Explainable Planning Using Answer Set Programming

Van Nguyen[1] , **Stylianos Loukas Vasileiou**[2] , **Tran Cao Son**[1] , **William Yeoh**[2]
[1]New Mexico State University
[2]Washington University in St. Louis
{vnguyen,tson}@cs.nmsu.edu, {v.stylianos,wyeoh}@wustl.edu

Abstract

In human-aware planning problems, the planning agent may need to explain its plan to a human user, especially when the plan appears infeasible or suboptimal for the user. A popular approach to do so is called *model reconciliation*, where the planning agent tries to reconcile the differences between its model and the model of the user such that its plan is also feasible and optimal to the user. This problem can be viewed as an optimization problem, where the goal is to find a *subset-minimal explanation* that one can use to modify the model of the user such that the plan of the agent is also feasible and optimal to the user. This paper presents an algorithm for solving such problems using answer set programming.

1 Introduction

In human-aware planning problems (Kambhampati 2019), the planning agent, which we refer to as a robot in this paper, needs to find ways to ensure that its plans are understood and accepted by human users. A typical assumption is that the model or knowledge base of the robot differs from that of the user. As such, a plan that is optimal to the robot may be suboptimal or, worse, infeasible to the user. A popular approach to solve this problem is called the *model reconciliation problem* (MRP), where the robot needs to provide *explanations* to the user and reconcile their two models such that the plan of the robot is also optimal in the reconciled model of the user (Chakraborti *et al.* 2017; Sreedharan *et al.* 2018; Sreedharan *et al.* 2019). A common thread across most of these works is that they, not surprisingly, employ mostly automated planning approaches.

In this paper, we are interested in tackling MRP from the perspective of *knowledge representation and reasoning* (KR), specifically through *answer set programming* (ASP) (Marek and Truszczyński 1999; Niemelä 1999). Our approach is motivated by the fact that the planning models of the robot and the user can be represented as logic programs via answer set planning (Gebser *et al.* 2013). Given that a planning problem can be viewed as a set of facts, we solve MRP by developing different ASP programs, which are then glued together via multi-shot ASP, for computing subset-minimal explanations. We empirically evaluate our ASP-based approach against the current state of the art by Chakraborti *et al.* (2017) on the same benchmark domains. Our empirical results show that our ASP-based approach is faster when the explanations are long.

2 Background: ASP and Planning

Logic Programming: *Answer set programming* (ASP) (Marek and Truszczyński 1999; Niemelä 1999) is a declarative programming paradigm based on logic programming under the answer set semantics.

A logic program π is a set of rules of the form $a_0 \leftarrow a_1, \ldots, a_m$, *not* $a_{m+1}, \ldots,$ *not* a_n, where $0 \leq m \leq n$, each a_i is an atom of a propositional language and *not* represents (default) negation. Semantically, a program π induces a collection of so-called *answer sets*, which are distinguished models of π determined by answer sets semantics; see the work by Gelfond and Lifschitz (1990) for details. ASP is combined with imperative means in *clingo* (Gebser *et al.* 2014) that allows for a more efficient implementation of solvers for problems that requires more than one call to an ASP solver (Son *et al.* 2016; Nguyen *et al.* 2017).

Planning Problems as Facts in ASP: A planning problem (Ghallab *et al.* 1998) is a triple (I, G, D), where I and G encode the initial states of the world and the goal, respectively; and the domain D specifies the actions and their preconditions and effects. Each problem P can be represented as a set of ASP facts. These atoms define object constants, types of objects, actions, the initial state, and the goal state. In this paper, we make use of the representation output by the system available at https://github.com/potassco/plasp. We assume that PDDL problems are typed. The program $\pi(P)$ consists of the following atoms:[1]

(*i*) type description (`type/1`);
(*ii*) constant and fluent declarations (`constant/1` and `variable/1`);
(*iii*) action declaration (`action/1`);
(*iv*) action precondition and effect (`precondition/3`)) and `postcondition/4`);
(*v*) initial state (`initialState/3`); and
(*vi*) goal state (`goal/2`).

[1]The use of double keywords such as *type*(*type*(.)) or *variable*(*variable*(...)) in `plasp` could be simplified. To allow them to be automatically processed, we leave them as they are.

