Comparing Planning Domain Models Using Answer Set Programming

Lukáš Chrpa[1](✉), Carmine Dodaro[2], Marco Maratea[2], Marco Mochi[3], and Mauro Vallati[4]

[1] Czech Technical University in Prague, Prague, Czechia
chrpaluk@cvut.cz
[2] University of Calabria, Arcavacata, Italy
{carmine.dodaro,marco.maratea}@unical.it
[3] University of Genova, Genova, Italy
marco.mochi@edu.unige.it
[4] University of Huddersfield, Huddersfield, UK
m.vallati@hud.ac.uk

Abstract. Automated planning is a prominent area of Artificial Intelligence, and an important component for intelligent autonomous agents. A critical aspect of domain-independent planning is the domain model, that encodes a formal representation of domain knowledge needed to reason upon a given problem. Despite the crucial role of domain models in automated planning, there is lack of tools supporting knowledge engineering process by comparing different versions of the models, in particular, determining and highlighting differences the models have.

In this paper, we build on the notion of *strong equivalence* of domain models and formalise a novel concept of *similarity* of domain models. To measure the similarity of two models, we introduce a directed graph representation of lifted domain models that allows to formulate the domain model similarity problem as a variant of the graph edit distance problem. We propose an Answer Set Programming approach to optimally solve the domain model similarity problem, that identifies the minimum number of modifications the models need to become strongly equivalent, and we demonstrate the capabilities of the approach on a range of benchmark models.

Keywords: Automated Planning · Answer Set Programming · Domain Model

1 Introduction

Automated planning is a research discipline that addresses the problem of generating a totally- or partially-ordered sequence of actions that transforms the environment from an initial state to a desired goal state. It has matured to such a degree that there exists a wide range of applications utilising planning, including UAV manoeuvring [21], space exploration [1], and train dispatching [7].

A critical aspect of domain-independent planning is the domain knowledge that must be fed into a planning engine that comes under the form of a domain model, a symbolic representation of the environment and actions, that has to be engineered prior its use [16]. The importance of good quality domain models in planning, and of the corresponding knowledge engineering process, has been well-argued [17,27,28]. However, there is a lack of approaches to support the knowledge engineering process. In particular, there is no “diff” tool that compares different versions of a domain model and highlights differences among them. Tools such as D-VAL [25] or a recent work of Coulter et al. [10] provide some limited support to compare domain models focusing on the state space they can generate, and the model reconciliation problem focuses on explaining why two models cannot create the same optimal plans [8].

To address the highlighted research gap, we propose a novel concept of domain model similarity and present a theoretical framework underlying the concept, which employs an extension of the notion of *strong equivalence* informally introduced by Shoeeb and McCluskey [24], which determines whether domain models are the same except naming. We propose a directed graph representation of lifted domain models and we show that domain models are strongly equivalent if and only if the graphs representing them are isomorphic. Then, we define *distance* between domain models as the minimum number of modifications that have to be made to both models to make them strongly equivalent. It corresponds to the notion of *edit distance* between two graphs (representing the domain models). The introduced theoretical framework gives us the notion of *similarity* by measuring the distance between domain models and enumerating the modifications that need to be done to make the models (strongly) equivalent. Then, we present an approach based on Answer Set Programming (ASP) [3,5,14,20] that allows to compare planning domain models to assess their similarity. This is not the first time that declarative programming, in particular ASP, is employed in this context, but considering either different problems in the planning domain (the already mentioned [25]), or not focused on planning [26]. Our solution relies on a directed graph representation of the lifted domain models, and is capable of providing optimally minimal sets of changes to transform one model into the other. Beside providing the first concrete approach to assess if two domain models are strongly equivalent, the proposed notion of similarity, and the ASP-based approach to measure it, have several practical implications: (i) it can be incorporated into a “diff” tool for highlighting differences between two versions of a domain model, to help knowledge engineers in understanding modifications; (ii) it can support the evaluation of tools for automated domain model acquisition (e.g., LOCM [11]) by comparing acquired domain models to the reference domain models; (iii) it can be exploited as an advanced plagiarism checker, where it can provide a “similarity” score to flag potential cases of plagiarism, and (iv) it can support the evaluation of models in competitions on domain modelling such as ICKEPS [9] and provide useful insights into how groups of experts differ in developing models.

We evaluate the approach on well-known benchmark domains from international competitions, of different size with regards to the number of models’ predicates and operators. We present a fully declarative approach, which is able to compare a number of planning domains, except the largest, and an improved solution, that exploits a preprocessor via imperative programming that acts as a sort of “problem-aware pregrounder”, which complements the declarative encoding. The related empirical evaluation shows that, by employing the improved solution, the comparison can be performed in less than a CPU-time second for all evaluated models, hence suggesting that it can be fruitfully exploited to support the knowledge engineering process of domain models in real time.

2 Background

In this section we present, in two separate subsections, needed preliminaries about automated planning and graph similarity, respectively.

Automated Planning. In the STRIPS representation, the environment is represented by *propositions*. *States* are defined as sets of these propositions (or *atoms*). An *action* is a quadruple $a = (\mathit{name}(a), \mathit{pre}(a), \mathit{del}(a), \mathit{add}(a))$, where $\mathit{name}(a)$ represents a unique action name, $\mathit{pre}(a)$, $\mathit{del}(a)$ and $\mathit{add}(a)$ are sets of atoms representing the *precondition* of a, the *delete* and *add effects* of a, respectively. We assume a is always *well defined*, i.e., $\mathit{add}(a) \neq \emptyset$ (as an action without any add effect would be useless). We say that an action a is *applicable* in a state s if and only if $\mathit{pre}(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus \mathit{del}(a)) \cup \mathit{add}(a)$.

In the lifted STRIPS representation, the environment is represented by first-order logic *predicates*. A *planning operator* $o = (\mathit{name}(o), \mathit{pre}(o), \mathit{del}(o), \mathit{add}(o))$ is specified such that $\mathit{name}(o) = \mathit{op\_name}(x_1, \ldots, x_k)$ ($\mathit{op\_name}$ represents a unique operator name and $x_1, \ldots x_k$ are variable symbols (parameters) appearing in the operator), $\mathit{pre}(o)$ is a set of predicates representing the operator's *preconditions*, $\mathit{del}(o)$ and $\mathit{add}(o)$ are sets of predicates representing the operator's *delete* and *add* effects, respectively. Again, we assume o is always *well defined*, i.e., $\mathit{add}(o) \neq \emptyset$. A *(lifted) domain model* $\mathcal{D} = (P, O)$ is specified via a set of predicates P and a set of operators O. A *problem instance* $\mathcal{P} = (\mathit{Obj}, I, G)$ for a lifted domain model $\mathcal{D}$ is specified via a set of objects Obj , the initial state I and a set of atoms representing the goal G. Atoms are obtained by grounding of the predicates from P, i.e., by substituting objects for predicates' variables. Actions are grounded instances of planning operators.

