Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)

On the Gains from Using Action Observations in Domain Repair

Alba Gragera, Raquel Fuentetaja, Ángel García-Olaya, Fernando Fernández

Computer Science and Engineering Department, Universidad Carlos III de Madrid, Spain
agragera@pa.uc3m.es, {rfuentet, agolaya, ffernand}@inf.uc3m.es

Abstract

Designing a PDDL planning domain is an error-prone task, which can result in unsolvable planning tasks or unexpected plans. Existing domain repair methods either rely on a complete plan to identify unsatisfied preconditions or operate without any input plan by compiling the flawed planning task into a new planning task with self-repair actions. In contrast, learning approaches often benefit from a range of input observations to infer domain models. In this paper, we extend the self-repair compilation to also accept as input a variable number of action observations. Experimental results show improved domain repair quality and generally strong performance compared to previous domain repair and learning methods.

Introduction

Automated Planning typically relies on complete model descriptions to generate solution plans, using the standard PDDL format (McDermott et al. 1998), which separates the planning task into domain and problem. However, creating these planning models is a complex knowledge engineering task, even for planning experts (McCluskey, Vaquero, and Vallati 2017). Incomplete domains can lead to unsolvable planning tasks or generate non-desired plans, emphasizing the need for a model-lite approach that can work with shallow or flawed domain specifications (Kambhampati 2007).

There are two main approaches to fixing planning domains. Gragera et al. (2023) compile the planning task into a new task with self-repair actions to fix the domain while searching for the solution plan, adding necessary effects to achieve the goals. Yet, the lack of further input data runs the risk that repaired domains may generate plans that deviate from a hypothetical expected plan. In contrast, Lin, Grastien, and Bercher (2023) work with a complete plan and a flawed domain, identifying conflicts through unsatisfied preconditions and using hitting sets to find repair candidates. However, it only repairs the domain to validate the plan and does not ensure goal achievement. FAMA (Aineto, Jiménez, and Onaindia 2018) is a domain learning method based on a planning compilation that accepts a variable amount of actions and state observations as input. However, it achieves good performance when fully observed states are provided.

Repairing unsolvable planning tasks without additional input knowledge can lead to imprecise repairs: the task becomes solvable, but some actions that are expected to appear in solution plans are not applicable when they should. Allowing for partial observability of plans acknowledges that defining fully valid plans can be difficult for users in many scenarios, but they often know some specific actions that should be included in the plan. Motivated by the growing interest in domain repair and learning, as well as the prospect of a new IPC track on this topic (Behnke and Bercher 2024), our objective is to contribute to the field by exploring how action observations can guide repairs and enhance the quality of the resulting models. Building on the domain repair approach by Gragera et al. (2023), we extend the compilation to accept action observations. Our experiments show competitive results against previous methods using partial and fully observable input plans.

Background

This work adopts the domain repair framework introduced by Gragera et al. (2023), which consider unsolvable planning tasks due to missing action effects. They define an *effect-incomplete* domain as tuple $D^- = \langle \mathcal{H}, \mathcal{C}, Pred, \mathcal{A}^- \rangle$, where $\mathcal{H}$ is a *type hierarchy*; $\mathcal{C}$ is a set of (domain) *constants*; $Pred$ is a set of *predicates* defined by the predicate name p and the types of its arguments $t = t_1, \ldots, t_n$; and $\mathcal{A}^-$ is a set of action schemes $a = \langle name(a), par(a), pre(a), add(a), del(a), cost(a) \rangle$ that can lack some effects. A problem is a tuple $P = \langle \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{I}$ is the set of ground atoms in the initial state and $\mathcal{G}$ is the set of ground atoms defining the goals. A ground action $\underline{a}$ is applicable in a state s if $pre^+(\underline{a}) \subseteq s$ and $pre^-(\underline{a}) \cap s = \emptyset$, transitioning to $s' = (s \setminus del(\underline{a})) \cup add(\underline{a})$. A *plan* π is a sequence $\underline{a}_1, \ldots, \underline{a}_n$ such that $\underline{a}_1$ is applicable in $\mathcal{I}$ and the consecutive application of all actions in the plan generates $s_n, \mathcal{G} \subseteq s_n$. A plan's cost is $cost(\pi) = \sum_{\underline{a}_i \in \pi} cost(\underline{a}_i)$.