Further details about each atom will be apparent from the use in the next sections.

Explainable Planning: *Explainable planning* as discussed by Chakraborti *et al.* (2017) assumes that a planning problem $P = (I, G, D)$ is given, and it is identical to the model of the robot $P_r = (I_r, G_r, D_r)$. The model of the human $P_h = (I_h, G_h, D_h)$ may be different from the model of the robot. The focus of this paper is in the *model reconciliation process*, i.e., to bring the model of the human closer to the model of the robot by means of explanations in the form of model updates. Given P_r and P_h, a *model reconciliation problem* (MRP) is defined by a tuple $\langle p^*, P_r, P_h \rangle$, where p^* is a cost-minimal solution for P_r. The goal of MRP is to find a solution (i.e., an explanation) ε that can be used to update P_h to $\cap P_h$ such that p^* is also a cost-minimal solution of $\cap P_h$.

This update process involves inserting into P_h (and/or removing from P_h) some initial conditions, action preconditions, action effects, or goals. It is required that the changes in the model of the human must be consistent with the robot's model. ("Consistent" means that if some information (e.g., preconditions or effects) is added to the human's model, then this information must be present in the robot's model; if some information is removed from the human's model, then this information must not be present in the robot's model.)

3 Explainable Planning Using ASP

3.1 Planning Engine

Let $P = (I, G, D)$ denote a planning problem and $\pi(P)$ be its representation in ASP. We now present a program $\pi(n)$ that could be used as an ASP-based planning engine that work with $\pi(P)$ (see, e.g., Lifschitz (2002)). $\pi(n)$, together with $\pi(P)$, can be used to generate all solutions of P with lengths that are no larger than n. It consists of two groups of rules:

- **Reasoning About Effects of Actions: (Listing 1)** Rules in this group ensure that an action can only be executed if all of its conditions are true and all of the effects of the actions become true. We use $h(l,t)$ to denote that the fluent l is true at step t for $1 \leq t \leq n$ and $occurs(a,t)$ to denote that the action a occurs at step t.

Listing 1: Reasoning About Effects of Actions

```
1  h(X,1)   :- initialState(X,value(X,true)).
2  -h(X,1)  :- not initialState(X,value(X,true)).
3  h(X,T+1) :- action(action(A)),occurs(A,T),
4      postcondition(action(A),
5      effect(unconditional),X,value(X,true)).
6  -h(X,T+1) :- action(action(A)),occurs(A,T),
7      postcondition(action(A),
8      effect(unconditional),X,value(X,false)).
9  h(X,T+1)  :- h(X,T), not -h(X,T+1).
10 -h(X,T+1) :- -h(X,T), not h(X,T+1).
11 non_exec(A,T) :- action(action(A)), not h(X,T),
12     precondition(action(A),X,value(X, true)).
13 non_exec(A,T) :- action(action(A)), not -h(X,T),
14     precondition(action(A),X,value(X, false)).
15 :- action(action(A)),occurs(A,T), non_exec(A,T).
```

The first two rules on Lines 1 and 2 encode the initial state. This encoding employs the Closed-World-Assumption (Reiter 1978). The next two rules on Lines 3-5 and 6-8 define the effect of an action. The rules on Lines 9 and 10 encode the inertia principle. The rules on Lines 11-12 and 13-14 define the predicate *non_exec(a,t)*, which states when an action cannot be executed. The constraint on Line 15 prevents non-executable actions from occurring.

- **Goal Enforcement and Action Generation: (Listing 2)** The rule on Line 1 generates action occurrences. The rules on Lines 2 and 3 specify that the goal is not achieved at time step T if one of the subgoals has not been achieved. The rules on Lines 4 and 5 enforce that the goal must be satisfied at the end of the horizon, at time step n.