A *planning task* $(\mathcal{D}, \mathcal{P})$ consists of a domain model $\mathcal{D}$ and a problem instance $\mathcal{P}$. A *solution plan* for a planning task is a sequence of actions such that consecutive application of the actions in the plan (starting in the initial state) results in a state in which all the goal atoms are true. We say that predicates are *equal* if they have the same name and their parameters including their order are identical. We define a function $pars(\cdot)$ that returns the set of variable symbols of a predicate or an operator. We also define a function $arity(\cdot)$ that returns the number of variable symbols of a predicate or an operator. With regards to *substitution mappings* that map free variables into terms (variables or constants in our case), we use a specific notation in order to disambiguate with other types of mappings. In particular, for a substitution mapping χ and a predicate (or an operator) $p(x_1, \ldots, x_n)$, $(p|\chi)$ refers to substituting $x_1, \ldots, x_n$ for terms according to χ, i.e., $(p|\chi) \equiv p(\chi(x_1), \ldots, \chi(x_n))$.

Graph Similarity. Comparing graphs, in terms of how similar they are, belongs under of the umbrella of *graph matching* [4]. For our purpose, we will consider (labelled) directed graphs with different types of edges. Let $\mathcal{G}_1 = (V_1, E_1^1, E_1^2, \ldots, E_1^k)$ and $\mathcal{G}_2 = (V_2, E_2^1, E_2^2, \ldots, E_2^k)$ be directed graphs with k different types of edges, and $\mathcal{L}_1$ and $\mathcal{L}_2$ be the sets of their edge labels. We say that $\mathcal{G}_1$ and $\mathcal{G}_2$ are *isomorphic* if and only if there exist bijective mappings $\xi : V_1 \to V_2$ and $\nu : \mathcal{L}_1 \to \mathcal{L}_2$ such that for each $1 \leqslant i \leqslant k : (x, l, y) \in E_1^i \Leftrightarrow (\xi(x), \nu(l), \xi(y)) \in E_2^i$. Note that for unlabelled directed graphs it is the case that $\mathcal{G}_1$ and $\mathcal{G}_2$ are *isomorphic* if and only if there exist a bijective mapping $\xi : V_1 \to V_2$ such that for each $1 \leqslant i \leqslant k : (x, y) \in E_1^i \Leftrightarrow (\xi(x), \xi(y)) \in E_2^i$.

Let $\mathcal{G}'_1$ and $\mathcal{G}'_2$ be subgraphs of $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. We say that $\mathcal{G}'_1$ and $\mathcal{G}'_2$ are *common isomorphic subgraphs* of $\mathcal{G}_1$ and $\mathcal{G}_2$ if and only if $\mathcal{G}'_1$ and $\mathcal{G}'_2$ are isomorphic.

Let $elem$ denote the number of elements in a graph (with k different types of edges), i.e., for $\mathcal{G} = (V, E^1, \ldots, E^k)$, $elem(\mathcal{G}) = |V| + \sum_{i=1}^{k} |E^i|$. We say that $\mathcal{G}'_1$ and $\mathcal{G}'_2$ are *maximum common isomorphic subgraphs* of $\mathcal{G}_1$ and $\mathcal{G}_2$ if and only if (i) $\mathcal{G}'_1$ and $\mathcal{G}'_2$ are common isomorphic subgraphs of $\mathcal{G}_1$ and $\mathcal{G}_2$ and (ii) for every pair $\mathcal{G}''_1$ and $\mathcal{G}''_2$ being also common isomorphic subgraphs of $\mathcal{G}_1$ and $\mathcal{G}_2$ it is the case that $elem(\mathcal{G}'_1) \geqslant elem(\mathcal{G}''_1)$ (and $elem(\mathcal{G}'_2) \geqslant elem(\mathcal{G}''_2)$). Then, we define a function *dist* representing a *distance* between graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ as $dist(\mathcal{G}_1, \mathcal{G}_2) = elem(\mathcal{G}_1) + elem(\mathcal{G}_2) - 2 * elem(\mathcal{G}'_1)$ with $\mathcal{G}'_1$ and $\mathcal{G}'_2$ being maximum common isomorphic subgraphs of $\mathcal{G}_1$ and $\mathcal{G}_2$. Note that our notion of distance is a variant of *Graph Edit Distance* [22] in which vertex and edge substitutions are not explicitly counted.

3 Strong Equivalence of Domain Models

Equivalence of domain models can be understood in a similar fashion as equivalence of grammars, i.e., two domain models are equivalent if a planning task specified in one model can be also specified in the other model and both models generate same plans for the corresponding planning tasks [24]. An alternative understanding of domain model equivalence, "functional equivalence", compares corresponding state-transition systems such that two domain models are (functionally) equivalent if and only if for corresponding planning tasks the sets of reachable states are equivalent [25]. In this paper, we focus on *strong equivalence* of domain models that has been informally defined in [24] as models being logically identical up to naming. It assumes that there exist bijective mappings between particular elements (e.g., atoms, action names).

To formally define *strong equivalence* for lifted domain models, we have to make sure that for each corresponding grounded instance of two strongly equivalent lifted domain models it is the case that those instances are strongly equivalent too. Whereas the (bijective) mapping between predicates needs to consider only naming and arity (without loss of generality we assume that free variables in each predicate are distinct), the (bijective) mapping between planning operators has to take into account ordering of their parameters (free variables). We formally define *strong equivalence* for two lifted domain models as follows.

Definition 1. *Let $\mathcal{D} = (P, O)$ and $\mathcal{D}' = (P', O')$ be lifted domain models. If there exist bijective mappings $\mathcal{P} : P \rightarrow P'$ and $\mathcal{O} : \{name(o) \mid o \in O\} \rightarrow \{name(o') \mid o' \in O'\}$ such that*

- *for each $p \in P$, $arity(p) = arity(\mathcal{P}(p))$*
- *for each $o \in O$, $arity(name(o)) = arity(\mathcal{O}(name(o)))$ and there exists $o' \in O'$ and a bijective substitution mapping $\chi^o : pars(o) \rightarrow pars(o')$, where*
  - *$name(o') = \mathcal{O}(name(o))$*
  - *$pre(o') = \{(\mathcal{P}(p)|\chi^o) \mid p \in pre(o)\}$*
  - *$del(o') = \{(\mathcal{P}(p)|\chi^o) \mid p \in del(o)\}$*
  - *$add(o') = \{(\mathcal{P}(p)|\chi^o) \mid p \in add(o)\}$*

then $\mathcal{D}$ and $\mathcal{D}'$ are **strongly equivalent***.*

Next, we will construct a *Lifted Domain Model Graph* (LDMG) which is a labelled directed graph connecting operators with predicates in a given lifted domain model. LDMG has vertices standing for both predicates and operator names, and three types of edges referring to preconditions, delete, and add effects, respectively. Edge labels represent matchings between operators' and predicates' variables. To show that two domain models are strongly equivalent their respective LDMGs have to be isomorphic.

