Isomorphisms Between STRIPS Problems and Sub-Problems

Martin C. Cooper
IRIT, University of Toulouse, France

Arnaud Lequen
IRIT, University of Toulouse, France

Frédéric Maris
IRIT, University of Toulouse, France

Abstract

Determining whether two STRIPS planning instances are isomorphic is the simplest form of comparison between planning instances. It is also a particular case of the problem concerned with finding an isomorphism between a planning instance P and a sub-instance of another instance P'. One application of such an isomorphism is to efficiently produce a compiled form containing all solutions to P from a compiled form containing all solutions to P'. In this paper, we study the complexity of both problems. We show that the former is GI-complete, and can thus be solved, in theory, in quasi-polynomial time. While we prove the latter to be NP-complete, we propose an algorithm to build an isomorphism, when possible. We report extensive experimental trials on benchmark problems which demonstrate conclusively that applying constraint propagation in preprocessing can greatly improve the efficiency of a SAT solver.

2012 ACM Subject Classification Computing methodologies → Planning for deterministic actions; Mathematics of computing → Matchings and factors

Keywords and phrases planning, isomorphism, complexity, constraint propagation

Digital Object Identifier 10.4230/LIPIcs.CP.2022.13

Supplementary Material The material to reproduce the experiments can be found here:
Software (Source Code): `https://github.com/arnaudlequen/PDDLIsomorphismFinder`

Funding This work was supported by the AI Interdisciplinary Institute ANITI, funded by the French program "Investing for the Future – PIA3" under Grant agreement no. ANR-19-PI3A-0004.

Acknowledgements The authors would like to thank the anonymous reviewers, whose insightful comments helped improve this paper.

1 Introduction

Models used for STRIPS [7] planning encode sizeable state-spaces that can rarely be represented explicitly, but that have a clear and somewhat regular structure. Parts of this structure can be, however, common to multiple planning instances, although this similarity is often far from immediate to identify by looking at the STRIPS representation. Indeed, finding whether or not an instance P is a sub-instance of another problem P' requires to map every fluent and every operator of P to their counterpart in P', while respecting a morphism property. This requires the exploration of the exponential search space of mappings from P to P'. Finding such a mapping, however, allows us to carry over significant pieces of information from one problem to the other. In particular, any solution-plan for P can then be translated into a plan for P' efficiently.

A classical technique in constraint programming is to store all solutions to a CSP or SAT instance in a compact compiled form [1]. This is performed off-line. A compilation map indicates which operations and transformations can be performed in polynomial time during the on-line stage [6]. STRIPS fixed-horizon planning can be coded as a SAT instance using the classical SATPLAN encoding [9]. So, for a given instance, all plans can be stored in a compiled form, at least in theory. In practice, the compiled form will often be too large to be stored. Types of planning problems which are nevertheless amenable to compilation are those where the number of plans is small or, at the other extreme, there are few constraints on the order of operators. If we have a compiled form C' representing all solution-plans to an instance P' and we encounter a similar problem P, it is natural to ask whether we can synthesize a plan for P from C'. If P is isomorphic to a subproblem of P', then it suffices to apply a sequence of conditioning operations to C' to obtain a compiled form C representing all solutions to P. This is our main motivation for studying isomorphisms between subproblems. A trivial but important special case occurs when C' is empty, i.e. P' has no solution. In this case, an isomorphism from P' to a subproblem of P is a proof that P' has no solution.

In this paper, we first focus on problem SI, which is concerned with finding an isomorphism between two STRIPS instances of identical size. As we show that the problem is GI-complete, we prove that a quasi-polynomial time algorithm exists [2]. We then consider problem SSI, which is concerned with finding an isomorphism between a STRIPS instance and a subinstance of another STRIPS instance. We call such a mapping a *subinstance isomorphism*. After showing that this problem is NP-complete, we propose an algorithm that finds a subinstance isomorphism if one exists, or that detects that none exists. This algorithm is based on constraint propagation techniques, that allow us to prune impossible associations between elements of P and P', as well as on a reduction to SAT.

So far we have assumed that the two planning instances P and P' have the same initial states and goals (modulo the isomorphism). Even when this is not the case, an isomorphism from P to a subinstance of P' can still be of use. For example, if π is a solution-plan for P, then its image in P' can be converted to a single new operator which could be added to P' to facilitate its resolution. Such an operator would have the image of the initial state I of P for precondition, and the image of the result of the application of π to I for effect, thus abstracting away the application of the sequence of operators π. We therefore also consider this weaker notion of subinstance isomorphism, that we call *homogeneous subinstance isomorphism*, and the corresponding computational problem SSI-H.

Previous work investigated the complexity of various problems related to finding solution-plans for STRIPS planning instances [4], or focused on the complexity of solving instances from specific domains [8]. More scarcely, problems focused on altering planning models have been studied from a complexity theory point of view, such as the problem concerned with adapting a planning model so that some user-specified plans become feasible [10].

The paper is organized as follows. In Section 2, we introduce general notations, concepts and constructions that we use throughout this paper. In Section 3 and Section 4, we present our complexity results, for SI and SSI respectively. In Section 5, we present the outline of our algorithm for SSI. Section 6 is dedicated to the experimental evaluation and discussion.

2 Preliminaries

2.1 Automated Planning

A STRIPS planning problem is a tuple $P = \langle F, I, O, G \rangle$ such that F is a set of *fluents* (propositional variables whose values can change over time), I and G are sets of literals of F, called the *initial state* and *goal*, and O is a set of *operators*. Operators are of the form $o = \langle \mathsf{pre}(o), \mathsf{eff}(o) \rangle$. $\mathsf{pre}(o)$ and $\mathsf{eff}(o)$ are, respectively, the *precondition* and *effect* of o, which are sets of literals of F. We will denote $\mathsf{pre}^+(o) = \{f \in F \mid f \in \mathsf{pre}(o)\}$ the positive fluents of $\mathsf{pre}(o)$, and $\mathsf{pre}^-(o) = \{f \in F \mid \neg f \in \mathsf{pre}(o)\}$ the negative fluents. Similarly, $\mathsf{eff}^+(o) = \{f \in F \mid f \in \mathsf{eff}(o)\}$ and $\mathsf{eff}^-(o) = \{f \in F \mid \neg f \in \mathsf{eff}(o)\}$. By a slight abuse of notation, we will denote $\mathsf{pre} : O \to 2^F \cup 2^{\neg F}$ the function $o \mapsto \mathsf{pre}(o)$, and use similar notations for $\mathsf{pre}^+, \mathsf{pre}^-, \mathsf{eff}, \mathsf{eff}^+$, and eff^-. In the rest of this paper, we will note $\mathcal{C} = \{\mathsf{pre}^+, \mathsf{pre}^-, \mathsf{eff}^+, \mathsf{eff}^-\}$. We will also use the notation that, for any set S of literals of F, $\neg S = \{\neg l \mid l \in S\}$.

