conditions that should be made true, while $\mathcal{N}$ is the set of conditions that should be made false.

The *operators* are actions that can be performed to achieve the goal. Each operator is a 4-tuple $\langle \phi, \eta, \alpha, \beta\rangle$, where ϕ, η, α, and β are sets of conditions. When executed, such an operator makes the conditions in α true, and those in β false, but only if the conditions in ϕ are currently true and those in η are currently false. The conditions in ϕ and η are called the positive and negative *preconditions* of the operator. The conditions in α and β are called the positive and negative *effects* or *postconditions* of the operator.

Given an instance of a STRIPS planning problem $\langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G}\rangle$, we define a plan for it as a sequence of operators that, when executed in sequence from the initial state, lead to a state where all the conditions in $\mathcal{M}$ are true and all those in $\mathcal{N}$ are false. More details about the definition of STRIPS can be found in [Fikes and Nilsson, 1971] and [Bylander, 1991].

The instances of a STRIPS planning problem have a clear fixed-varying part pattern. The set of conditions and operators are the fixed part of the input, since they represent the general description of the world. It is very likely that we need to solve many planning problems w.r.t. the same world. Consider, for example, a robot that delivers the mail on a floor of a building. After delivered the first mail, it is necessary to find a new plan to deliver the second one, etc. Even if we formalize the problem in such a way the goal is to deliver all the mail of the day, the day after there is a

need of a new plan, as the new mail to be delivered is different. This example shows that, given a general description of the world (conditions and operators), it is likely that there are many planning problems that differ only for the initial state and the goal.

As a result, STRIPS planning is a problem for which compilation may be useful. The fixed part of the input is composed of $\mathcal{P}$ and $\mathcal{O}$, where a compilation is allowed, while the varying part is composed of the initial state $\mathcal{I}$ and the goal $\mathcal{G}$.

The problem analyzed here is PLANSAT+: given a STRIPS instance such that all the operators have only positive postconditions, decide whether there exists a plan. This problem has been proved NP complete by Bylander [1991]. We prove that Bylander's proof is monotonic. First of all, a STRIPS problem is expressed as a language of pairs as

$$\mathsf{PLANSAT}+ = \{\langle\langle\mathcal{P},\mathcal{O}\rangle,\langle\mathcal{I},\mathcal{G}\rangle\rangle \mid \text{there exists a plan}\}$$

Bylander's reduction is from 3sat. Let Π_1 be a set of clauses, each composed of three literals, over the alphabet X. For each variable in X there are two conditions T_i and F_i. There is also a condition C_j for each clause in Π_1. For each variable there are two operators $\langle\emptyset,\{F_i\},\{T_i\},\emptyset\rangle$ and $\langle\emptyset,\{T_i\},\{F_i\},\emptyset\rangle$. There is also an operator for each literal of each clause. If the literal is x_i in the clause γ_j then the operator is $\langle\{T_i\},\emptyset,\{C_j\},\emptyset\rangle$, otherwise it is $\langle\{F_i\},\emptyset,\{C_j\},\emptyset\rangle$. The initial state is empty, while the goal is $\mathcal{G} = \langle\mathcal{M},\emptyset\rangle$, where $\mathcal{M} = \{C_j \mid \gamma_j \in \Pi\}$. As in the example of diagnosis, there is no need to prove that this reduction works. We trust Bylander.

Given two set of clauses of three literals Π_1 and Π_2 such that $\Pi_1 \subseteq \Pi_2$, the result of $r(\Pi_2)$ differs from $r(\Pi_1)$ only for the fact that there are new conditions C_j corresponding to the clauses $\gamma_j \in \Pi_2 \backslash \Pi_1$. Moreover, there are new operators with C_j as postcondition.

Let us compare $\langle r(\Pi_1), h(\Pi_1)\rangle$ and $\langle r(\Pi_2), h(\Pi_1)\rangle$. The new conditions have no effect on the existence of a plan, since a. they are not precondition of any operator, and b. they are not part of the goal. As a result, if a plan contains some of the new operators, then they can be removed from the plan. This reduced plan still leads to a state in which the goal is satisfied, because the new operators do not affect the preconditions of the old operators, neither they make true or false the conditions of the goal.