Listing 2: Goal Enforcement and Action Generation

```
1  1{occurs(A,T):action(action(A))}1:-nok(T).
2  nok(T):-goal(X,value(X, true)),not h(X,T).
3  nok(T):-goal(X,value(X, false)),not -h(X,T).
4  goal :- not nok(n).
5  :- not goal.
```

Let S be a set of atoms in $\pi(P) \cup \pi(n)$ and $plan(S)$ denote the sequence $[a_{t_1}, \ldots, a_{t_k}]$ such that $occurs(a_{t_i}, t_i) \in S$, where $t_1 < \ldots < t_k$, and for every $occurs(a_i, i) \in S$, $i \in \{t_1, \ldots, t_k\}$. For a plan $p = [a_1, \ldots, a_t]$, $occurs^*(p) = \{occurs(a_i, i) \mid i = 1, \ldots, t\}$. It can be shown that for each answer set A of $\pi(P) \cup \pi(n)$, $plan(A)$ is a solution of $P = (I, G, D)$; and if $p = [a_1, \ldots, a_t]$ with $t < n$ is a solution of (I, G, D), then $\pi(P) \cup \pi(n) \cup occurs^*(p)$ has an answer set A such that $p = plan(A)$. $\pi(P) \cup \pi(n)$ has p as a solution if $\pi(P) \cup \pi(n) \cup occurs^*(p)$ has an answer set. We use $\pi(P)$ to compute minimal length plan of P by computing answer set of $\pi(P) \cup \pi(k)$ for $k = 1, \ldots$, and stop when the first answer set is found.

3.2 Computing Optimal Solution of MRPs

Consider an MRP $M = (p^*, P_r, P_h)$, where $P_r = (I_r, G, D_r)$, $P_h = (I_h, G, D_h)$, and $p^* = [a_1, a_2, ..., a_k]$ is an optimal (in terms of length) solution of P_r but not an optimal solution of P_h. An *explanation* for M is a pair $(\varepsilon^+, \varepsilon^-)$, where $\varepsilon^+ \subseteq I_r \cup D_r$ and $\varepsilon^- \subseteq I_h \cup D_h$, that can be used to update P_h to $\hat{P}_h = (\hat{I}_h, G, \hat{D}_h)$ such that p^* is an optimal plan of $\hat{P}_h$. An explanation $(\varepsilon^+, \varepsilon^-)$ is *optimal* if there exists no explanation (θ^+, θ^-) such that $\theta^+ \cup \theta^-$ is a proper subset of $\varepsilon^+ \cup \varepsilon^-$.

In the updated $\hat{P}_h$, $\hat{I}_h = (I_h \setminus (\varepsilon^- \cap I_h)) \cup (\varepsilon^+ \cap I_r)$ and $\hat{D}_h = D_h \setminus (\varepsilon^- \cap D_h) \cup (\varepsilon^+ \cap D_r)$. Let $pre(x, D_r)$ and $post(x, D_r)$ be the collection of preconditions and postconditions, respectively, of the action x in D_r.

Algorithm 1 gives an overview of the process for computing a solution for a problem (p^*, P_r, P_h) and makes use of three ASP programs:

(*i*) π_1: It computes the explanation ε of a plan (p^*) with respect to a planning domain (P_r);

(*ii*) π_2: It updates a planning domain (P_h) with changes (from ε) and computes an optimal solution; and

Algorithm 1: `Explanation`(P_r, P_h, p^*)

Input: A MRP: (p^*, P_r, P_h), k is the length of p^*
Output: An explanation $(\varepsilon^+, \varepsilon^-)$

1 Compute the first set of explanation $\varepsilon = (\varepsilon^+, \varepsilon^-)$ using π_1
2 Update P_h with ε to create $\hat{P}_h$
3 % $\hat{P}_h$ will have p^* as one of its solutions
4 **while** $\hat{P}_h$ *has optimal plan of length n* **do**
5 **if** $n == k$ **then**
6 **break**
7 **else**
8 **for** *each action b in p* **do**
9 **if** *b in* D_r **then**
10 $\varepsilon^+ = \varepsilon^+ \cup pre(b, D_r) \cup post(b, D_r)$
11 **else**
12 $\varepsilon^- = \varepsilon^- \cup pre(b, D_h) \cup post(b, D_h)$
13 Update P_h with ε to create $\hat{P}_h$
14 Compute minimal $(\varepsilon^+, \varepsilon^-)$ using π_3
15 **return** $(\varepsilon^+, \varepsilon^-)$

(*iii*) π_3: Given a set of potential changes (ε) to a problem (P_h) that guarantee that the problem does not have an optimal plan shorter than the length of p^*, it computes a set of minimal changes that still guarantee this property.