The incomplete planning task is reformulated into a new self-repair planning task $\Pi^{\circlearrowleft} = \langle D^{\circlearrowleft}, P^{\circlearrowleft} \rangle$, where the set of original actions $\mathcal{A}^-$ is compiled into $\mathcal{A}^{\circlearrowleft}$, equipped with new operators to fix actions in D^- with additional effects. The reparation is a set $\hat{R} = \{(a_i^-, p(t)^X) \mid \{p(t) \mid p \in Pred, t \subseteq par(a_i^-)\}, X \in \{+, -\}\}$ indicating how to extend the positive $(+)$ and negative $(-)$ effects of the actions $a_i^- \in \mathcal{A}^-$

with new atoms to generate the actions $a_i^{\hat{R}}$ of the repaired domain $D^{\hat{R}}$, where $add(a_i^{\hat{R}}) = add(a_i^-) \cup \{(a_i^-, p(t)) \mid (a_i^-, p(t)^+) \in \hat{R}\}$, and $del(a_i^{\hat{R}}) = del(a_i^-) \cup \{(a_i^-, p(t)) \mid (a_i^-, p(t)^-) \in \hat{R}\}$.

The plan below is a part of the solution for a compiled BLOCKSWORLD task, where the *stack* and *pick-up* actions lacked the *on* and *holding* effects, respectively.

```
(stack b3 b4)
(fix on stack)
(add-fix-2par on stack b3 b4 t_block t_block)
(completed_fixed stack)
(pick-up b1)
(fix holding pick-up)
(add-fix-1par holding pick-up b1 t_block)
(completed_fixed pick-up)
```

When actions lack the necessary effects and cannot stack or hold the block, self-repair actions are introduced to address this gap. These self-repair actions link the original action to the missing predicate and incorporate it into the current state with the appropriate objects. Then, the *completed* action allows the process to move on to the next action. The domain will be repaired by modifying the actions specified in the fix actions with the associated predicates, using the parameters identified from the add-fix actions.

The representation of the self-repairing planning task involves, among other things, new types to have access to the task elements (actions, predicates, types, etc.). In the remainder of the paper we present the extensions made to adapt the compiled domain $D^{\circlearrowleft}$ and problem $P^{\circlearrowleft}$ (see (Gragera et al. 2023) for more details) to consider action observations.

Incorporating Action Observations

We aim to achieve a repaired domain that ensures the existence of a plan that complies with the observations. Then, we redefine the repair problem as follows.

Definition 1 (Repair problem). *Given an effect-incomplete domain D^-, a problem instance P for the ground truth domain D and a sequence of observations O of a plan solving $\Pi = \langle D, P \rangle$, the repair problem is to compute a repaired domain $D^{\hat{R}}$ such that the planning task $\Pi^{\hat{R}} = \langle D^{\hat{R}}, P \rangle$ is solvable and there is at least a plan $\pi^{\hat{R}}$ for $\Pi^{\hat{R}}$ fulfilling the observations.*

The sequence of observations O represents either a totally or a partially specified plan. We adopt the notation from Ramírez and Geffner (2009), where action observations $\{obs_1, \ldots, obs_m\} \in O$ must preserve their relative order in $\{\underline{a}_1, \ldots, \underline{a}_n\} \in \pi$. For example, the action sequence $\pi = \{a, b, c, d, e, a\}$ satisfies the action observation sequences $O_1 = \{b, e, a\}$ and $O_2 = \{b, d\}$, but not $O_3 = \{d, e, b\}$ (for simplicity, we abuse the notation and represent sequences as sets, as in previous work (Ramírez and Geffner 2009)). Therefore, an action sequence satisfies an observation sequence iff there is a monotonic function f that maps the observation indices into action indices, such that $a_{f(j)} = obs_j$.