A state s is an assignment of truth values to all fluents in F. For notational convenience, we associate s with the set of literals of F which are true in s. Given an instance $P = \langle F, I, O, G\rangle$, a *solution-plan* is a sequence of operators $o_1, \ldots, o_k$ from O such that the sequence of states $s_0, \ldots, s_k$ defined by $s_0 = I$ and $s_i = (s_{i-1} \setminus \mathsf{eff}^-(o_i)) \cup \mathsf{eff}^+(o_i)$ $(i = 1, \ldots, k)$ satisfies $\mathsf{pre}^+(o_i) \subseteq s_{i-1}$, $\mathsf{pre}^-(o_i) \cap s_{i-1} = \varnothing$ $(i = 1, \ldots, k)$ and $G \subseteq s_k$. A *plan* is defined similarly but without the conditions concerning I and G.

2.2 Complexity Class GI

This section introduces the complexity class GI, for which SI is later shown to be complete. GI is built around the Graph Isomorphism problem, which consists in finding a bijection $u : V \to V'$ between the vertices of two graphs $\mathcal{G}(V, E)$ and $\mathcal{G}'(V', E')$, such that the images of vertices linked by an edge in $\mathcal{G}$ are also linked by an edge in $\mathcal{G}'$ (and vice-versa). Formally, we require that the following condition holds:

$$\{x, y\} \in E \text{ iff } \{u(x), u(y)\} \in E' \tag{1}$$

▶ **Definition 1.** *The complexity class GI is the class of problems with a polynomial-time Turing reduction to the Graph Isomorphism problem.*

Complexity class GI contains numerous problems concerned with the existence of an isomorphism between two non-trivial structures encoded explicitly. Such problems are often complete for the class: finding an isomorphism between colored graphs, hypergraphs, automata, etc. are GI-complete problems [15]. In particular, we later use the following result:

▶ **Proposition 2** ([15], Ch. 4, Sec. 15)**.** *The* Oriented Graph Isomorphism problem *is GI-complete.*

As with the Graph Isomorphism problem, an isomorphism between oriented graphs $\mathcal{G}(V, E)$ and $\mathcal{G}'(V', E')$ is a bijection $u : V \to V'$ such that $(x, y) \in E$ iff $((u(x), u(y)) \in E'$.

In this paper, we consider another category of structures, called Finite Model, defined below. Finite models are also such that the related isomorphism existence problem is GI-complete.

▶ **Definition 3.** *A Finite Model is a tuple $M = \langle V, R_1, \ldots, R_n\rangle$ where V is a finite non-empty set and each R_i is a relation on elements of V with a finite number of arguments.*

Let $M = \langle V, R_1, \ldots, R_n\rangle$ and $M' = \langle V', R'_1, \ldots, R'_n\rangle$ be two finite models. An isomorphism between M and M' is a bijection $u : V \to V'$ such that, for any $i \in \{1, \ldots, n\}$, for any set of elements $v_1, \ldots, v_m$ with m the arity of R_i, $R_i(v_1, \ldots, v_m)$ iff $R'_i(u(v_1), \ldots, u(v_m))$.

▶ **Proposition 4** ([15], Ch. 4, Sec. 15)**.** *The* Finite Model Isomorphism problem *is GI-complete.*

Class GI is believed to be an intermediate class between P and NP: the Graph Isomorphism problem can indeed be solved in quasi-polynomial time [2]. Although the problem is thought not to be NP-complete, no polynomial time algorithm is known.

2.3 Graph encodings into STRIPS

In this section, we present two ways to encode a graph $\mathcal{G} = (V, E)$ into a planning problem $P = \langle F, I, O, G\rangle$. These constructions are needed at various points in the rest of this paper, and only differ in that they take into account, or not, the orientation of the edges of $\mathcal{G}$. The intuition behind these constructions is that they model an agent that would move on the graph, resting on vertices and moving along edges. An agent being on vertex v would thus be denoted by the state $\{v\}$, where all fluents other than v are false.

In order to make the construction and resulting proofs simpler to read, for any pair $(v_s, v_t) \in F^2$, we will denote $\mathsf{move}(v_s, v_t)$ the operator that represents a movement from vertex v_s to vertex v_t. Keeping in mind that $F = V$, we have, more formally:

$$\mathsf{move}(v_s, v_t) = \langle \{v_s\}, \{v_t\} \cup \neg(V \setminus \{v_t\})\rangle$$

Where $\{v_s\}$ is the precondition of the operator, and $\{v_t\} \cup \neg(V \setminus \{v_t\})$ its effects. In the following construction, the vertices (resp. edges) of $\mathcal{G}$ are in bijection with the fluents (resp. operators) of P. In particular, we do not allow multi-edges. Other alternative constructions for move could have been used, as long as the encoding of each edge is unique. The one we propose is sufficient for our theoretical use, even though they encode trivial planning tasks.

▶ **Construction 5.** *Let $\mathcal{G} = (V, E)$ be an oriented graph. Let us build the planning problem $P_{\mathcal{G}} = \langle F, I, O, G\rangle$, where:*

$$F = V$$
$$O = \{\mathsf{move}(v_s, v_t) \mid (v_s, v_t) \in E\}$$
$$G = I = \varnothing$$

In the case of non-oriented graphs, the construction is essentially the same, except that moves are possible in both directions. This gives us the following definition:

▶ **Construction 6.** *Let $\mathcal{G} = (V, E)$ be a non-oriented graph. Let us build the planning problem $P_{\mathcal{G}} = \langle F, I, O, G\rangle$, where F, I and G are defined as in Construction 5, but where:*

$$O = \{\mathsf{move}(v_s, v_t), \mathsf{move}(v_t, v_s) \mid \{v_s, v_t\} \in E\}$$

3 STRIPS Isomorphism Problem

This section is concerned with the problem of finding an isomorphism between two STRIPS planning problems. After introducing the notion of isomorphism between STRIPS instances that we use throughout this paper, we formally introduce problem SI, and settle its complexity.

▶ **Definition 7** (Isomorphism between STRIPS instances)**.** *Let $P = \langle F, I, O, G\rangle$ and $P' = \langle F', I', O', G'\rangle$ be two STRIPS instances. An* isomorphism *from P to P' is a pair (υ, ν) of bijections $\upsilon : F \to F'$ and $\nu : O \to O'$ that respect the following three conditions:*

$$\forall o \in O,\ \nu(o) = \langle \upsilon(\mathsf{pre}(o)), \upsilon(\mathsf{eff}(o))\rangle \tag{2}$$
$$\upsilon(I) = I' \tag{3}$$
$$\upsilon(G) = G' \tag{4}$$

Where, by a slight abuse of notation, for any two sets F_1 and F_2 of fluents of F,

$$\upsilon(F_1 \cup \neg F_2) = \upsilon(F_1) \cup \neg\upsilon(F_2)$$

An immediate property of this definition is that it carries over all plans: any sequence $o_1, \ldots, o_n$ of operators of O is a plan for P if, and only if, the corresponding plan $\nu(o_1), \ldots, \nu(o_n)$ is a plan for P'. This homomorphism property is enforced by equation (2). Similarly, all solution-plans carry over, as enforced by the additional conditions defined in equations (3) and (4). We now introduce the problem SI formally, and analyze its complexity:

▶ **Problem 8.** *STRIPS Isomorphism problem SI*

Input: *Two STRIPS instances P and P'*

Output: *An isomorphism (υ, ν) between P and P', if one exists*

▶ **Proposition 9.** *SI is GI-complete*

The rest of this section is dedicated to the proof of this result. We first show the GI-hardness of the problem, and then that it belongs to GI.