π_1 is used to compute the initial explanation (Line 1). Given a plan, its explanation is essentially all the preconditions and effects of the actions in the plan.

π_2 is used to update P_h with ε, create $\hat{P}_h$, and check whether $\hat{P}_h$ has a plan of length smaller than the length of p^* (Lines 2-4). If it is the case, then it updates ε and repeats (**while-loop**, Lines 4-13).

Finally, π_3 is used to find a minimal set of changes and return it (Lines 14-15).

As we will describe next, π_1, π_2, and π_3 share some subprograms. Thus, to avoid grounding a sub-program multiple times, Lines 4-13 employs a multi-shot solver feature of `clingo`. The computation of the set of explanations (Lines 10, 12 and 14) is purely encoded in the logic program in Listing 6. We next describe these programs in detail.

π_2: Computing An Explanation: The program π_1 is divided into four programs: π_{select}, π_{update}, $\pi_{engine} = \pi(n)$ (as described in the previous subsection, with the modification that *postcondition(.)* and *precondition(.)* are changed to *true(postcondition(.))* and *true(precondition(.))*), and $\pi_{optimal}$. Given (p^*, P_r, P_h), π_1 computes changes to P_h so that p^* is a plan of $\hat{P}_h$.

The program π_{select} below has as inputs p^*, D_h, and D_r, which are represented by the set of atoms of the form $occurs(a,t)$, $human(l)$ for $l \in \pi(D_h)$, and $robot(l)$ for $l \in \pi(D_r)$, respectively. Lines 1-2 set up the time steps from p^*. Lines 3-4 (5-6) encode a choice rule which states that an action in D_r (D_h) can be added to (removed from) D_h if it is not (is) in D_h (e.g., $add(action(a))$ is true means that action a from D_r is added to D_h). Lines 7-15 (16-23) encode a choice rule that defines $add/1$ ($remove/1$) for adding (removing) preconditions and postconditions from D_r (D_h). Lines 24-34 are similar to Lines 7-23 and are for adding/deleting information to/from the initial state I_h.

Listing 3: Program π_{select}

```
1  maxTime(N) :- N = #max {T : time(T)}.
2  preTime(T) :- time(T), maxTime(N), T < N.
3  {add(action(A))} :- robot(occurs(A,_)),
4    not human(action(A)).
5  {remove(action(A))} :- human(action(A)),
6    not robot(action(A)).
7  {add(postcondition(action(A),B,X,value(X,BoolV)))} :-
8    robot(postcondition(action(A),B,X,
9    value(X,BoolV))), robot(occurs(A,_)),
10   not human(postcondition(action(A),B,X,
11   value(X, BoolV))).
12 {add(precondition(action(A),X,value(X,BoolV)))} :-
13   robot(precondition(action(A),X,value(X,BoolV))),
14   robot(occurs(A,_)), not human(precondition(action(A),
15   X, value(X, BoolV))).
16 {remove(postcondition(action(A),B,X,value(X,BoolV)))}
17   :- human(postcondition(action(A),B,X,value(X, BoolV))),
18   contradictory(BoolV,NBoolV),
19   robot(postcondition(action(A),B,X,value(X, NBoolV))).
20 {remove(precondition(action(A),X,value(X, BoolV)))} :-
21   human(precondition(action(A),X,
22   value(X,BoolV))), contradictory(BoolV, NBoolV),
23   robot(precondition(action(A), X, value(X, NBoolV))).
24 {add(initialState(X,value(X,true)))} :-
25   robot(initialState(X,value(X,true))),
26   human(initialState(X,value(X,false))).
27 {remove(initialState(X,value(X,true)))} :-
28   robot(initialState(X,value(X,false))),
29   human(initialState(X,value(X,true))).
30 h(X,1) :- robot(initialState(X,value(X,true))),
31   human(initialState(X,value(X,true))).
32 h(X,1) :- human(initialState(X,value(X,true))),
33   not remove(initialState(X,value(X,true))).
34 h(X,1):- add(initialState(X,value(X,true))).
```

The program π_{update} below updates a planning problem (in this case, P_h) with the changes stipulated by π_{select}. It defines, for each action a, a set of atoms of the from $true(l)$, where l is a precondition or postcondition of a to indicate the elements of $\hat{P}_h$ that should be used to compute plans in the next iteration.