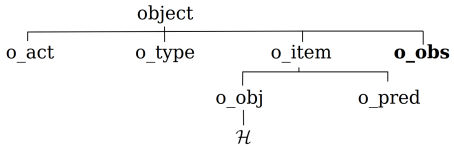


Figure 1: Type hierarchy of the new extended planning task.

To generate repaired domains able to produce solution plans incorporating the desired observations, we must integrate these observations into the compilation. This allows the planner to consider their reparation while completing the plan, ensuring that the observations are applicable. Then, the compiled domain $D^{\circlearrowleft} = \langle \mathcal{H}^{\circlearrowleft}, \mathcal{C}^{\circlearrowleft}, Pred^{\circlearrowleft}, \mathcal{A}^{\circlearrowleft} \rangle$ is extended as $D^{\circlearrowleft}_{obs} = \langle \mathcal{H}^{\circlearrowleft}_{obs}, \mathcal{C}^{\circlearrowleft}_{obs}, Pred^{\circlearrowleft}_{obs}, \mathcal{A}^{\circlearrowleft}_{obs} \rangle$:

TYPE HIERARCHY. We represent action observations with a new type *o_obs* in the extended type hierarchy (Figure 1), in which the type *o_obj* summarizes any object type in the type of the original domain. Thus, $\mathcal{H}^{\circlearrowleft}_{obs} = \mathcal{H}^{\circlearrowleft} \cup \{o\_obs\}$.

CONSTANTS. We introduce $\mathcal{C}_{obs} = \{obs\langle i \rangle \mid i \in \{1, \ldots, |O|+1\}\}$, of type ***o_obs***, to label observations. Thus, $\mathcal{C}^{\circlearrowleft}_{obs} = \mathcal{C}^{\circlearrowleft} \cup \mathcal{C}_{obs}$.

PREDICATES. We include the following predicates, denoted as $Pred_{obs}$, which allow us to sequentially listing the observations and navigate through them.

- $\mathbf{obs\_\langle name(a) \rangle}(o\_obs, o\_obj_1, \ldots, o\_obj_n)$ serves to parse observations and include them in the initial state. For example, the observation *(unstack b3 b2)* becomes *(obs_unstack obs2 b3 b2)* in the initial state, where the *2* in *obs2* indicates it is the second observation.
- $\mathbf{active\_obs}(o\_obs)$ represents the current observation to be included in the plan.
- $\mathbf{next\_obs}(o\_obs_i, o\_obs_{i+1})$ allow us to move forward through ordered observations.

Thus, $Pred^{\circlearrowleft}_{obs} = Pred^{\circlearrowleft} \cup Pred_{obs}$.

ACTIONS. In Gragera et al. (2023) compilation, original action schemes $\mathcal{A}^-$ are compiled into $\mathcal{A}_\alpha$ to adjust them for the new representation along with control predicates that guide the repair process. To manage observations, we replace actions in $\mathcal{A}_\alpha$ by two sets of actions $\mathcal{A}^{obs}_\alpha$ and $\mathcal{A}'_\alpha$. For each $\alpha_a \in \mathcal{A}_\alpha$, $\alpha^{obs}_a \in \mathcal{A}^{obs}_\alpha$ is defined as:

$$
\begin{aligned}
name(\alpha^{obs}_a) &= name(\alpha_a)\textbf{\_obs} \\
par(\alpha^{obs}_a) &= par(\alpha_a) \cup \{\textbf{obs1}, \textbf{obs2} : o\_obs\} \\
pre(\alpha^{obs}_a) &= pre(\alpha_a) \cup \\
&\quad \{\textbf{obs\_}\langle\textbf{name(a)}\rangle\textbf{(obs1, x)} \mid \textbf{x} \subseteq \textbf{par(a)}\} \cup \\
&\quad \{\textbf{active\_obs(obs1)}, \textbf{next\_obs(obs1, obs2)}\} \\
add(\alpha^{obs}_a) &= add(\alpha_a) \cup \textbf{active\_obs(obs2)}\} \\
del(\alpha^{obs}_a) &= del(\alpha_a) \cup \{\textbf{active\_obs(obs1)}\}
\end{aligned}
$$

The only distinction between α^{obs}_a and α_a lies in the bold part. Actions $\alpha^{obs}_a \in \mathcal{A}^{obs}_\alpha$ delete the active observation and activate the subsequent one.