Definition 2. *Let $\mathcal{D} = (P, O)$ be a lifted domain model. We assume, without loss of generality, that all variable symbols defined in $\mathcal{D}$ are distinct. We say that $\mathcal{G} = (V, E_{pre}, E_{del}, E_{add})$ is a* **Lifted Domain Model Graph (LDMG)** *of $\mathcal{D}$, where $V = P \cup \{name(o) \mid o \in O\}$ is a set of vertices, $E_{pre} = \{(name(o), \Theta^o, p) \mid p \in (pre(o|\Theta^o)), o \in O, p \in P\}$, $E_{del} = \{(name(o), \Theta^o, p) \mid p \in (del(o|\Theta^o)), o \in O, p \in P\}$ and $E_{add} = \{(name(o), \Theta^o, p) \mid p \in ((add(o|\Theta^o)), o \in O, p \in P\}$ are sets of labelled directed edges, where Θ^o is the substitution mapping from $pars(o)$ to $\bigcup_{p \in P} pars(p)$ for each operator o.*

Theorem 1. *Let $\mathcal{D} = (P, O)$ and $\mathcal{D}' = (P', O')$ be lifted domain models. We assume, without loss of generality, that all variable symbols defined in both $\mathcal{D}$ and $\mathcal{D}'$ are distinct. Let $\mathcal{G} = (V, E_{pre}, E_{del}, E_{add})$ and $\mathcal{G}' = (V', E'_{pre}, E'_{del}, E'_{add})$ be LDMGs of $\mathcal{D}$ and $\mathcal{D}'$, respectively. $\mathcal{D}$ and $\mathcal{D}'$ are strongly equivalent if and only if $\mathcal{G}$ and $\mathcal{G}'$ are isomorphic with a bijective mapping $\xi : V \to V'$ such that for each $x \in V : arity(x) = arity(\xi(x))$.*

Proof. The "if" part: If $\mathcal{D}$ and $\mathcal{D}'$ are strongly equivalent, then there exist bijective mappings $\mathcal{P}$ and $\mathcal{O}$ between atoms and operator names of both domain models as in Definition 1. We can combine $\mathcal{P}$ and $\mathcal{O}$ into ξ such that for each $f \in P : \xi(f) = \mathcal{P}(f)$ and for each $o \in O : \xi(name(o)) = \mathcal{O}(name(o))$. Hence, ξ is a bijective mapping from V to V'. Then, we can observe that for each $o \in O$ there exists $o' \in O'$ such that $\xi(name(o)) = name(o')$, $arity(\xi(name(o))) = arity(name(o'))$. There also exist substitution mappings χ^o for each $o \in O$ as in Definition 1 and $\Theta^o : pars(o) \to \bigcup_{p \in P} pars(p)$ and $\Theta^{o'} : pars(o') \to \bigcup_{p' \in P'} pars(p')$ for each $o \in O$ and $o' \in O'$ as in Definition 2. Now we can define a bijective substitution mapping $\nu : \bigcup_{o \in O} pars(o) \times \bigcup_{p \in P} pars(p) \to \bigcup_{o' \in O'} pars(o') \times \bigcup_{p' \in P'} pars(p')$ (since variable symbols are distinct) such that for all $o \in O$ and $x \in pars(o)$, $((x, (x|\Theta^o))|\nu) = ((x|\chi^o), ((x|\chi^o)|\Theta^{o'})$. Then, if for $o \in O$ and $p \in P$ it is the case that $p \in (pre(o|\Theta^o))$, then there exists $o' \in O'$ such that $(\xi(p)|\chi^o) \in (pre(o'|\Theta^{o'}))$. Hence, if $(name(o), \Theta^o, p) \in E_{pre}$, then $(\xi(name(o)), (\Theta^o|\nu), \xi(p)) \in E'_{pre}$. For E_{del} and E_{add}, it can be proven analogously.

The "only if" part: From Definition 2 and the fact that every operator is well defined, we can derive that $X = \{x \mid (x, y) \in E_{add}\} = \{name(o) \mid o \in O\}$. If $\mathcal{G}$ and $\mathcal{G}'$ are isomorphic, then there exist a bijective mapping $\xi : V \to V'$ and a bijective substitution mapping $\nu : \bigcup_{o \in O} pars(o) \times \bigcup_{p \in P} pars(p) \to \bigcup_{o' \in O'} pars(o') \times \bigcup_{p' \in P'} pars(p')$. Hence, we can "split" ξ into two bijective mappings $\mathcal{P}$ and $\mathcal{O}$ such that for each $x \in X : \mathcal{O}(x) = \xi(x)$ and for each $y \in (V \setminus X) : \mathcal{P}(y) = \xi(y)$. Also, with $\forall x \in V : arity(x) = arity(\xi(x))$ we can derive $arity(p) = arity(\mathcal{P}(p))$ and $arity(name(o)) = arity(\mathcal{O}(name(o)))$. We can also observe (from the isomorphism of $\mathcal{G}$ and $\mathcal{G}'$) that for each $o \in O$ and $x \in pars(o)$, $((x, (x|\Theta^o)|\nu) = (y, (y|\Theta^{o'}))$ for some $o' \in O'$ and $y \in pars(o')$. Since Θ^o and $\Theta^{o'}$ are substitution mappings, then we can define a substitution mapping $\chi^o : pars(o) \rightarrow pars(o')$ such that $(x|\chi^o) = y$ (with $((x, (x|\Theta^o))|\nu) = (y, (y|\Theta^{o'})))$ and χ^o is bijective. Hence, we can derive for each $o \in O$ and $p \in pre(o)$ that there exists $o' \in O'$ such that $(\mathcal{P}(p)|\chi^o) \in pre(o')$. For $del(o')$ and $add(o')$, it can be proven analogously. $\square$

Example 1 (Running example). We consider as running example a simplified version of the well-known Logistics domain, originally introduced in the IPC 2000. In the simplified version, a number of trucks are used to deliver packages from a location of origin to a destination location. The domain model includes 4 predicates: (at-truck ?Loc ?Truck), (at-package ?Loc ?Pkg), (in-package ?Pkg ?Truck), (in-city ?Cty ?Loc) and 3 operators: load(?Loc ?Pkg ?Truck), unload(?Loc ?Pkg ?Truck), and move(?Cty ?Loc1 ?Loc2 ?Truck).

We can define another domain model that concerns transporting passengers from one location to another by shuttles. The domain model includes 4 predicates: (at-shuttle ?Loc ?Shtl), (at-passenger ?Loc ?Psg), (in-passenger ?Psg ?Shtl), (in-city ?Cty ?Loc) and 3 operators: embark(?Loc ?Psg ?Shtl), debark(?Loc ?Psg ?Shtl), and move(?Cty ?Loc1 ?Loc2 ?Shtl). We can observe that the structure of the domain model is identical to the Logistics model apart of naming of (most of) predicates and operators. Hence, the domain models are strongly equivalent.