Listing 4: Program π_{update}

```
1  action(action(A)) :- add(action(A)).
2  true(action(A)) :- action(action(A)),
3    not remove(action(A)).
4  true(postcondition(action(A), B, C, D)) :-
5    add(postcondition(action(A), B, C, D)).
6  true(precondition(action(A), C, D)) :-
7    add(precondition(action(A), C, D)).
8  true(postcondition(action(A),B,X,value(X, BoolV))) :-
9    human(postcondition(action(A), B, X,
10   value(X, BoolV))), contradictory(BoolV, NBoolV),
11   not add(postcondition(action(A), B, X,
12   value(X, NBoolV))),
13   not remove(postcondition(action(A), B, X,
14   value(X, BoolV))).
15 true(precondition(action(A), X, value(X, BoolV))) :-
16   human(precondition(action(A),X,value(X, BoolV))),
17   contradictory(BoolV, NBoolV),
18   not add(precondition(action(A),X,value(X, NBoolV))),
19   not remove(precondition(action(A),X,value(X, BoolV))).
```

Finally, the program $\pi_{optimal}$ below uses the optimization feature of `clingo` to find the minimal number of changes to D_h that are needed to explain the optimality of p^*.

Listing 5: Program $\pi_{optimal}$

```
change(N) :-
  N1=#count{1,(A,B,C,D):add(postcondition(A,B,C,D))},
  R1=#count{1,(A,B,C,D):remove(postcondition(A,B,C,D))},
  N2=#count{1,(A,B,C)   :add(precondition(A,B,C))},
  R2=#count{1,(A,B,C)   :remove(precondition(A,B,C))},
  N3=#count{1,A         :add(action(A))},
  R3=#count{1,A         :remove(action(A))},
  I1=#count{X   :add(initialState(X,value(X,true)))},
  I2=#count{X   :remove(initialState(X,value(X,true)))},
  N = N1 + N2 + N3 + R1 + R2 + R3 + I1 + I2.
#minimize {N : change(N)}.
```

Given the correctness of $\pi(n)$ and the fact that an action is executable when its precondition is satisfied and will produce its postconditions, we can verify that π_1 produces an explanation ε so that $\hat{P}_h$ has p^* as one of its solutions.

π_2: Updating and Checking: π_2 consists of π_{update}, π_{engine}, and the code in Listing 6. To avoid the grounding of π_2 in the **while-loop**, we use external atoms of the form *considered(a)* to indicate that a should be added and set it to true in accordance to the output of π_2 at the end of the loop (Line 13, Algorithm 1). Following Chakraborti *et al.* (2017), if an action of the form $a(\vec{X})$ is considered, then all of its instantiations will be considered. We ensure this property by creating a set of atoms of the form $name(a(\vec{X}), a)$ and the rules in Lines 1-12 of Listing 6.

Listing 6: A part of program π_2

```
add(postcondition(action(A),B,X,value(X, BoolV))) :-
  considered(A),
  robot(postcondition(action(A),B,X,value(X,BoolV))).
add(precondition(action(A),X,value(X, BoolV))) :-
  considered(A),
  robot(precondition(action(A), X, value(X, BoolV))).
add(postcondition(action(B),E,X,value(X, BoolV))) :-
  considered(A), name(A,NA), name(B,NA), B != A,
  robot(postcondition(action(B),E,X,value(X,BoolV))).
add(precondition(action(B),X,value(X,BoolV))) :-
  considered(A), name(A,NA), name(B,NA), B != A,
  robot(precondition(action(B), X, value(X, BoolV))).
remove(action(A)):-considered(A),not robot(action(A)).
considered(A) :- robot(occurs(A, T)).
```

Observe that $\hat{P}_h$ has a solution p shorter than p^* only if it contains actions that are different from those in D_r. Hence, it is clear that if actions in p are modified to match their counterparts in D_r or removed, p will not be regenerated by π_2 in the next iteration. Therefore, Algorithm 1 will terminate, i.e., the condition for the **while-loop** to continue will be false eventually.