▶ **Lemma 10.** *SI is GI-hard*

Proof. The proof consists in a reduction from the Oriented Graph Isomorphism problem to SI. Let $(\mathcal{G}, \mathcal{G}')$ be an instance of the Oriented Graph Isomorphism problem, where $\mathcal{G} = (V, E)$ and $\mathcal{G}' = (V', E')$. The proof relies on Construction 5, which gives us in polynomial time the STRIPS planning problems $P_{\mathcal{G}}$ and $P_{\mathcal{G}'}$.

We show that there exists an isomorphism $u : V \to V'$ between $\mathcal{G}$ and $\mathcal{G}'$ iff there exists an isomorphism (υ, ν) between $P_{\mathcal{G}}$ and $P_{\mathcal{G}'}$. The main idea consists in, first, identifying mappings u and υ, and second, showing that the morphism condition between the edges of graphs $\mathcal{G}$ and $\mathcal{G}'$ is enforced by the morphism condition on the operators of STRIPS instances $P_{\mathcal{G}}$ and $P_{\mathcal{G}'}$, and vice-versa.

($\Rightarrow$) Suppose that there exists a graph isomorphism $u : V \to V'$ between $\mathcal{G}$ and $\mathcal{G}'$, and let us show that there exists an isomorphism between $P_{\mathcal{G}}$ and $P_{\mathcal{G}'}$. We define the transformation ν on elements of O by $\nu(\langle \mathsf{pre}(o), \mathsf{eff}(o) \rangle) = \langle u(\mathsf{pre}(o)), u(\mathsf{eff}(o)) \rangle$. We will show that $\nu : O \to O'$ is well-defined and that the pair (u, ν) forms an isomorphism between $P_{\mathcal{G}}$ and $P_{\mathcal{G}'}$. For any $o \in O$, by construction, there exists a unique pair $(v_1, v_2) \in V^2$ such that $o = \mathsf{move}(v_1, v_2)$. Thus, we have that

$$
\begin{aligned}
o \in O \ \textit{iff}\ & (v_1, v_2) \in E \\
\textit{iff}\ & (u(v_1), u(v_2)) \in E' \\
\textit{iff}\ & \mathsf{move}(u(v_1), u(v_2)) \in O' \\
\textit{iff}\ & \nu(\mathsf{move}(v_1, v_2)) \in O' \\
\textit{iff}\ & \nu(o) \in O'
\end{aligned}
$$

The arguments from one line to the other stem from the construction of the various objects we use. Thus, we have shown that $P_{\mathcal{G}}$ and $P_{\mathcal{G}'}$ are isomorphic.

($\Leftarrow$) Suppose that $P_{\mathcal{G}}$ and $P_{\mathcal{G}'}$ are isomorphic, and that there exists an isomorphism (υ, ν) between them. We will show that there exists an isomorphism between $\mathcal{G}$ and $\mathcal{G}'$. By hypothesis, we have $\upsilon : F \to F'$ (or $\upsilon : V \to V'$) and $\nu : O \to O'$ two bijections.

In the following, we will denote by g and g' the bijections $g : E \to O$ and $g' : E' \to O'$, that exist by the construction (e.g. $g((v_1, v_2)) = \mathsf{move}(v_1, v_2)$).

Let us show that the function υ is a graph isomorphism between $\mathcal{G}$ and $\mathcal{G}'$. We have that, for any $e = (v_1, v_2) \in E$, $g(e) = \mathsf{move}(v_1, v_2) \in O$. So $\nu \circ g(e) = \nu(\mathsf{move}(v_1, v_2))$, and as such, $\nu \circ g(e) = \mathsf{move}(\upsilon(v_1), \upsilon(v_2)) \in O'$. Then, $g'^{-1} \circ \nu \circ g(e) = (\upsilon(v_1), \upsilon(v_2))$, but also $g'^{-1} \circ \nu \circ g(e) \in E'$. As a consequence, $(\upsilon(v_1), \upsilon(v_2)) \in E'$.

With similar arguments, as g, g' and ν are bijections, the converse can be shown. As a consequence, ν is a graph isomorphism between $\mathcal{G}$ and $\mathcal{G}'$. ◀

▶ **Lemma 11.** *SI is in GI.*

Proof. The proof follows a reduction from SI to the Finite Model isomorphism problem, as defined in Definition 3. It is based on the following construction: for any planning problem $P = \langle F, I, O, G\rangle$, we build the finite model M_P, such that:

$$M_P = \langle V, \mathcal{R}_F, \mathcal{R}_I, \mathcal{R}_O, \mathcal{R}_G, \mathcal{R}_{\mathsf{pre}^+}, \mathcal{R}_{\mathsf{pre}^-}, \mathcal{R}_{\mathsf{eff}^+}, \mathcal{R}_{\mathsf{eff}^-}\rangle$$
$$V = F \sqcup O$$
$$\text{For } X \in \{F, I, O, G\},\ \mathcal{R}_X = X$$
$$\text{For each } \mathcal{S} \in \mathcal{C},\ \mathcal{R}_\mathcal{S} = \big\{(o, f) \in V^2 \mid o \in O \textit{ and } f \in \mathcal{S}(o)\big\}$$

where $\mathcal{C} = \{\mathsf{pre}^+, \mathsf{pre}^-, \mathsf{eff}^+, \mathsf{eff}^-\}$. We will show that any two STRIPS planning problems P and P' are isomorphic iff M_P and $M_{P'}$ are isomorphic.

Let us denote $M_P = \langle V, \mathcal{R}_F, \dots, \mathcal{R}_{\mathsf{eff}^-}\rangle$ and $M_{P'} = \langle V', \mathcal{R}'_F, \dots, \mathcal{R}'_{\mathsf{eff}^-}\rangle$

($\Rightarrow$) Suppose that there exists an isomorphism (υ, ν) between P and P'. Define the mapping $g : V \to V'$ such that, for $x \in V$,

$$g(x) = \begin{cases} \upsilon(x) & \text{if } x \in F \\ \nu(x) & \text{if } x \in O \end{cases} \tag{5}$$

g is immediately a bijection, by hypothesis on (υ, ν). In addition, for $X \in \{F, I, O, G\}$, $\mathcal{R}_X(v)$ iff $\mathcal{R}'_X(g(v))$, by hypothesis on (υ, ν).

Let $o \in O$, $p \in F$. We have that, for any $\mathcal{S} \in \mathcal{C} = \{\mathsf{pre}^+, \mathsf{pre}^-, \mathsf{eff}^+, \mathsf{eff}^-\}$,

$$\begin{aligned} \mathcal{R}_\mathcal{S}(o, p) &\textit{ iff } o \in O \textit{ and } p \in \mathcal{S}(o) && (6)\\ &\textit{ iff } \nu(o) \in O' \textit{ and } \upsilon(p) \in \mathcal{S}(\nu(o)) && (7)\\ &\textit{ iff } \mathcal{R}'_\mathcal{S}(\nu(o), \upsilon(p)) \\ &\textit{ iff } \mathcal{R}'_\mathcal{S}(g(o), g(p)) \end{aligned}$$

The passage from (6) to (7) is by definition of the isomorphism. The other equivalences follow mostly by definition. This proves that M_P and $M_{P'}$ are isomorphic.