This proves that the given reduction is monotonic, thus the problem PLANSAT+ is $\|\!\rightarrow$NP hard, and thus it cannot be compiled into P.

5 REASONING ABOUT ACTIONS

In this paper we consider the language $\mathcal{A}$ which expresses domains of actions. A detailed description of $\mathcal{A}$ is given in [Gelfond and Lifschitz, 1993], where more examples are also provided. For the sake of completeness, we recall here the syntax of $\mathcal{A}$, with a short explanation of the semantics.

There are two nonempty sets of symbols, called fluent names and action names. Fluents are facts that can be true or false. They are called "fluents" because their value may change over time. A fluent expression is a fluent name possibly preceded by the negation symbol $\neg$. If F is a fluent expression and $A_1,\ldots,A_m$ are action names, then

$$F \text{ after } A_1;\ldots;A_m$$

is a value proposition. Such a proposition means that after the execution of the sequence of actions $A_1;\ldots;A_m$ the fluent expression F becomes true. If $m = 0$, the above statement is written

$$\text{initially } F$$

This expresses the value of a fluent in the initial state.

An effect proposition is an expression of the form

$$A \text{ causes } F \text{ if } P_1,\ldots,P_m$$

where A is an action name and $F, P_1,\ldots,P_m$ are fluent expressions. It means that the action A, when performed, makes the fluent expression F true, if the fluent expressions $P_1,\ldots,P_m$ are true. The fluents $P_1,\ldots,P_m$ are called preconditions, while F is the effect or postcondition of the proposition.

A domain description is a set of value and effect propositions.

A model of a domain description is a pair (initial state, transition function), where the transition function is a function that expresses how the states change in response to actions. We do not formally give the semantics of $\mathcal{A}$ here. The problem of interest is the entailment: $D \models V$ holds if and only if all the models of D are also models of V, where D is a domain description and V is a single value proposition.

The problem of entailment $D \models V$ in $\mathcal{A}$ is proved to be coNP complete in [Liberatore, 1997b]. Since a single domain description D may be queried many times w.r.t. several different propositions V, it makes sense to compile D once, if this simplifies the problem. Thus, it is worthwhile to analyze entailment in $\mathcal{A}$ w.r.t. our framework of compilation: the domain description D is the fixed part of the input, while V is the varying part.

For the sake of simplicity, in this paper we only consider NP problems. This does not prevent us from analyzing the problem of entailment in $\mathcal{A}$. Indeed, $D \models F$ after $A_1; \ldots; A_m$ if and only if $D \cup \{\neg F \text{ after } A_1; \ldots; A_m\}$ is not consistent, and consistency in $\mathcal{A}$ is NP complete. As a result, if we can show that the problem of consistency of $D \cup \{V\}$ (where V is a value proposition) is compilable (where D is the fixed part) then the problem of entailment is compilable, and vice versa. Indeed, what we prove is that the problem of deciding the consistency of a domain description with a value proposition is not compilable. In terms of languages of pairs, the problem can be formalized as:

$$\mathcal{A}\mathsf{sat} = \{\langle D, V\rangle \mid D \cup \{V\} \text{ is consistent}\}$$

The reduction that proves NP hard the problem of consistency given in [Liberatore, 1997b] is not monotonic. Consider the two set of clauses $\Pi_1 = \{x\}$ and $\Pi_2 = \{x, \neg x\}$ which are built on the same alphabet. We have

$$\begin{aligned} r(\Pi_1) &= \{ \text{ initially } \neg F,\ A \text{ causes } F \text{ if } x\} \\ r(\Pi_2) &= \{ \text{ initially } \neg F,\ A \text{ causes } F \text{ if } x, \\ &\qquad A \text{ causes } F \text{ if } \neg x\} \\ h(\Pi_1) &= \neg F \text{ after } A \end{aligned}$$

Thus $r(\Pi_1) \cup \{h(\Pi_1)\}$ is satisfiable while $r(\Pi_2) \cup \{h(\Pi_1)\}$ is not. As a result, the reduction is not monotonic.