π_3: Computing the Final Explanation: The program π_3 consists of of π_2 and $\pi_{optimal}$. It is called when π_2 is terminated and finalizes the computation by minimizing the total number of elements that should be added/removed. Observe that because we minimize the cardinality of the set of changes, the output of π_3 is a minimal solution.

Problem	$\|p^*\|$	Mod. 1			Mod. 2			Mod. 3			Mod. 4			Mod. 5	Mod. 6	
		$\|\varepsilon\|$	CSZK	ASP	$\|\varepsilon\|$	CSZK	ASP	$\|\varepsilon\|$	CSZK	ASP	$\|\varepsilon\|$	CSZK	ASP	$\|\varepsilon\|$ ASP	$\|\varepsilon\|$	ASP
BLOCKS-WORLD	10	17	3	**7**	86	3	**0.4**	17	6	36	**17**	9	462	**84**	9 46	2	1
	15	19	3	**44**	83	2	**0.3**	83	5	87	**84**	9	433	**84**	10 45	4	0.1
	14	21	3	**3**	270	3	**0.5**	271	6	**17**	272	10	656	**268**	11 47	5	3
	13	17	3	**13**	218	3	**0.5**	218	6	**67**	217	10	516	**217**	9 45	5	105
LOGIS-TICS	1	21	5	**20**	115	3	**0.7**	120	10	465	**148**	4	**17**	139	5 58	5	128
	5	18	5	**20**	100	4	**3**	101	10	404	**107**	4	**15**	110	5 57	4	135
	10	15	5	**20**	1390	4	**2**	1172	10	**401**	1178	4	**15**	1214	5 57	7	134
	8	25	5	22	**19**	3	**0.7**	18	10	472	**17**	4	**15**	18	5 58	6	144
ROVER	1	11	3	47	**5**	5	5	5	5	561	**5**	6	590	**5**	6 333	4	5
	2	09	1	**1**	3	5	5	**3**	4	378	**3**	7	1521	**3**	6 373	4	3
	3	12	3	48	**11**	6	**11**	14	5	564	**37**	7	1522	**15**	6 1501	6	17
	4	09	1	**1**	11	4	**4**	10	4	63	**16**	4	37	**10**	6 2556	10	20

Table 1: Varying Modifications and Domains

$\|p^*\|$	Explanation Length $\|\varepsilon\|$													
	2		4		6		8		10		12		14	
	CSZK	ASP	CSZK	ASP	CSZK	ASP	CSZK	ASP	CSZK	ASP	CSZK	ASP	CSZK	ASP
17	**0.31**	106	**8**	103	**38**	104	**95**	99	508	**102**	1850	**104**	6163	**99**
20	**0.30**	117	**8**	122	**30**	122	**73**	128	508	**120**	1850	**120**	6161	**122**

Table 2: Varying Explanation and Plan Lengths for LOGISTICS

4 Experimental Results

We empirically evaluated our ASP-based implementation of Algorithm 1, labeled ASP, to find cost-minimal explanations against the current state of the art by Chakraborti *et al.* (2017), labeled CSZK. We evaluated them on the same three planning benchmarks used by Chakraborti *et al.* (2017), i.e., BLOCKSWORLD, LOGISTICS, and ROVER, where we used problem instances from the International Planning Competition (IPC). In all our experiments, we used the actual IPC instances as the model of the robot D_r and considered the following six modifications to the model of the human D_h:

(1) Removal of one random precondition from every action;
(2) Removal of one random effect from every action;
(3) Removal of one random precondition and one random effect from every action;
(4) Removal of all but one random precondition and one random effect from two actions;
(5) Removal of some random actions that are used in the optimal plan; and
(6) Modification of some initial states.

Table 1 tabulates the optimal plan lengths $|p^*|$, explanation lengths $|\varepsilon|$, and runtimes in seconds. The problem IDs in the table correspond to the IDs of instances we used from the IPC benchmark. We omit the runtimes of CSZK in Modifications 5 and 6 as it was not designed for those scenarios. In general, CSZK is faster in most LOGISTICS and BLOCKSWORLD instances but ASP is faster in most ROVER instances. We suspect that the reason is because optimal plan lengths in ROVER tend to be shorter than optimal plan lengths in LOGISTICS and BLOCKSWORLD. We thus conducted an experiment where we varied the optimal plan and the explanation length in the LOGISTICS domain to verify that correlation. Table 2 shows the results and we make the following observations:

- These results show a clear trend that the runtimes of CSZK increases as the explanation lengths increase. The reason is that CSZK needs to search over a larger search space as the explanation increases. As such, its runtime also increases.