($\Leftarrow$) Suppose that M_P and $M_{P'}$ are isomorphic, and that $g : V \to V'$ is an isomorphism between the two models. Let us define $\upsilon = g_{|F}$ (resp. $\nu = g_{|O}$) the restriction of g on the subdomain F (resp. O). Clearly, we have that $\upsilon : F \to F'$, as otherwise there would exist an element $v \in V$ such that $\mathcal{R}_F(v)$ but *without* $\mathcal{R}'_F(g(v))$, violating the isomorphism hypothesis on g. Similarly, we have $\nu : O \to O'$. We have, as above, for any $o \in O$,

$$\begin{aligned} & o = \langle \mathsf{pre}(o), \mathsf{eff}(o)\rangle \\ \textit{iff } & \forall p \in F, \forall \mathcal{S} \in \mathcal{C}, p \in \mathcal{S}(o) \Leftrightarrow \mathcal{R}_S(o, p) && (8)\\ \textit{iff } & \forall p \in F, \forall \mathcal{S} \in \mathcal{C}, p \in \mathcal{S}(o) \Leftrightarrow \mathcal{R}'_\mathcal{S}(g(o), g(p)) && (9)\\ \textit{iff } & \forall p \in F, \forall \mathcal{S} \in \mathcal{C}, g(p) \in g(\mathcal{S}(o)) \Leftrightarrow \mathcal{R}'_\mathcal{S}(g(o), g(p)) && (10)\\ \textit{iff } & \forall p' \in F', \forall \mathcal{S} \in \mathcal{C}, p' \in g(\mathcal{S}(o)) \Leftrightarrow \mathcal{R}'_\mathcal{S}(g(o), p') && (11)\\ \textit{iff } & g(o) = \langle g(\mathsf{pre}(o)), g(\mathsf{eff}(o))\rangle && (12)\\ \textit{iff } & \nu(o) = \langle \upsilon(\mathsf{pre}(o)), \upsilon(\mathsf{eff}(o))\rangle \end{aligned}$$

The relations between the first line and (8), as well as between (11) and (12) hold by construction of M_P and $M_{P'}$. The equivalence between (8) and (9) comes from the hypothesis that g is an isomorphism. Between (9) and (10), we use that g is a bijection. For the equivalence between (10) and (11), we use that g is surjective over F'.

This finally proves that (υ, ν) is a homomorphism, and thus an isomorphism by choice of its domain and codomain. ◀

The results still hold if we do not enforce conditions (3) and (4), that the initial and goal states of P and P' are in bijection. Indeed, the hardness proof relies on a reduction from the Graph Isomorphism problem, with graphs that do not have initial or goal nodes, which renders trivial the initial and goal states of the construction. Conversely, the proof that SI belongs to class GI can include, or not, the relations $\mathcal{R}_I$ and $\mathcal{R}_G$ that take into account the information concerning initial and goal states, and still remain correct for the version of SI without conditions on the initial and goal states. This means that the hardness of SI comes from matching the inner structure of the state-space, and that additional properties on some states (like being initial states or goal states) do not impact significantly the complexity of the problem. This is consistent with our intuition of class GI: it is known that finding a color-preserving isomorphism between colored graphs (i.e., an isomorphism that conserves a given property on nodes) is also a problem that is complete for this class [15].

4 The STRIPS Subinstance isomorphism problem

Let us now introduce problems SSI-H and SSI, which are concerned with finding (different kinds of) isomorphisms between a planning instance P and some subinstance of another STRIPS instance P'. In this section, we settle the complexity of both problems, and show that they are NP-complete. We use this result in order to propose, in the next section, an algorithm for SSI and SSI-H. This algorithm is based on a reduction to SAT, assisted by a preprocessing phase that relies on constraint propagation.

We begin by introducing the notion of *homogeneous subinstance isomorphism*, which is concerned with finding an isomorphism between P and a subinstance of P', but does not conserve the initial state and goal. It maps the whole state-space of problem P to a part of the state-space of problem P', regardless of the initial state and goal of either problem.

▶ **Definition 12** (Homogeneous subinstance isomorphism). *Consider two STRIPS instances $P = \langle F, I, O, G\rangle$ and $P' = \langle F', I', O', G'\rangle$. A* homogeneous subinstance isomorphism *from P to P' is a pair (υ, ν) of* injective *mappings $\upsilon : F \to F'$ and $\nu : O \to O'$ that respect condition (2) of Definition 7.*

▶ **Problem 13.** *STRIPS Homogeneous Subinstance Isomorphism SSI-H*

Input	*Two STRIPS instances P and P'*
Output	*A* homogeneous subinstance isomorphism *(υ, ν) between P and P', if one exists*

A homogeneous subinstance isomorphism between P and P' is useful, for instance, in the case where we managed to compile all plans for P', and wish to extract a plan for P. The following more precise notion of isomorphism between P and a subinstance of P' takes into account the information provided by the initial state and goal. This allows us to carry over only *solution*-plans from one problem to the other.

▶ **Definition 14** (Subinstance isomorphism). *A* subinstance isomorphism *from P to P' is a homogeneous subinstance isomorphism that respects conditions (3) and (4) of Definition 7.*

▶ **Problem 15.** *STRIPS Subsintance Isomorphism SSI*

Input	*Two STRIPS instances P and P'*
Output	*A subinstance isomorphism (υ, ν) between P and P', if one exists*

The main difference between SI and SSI is that, in SSI, we relax the condition on the bijectivity of υ and ν, to account for the difference in size between P and P'. Their injectivity is still required in order to prevent fluents (or operators) being merged together by the mapping. All other conditions remain the same.

The main result of this section is presented below. The proof is based on a reduction from the Subgraph Matching problem, which is known to be NP-complete [5]. As such, we introduce that problem before stating our result. Essentially, it consists in finding a mapping g, that defines an isomorphism between $\mathcal{G}$ and the subgraph $(g(V), E' \cap g(V) \times g(V))$ of $\mathcal{G}'$.

▶ **Problem 16.** *Subgraph Matching problem*

Input *Two non-oriented graphs $\mathcal{G}(V, E)$ and $\mathcal{G}'(V', E')$*

Output *An injective mapping $g : V \to V'$ such that, for any $v_1, v_2 \in V$, $\{v_1, v_2\} \in E$ iff $\{g(v_1), g(v_2)\} \in E'$.*

▶ **Proposition 17.** *SSI is NP-complete*

Proof. In order to prove that SSI is in NP, it suffices to resort to the certificate-based definition of the class NP, and observe that the mappings υ and ν constitute a polynomial size certificate that can be checked in polynomial time.

The proof that SSI is NP-hard consists in a reduction from the Subgraph Matching problem, which is straightforward with the construction that we proposed earlier.