4 Domain Model Similarity

Informally speaking, *domain model similarity* stands for quantifying how close domain models are to each other, in terms of how many manipulations (adding/modifying an element in either of the models) are needed to make the models strongly equivalent.

Initially, we define the notion of *submodel* that describes the relation between domain models based on the subgraph relation between their LDMGs.

Definition 3. *Let $\mathcal{D}$ and $\mathcal{D}'$ be domain models. We say that $\mathcal{D}'$ is a* **submodel** *of $\mathcal{D}$ if the LDMG of $\mathcal{D}'$ is a subgraph of the LDMG of $\mathcal{D}$. We say that a domain model $\mathcal{D}''$ is a* **strongly equivalent submodel** *of $\mathcal{D}$ if $\mathcal{D}''$ is strongly equivalent with $\mathcal{D}'$ (being a submodel of $\mathcal{D}$).*

In more general cases, domain models share the same structure only partially. In other words, they share common (strongly equivalent) submodels.

Definition 4. *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be domain models. We say that submodels $\mathcal{D}'_1$ and $\mathcal{D}'_2$ of $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively, are* **common strongly equivalent submodels** *of $\mathcal{D}_1$ and $\mathcal{D}_2$ if and only if $\mathcal{D}'_1$ and $\mathcal{D}'_2$ are strongly equivalent. We say that $\mathcal{D}'_1$ and $\mathcal{D}'_2$ are* **maximum common strongly equivalent submodels** *of $\mathcal{D}_1$ and $\mathcal{D}_2$ if and only if they are common strongly equivalent submodels and the value of $elem$ is maximum for their LDMGs compared to other common strongly equivalent submodels of $\mathcal{D}_1$ and $\mathcal{D}_2$.*

Proposition 1. *Let $\mathcal{D}'_1$ and $\mathcal{D}'_2$ be submodels of domain models $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. It holds that $\mathcal{D}'_1$ and $\mathcal{D}'_2$ are (maximum) common strongly equivalent submodels of $\mathcal{D}_1$ and $\mathcal{D}_2$ if and only if the LDMG of $\mathcal{D}'_1$ and the LDMG of $\mathcal{D}'_2$ are (maximum) common isomorphic subgraphs of the LDMGs of $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively.*

Proof. The claim of the proposition is directly implied from the definition of (maximum) common isomorphic subgraphs (see the Background Section) and Theorem 1. □

The above proposition connects our variant of edit distance of graphs and the distance of domain models. The definition below summarizes the concept.

Definition 5. *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be domain models and $\mathcal{G}_1$ and $\mathcal{G}_2$ be their LDMGs, respectively. We define a dist function representing the* **distance** *between $\mathcal{D}_1$ and $\mathcal{D}_2$ as $dist(\mathcal{D}_1, \mathcal{D}_2) = dist(\mathcal{G}_1, \mathcal{G}_2)$.*

The notion of distance between two domain models determines a minimum number of *elementary operations* to modify these two models to make them strongly equivalent. That corresponds to adding vertices and edges to the LDMGs of these two models. Let $\mathcal{D} = (P, O)$ be a domain model and $\mathcal{G} = (V, E_{pre}, E_{del}, E_{add})$ its LDMG. The *elementary operations* over $\mathcal{D}$ and $\mathcal{G}$ are defined as follows:

(1) Add p into $pre(o)$ (resp. $del(o)$, resp. $add(o)$) iff (o, p) is added into E_{pre} (resp. E_{del}, resp. E_{add}).
(2) Add o into O iff o is added into V and (o, p) is added into E_{add} for some p.
(3) Add p into P iff p is added into V and no edge from p is added to E_{add}.

We would like to emphasise that we do not explicitly distinguish operator and predicate nodes. We can observe that a well defined operator has to have at least one add effect and hence the corresponding vertex in the underlying LDMG has to have at least one outgoing "add" edge. Note, again, that we assume that all operators are well defined. On the other hand, each predicate node has no outgoing edge.

It is known that the problem of graph edit distance is NP-hard [29]. Due to specific structure of LDMGs, again, the question whether determining distance between domain models is NP-hard is still open.

Example 2 (Example 1 cont'd). Let us simplify the model introduced in Example 1 by removing the (in-city ?Cty ?Loc) predicate. The simplified model is a submodel of the original one. Now, let us add a macro-operator move-load(?Pkg ?Truck ?Loc1 ?Loc2 ?Cty) encapsulating the sequence of move and load operators into the simplified model. The simplified model is a submodel of the "macro" model. Finally, we can compare the "macro" Logistics model with the "passenger" model from Example 1. The models are not strongly equivalent. We can, on the other hand, find their maximum common strongly equivalent submodels, i.e., the simplified Logistic model and the "passenger" model which is simplified by removing (in-city ?Cty ?Loc).

5 Comparing Domain Models via ASP

In this section, we describe our approach based on Answer Set Programming (ASP), and its results. Note that the ASP terminology may be not perfectly aligned to the one of planning in the usage of some terms, e.g., atoms and predicates.

Algorithm 1: Comparing Domain Models

```
Input  : A graph $\mathcal{G}_1$ and graph $\mathcal{G}_2$.
Output: Differences between $\mathcal{G}_1$ and $\mathcal{G}_2$
1  $\Pi := preprocessing(\mathcal{G}_1, \mathcal{G}_2)$;
2  $\Pi := \Pi \cup \Pi'$ ;                          // $\Pi'$ reported in Figure 1
3  $A := ASPSolver(\Pi)$;
4  for $p \in A$ do
5      if $p$ is $ver(, add, g, name, \ldots)$ then
6          Print("Add vertex " + $name$ + " in " + $g$);
7      if $p$ is $edge(add, g, n1, n2, t, l)$ then
8          Print("Add " + $t$ + " edge with label " + $l$ + " from vertex " + $n1$ + " to " + $n2$ +
           " in " + $g$ );
9      if $p$ is $map(n1, n2, e1, e2)$ then
10         Print("Map vertex " + $n1$ + " of $g_2$ to vertex " + $n2$ + "of $g_1$");
11         if $e1 \neq e2$ then
12             Print("Remapping " + $e1$ + " to " + $e2$);
```

```
r1 {map(ID1, ID2, N1, N2) : ver(ID2, _, gr1, _, T, P, N1), ver(ID1, _, gr2, _, T, P,
        N2)} = 1 :- ver(ID2, _, gr2, _, T, _, _).
r2 :- ver(ID, _, gr1, _, _, _, _), #count{X: map(X, ID, _, _)} != 1.
r3 :- #sum{1,ID: ver(ID, _, gr2, _, _, _, _); -1,ID: ver(ID, _, gr1, _, _, _, _)} !=
        0.
r4 edge(add, gr1, X2, X4, L, R) :- map(X1, X2, _, _), map(X3, X4, _, _), edge(orig,
        gr2, X1, X3, L, R), not edge(orig, gr1, X2, X4, L, R).
r5 edge(add, gr2, X1, X3, L, R) :- map(X1, X2, _, _), map(X3, X4, _, _), edge(orig,
        gr1, X2, X4, L, R), not edge(orig, gr2, X1, X3, L, R).
r6 :~ ver(ID, orig, G, _, _, _, N1), ver(ID, orig, G, _, _, _, N2), N1 != N2.
        [1@1,ID]
r7 :~ ver(ID, add, G, _, _, _, _). [1@2, ID, G]
r8 :~ edge(add, G, N1, N2, L, R). [1@2, ID, G, N1, N2, L, R]
```

Fig. 1. ASP Program Π'.