We give now a monotonic reduction. Let Π_1 be a set of clauses, each composed by three literals. The functions r and h are defined as follows. For each clause $\gamma_i = l_{i_1} \vee l_{i_2} \vee l_{i_3}$ there is an action A_i.

$$\begin{aligned} r(\Pi_1) &= \{ \text{ initially } \neg F\} \cup \\ &\qquad \bigcup_{l_{i_1} \vee l_{i_2} \vee l_{i_3} \in \Pi_1} \{A_i \text{ causes } F \text{ if } \neg l_{i_1}, \neg l_{i_2}, \neg l_{i_3}\} \\ h(\Pi_1) &= \neg F \text{ after } A_1; \ldots; A_n \\ &\qquad \text{where } \Pi_1 = \{\gamma_1, \ldots, \gamma_n\} \end{aligned}$$

This is indeed a reduction from 3sat to $\mathcal{A}$sat.

Theorem 6 *The set Π_1 is satisfiable if and only if $r(\Pi_1) \cup \{h(\Pi_1)\}$ is consistent.*

We can also prove that the given reduction is monotonic. Let Π_2 be a set of clauses over the same alphabet of Π_1, and such that $\Pi_1 \subseteq \Pi_2$. The function r on Π_2 gives:

$$\begin{aligned} r(\Pi_2) &= \{ \text{ initially } \neg F\} \cup \\ &\qquad \bigcup_{\gamma_i = l_{i_1} \vee l_{i_2} \vee l_{i_3} \in \Pi_2} \{A_i \text{ causes } F \text{ if } \neg l_{i_1}, \neg l_{i_2}, \neg l_{i_3}\} \end{aligned}$$

That is, there are some new effect propositions: for each $\gamma_i \in \Pi_2 \backslash \Pi_1$ there is a proposition A_i causes F if $\neg l_{i_1}, \neg l_{i_2}, \neg l_{i_3}$.

However, these new actions do not affect the consistency, since a. there is only one effect proposition for each A_i, and b. there is no value proposition in which A_i appears. In semantical terms, such new effect propositions only modify the transition function but not the initial state. This implies that the consistency is not affected.

In this case, we needed to find a monotonic reduction. However, we did it in two steps: first we found the reduction, and afterwards we proved that it is monotonic. These steps are easier than finding a nucomp reduction.

6 BELIEF REVISION

So far we proved only "negative" results, that is, we showed problems for which a compilation does not give a gain in efficiency. In this section we analyze the problem of iterated belief revision. Due to the lack of space, we omit the definitions and the proofs. The basic model we refer to is that of ordinal conditional functions, as can be found, for example, in [Williams, 1994]. The results presented here can be easily extended to other revision operators [Boutilier, 1993, Nayak, 1994, Spohn, 1988, Nayak and Foo, 1997].

The formalism is the following: there is a current knowledge base K and an ordering of plausibility (the ordinal conditional function) that associates a number to each interpretation. This knowledge base K has to be revised with a sequence of revising formulas[4] $P_1, \ldots, P_m$. The problem we are interested in is to determine whether K, revised according to $P_1, \ldots, P_m$, implies another formula Q. This problem is Δ_2^p complete [Liberatore, 1997a].

The result of the revision may be queried many times w.r.t. several formulas Q. As a result, it is worthwhile to compile K and $P_1, \ldots, P_m$ if this allows for more efficient algorithms.

We can prove that, given K and $P_1, \ldots, P_m$, there exists a polynomial size formula K' such that K, revised with $P_1, \ldots, P_m$, implies Q if and only if $K' \models Q$

[4] Technically, we assume that the plausibility of each of these formulas is 1.

This is a proof of compcoNP membership of the problem. We can determine K' from K and $P_1, \ldots, P_m$, during the compilation, and then the problem reduces to $K' \models Q$, which is in coNP. As a result, the compilation reduces the complexity from Δ_2^p to coNP. Note that it is impossible to do any better, since deciding $\models Q$ is already a coNP complete problem (this argument can be used to prove that the problem is compcoNP complete).

7 OTHER APPLICATIONS

In this section we show that several problems from various fields cannot be made tractable by preprocessing part of the input.

7.1 STEINER TREE

The Steiner Tree problem is defined as: given a graph G whose edges are labeled with integers, a number k and a set of nodes N', decide whether G has a subtree of weight less or equal than k that contains all the nodes of N'.