Let $(\mathcal{G}, \mathcal{G}')$ be an instance of the Subgraph Matching problem, and let us follow Construction 6 to build planning problems $P_{\mathcal{G}}$ and $P_{\mathcal{G}'}$. We show that there exists a subgraph matching g between $\mathcal{G}$ and $\mathcal{G}'$ iff there exists a subinstance isomorphism of $P_{\mathcal{G}}$ and $P_{\mathcal{G}'}$.

($\Rightarrow$) Suppose that there exists a subgraph matching $g : V \to V'$ between $\mathcal{G}$ and $\mathcal{G}'$. Then by construction, as $F = V$ and $F' = V'$, g is also an injective mapping between F and F'. In addition, let us define the mapping $\nu : O \to O'$ such that $\nu : \mathsf{move}(v_1, v_2) \mapsto \mathsf{move}(g(v_1), g(v_2))$. ν is well-defined, as $\{v_1, v_2\} \in E$ iff $\{g(v_1), g(v_2)\} \in E'$, so $\mathsf{move}(v_1, v_2) \in O$ iff $\mathsf{move}(g(v_1), g(v_2)) \in O'$. In addition, as g is injective, so is ν. As a consequence, (g, ν) is a subinstance isomorphism between $P_{\mathcal{G}}$ and $P_{\mathcal{G}'}$.

($\Leftarrow$) Suppose that there exists a subinstance isomorphism (υ, ν) between $P_{\mathcal{G}}$ and $P_{\mathcal{G}'}$. As above, $\upsilon : V \to V'$ is an injective mapping. In addition, we have that

$$\begin{aligned}
&(v_1, v_2) \in E \\
\text{iff } &\mathsf{move}(v_1, v_2) \in O \\
\text{iff } &\nu\left(\mathsf{move}(v_1, v_2)\right) \in O' \\
\text{iff } &\mathsf{move}(\upsilon\left(v_1\right), \upsilon\left(v_2\right)) \in O' \\
\text{iff } &\left(\upsilon\left(v_1\right), \upsilon\left(v_2\right)\right) \in E'
\end{aligned}$$

As a consequence, υ is a subgraph matching between $\mathcal{G}$ and $\mathcal{G}'$. ◀

In addition, it is clear that SSI-H is in NP. As the above proof of NP-hardness of SSI is independent of the initial and goal states, it also applies to the problem SSI-H.

▶ **Corollary 18.** *SSI-H is NP-complete*

5 An algorithm for SSI

In this section, we present an algorithm for problem SSI, for which the pseudo-code is presented in Algorithm 1. This algorithm is based on a compilation of the problem into a propositional formula, which is then passed to a SAT solver. It is completed by a preprocessing step, based on constraint propagation, that allows us to prune impossible mappings early on.

Algorithm 1 to find a subinstance isomorphism.

```
Input: Two STRIPS instances $P$ and $P'$
Output: A subinstance isomorphism between $P$ and $P'$ if one exists
 1: Initialize_domains($F, O$)
    /* Prune impossible associations */
 2: $Q := F \cup O$
 3: while $Q \neq \varnothing$ do
 4:     $v := Q$.Pop()
 5:     $r :=$ Revise($v$)
 6:     if $r$ then
 7:         if $\mathcal{D}(v) = \varnothing$ then return UNSAT
 8:         else $Q$.Add($\{v' \mid v'$ related to $v\}$)
    /* Search phase through a SAT solver */
 9: $\varphi :=$ Encode_to_SAT($P, P', \mathcal{D}$)
10: return Interpret(Solver.Find_model($\varphi$))
```

Given two STRIPS instances P and P', the algorithm outputs, when possible, a subinstance isomorphism (υ, ν). Algorithm 1 consists in two main phases. The first phase, that spans lines 2 to 8, consists in pruning as many associations between fluents (resp. operators) of problem P and fluents (resp. operators) of problem P' that are impossible, because of some syntactical inconsistencies (described below) that are then propagated. The second phase, that starts at line 9, consists in a search phase, by means of an encoding of the problem into a CNF formula, that is then passed to a SAT solver.

5.1 Pruning invalid associations

By *association* between fluents, we mean a pair $(f, f') \in F \times F'$ such that f' is a candidate for the value of $\upsilon(f)$. Similarly, we call an *association* between operators a pair $(o, o') \in O \times O'$ such that o' is a candidate for the value of $\nu(o)$. Detecting early on associations that can not be part of a valid subinstance isomorphism reduces the size of the search space.

In order to prune as many inconsistent associations as possible, we use a technique similar to constraint propagation, as commonly found in the constraint programming literature. The general idea is to maintain, for each fluent $f \in F$ of P, a *domain* $\mathcal{D}(f) \subseteq F'$ of fluents of P', that consists of the plausible candidates for the value of $\upsilon(f)$. Similarly, each operator $o \in O$ is assigned a domain $\mathcal{D}(o) \subseteq O'$, containing the plausible candidates for $\nu(o)$. In the following, we call fluents and operators *variables*. The aim of the procedure presented below is to trim the domains of the variables, thus alleviating the load left to the SAT solver.

The first step is to initialize the domains. For each fluent $f \in F$, we set $\mathcal{D}(f) = F'$. The initial assignment of the domains of operators $o \in O$, however, is based on *operator profiles*. For each operator $o \in O \cup O'$, we define the vector $\mathsf{profile}(o) \in \mathbb{N}^6$, called the *profile* of o. This vector numerically abstracts some characteristics of the operator, so that an operator $o \in O$ cannot be associated to operator $o' \in O'$ if $\mathsf{profile}(o) \neq \mathsf{profile}(o')$. In practice, $\mathsf{profile}(o)$ consists of the number of positive and negative fluents in the precondition and effect of o, as well as its number of *strict-add* and *strict-delete* fluents. A fluent f is said to be *strict-add* for operator o if $f \in \mathsf{pre}^-(o) \wedge f \in \mathsf{eff}^+(o)$, and *strict-delete* if $f \in \mathsf{pre}^+(o) \wedge f \in \mathsf{eff}^-(o)$. Then, we initialize the domain of each $o \in O$ such that

$$\mathcal{D}(o) = \{o' \in O' \mid \mathsf{profile}(o') = \mathsf{profile}(o)\}$$

The second step is to propagate the additional constraints posed by these newly-found restrictions of the domains. The technique we propose is based on the concept of arc consistency, which is ubiquitous in the field of constraint programming. The idea consists in eliminating, from the domains of fluents (resp. operators), the candidate fluents (resp. operators) that have no support in the domain of some operator (resp. fluent).

More specifically, let us consider a fluent $f \in F$. When an operator $o \in O$ is such that f appears, negated or not, in its precondition or effect, then we say that o *depends* on f. Let us denote $d(f)$ the set of operators that depend on f. When $f' \in F'$, we define $d(f')$ in a similar fashion. Now suppose that $\upsilon(f) = f'$. As a consequence of equation (2) of Definition 7, each operator of $d(f)$ must have its image by ν in $d(f')$. Otherwise, f would appear in $\mathsf{pre}(o)$ or $\mathsf{eff}(o)$, but $\upsilon(f)$ would not appear in $\upsilon(\mathsf{pre}(o))$ nor $\upsilon(\mathsf{eff}(o))$. Thus, if for some operator $o \in d(f)$ no candidate operator for its image is in $d(f')$ (*i.e.*, $\mathcal{D}(o) \cap d(f') = \varnothing$), then it means that f' can not be chosen as the image of f.