Answer Set Programming. ASP is a well-known declarative language. An ASP program [6] is made of (a combination of): (1) facts of the form `head.`; (2) rules of the form `head :- body.`; (3) choice rules of the form `tomsatoms = 1 :- body.`; (4) constraints of the form `:- body.`; and (5) weak constraints of the form `:~ body. [weight@level,terms]`; where `head` is an atom, `atoms` is a set of atoms, and `body` is a set of (possibly negated) atoms, also including aggregate functions, such as `#sum`, and `terms` is a sequence of terms, i.e., variables (strings starting with uppercase letter) or constants (non-negative integers or strings starting with lowercase letters). Atoms can be made over terms. The semantics is given in terms of its *answer sets*, that is, sets A of ground atoms, where atoms in A are said to be true (false, otherwise), such that: (1) `head` is in A; (2) whenever the `body` is true (i.e., all positive atoms are in A and all negated atoms are not in A), `head` is in A; (3) exactly one of the atoms in `atoms` is in A whenever the `body` is true; or if `= 1` is omitted then one of the atoms in `atoms` can be in A whenever the `body` is true; (4) the `body` must be false. Moreover, weak constraints of the form (5) allow expressing preferences among answer sets, where `level` represents the priority and `weight` is a numerical cost that is paid whenever the body of a weak constraint is true w.r.t. an answer set. Overall, the preferred weak constraints are the ones with the lowest costs at the highest levels. For formal details about syntax and semantics of ASP programs, the reader is referred to [5,6].

ASP-based Comparison. Following the theory presented in previous sections, we implemented an ASP-based approach depicted in Algorithm 1. The algorithm receives two LDMGs (referred to as $\mathcal{G}_1$ and $\mathcal{G}_2$) as input, and prints a minimal number of changes to the graphs to make them isomorphic as output. In the following, we assume that the number of vertices representing an operator (resp. a predicate) of $\mathcal{G}_1$ is less than or equal to the number of vertices representing an operator (resp. a predicate) of $\mathcal{G}_2$. The idea of the algorithm is as follows: Firstly, a processing step creates an ASP program Π starting from the input graphs; then, Π is combined with the ASP encoding reported in Fig. 1, and an ASP solver is invoked on the resulting program. Finally, the output of the ASP solver is processed by a postprocessing part which produces human-readable instructions to make the two graphs isomorphic. In more detail, the ASP program operates on atoms over the predicates ver, $edge$, and map, as follows. Atoms of the form $ver(id,\ status,\ graph,\ name,\ type,\ parameters,\ changes)$ denote the vertices of the input graphs, where id is a unique identifier of the vertex, $status$ denotes if the vertex was in the input graph ($orig$) or if it must be added (add), $graph$ indicates the graph of the vertex (between $\mathcal{G}_1$ and $\mathcal{G}_2$), $name$ is the name of the vertex, $type$ indicates if vertex is a predicate or an operator, $parameters$ is a string representing the parameters of the predicate, and $changes$ indicates if (and how) the parameters of the vertex must be changed for a correct match. Atoms of the form $edge(status,\ graph,\ id\_ver1,\ id\_ver2,\ type,\ label)$ denote the edges of the input graphs, where $status$ and $graph$ are as the ones of ver, id_ver1 and id_ver2 are the identifiers of the vertices connected by the edges, $type$ denotes if the edge is in E_{pre}, E_{del}, or E_{add}, respectively, and $label$ is the label of the edge. Atoms of the form $map(id\_ver1, id\_ver2, params1, params2)$ denote the mapping from vertices of the different graphs, where id_ver1 and id_ver2 belong to $\mathcal{G}_2$ and $\mathcal{G}_1$, respectively, $params1$ and $params2$ denote the parameters of the two matched vertices, respectively. The preprocessing step creates the rules:

(1) $ver(id,\ orig,\ gr_i,\ name,\ t,\ \text{“}x_1, \ldots, x_k\text{”}, \text{“}x_1, \ldots, x_k\text{”}).$
for each vertex $name(x_1, \ldots, x_k)$ in $\mathcal{G}_i (i \in \{1, 2\})$ representing a vertex of type t, where t is $predicate$ or $operator$, and id is an identifier of the vertex;

(2) $edge(orig,\ gr_i,\ v_1,\ v_2,\ x,\ label).$
for each edge (v_1, v_2) in $E_x (x \in \{pre, del, add\})$ of the graph $\mathcal{G}_i (i \in \{1, 2\})$, where $label$ is the label of the edge;

(3) $\{ver(id,\ add,\ gr_1,\ name,\ t,\ \text{“}x_1, \ldots, x_k\text{”}, \text{“}x_1, \ldots, x_k\text{”})\}.$
for each vertex $name(x_1, \ldots, x_k)$ in $\mathcal{G}_2$ of type t such that there is no vertex $name(x_1, \ldots, x_k)$ of type t in $\mathcal{G}_1$, where t can be either $predicate$ or $operator$;

(4) $\{ver(id, orig, gr_1, name, t, \text{“}x_1, \ldots, x_k\text{”}, \text{“}y_1, \ldots, y_z\text{”})\}.$
for each vertex $name(x_1, \ldots, x_k)$ in $\mathcal{G}_1$ of type t and for each vertex $name(y_1, \ldots, y_z)$ in $\mathcal{G}_2$ of type t, with the set of the parameters $x_1, \ldots, x_k$ different from the set of the parameters $y_1, \ldots, y_z$, where t can be either $predicate$ or $operator$.

The program Π produced by the preprocessing is combined with the ASP encoding of Fig. 1, whose behaviour is described in the following. Rules r_1 and r_2 associate each vertex of $\mathcal{G}_2$ to exactly one vertex of the same type of $\mathcal{G}_1$. Rule r_3 ensures that the number of vertices of the two graphs are the same. Rules r_4 and r_5 generate missing edges between the vertices after the mapping. Finally, weak constraints r_6, r_7, and r_8 minimise the number of vertices with different parameters, i.e., the number of added vertices, and the number of added edges, respectively. Observe that weak constraints r_7, and r_8 have a higher level than r_6, that is, preserving the same number of vertices and edges has a higher priority than changing the parameters of the vertices.