This is a classical problem of graph theory, and is used to formalize problems of deciding whether a set of points can be connected without exceeding a certain cost. For example, in network design, one wants to know if a set of nodes in a network will remain connected with a given probability (given the probability of failure of the edges). The graph and the weight of the edges are in general fixed (they depends on the structure of the network), while one may ask about the reliability of the network w.r.t. many possible sets of nodes. Thus, the variable part is the set of nodes that the tree must connect.

7.2 NETWORK FLOW

This problem is defined as: given a graph and a collection of disjoint source-sink pairs $\{(s_1, t_1), \ldots, (s_k, t_k)\}$, decide if there exists a set of k disjoint paths, each from s_i to t_i.

This problem deals with the ability of a network to support a given traffic. The set of source-sink pairs represents the set of nodes that want to communicate, and thus they are the varying part of the problem.

7.3 REQUIRED PAIRS

The definition of this problem is: given a graph G, a set of pairs $\{(s_1, t_1), \ldots, (s_l, t_l)\}$, two nodes s and t and an integer k, decide if there exist a set of k paths from s to t such that for any pair (s_i, t_i) there is a path containing both s_i and t_i.

This problem comes from program testing. To certify a program, one must test all the execution sequences in the program. Even for small programs, the number of them can be extremely large. A compromise is to test only a subset of the execution sequences. Asking that a pair of nodes are together in at least one path is equivalent to ask for a test set that ensure that two segments of code interact in the correct way (see [Ntafos and Hakimi, 1979] for a more detailed explanation). A given program must be tested many times, thus it would be useful to compile its structure in order to speed up the search of test sets.

7.4 HAMILTONIAN CYCLE

The classical problem of Hamiltonian Cycle is: given a graph, decide if there exists a cycle that contains each node exactly once. Here we consider the problem in which we request that the cycle contains exactly a subset of nodes $N' \subseteq N$.

The problem Hamiltonian Cycle is a formalization of problems in which one must visit a set of points, minimizing the total distance traveled. One can hardly see a fixed and a varying part in this problem. However, the variant in which only a subset of nodes must be reached is a typical example of a problem with a fixed and a varying part: the positions of the points (and thus the underlying graph) is fixed, while the set of points that must visited may change from time to time. An example is that of the traveling salesman who has to visit a subset of the cities, or a postman who has to deliver mail only to a subset of people that live in a city. If this is the case, one can afford a long preprocessing time on the graph, if this make easier the finding of the Hamiltonian Cycle.

7.5 CONJUNCTIVE QUERY

Given a conjunctive query in the relational calculus, and a set of relations, the problem is to decide whether the query is true.

This is the decision problem derived from a problem that occurs in relational database systems: given a query, find the tuples that satisfies it. In practical settings, there is a set of typical queries that occur often, while the specific database is not known in advance. In this case it makes sense to compile the query in order to allow a fast answering.

Note that Theorem 7, proving the incompilability of this problem, is not in contrast to a result well known

to databases researchers, that proves that Conjunctive Query can be solved in logarithmic space (thus in polynomial time) if the query is constant. This is a good opportunity to remark the difference between "constant" and "fixed". When we say that part of the input data is constant, we mean that this part will be small, or that we are not interested in analyzing the complexity when the size of it becomes large. Instead, when part of the input is fixed, we mean that it can be arbitrarily large but we can afford a long preprocessing on it, because either it is known in advance, or there will be many input instances with the same fixed part.

7.6 SUBGRAPH ISOMORPHISM

The problem of subgraph isomorphism is defined as: given two graphs G and H, decide if there exists a subgraph of G that is isomorphic to H.

This problem is not compilable, either if the fixed part is the "big" graph, or the "small" one. We conjecture that the graph isomorphism problem (given two graphs, decide if they are isomorphic) is compilable into P.

Theorem 7 *The problems: Steiner Tree, Network Flow, Required Pairs, Sub-Hamiltonian Cycle, Conjunctive Query, and Subgraph Isomorphism are* $\|\!\rightsquigarrow$NP *hard, and thus they cannot be compiled into* P *(unless the polynomial hierarchy collapses).*

8 CONCLUSIONS

In this paper we have shown how the framework of compilation can be used for many problems of Artificial Intelligence. Indeed, we shown that it is theoretically impossible to reduce the complexity of diagnosis, planning, and reasoning about actions via a precompilation of part of the input. A positive result is the reduction from Δ_2^p to coNP of iterated belief revision.