In the following, we refine the argument of last paragraph by identifying $\mathsf{pre}^+(o)$ with $\mathsf{pre}^+(o'), \ldots, \mathsf{eff}^-(o)$ with $\mathsf{eff}^-(o')$. We thus have the following constraint for $\mathcal{D}(f)$, where $\mathcal{C} = \{\mathsf{pre}^+, \mathsf{pre}^-, \mathsf{eff}^+, \mathsf{eff}^-\}$:

$$\mathcal{D}(f) \subseteq \left\{ f' \;\middle|\; \begin{array}{l} \forall o \in O, \forall \mathcal{S} \in \mathcal{C} \text{ s.t. } f \in \mathcal{S}(o), \\ \exists o' \in \mathcal{D}(o) \text{ s.t. } f' \in \mathcal{S}(o') \end{array} \right\} \tag{13}$$

A similar case can be made for operators. Let $o \in O$ be any operator, and consider a candidate operator $o' \in O'$. In order for the morphism property to hold, in the case where $\nu(o) = o'$, for every fluent f of $\mathsf{pre}^+(o)$, for instance, there must exist in $\mathsf{pre}^+(o')$ a fluent that belongs to $\mathcal{D}(f)$. More generally and more formally, we have the following:

$$\mathcal{D}(o) \subseteq \{o' \mid \forall \mathcal{S} \in \mathcal{C}, \forall f \in \mathcal{S}(o), \exists f' \in \mathcal{D}(f) \cap \mathcal{S}(o')\} \tag{14}$$

Algorithmically, we enforce these constraints using an adaptation of AC3 [12, 14]. The algorithm revolves around the revision of the coherence of the variables' domains. Revising a variable v boils down to checking that all elements of its domain still comply with the necessary condition evoked earlier, which is either equation (13) if v is a fluent, or equation (14) if v is an operator. The main loop, depicted in Algorithm 1, then consists in revising all fluents and operators iteratively, by maintaining a queue Q of variables to revise (line 1). The algorithm begins by revising once each variable. If, during the revision of a variable v, the domain of v is altered by the procedure, then all variables that are related to v are added to the set of variables to revise later on (lines 5 to 9). We say that v' is related to v if v is a fluent and $v' \in d(v)$, or conversely. If the domain of a variable is empty, then no isomorphism exists, and the procedure ends prematurely (line 6 and 7). Otherwise, the loop ends when there is no variable left to revise.

This procedure is often not sufficient to conclude, but greatly alleviates the pressure on the search phase, which we present in the following section.

5.2 Encoding into a SAT instance

In this section, we build the propositional formula φ evoked earlier, from the models of which an isomorphism can be extracted. φ is built on the set of variables $\mathit{Var}(\varphi)$, such that:

$$\mathit{Var}(\varphi) = \left\{ f_i^j \;\middle|\; i \in F, j \in F' \right\} \cup \{ o_r^s \mid r \in O, s \in O' \}$$

The propositional variable f_i^j represents the association of fluent $i \in F$ to fluent $j \in F'$. Likewise, o_r^s represents the association of $r \in O$ to $s \in O'$.

In the rest of this section, we show how to build formula φ, which encodes the SSI problem input to Algorithm 1. φ consists in the conjunction of the formulas presented below.

The formula presented in (15) enforces that each fluent has an image which is unique. Similarly, by swapping f_i^j variables for o_i^j and adapting the domains of i and j, we enforce that each operator has an image by ν.

$$\bigwedge_{i \in F} \left(\bigvee_{j \in \mathcal{D}(i)} f_i^j \wedge \bigwedge_{\substack{j,k \in \mathcal{D}(i) \\ j \neq k}} (\neg f_i^j \vee \neg f_i^k) \right) \tag{15}$$

We now need to ensure that υ and ν are injective. For fluents, this is done through (16). A similar formula is used to ensure the injectivity of ν on operators.

$$\bigwedge_{i \in F'} \bigwedge_{\substack{j,k \in F \\ j \neq k}} \neg f_j^i \vee \neg f_k^i \tag{16}$$

The morphism property is enforced by formulas (17) and (18), for each $\mathcal{S} \in \mathcal{C}$, where $\mathcal{C} = \{\mathsf{pre}^+, \mathsf{pre}^-, \mathsf{eff}^+, \mathsf{eff}^-\}$. More precisely, (17) ensures that, for any $\mathcal{S} \in \mathcal{C}$ and for any operator $o \in O$, we have $\upsilon(\mathcal{S}(o)) \subseteq \mathcal{S}(\nu(o))$. Conversely, (18) ensures that $\mathcal{S}(\nu(o)) \subseteq \upsilon(\mathcal{S}(o))$.

$$\bigwedge_{\substack{r \in O \\ s \in O'}} \left(o_r^s \longrightarrow \bigwedge_{i \in \mathcal{S}(r)} \bigvee_{j \in \mathcal{S}(s)} f_i^j \right) \tag{17}$$

$$\bigwedge_{\substack{r \in O \\ s \in O'}} \left(o_r^s \longrightarrow \bigwedge_{j \in \mathcal{S}(s)} \bigvee_{i \in \mathcal{S}(r)} f_i^j \right) \tag{18}$$

Finally, we need to conserve the initial and the goal state (i.e., respect equations (3) and (4)). Let us denote I^+ (resp. I^-) the set of fluents appearing positively (resp. negatively) in I, and use similar notation for G, I' and G'. For every $\mathcal{T} \in \{I^+, I^-, G^+, G^-\}$, and for the corresponding $\mathcal{T}' \in \{I'^+, I'^-, G'^+, G'^-\}$, we then add the following formulas:

$$\bigwedge_{i \in \mathcal{T}} \bigvee_{j \in \mathcal{T}'} f_i^j \quad \wedge \quad \bigwedge_{j \in \mathcal{T}'} \bigvee_{i \in \mathcal{T}} f_i^j \tag{19}$$

The formulas presented in (15), (16), and (19) are immediately in CNF, and the size of their conjunction is in $\mathcal{O}(|F| \cdot |F'|^2 + |O| \cdot |O'|^2)$ assuming $|F| \leq |F'|$ and $|O| \leq |O'|$. In addition, the formulas presented in (17) and (18) can be readily converted into CNF by duplicating the implication in each clause, and then have a size $\mathcal{O}(|O| \cdot |O'| \cdot |F| \cdot |F'|)$.

The preprocessing step presented in Section 5.1 allows us to simplify φ. Indeed, if it is known that fluent $i \in F$ (resp. $r \in O$) cannot be mapped to fluent $j \in F'$ (resp. $s \in O'$), then f_i^j (resp. o_r^s) is necessarily false in any model of φ. As a consequence, as all formulas are in CNF, every positive occurrence of f_i^j is removed in the clauses of φ, while clauses where f_i^j appears negatively are simplified.

In order to adapt the algorithm for SSI-H, it suffices to remove the set of formulas presented in (19). The others formulas and the rest of the algorithm remains the same.