Actions $\alpha'_a \in \mathcal{A}'_\alpha$ restrict their application if it coincides with the active observation, which forces to apply the corresponding action in $\mathcal{A}^{obs}_\alpha$ instead.

$$
\begin{aligned}
par(\alpha'_a) =& par(\alpha_a) \cup \{\textbf{obs1} : o\_obs\} \\
pre(\alpha'_a) =& pre(\alpha_a) \cup \{\textbf{active\_obs(obs1)}\} \cup \\
& \{\neg \textbf{obs\_}\langle \textbf{name(a)}\rangle \textbf{(obs1, x)} \mid \textbf{x} \subseteq \textbf{par(a)}\} \\
add(\alpha'_a) =& add(\alpha_a) \\
del(\alpha'_a) =& del(\alpha_a)
\end{aligned}
$$

Thus, $\mathcal{A}^{\circlearrowleft}_{obs} = (\mathcal{A}^{\circlearrowleft} \setminus \mathcal{A}_\alpha) \cup \mathcal{A}^{obs}_\alpha \cup \mathcal{A}'_\alpha$. The presence of both $\mathcal{A}'_\alpha$ and $\mathcal{A}^{obs}_\alpha$ in the domain corresponds to the case of partial observability. In case of full observability, all actions align with the application of an observation and $\mathcal{A}'_\alpha = \emptyset$. The general fix and close actions of the original compilation are kept unchanged.

PROBLEM. The compiled problem $P^{\circlearrowleft} = \langle \mathcal{I}^{\circlearrowleft}, \mathcal{G}^{\circlearrowleft} \rangle$ is extended to $P^{\circlearrowleft}_{obs} = \langle \mathcal{I}^{\circlearrowleft}_{obs}, \mathcal{G}^{\circlearrowleft}_{obs} \rangle$, with:

$$
\begin{aligned}
\mathcal{I}^{\circlearrowleft}_{obs} =& \mathcal{I}^{\circlearrowleft} \cup \mathcal{I}_{obs} \cup \mathcal{I}_{order} \cup \{active\_obs(obs_1)\} \\
\mathcal{G}^{\circlearrowleft}_{obs} =& \mathcal{G}^{\circlearrowleft} \cup \{active\_obs(obs_{|O|+1})\}
\end{aligned}
$$

- The active observation ($active\_obs$) is set to the first and last observations in $\mathcal{I}'$ and $\mathcal{G}'$ respectively.
- $\mathcal{I}_{obs} = \{obs\_\langle name(a_j)\rangle(obs_i, obj_1, \ldots, obj_n)\}$, represents the action name and objects from each observation $obs_i \in O$. Here, $obs_i \in \mathcal{C}_{obs}$ is a constant denoting the i-th observation in O, which is used to ensure that the action order in the observations is preserved.
- $\mathcal{I}_{order} = \{next\_obs(obs_i, obs_{i+1})\}$ establishes the observation order for all $obs_i \in \mathcal{C}_{obs}$.

All observations are sequentially numbered to be included in the plan while preserving their order. In case of partial observability, other actions may appear before, after, or between these observations, leaving the solution unbounded. Including the last observation in the problem's goals ensures that all solutions to the compiled task incorporate each specified observation, guaranteeing that at least one plan generated using the repaired domain satisfies these observations.

Evaluation

Approaches. We compare the results of the proposed approach (DR) with FAMA, which also compiles the problem into a planning task, and the approach by Lin, Grastien, and Bercher (2023) using hitting sets, which we refer to as HIT. To ensure a fair comparison, we limited FAMA and HIT to only add missing effects, as both methods can also repair preconditions and delete effects, although this last feature is initially disabled in FAMA's learning task.