The negative results are obtained by employing the concept of monotonic polynomial reduction. The technique of monotonic polynomial reductions is useful for proving the non-compilability of problem. We proved the generality of the method by applying it to some problems coming from various fields.

Acknowledgments

Many thanks to Marco Schaerf for revising a previous draft of this paper. Part of the work reported here has been done while the author was visiting AT&T Bell Labs, Murray Hill, NJ. Thanks to Henry Kautz for his support. This work has been partially supported by MURST, ASI, and CNR grants.

References

[Allemang *et al.*, 1987] D. Allemang, M. C. Tanner, T. Bylander, and J. R. Josephson. Computational complexity of hypothesis assembly. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 1112–1117, 1987.

[Boutilier, 1993] C. Boutilier. Revision sequences and nested conditionals. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 519–525, 1993.

[Bylander, 1991] T. Bylander. Complexity results for planning. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 274–279, 1991.

[Cadoli *et al.*, 1995] M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. The size of a revised knowledge base. In *Proceedings of the Fourteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS-95)*, pages 151–162, 1995. Extended version available as Technical Report DIS 34-96, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", November 1996.

[Cadoli *et al.*, 1996] M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. Feasibility and unfeasibility of off-line processing. In *Proceedings of the Fourth Israeli Symposium on Theory of Computing and Systems (ISTCS-96)*, pages 100–109. IEEE Computer Society Press, 1996. URL = http://www.dis.uniroma1.it/PUB/AI/papers/cado-etal-96.ps.gz.

[Cadoli *et al.*, 1997] M. Cadoli, F. M. Donini, M. Schaerf, and R. Silvestri. On compact representations of propositional circumscription. *Theoretical Computer Science*, 182:183–202, 1997.

[Chandra and Merlin, 1977] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Fifth ACM Symposium on Theory of Computing (STOC-77)*, pages 77–90, 1977.

[Fikes and Nilsson, 1971] R. Fikes and N. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Garey and Johnson, 1979] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to*

the Theory of NP-Completeness. W.H. Freeman and Company, San Francisco, Ca, 1979.

[Gelfond and Lifschitz, 1993] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.

[Gogic *et al.*, 1995] G. Gogic, H. A. Kautz, C. Papadimitriou, and B. Selman. The comparative linguistics of knowledge representation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 862–869, 1995.

[Johnson, 1990] D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.

[Karp and Lipton, 1980] R. M. Karp and R. J. Lipton. Some connections between non-uniform and uniform complexity classes. In *Proceedings of the Twelfth ACM Symposium on Theory of Computing (STOC-80)*, pages 302–309, 1980.

[Karp, 1972] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. New York: Plenum, 1972.

[Kautz and Selman, 1992] H. A. Kautz and B. Selman. Forming concepts for fast inference. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 786–793, 1992.

[Liberatore, 1997a] P. Liberatore. The complexity of iterated belief revision. In *Proceedings of the Sixth International Conference on Database Theory (ICDT-97)*, pages 276–290, 1997.

[Liberatore, 1997b] P. Liberatore. The complexity of the language $\mathcal{A}$. *Electronic Transactions on Artificial Intelligence*, 1(1–3):13–37, 1997. Available at http://www.ep.liu.se/ea/cis/1997/006/.

[Nayak and Foo, 1997] A. Nayak and N. Foo. Reasoning without minimality. In *IJCAI-97 Workshop on Nonmonotonic Reasoning, Action and Change*, pages 127–138, 1997.

[Nayak, 1994] A. Nayak. Iterated belief change based on epistemic entrenchment. *Erkenntnis*, 41:353–390, 1994.

[Ntafos and Hakimi, 1979] S. Ntafos and S. Hakimi. On path cover problems in digraphs and applications to program testing. *IEEE Transaction on Software Engigneering*, SE-5:520–529, 1979.

[Peng and Reggia, 1986] Y. Peng and J. Reggia. Plausibility of diagnostic hypothesis. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 140–145, 1986.