Table 1 Number of instances of SSI-H and SSI on which our implementation of our method terminates within 600 seconds. For each problem, the first pair of columns shows the number of STRIPS matching instances solved with and without the constraint propagation-based preprocessing, respectively. The last column shows the average percentage of clauses that have been eliminated from the propositional encoding, thanks to the pruning step.

Domain	SSI-H			SSI		
	CP	NoCP	Av. Simp.	CP	NoCP	Av. Simp.
blocks	172	96	76.1%	166	93	76.2%
gripper	210	189	74.9%	90	84	75.1%
hanoi	74	75	0.2%	85	82	0.2%
rovers	19	6	97.4%	16	6	97.3%
satellite	34	22	79.1%	38	23	78.4%
sokoban	204	0	98.6%	205	4	98.6%
tsp	376	374	0.7%	265	266	1.0%

6 Experimental evaluation

We implemented Algorithm 1 in Python 3.10, and used it to solve SSI and SSI-H. In order to parse planning instances in PDDL and convert them into a STRIPS representation, we used the parser of TouISTPlan [3]. The SAT solver we used was Maple LCM [11], winner of the main track at SAT 2017. The code and sets of benchmarks are available online.

Experiments were run on a machine running Rocky Linux 8.5, powered by an Intel Xeon E5-2667 v3 processor, using at most 8GB of RAM and 4 threads per test. Our set of benchmarks is based on eight sets found in previous International Planning Competitions, namely Blocks, Gripper, Hanoi, Rovers, Satellite, Sokoban, TSP and Visitall. For each of these domains, we created what we call *STRIPS matching instances*, which are pairs of instances of the same domain. We did this for each possible pair of planning instances of each considered domain. A STRIPS matching instance is an instance of both SSI and SSI-H. We thus evaluated our algorithm adapted for both problems on the same set of benchmarks.

The goal of the experiments is twofold. First, the aim is to demonstrate that, despite the theoretical hardness of the problem, it is possible to find a (homogeneous) subinstance isomorphism in reasonable time for problems of non-trivial size. Second, the goal is to show the efficiency of the pruning technique presented in Section 5.1, i.e., to prove that the additional cost of the preprocessing is outbalanced by the speed-up it provides during search.

The coverage of our implementation on our set of benchmarks is shown in Table 1 for both SSI and SSI-H. The table shows the absolute and relative numbers of instances of SSI (resp. SSI-H) on which our implementation terminates within the time and memory cutoffs. Note that we tested our algorithm on a handful of other domains, but we only report those for which at least one instance was solved. Domains where even the smallest problem timeouts include Visitall, Barman and Woodworking, for instance.

The first point we notice is that problems SSI-H and SSI are often closely comparable in terms of hardness, except for some particular domains. These include domains TSP and Gripper, for which 40% and 133% more instances are solved when requiring no condition on the initial state and goal. For both domains, this is due to the additional constraints in SSI. Indeed, because of these constraints all pairs of non-identical TSP planning instances (or Gripper planning instances) constitute negative SSI instances, which turn out to be harder for the SAT solver to detect than positive ones.

A crucial observation is that the preprocessing step almost never holds back the algorithm: almost all instances of our test sets that can be solved without preprocessing are also solved when the preprocessing step is performed. Furthermore, in many sets of benchmarks, the preprocessing greatly improves the overall performance of our implementation, so much so that some previously infeasible domains are now within the range of our algorithm. Such extreme cases include Sokoban, for which our algorithm is powerless without the pruning step: all 204 instances solved by our implementation are outside the range of the preprocessing-less version of the algorithm. In most cases, however, we observe a significant increase in the coverage of the algorithm, that remains nonetheless within the same order of magnitude. For example, for domain Satellite in the case of SSI, 34 instances are solved when constraint propagation is enabled, whereas only 22 can be settled without it.

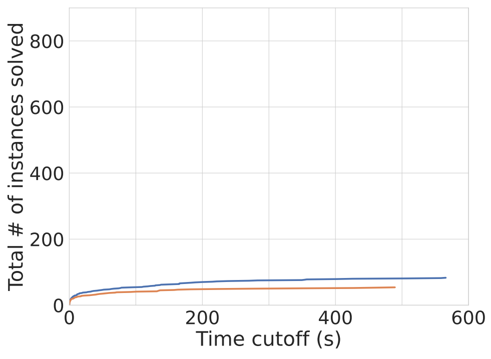
More specifically, in most cases, the preprocessing step leads to a reduction of the size of the propositional encoding. This is shown by the columns labeled "Av. Simp." in Table 1, which represent the average proportion of clauses that are simplified as a consequence of the pruning step. The highest percentages of simplified clauses are found in domains that contain little to no symmetries. For example, in Rovers, fluents represents entities that often have different types, and that are affected in different ways by operators. For instance, operators of the form `navigate(rover, x, y)` have a unique profile, and are not numerous. Consequently, their respective domains remain small, which is something our algorithm makes the most of.

On the contrary, for domains that contain lots of symmetries, the pruning step does not remove a significant number of associations. This is the case in Hanoi, where all operators have the same profile: except for the information provided by the initial and goal state, all disks are interchangeable, which does not allow our preprocessing to draw any conclusive result. The only bits of information that can guide the search are encoded in the initial state, which we believe partly explains the slightly greater coverage of SSI over SSI-H.

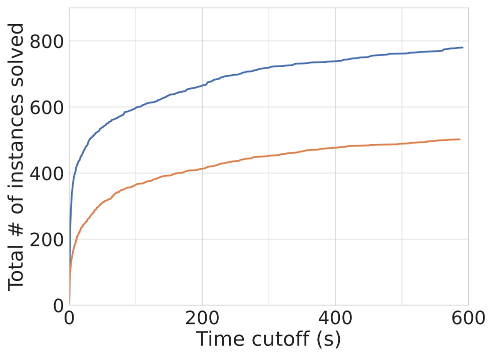
In some instances of our set of benchmarks, pruning suffices to find that no (homogeneous) subinstance isomorphism exists: as the majority of associations between fluents or between operators are ruled out, the domains of some variables become empty. As a direct consequence, our algorithm is most effective in the case where no (homogeneous) subinstance isomorphism exists. In many of these cases, an empty domain is found for a variable, which allows the algorithm to return UNSAT prematurely, and skip the search phase altogether. This is why the pruning step allows us to significantly increase our coverage on STRIPS matching instances that are negative, as shown in Figure 1, while our performance on positive instances is more modest, although significant.

In Table 2, we also show that the additional time required by the constraint propagation phase is negligible compared to the rest of the algorithm. In fact, be it in domains where it prunes out lots of associations or in domains where its efficiency is limited, constraint propagation rarely takes more than a handful of seconds. As a consequence, some instances that would otherwise require a substantial amount of time are now solved almost immediately. In addition, as shown in Figure 1b, solving any 500 negative SSI instances requires 10 minutes when pruning is not enabled, while it requires less than a minute when pruning is enabled.