An ASP solver is then executed on the resulting program, and its output is processed to produce human-readable instructions on how to make the two graphs isomorphic.

Example 3 (Example 2 cont'd)**.** Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be the LDMGs of the Logistics domains considered in Example 1 and the "macro" variant with the incity predicate (see Example 2), respectively. For the sake of compactness, we shorten the name of variables of STRIPS predicates and operators to a single capital letter. Thus, $\mathcal{G}_2$ consists of the vertices of $\mathcal{G}_1$ extended with a vertex of the form $\mathit{moveload}(?P, ?T, ?L1, ?L2, ?C)$. Moreover, $E^2_t = E^1_t \cup E'_t$ ($t \in \{\mathit{pre}, \mathit{del}, \mathit{add}\}$), where E'_{pre} is

$$\begin{aligned}\{&(\mathit{moveload}, ?T =?T, ?L1 =?L, \mathit{attruck}),\\ &(\mathit{moveload}, ?L1 =?L, ?C =?C, \mathit{incity}),\\ &(\mathit{moveload}, ?L2 =?L, ?C =?C, \mathit{incity}),\\ &(\mathit{moveload}, ?P =?P, ?L1 =?L, \mathit{atpackage})\}\end{aligned}$$

E'_{add} is $\{(\mathit{moveload}, ?T =?T, ?L1 =?L, \mathit{atpackage}), (\mathit{moveload}, ?P =?P, ?T =?T, \mathit{inpackage})\}$ and E'_{del} is $\{(\mathit{moveload}, ?T =?T, ?L1 =?L, \mathit{attruck}), (\mathit{moveload}, ?P =?P, ?L1 =?L, \mathit{atpackage})\}$. Note that rules (1) and (2) produced by the preprocessing step of Algorithm 1 encode the vertices and the edge, for instance $\mathit{ver}(0, \mathit{orig}, \mathit{gr}_1, \mathit{load}, \mathit{operator}, \text{“}L, P, T'', \text{“}L, P, T'')$ represents the vertex load of $\mathcal{G}_1$, where 0 is a unique identifier of the vertex, and "L, P, T" corresponds to the (ordered) parameters. Moreover, there is only one rule of type (3), i.e., $\{\mathit{ver}(7, \mathit{add}, \mathit{gr}_1, \mathit{moveload}, \mathit{operator}, \text{“}C, L, L, P, T\text{”}, \text{“}C, L, L, P, T\text{”})\}$, since there is only one vertex in $\mathcal{G}_2$ that is not in $\mathcal{G}_1$. Finally, an excerpt of the rules of type (4), are the following (where or, o, and p stand for orig, $\mathit{operator}$, and $\mathit{predicate}$, respectively):

$$\{ver(0, or, gr_1, load, o, \text{“}L, P, T\text{”}, \text{“}C, L, L, T\text{”})\}.$$
$$\{ver(0, or, gr_1, load, o, \text{“}L, P, T\text{”}, \text{“}C, L, L, P, T\text{”})\}.$$
$$\{ver(1, or, gr_1, unload, o, \text{“}L, P, T\text{”}, \text{“}C, L, L, T\text{”})\}.$$
$$\{ver(1, or, gr_1, unload, o, \text{“}L, P, T\text{”}, \text{“}C, L, L, P, T\text{”})\}.$$
$$\{ver(2, or, gr_1, move, o, \text{“}C, L, L, T\text{”}, \text{“}L, P, T\text{”})\}.$$
$$\{ver(2, or, gr_1, move, o, \text{“}C, L, L, T\text{”}, \text{“}C, L, L, P, T\text{”})\}.$$
$$\{ver(3, or, gr_1, attruck, p, \text{“}L, T\text{”}, \text{“}L, P\text{”})\}.$$
$$\{ver(3, or, gr_1, attruck, p, \text{“}L, T\text{”}, \text{“}P, T\text{”})\}.$$
$$\{ver(3, or, gr_1, attruck, p, \text{“}L, T\text{”}, \text{“}C, L\text{”})\}.$$
$$\{ver(4, or, gr_1, atpackage, p, \text{“}L, P\text{”}, \text{“}L, T\text{”})\}.$$
$$\{ver(4, or, gr_1, atpackage, p, \text{“}L, P\text{”}, \text{“}P, T\text{”})\}.$$
$$\{ver(4, or, gr_1, atpackage, p, \text{“}L, P\text{”}, \text{“}C, L\text{”})\}.$$
$$\{ver(5, or, gr_1, inpackage, p, \text{“}P, T\text{”}, \text{“}L, P\text{”})\}.$$
$$\{ver(5, or, gr_1, inpackage, p, \text{“}P, T\text{”}, \text{“}L, T\text{”})\}.$$
$$\{ver(5, or, gr_1, inpackage, p, \text{“}P, T\text{”}, \text{“}C, L\text{”})\}.$$
$$\{ver(6, or, gr_1, incity, p, \text{“}C, L\text{”}, \text{“}L, P\text{”})\}.$$
$$\{ver(6, or, gr_1, incity, p, \text{“}C, L\text{”}, \text{“}L, T\text{”})\}.$$
$$\{ver(6, or, gr_1, incity, p, \text{“}C, L\text{”}, \text{“}P, T\text{”})\}.$$

The preprocessing step allows generating only the meaningful combinations of vertices and terms in a graph via imperative programming. These combinations are added as choice rules, which can then be utilised by the solver as possible newly added vertices. Without preprocessing, all choice rules for mapping all terms/operator combinations are instead generated, and is left to the solver to derive new vertices by combining original vertices with every choice to potentially change terms. After the preprocessing (line 1 of Algorithm 1), the ASP solver is executed (line 3) on the resulting program extended with Π' (line 2). Then, Algorithm 1 analyses its output (from line 4 on) and produces the following instructions:

- Add vertex $moveload$ in g_1.
- Add pre edge with label T=T,L1=L from vertex $moveload$ to vertex $attruck$ in g_1.
- Add pre edge with label L1=L,C=C from vertex $moveload$ to vertex $incity$ in g_1.
- Add pre edge with label L2=L,C=C from vertex $moveload$ to vertex $incity$ in g_1.
- Add pre edge with label P=P,L1=L from vertex $moveload$ to vertex $atpackage$ in g_1.
- Add add edge with label T=T,L1=L from vertex $moveload$ to vertex $attruck$ in g_1.
- Add add edge with label P=P,T=T from vertex $moveload$ to vertex $inpackage$ in g_1.
- Add del edge with label T=T,L1=L from vertex $moveload$ to vertex $attruck$ in g_1.
- Add del edge with label P=P,L1=L from vertex $moveload$ to vertex $atpackage$ in g_1.
- Map vertex v of g_2 to vertex v of g_1, where $v \in \{load, unload, move, attruck, atpackage, inpackage, incity, moveload\}$.