[Preparata and Shamos, 1985] F. P. Preparata and M. I. Shamos. *Computational Geometry: An introduction*. Springer-Verlag, 1985.

[Spohn, 1988] W. Spohn. Ordinal conditional functions: A dynamic theory of epistemic states. In *Causation in Decision, Belief Change, and Statistics*, pages 105–134. Kluwer Academics, 1988.

[Williams, 1994] M. Williams. Transmutations of knowledge systems. In *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 619–629, 1994.

A PROOFS

In this appendix we report the proofs of two of the theorems stated in the text. The first is Theorem 2, which state that the existence of a monotonic reduction from 3sat to a problem implies its $\|\!\rightarrowtail$NP hardness. Then, we prove the $\|\!\rightarrowtail$NP hardness of the Steiner Tree problem.

Theorem 2 *If there exists a monotonic polynomial reduction from* 3sat *to a problem of pairs S, then S is* $\|\!\rightarrowtail$NP *hard.*

Proof. Suppose there exists a monotonic polynomial reduction $\langle r, h\rangle$ from 3sat to B. We prove that there exists a nu-comp reduction f_1, f_2, g from the problem $*$3sat (which is compNP hard: see [Cadoli *et al.*, 1996]) to B.

Let $||\Pi||$ denote the size of Π (that is, the size of the string used to represent it), and $Var(\Pi)$ be the number of atoms in it. Furthermore Π_m is the set of all the clauses of three literals over a set of variables $\{x_1, \ldots, x_m\}$.

Let f_1, f_2, g be defined as follows

$$\begin{aligned} f_1(s,m) &= r(\Pi_m) \\ f_2(s,m) &= \epsilon \\ g(a,\Pi) &= h(\Pi \cup \{x_{Var(\Pi)+1} \vee \neg x_{Var(\Pi)+1}, \ldots, \\ &\qquad x_{||\Pi||} \vee \neg x_{||\Pi||}\}) \end{aligned}$$

Let $\Pi' = \Pi \cup \{x_{Var(\Pi)+1} \vee \neg x_{Var(\Pi)+1}, \ldots, x_{||\Pi||} \vee \neg x_{||\Pi||}\}$. Using the fact that $\langle r, h\rangle$ is a monotonic polynomial reduction we have that

$$\begin{array}{lll} \langle x, \Pi\rangle \in *\mathsf{3sat} & \text{iff} & \Pi \text{ is satisfiable} \\ & \text{iff} & \Pi' \text{ is satisfiable} \\ & \text{iff} & \langle r(\Pi'), h(\Pi')\rangle \in B \\ & \text{iff} & \langle r(\Pi_{Var(\Pi')}), h(\Pi')\rangle \in B \\ & \text{iff} & \langle r(\Pi_{||\Pi||}), h(\Pi')\rangle \in B \end{array}$$

But $r(\Pi_{||\Pi||}) = f_1(x, ||\Pi||)$ and $h(\Pi') = g(a, \Pi)$, thus $\langle x, \Pi\rangle \in *\mathsf{3sat}$ if and only if $\langle f_1(x, ||\Pi||), g(f_2(x, ||\Pi||), \Pi)\rangle \in B$. □

Theorem 8 *Steiner Tree (*st*) is* $\|\!\rightarrow$NP *hard.*

Proof. The reduction given by Karp in A.D. 1972 is not monotonic. The proof is by means of a reduction from Node Cover. The problem is that the instance of the Node Cover problem must be a graph with labels on edges, and this is not allowed by Theorem 4. However, with few changes the proof can be made monotonic.

Let $S = \{S_1, \ldots, S_m\}$ be a generic instance of the problem x3c. Let $\bigcup S = \{u_1, \ldots, u_n\}$. The corresponding instance of the problem st is

$$\begin{array}{rcl} N & = & \{n_0\} \cup S \cup \{\langle u_i, S_j\rangle \mid u_i \in S_j\} \\ E & = & \{\langle n_0, S_j\rangle\} \cup \{\langle S_j, \langle u_i, S_j\rangle\rangle\} \cup \\ & & \{\langle\langle u_i, S_j\rangle, \langle u_i, S_z\rangle\rangle\} \\ w & : & w(\langle n_0, S_j\rangle) = 3ln \\ & & w(\langle S_j, \langle u_i, S_j\rangle\rangle) = 0 \\ & & w(\langle\langle u_i, S_j\rangle, \langle u_i, S_z\rangle\rangle) = 1 \\ N' & = & \{n_0\} \cup \{\langle u_i, S_j\rangle\} \\ k & = & ln^2 + \sum(p(u_i) - 1) \end{array}$$

where l is the number of all the possible subsets of three elements of a set of m elements, and $p(u_i)$ is the number of sets in which u_i is present.