- In contrast, the runtimes of ASP remain relatively unchanged with varying explanation lengths. The reason is that the runtimes of ASP are dominated by the grounding of rules in its programs, which are independent of the explanation lengths.
- The results also show that the runtimes of CSZK remain relatively unchanged with varying optimal plan lengths. The reason is that CSZK runs an A* search over the explanation search space and as long as the explanation length remains unchanged, the runtime complexity of the search, which is exponential in the explanation length, remains relatively unchanged as well.
- In contrast, the runtimes of ASP increase as the optimal plan lengths increase. The reason is that the size of the ASP program increases with the optimal plan length. Thus, there is an increasing number of rules to ground, which results in an increase in runtime.
- The implementation of CSZK's system takes as input one problem file. In other words, the problem file has to be the same for the robot and the human. Therefore, CSZK does not work when the signature of the problem of the human differs from that of the robot and ASP does as in the last two scenarios.

These observations highlight that ASP is faster than CSZK when the explanations are long.

5 Conclusions

Research in explainable planning is becoming increasingly important as human-AI collaborations becomes more pervasive. The *model reconciliation problem* (MRP) in explainable planning is a problem where the plan of a planning agent is infeasible or suboptimal to a human user due to differences in their models of the problem. As such, the agent needs to provide an explanation to the user, which reconciles some of the differences in the two models such that its plan is now optimal to the user. Within this space, we demonstrate that MRP can be effectively solved using ASP technologies and empirically show that it outperforms the current state of the art when the explanations are long. The code for our system is available at https://github.com/tcson62/asp-mrp.

For future work, we plan to investigate two orthogonal directions. The first direction is on asserting how believable is the explanation provided by the agent to the human. The second direction is on integrating with *explainable scheduling* systems, where similar to explainable planning, the goal of a scheduling agent is to explain to a human user why its schedule is feasible or cost minimal. ASP has been used with some success on real-world scheduling problems (Abels *et al.* 2019) and we plan to investigate the applicability of our approach in those problems.

Acknowledgments

This research is partially supported by NSF grants 1757207, 1812619, 1812628, and 1914635. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

References

Dirk Abels, Julian Jordi, Max Ostrowski, Torsten Schaub, Ambra Toletti, and Philipp Wanko. Train scheduling with hybrid ASP. In *LPNMR*, pages 3–17, 2019.

Tathagata Chakraborti, Sarath Sreedharan, Yu Zhang, and Subbarao Kambhampati. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *IJCAI*, pages 156–163, 2017.

Martin Gebser, Benjamin Kaufmann, Javier Romero, Ramón Otero, Torsten Schaub, and Philipp Wanko. Domain-specific heuristics in answer set programming. In *AAAI*, pages 350–356, 2013.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014.

M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *LP*, pages 579–597, 1990.

Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – the planning domain definition language. Number TR-98-003, 1998.

Subbarao Kambhampati. Synthesizing explainable behavior for human-ai collaboration. In *AAMAS*, pages 1–2, 2019.

V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1–2):39–54, 2002.

V. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-year Perspective*, pages 375–398, 1999.

Van Duc Nguyen, Philipp Obermeier, Tran Cao Son, Torsten Schaub, and William Yeoh. Generalized target assignment and path finding using answer set programming. In *IJCAI*, 2017.

I. Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.

R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 119–140. Plenum Press, New York, 1978.

Tran Cao Son, Orkunt Sabuncu, Christian Schulz-Hanke, Torsten Schaub, and William Yeoh. Solving Goal Recognition Design using ASP. In *AAAI 2016*, 2016.

Sarath Sreedharan, Tathagata Chakraborti, and Subbarao Kambhampati. Handling model uncertainty and multiplicity in explanations via model reconciliation. In *ICAPS*, pages 518–526, 2018.

Sarath Sreedharan, Alberto Olmo Hernandez, Aditya Prasad Mishra, and Subbarao Kambhampati. Model-free model reconciliation. In *IJCAI*, pages 587–594, 2019.