Finally, note how Algorithm 1 can be easily extended in order to deal with the generation of multiple answer sets, corresponding to minimal sets of changes.

Evaluation. We selected 7 different domain models from well-known international competitions. In particular, we considered the domains of Barman, Blocksworld, (simplified) Logistics, Rovers, Satellite, and Sokoban from various editions of the International Planning Competition (IPC), and the RPG domain from the 2016 International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS).[1] The number of operators ranges between 2 and 12, and the number of predicates between 4 and 25. To obtain a different model for each benchmark domain, but RPG, we reformulated the original models by considering a mix of entanglements and macro-actions, and by modifying preconditions and effects of original operators. For the RPG domain, we compared two models crafted by two of the teams that took part in the competition: Such models present significant differences in terms of predicates and operators as they embody very different interpretations of the domain at hand. Details of the LDMGs corresponding to the models are shown in the left part of Table 1. Year indicates when the model was introduced. Finally, to perform a stress test of the proposed approach, we compared models designed for diverse domains too. Table is divided horizontally in two parts: The top part considering cases where a model has been reformulated, while the bottom part focuses on comparing very different models (RPG, Barman vs Logistics, Rovers vs Satellite). We performed experiments on an Intel Core i5-10210U machine with 1.6 GHz, 8 GB of RAM and Linux operating system. Each system run was given an overall memory limit of 6 GB and 5 CPU-time minutes. As ASP system we used the state-of-the-art tool CLINGO [13], configured with the option `--parallel-mode=4`, which enables the use of multiple threads (with different solving strategies). We used 4 threads and in our experiments this helps improve the performance of Clingo compared to the default configuration. Moreover, we used the open-source python library PYSPEL [2] which simplifies the implementation of Algorithm 1. We tested two approaches: The first one is the implementation of Algorithm 1 as described in Sect. 5, and the second one is the same implementation where rules (3) and (4) of the preprocessing (line 1 of Algorithm 1) are produced using plain ASP rules. We preliminary tested, on Logistics and Rover domains, the ability of our solution to compare models that are exactly the same but for the names of the involved operators and predicates, i.e., if they are strongly equivalent. The results indicate that the ASP solution employing preprocessing always identifies the compared graphs as isomorphic, and provides an appropriate mapping between the nodes of the compared LDMGs, in less than 0.5 CPU-time seconds.

Then, we move to the general case and the results of the experimental analysis are shown in the right part of Table 1 where, for each domain and tested approach, we report the number of optimal models found, the CPU time, and the number of rules generated by the grounding. Square brackets indicate that an optimal solution has been found in the reported CPU time (checked manually), but has not been proved by CLINGO. As a first observation, the approach employing preprocessing is extremely fast, since for all the tested benchmarks we are able to find an optimal solution (through not proved) in less than 1 CPU-time second. Instead, plain ASP encoding exceeds the memory limits when executed on large domains (Barman and Rovers) or in the presence of significant difference between the compared domains, showing that preprocessing is

[1] Encoding and benchmarks are available at: https://github.com/MarcoMochi/jelia-planning.

Table 1. Size of the generated graphs for each benchmark domain model (Left) and performance of the proposed ASP-based approach without/with preprocessing (Right). Vertices and Edges give information of the size of the graphs to compare, $\mathcal{G}_1$ is the graph obtained by considering the original domain model, $\mathcal{G}_2$ is obtained from the reformulated model for all the domains but RPG, where two original models independently crafted are compared. RPG is the only domain presented at ICKEPS, while other domains have been used as IPC benchmarks. Results are presented in terms of seconds needed to enumerate all the optimal solutions, and the number of optimal models (Opt Mod.). Square brackets indicate that the optimal solution was not proved. (# Rules) shows the size of the ASP program.

Domain	Year	Vertices		Edges		Opt. Mod.	No preprocessing		Preprocessing	
		$\mathcal{G}_1$	$\mathcal{G}_2$	$\mathcal{G}_1$	$\mathcal{G}_2$		CPU Time	# Rules	CPU Time	# Rules
Sokoban	2008	6	7	16	18	1	0.1	1,411	0.1	636
Logistics	1998	7	8	13	21	1	0.1	832	0.1	697
Blocksworld	2000	9	11	27	29	2	0.1	3,219	0.1	2,446
Satellite	2002	13	16	23	44	2	2.7	2,132,131	0.1	10,666
Barman	2011	27	29	97	123	1	–	–	0.3	98,676
Rovers	2002	34	39	75	103	4	–	–	0.4	146,857
RPG	2016	13	34	23	74	[1]	[1.9]	849,732	[0.4]	44,601
Barman_Log	–	–	–	–	–	[1]	–	–	[0.1]	93,893
Rovers_Sat	–	–	–	–	–	[1]	–	–	[0.8]	37,251

indeed necessary in challenging cases. Note that even the mid-size Satellite leads to more than 2 Million rules, whereas the approach using preprocessing produces only around 10 thousands rules. In models where differences are limited, (i.e., modification of 2-3 vertices and a few tens of edges), there is no substantial difference between finding one optimal solution and finding all of them. This result can be explained by the fact that the number of optimal solutions is rather small (maximum 4) and this paves the way for the development of more comparison techniques, e.g., by proposing preferences among the possible solutions.

Turning our attention more on the stress test, it is easy to notice that despite the fact that the optimal solution for models of different domains required over 100 modifications/additional elements (nodes and edges) and for the RPG domain required 4 additional nodes and 78 edges, the proposed solution was able to generate an optimal result in less than 1 CPU-time second. However, differently from the other tests, in these cases the approach was able to generate a single optimal solution and not to enumerate all the optimal ones. On the one hand, this result confirms that ASP is a viable tool to be used; on the other hand, it may suggest that additional optimisation could be beneficial for fully enumerate all optimal cases.

Summarising, the performed experimental analysis indicates that the presented ASP system, enhanced with the preprocessing step, is very efficient in generating optimally minimal sets of modifications that allows to transform a model into the compared one, on the basis of the corresponding LDMGs. Considering that results are generated efficiently, the proposed tool can also be exploited during the domain encoding step, for comparing alternative representations of a domain's dynamics.

6 Related Work and Discussion

The notion of strong domain model equivalence has been informally introduced in [24]. On a similar note, [25] proposed an automated approach, D-VAL, to check the functional equivalence of two domain models, i.e., their ability to parse the same set of problems. D-VAL focuses on comparing models of the same domain that has been reformulated, to ensure that the reformulation process did not undermine the domain model capabilities. The approach proposed in this paper is more general, and allows to compare even very different models in terms of their corresponding solution spaces, and to obtain a measure of their similarity.