Suppose there exists a subset $S' \subseteq S$ such that $\bigcup S' = \bigcup S$ and $S_j \cap S_z = \emptyset$ for each $S_j, S_z \in S'$. A Steiner tree in the graph above is the following.

$$\begin{array}{rcl} T & = & \{\langle n_o, S_j\rangle \mid S_j \in S'\} \cup \{\langle S_j, \langle u_i, S_j\rangle\rangle \mid S_j \in S'\} \\ & & \cup\{\langle\langle u_i, S_j\rangle, \langle u_i, S_z\rangle\rangle \mid S_j \in S'\} \end{array}$$

It is easy to see that this tree has weight

$$3ln \cdot n/3 + \sum(p(u_i) - 1) = k$$

Suppose there exists a Steiner tree T of weight less or equal than k. We define

$$S' = \{S_j \mid \langle n_0, S_j\rangle \in T\}$$

First of all, we have $|S'| \leq n/3$: otherwise, the weight of the tree would be greater than k, as one can easily prove.

We have also $\bigcup S' = \bigcup S$. The node n_0 is the root of the tree (we recall that the graph is direct). Given an element u_i, each node $\langle u_i, S_j\rangle$ must be reached by the tree, thus it must be the case that

$$\begin{array}{ll} & \langle n_0, S_j\rangle, \langle S_j, \langle u_i, S_j\rangle\rangle \in T \\ \text{or} & \langle n_0, S_z\rangle, \langle S_z, \langle u_i, S_z\rangle\rangle, \langle\langle u_i, S_z\rangle, \langle u_i, S_y\rangle\rangle, \ldots, \\ & \langle\langle u_i, S_x\rangle, \langle u_i, S_j\rangle\rangle \in T \end{array}$$

In the first case, $u_i \in S_j$ and $S_j \in S'$. In the second one, $u_i \in S_z$ and $S_z \in S'$. Thus in both cases $u_i \in \bigcup S'$.

Now, since $|S'| \leq n/3$ and $\bigcup S' = \bigcup S$, it must be also that the sets in S' are mutually disjoint.

The monotonicity of this reduction is not trivial to prove. Let S_2 be a set of sets of elements in $\{u_1, \ldots, u_n\}$ such that $S \subseteq S_2$. The instance $\langle r(S), h(S)\rangle$ is that of the previous construction. The instance $\langle r(S_2), h(S)\rangle$ has some new nodes S_z and $\langle u_i, S_z\rangle$ for each $S_z \in S_2 \backslash S$ and $u_i \in S_z$, and the corresponding edges. Since the weight of the tree and the set of nodes to reach are the same, it follows that a Steiner tree for the instance $\langle r(S), h(S)\rangle$ is also a Steiner tree for $\langle r(S_2), h(S)\rangle$.

The hard part of the proof is to show that if the instance $\langle r(S_2), h(S)\rangle$ has a Steiner tree of weight at most k, then the same holds for the instance $\langle r(S), h(S)\rangle$. This is proved by showing that the Steiner tree for $\langle r(S_2), h(S)\rangle$ does not contain the new nodes. Informally, the Steiner tree for $\langle r(S_2), h(S)\rangle$ may have at most $n/3$ nodes of the type S_z. Suppose that one of these nodes corresponds to a set $S_z \in S_2 \backslash S$. Since the tree must also contain all the nodes $\langle u_i, S_j\rangle$, it follows that each node $\langle u_i, S_z\rangle$ must be linked to all the corresponding nodes $\langle u_i, S_j\rangle$. This makes a total weight of $p(u_i)$. As a result, the total weight of the tree is greater than k. □