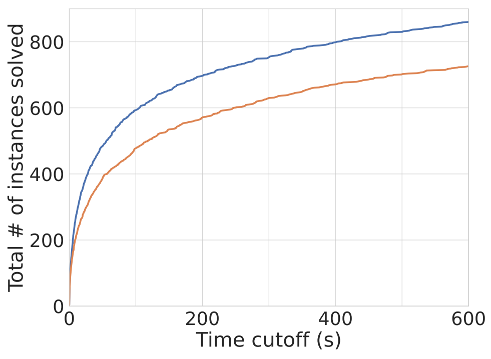
In Table 3, we present a few results on the absolute sizes of the problems that we solved during our experiments, within the time and memory limits. For a STRIPS planning problem $P = \langle F, I, O, G \rangle$, we denote $|P| = |F| + |O|$. As an SSI instance has two main dimensions, represented by the respective sizes of the planning instances that constitute it, we present two different ways of measuring it size6. In the first set of columns of Table 3, the sum of both planning instances is considered, and we report the size of the SSI instance that maximizes that sum. With this metric, P' is often disproportionately bigger than P. This



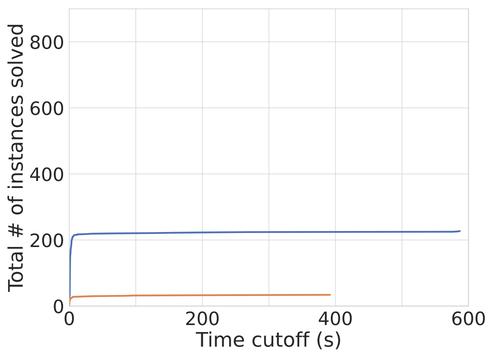
(a) SSI instances with positive outcome



(b) SSI instances with negative outcome



(c) SSI-H instances with positive outcome



(d) SSI-H instances with negative outcome

Figure 1 Number of SSI (top) and SSI-H (bottom) instances that can be solved by our implementation, as a function of the time cutoff. Blue/orange curves correspond respectively to with/without pruning (constraint propagation preprocessing).

imbalance can be explained by the fact that the encoding into a propositional formula is of time and size $\mathcal{O}(|O| \cdot |O'| \cdot |F| \cdot |F'|)$, as mentioned previously. Thus, in the second set of columns, we consider instead the lexicographic order on pairs $(|P|, |P'|)$, and report the biggest problem with respect to that metric.

7 Conclusion

In this article, we introduced the problem SI, which is concerned with finding an isomorphism between two planning problems, and showed that it is GI-complete. Afterwards, we introduced the notion of subinstance isomorphism, as well as the associated problem SSI. In addition to proving the NP-completeness of the problem, we proposed an algorithm for it, based on constraint propagation techniques and a reduction to SAT

The experimental evaluation of said algorithm shows that traditional constraint propagation in a preprocessing step can greatly improve the efficiency of SAT solvers. However, even though it is not costly to perform, not all planning domains benefited equally from this preprocessing.

On a more general note, various methods have been proposed to automatically reformulate general models, with the aim of rendering easier the task delegated to the solver [13]. It remains an interesting open question to identify which characteristics of problems in NP make them amenable to this hybrid CP-SAT approach.

Table 2 Average time, in seconds, spent in each of the main three steps of the algorithm: pruning (CP), compilation to SAT, and solving, respectively. The last column summarizes the average total running time of the algorithm. We only report instances that were successfully solved (either positively or negatively), and results for SSI-H and SSI are thus non-comparable.

Domain	SSI-H				SSI			
	CP	Comp.	Solving	Total time	CP	Comp.	Solving	Total time
blocks	0.5	93.3	76.8	170.3	0.4	83.3	94.6	178.1
gripper	0.2	23.5	9.8	33.4	0.1	11.2	35.2	46.5
hanoi	0.3	43.9	78.0	122.0	0.3	70.9	47.8	118.9
rovers	1.8	168.9	2.2	171.4	1.7	180.7	2.7	183.5
satellite	0.4	116.4	48.8	165.4	0.4	85.0	10.3	95.4
sokoban	1.7	222.7	2.3	225.2	1.7	220.6	1.4	222.3
tsp	0.2	50.7	46.7	97.5	0.1	14.6	26.6	41.2

Table 3 Sizes of the biggest instances that can be solved by our implementation within the time and memory limits, for both SSI-H and SSI. In the first set of columns, we consider the sum of the sizes of the planning instances that constitute the STRIPS matching instance. In the second set, we consider the size of P, the smallest planning instance among the pair that constitutes the instance.

Domain	SSI-H					SSI				
	Maximum sum			Max $\|P\|$		Maximum sum			Max $\|P\|$	
	$\|P\|$	$\|P'\|$	Sum	$\|P\|$	$\|P'\|$	$\|P\|$	$\|P'\|$	Sum	$\|P\|$	$\|P'\|$
blocks	57	4642	4699	534	534	57	4642	4699	534	534
gripper	510	510	1020	510	510	510	510	1020	510	510
hanoi	13	3328	3341	391	391	13	6953	6966	391	513
rovers	276	2667	2943	920	920	276	2667	2943	920	920
satellite	147	2066	2213	608	920	147	2610	2757	608	920
sokoban	2212	2286	4498	2212	2286	2212	2286	4498	2212	2286
tsp	182	930	1112	462	462	90	930	1020	380	380

References

1 Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis. Consistency restoration and explanations in dynamic csps application to configuration. *Artif. Intell.*, 135(1-2):199–234, 2002. doi:10.1016/S0004-3702(01)00162-X.

2 László Babai. Group, graphs, algorithms: the graph isomorphism problem. In *Proceedings of the International Congress of Mathematicians*, pages 3319–3336. World Scientific, 2018.

3 Djamila Baroudi, Maël Valais, and Frédéric Maris. Touistplan. URL: https://github.com/touist/touistplan.

4 Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

5 Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *3rd Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047.

6 Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002. doi:10.1613/jair.989.

7 Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971. doi:10.1016/0004-3702(71)90010-5.

8 Malte Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2):219–262, 2003. doi:10.1016/S0004-3702(02)00364-8.

9 Henry A. Kautz and Bart Selman. Planning as satisfiability. In Bernd Neumann, editor, *ECAI 92*, pages 359–363. John Wiley and Sons, 1992.

10 Songtuan Lin and Pascal Bercher. Change the world - how hard can that be? On the computational complexity of fixing planning models. In Zhi-Hua Zhou, editor, *IJCAI-21*, pages 4152–4159, August 2021. doi:10.24963/ijcai.2021/571.

11 Mao Luo, Chu-Min Li, Fan Xiao, Felip Manyà, and Zhipeng Lü. An effective learnt clause minimization approach for cdcl sat solvers. In *IJCAI-17*, pages 703–711, 2017. doi:10.24963/ijcai.2017/98.

12 Alan K Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8(1):99–118, 1977.

13 Peter Nightingale, Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Patrick Spracklen. Automatically improving constraint models in savile row. *Artif. Intell.*, 251:35–61, 2017. doi:10.1016/j.artint.2017.07.001.

14 Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

15 Viktor N Zemlyachenko, Nickolay M Korneenko, and Regina I Tyshkevich. Graph isomorphism problem. *Journal of Soviet Mathematics*, 29(4):1426–1481, 1985.