An application-specific investigation of the engineering of different models has been proposed in [23], and ICKEPS introduced metrics to manually compare models [9].

Notably, an approach based on ASP has been proposed also for the MRP in planning [19], while [26] deals with the MRP but specifically defined on two logic programs and their answer sets. Some authors of [26] followed a similar direction in [18], but in the context of (numerical) scheduling and employing CLINGO-DL language, which is an extension of ASP enriched with a limited form of arithmetic [15].

7 Conclusion

This paper contributes to theory and practice of the problem of comparing planning domain models: We defined the concept of similarity of domain models, which builds on the notion of strong equivalence of domain models, also introduced in the paper. We proposed an approach based on Answer Set Programming for specifying and solving the problem – with particular attention given to the identification of optimal minimal sets of modifications that allows to transform one model into the compared one. Experiments on well-known planning benchmarks of different size show that the approach can find a minimal set of corrections in very short time. Future work will focus on extending the approach to more expressive planning representation languages, such as PDDL+ [12], that can also consider hybrid discrete-continuous numeric changes, and to improve the explanations provided as output.

Acknowledgements. L. Chrpa was funded by the Czech Science Foundation (project no. 23-05575S). M. Vallati was supported by the UKRI Future Leaders Fellowship [grant number MR/T041196/1]. C. Dodaro was supported by Italian Ministry of Research (MUR) under PNRR projects FAIR “Future AI Research”, CUP H23C22000860006, and Tech4You “Technologies for climate change adaptation and quality of life improvement”, CUP H23C22000370006;

References

1. Ai-Chang, M., et al.: MAPGEN: mixed-initiative planning and scheduling for the mars exploration rover mission. IEEE Intell. Syst. **19**(1), 8–12 (2004)
2. Alviano, M., Dodaro, C., Previti, A.: Python Specification Language (2021). https://github.com/dodaro/pyspel
3. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge (2003). https://doi.org/10.1017/CBO9780511543357

4. Bengoetxea, E.: Inexact Graph Matching Using Estimation of Distribution Algorithms. Ph.D. thesis, Ecole Nationale Supérieure des Télécommunications, Paris, France, December 2002
5. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011)
6. Calimeri, F., et al.: ASP-Core-2 input language format. Theory Pract. Log. Program. **20**(2), 294–309 (2020)
7. Cardellini, M., Maratea, M., Vallati, M., Boleto, G., Oneto, L.: In-station train dispatching: a PDDL+ planning approach. In: Proceedings of ICAPS, pp. 450–458 (2021)
8. Chakraborti, T., Sreedharan, S., Zhang, Y., Kambhampati, S.: Plan explanations as model reconciliation: moving beyond explanation as soliloquy. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI, pp. 156–163 (2017)
9. Chrpa, L., McCluskey, T.L., Vallati, M., Vaquero, T.: The fifth international competition on knowledge engineering for planning and scheduling: summary and trends. AI Mag. **38**(1), 104–106 (2017)
10. Coulter, A., Ilie, T., Tibando, R., Muise, C.: Theory alignment via a classical encoding of regular bisimulation. In: Workshop on Knowledge Engineering for Planning and Scheduling (KEPS) (2022)
11. Cresswell, S., McCluskey, T.L., West, M.M.: Acquiring planning domain models using LOCM. Knowl. Eng. Rev. **28**(2), 195–213 (2013)
12. Fox, M., Long, D.: Modelling mixed discrete-continuous domains for planning. J. Artif. Intell. Res. **27**, 235–297 (2006)
13. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Wanko, P.: Theory solving made easy with Clingo 5. In: ICLP (Technical Communications). OASICS, vol. 52, pp. 2:1–2:15 (2016)
14. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. N. Gener. Comput. **9**(3/4), 365–386 (1991)
15. Janhunen, T., Kaminski, R., Ostrowski, M., Schellhorn, S., Wanko, P., Schaub, T.: Clingo goes linear constraints over reals and integers. Theory Pract. Log. Program. **17**(5–6), 872–888 (2017)
16. McCluskey, T.L., Porteous, J.M.: Engineering and compiling planning domain models to promote validity and efficiency. Artif. Intell. **95**(1), 1–65 (1997)
17. McCluskey, T.L., Vaquero, T.S., Vallati, M.: Engineering knowledge for automated planning: towards a notion of quality. In: Proceedings of K-CAP, pp. 14:1–14:8 (2017)
18. Nguyen, V., Son, T.C., Yeoh, W.: Explainable problem in clingo-dl programs. In: Ma, H., Serina, I. (eds.) Proceedings of the Fourteenth International Symposium on Combinatorial Search (SOCS 2021), pp. 231–232. AAAI Press (2021)
19. Nguyen, V., Stylianos, V.L., Son, T.C., Yeoh, W.: Explainable planning using answer set programming. In: Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR, pp. 662–666 (2020)
20. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. **25**(3–4), 241–273 (1999)
21. Ramírez, M., et al.: Integrated hybrid planning and programmed control for real time UAV maneuvering. In: Proceedings of the AAMAS, pp. 1318–1326 (2018)
22. Sanfeliu, A., Fu, K.: A distance measure between attributed relational graphs for pattern recognition. IEEE Trans. Syst. Man Cybern. **13**(3), 353–362 (1983). https://doi.org/10.1109/TSMC.1983.6313167
23. Shah, M.M.S., Chrpa, L., Kitchin, D.E., McCluskey, T.L., Vallati, M.: Exploring knowledge engineering strategies in designing and modelling a road traffic accident management domain. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence, pp. 2373–2379 (2013)

24. Shoeeb, S., McCluskey, T.: On comparing planning domain models. In: PlanSIG Workshop (2011)
25. Shrinah, A., Long, D., Eder, K.: D-VAL: an automatic functional equivalence validation tool for planning domain models. arXiv preprint arXiv:2104.14602 (2021)
26. Son, T.C., Nguyen, V., Vasileiou, S.L., Yeoh, W.: Model reconciliation in logic programs. In: Faber, W., Friedrich, G., Gebser, M., Morak, M. (eds.) JELIA 2021. LNCS (LNAI), vol. 12678, pp. 393–406. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75775-5_26
27. Vallati, M., Chrpa, L.: On the robustness of domain-independent planning engines: the impact of poorly-engineered knowledge. In: Proceedings of K-CAP, pp. 197–204 (2019)
28. Vallati, M., McCluskey, T.L.: A quality framework for automated planning knowledge models. In: Proceedings of the 13th International Conference on Agents and Artificial Intelligence, ICAART, pp. 635–644 (2021)
29. Zeng, Z., Tung, A.K.H., Wang, J., Feng, J., Zhou, L.: Comparing stars: on approximating graph edit distance. Proc. VLDB Endow. **2**(1), 25–36 (2009)