Benchmarks. We use domains increasing in the number of actions schemes (in brackets): TRANSPORT(3), BLOCKS(4), SATELLITE(5), CHILDSNACK(6), GOLDMINER(7), ROVERS(9) and BARMAN(12). For each domain, we use a problem generator[1] to create 10 problem instances of increasing difficulty (Seipp, Torralba, and Hoffmann 2022). For each domain, we generated flawed domains by randomly removing positive action effects from the lifted representation, creating four sets of 10 domains with one to four effects removed. This produced 40 incomplete domains per original domain. We consider four observability levels: 0% (no observability), 30%, 70% (partial observability) and 100% (full observability). Intermediate levels were generated by randomly omitting the corresponding percentage of actions from the optimal plans. Each domain is associated with a single problem-observation trace, demonstrating that our approach can successfully repair a domain using just a single trace, which highlights a key strength of our method. We use the RAGNAROK planning system (Drexler et al. 2023) to solve the compiled tasks. The experiments were run on an Ubuntu machine with Intel(R) Core(TM) i7-7700HQ CPU running at 2.80GHz and 8GB memory.

Coverage Analysis. Table 1 shows the coverage of each approach, out of a total of 40 tasks per cell. Most instances unsolved by FAMA are due to memory or time limits: our compilation focuses on repairing applicable actions to enable other actions, while FAMA starts by considering the repair of all actions. In larger domains, this makes the search unmanageable. In addition, FAMA does not represent if effects are positive or negative explicitely. It assumes that atoms appearing both in the preconditions and effects are deletions. This can make sense, but it is not necessarily true. For instance, in ROVERS, actions to communicate data have *(available ?r)*, where *?r* represents the current rover, both as precondition and positive effect. Since this effect is considered as a negative one, FAMA follows the plan, communicates the data, and permanently makes the rover unavailable for any further data transmissions. We also found that FAMA defines transitions between observations with (*inext* o_i o_j) and each action in the compilation requires both a current observation and a next one. However, for partial observability, the number of actions after the last observation is not known. So, this approach seems to only guarantee a solution when the last action in the plan is observed. Otherwise the absence of a next observation makes the task unsolvable. We anticipated the unbounded nature of the problem and included two types of actions to handle observations, so that actions can be applied without needing a next observation defined. Regarding times, none of the tasks finish within the first minute, but 82.1% complete within the next 30 seconds. In the case of HIT, it uses hitting sets to find unsatisfied preconditions following the plan. This approach provides a repair that validates the plan across all tasks and is very fast since it does not perform a search to achieve the goals. However, in domains repaired by HIT some actions might still be broken: the plan is applicable, but the goals are not actually achieved. This can be observed in GOLDMINER, where for 22 out of 40 tasks the action *pick-gold* lacks the *holds-gold* effect (which is the goal for all problems), so the plan can still be validated because the action is applicable, but it does not achieve the goal. Nonetheless, their method maintains an average computation time of 0.01 seconds or less.

[1] https://github.com/AI-Planning/pddl-generators

	0% obs		30%obs		70%obs		100%obs		
Domain	**DR**	**FAMA**	**DR**	**FAMA**	**DR**	**FAMA**	**DR**	**FAMA**	**HIT**
Transport	**33**	15	**30**	0	**29**	23	**40**	**40**	**40**
Blocks	38	**40**	**38**	32	38	**40**	**40**	**40**	**40**
Satellite	37	**40**	**34**	24	37	**40**	**40**	**40**	**40**
Childsnack	**29**	8	**24**	8	**27**	22	**40**	**40**	**40**
Goldminer	**40**	**40**	**39**	25	**34**	25	**40**	**40**	18
Rovers	**20**	0	**19**	2	**15**	0	**40**	0	**40**
Barman	**13**	0	**11**	0	**10**	2	38	**40**	**40**
Total	**210**	143	**195**	91	**190**	152	**278**	240	258

Table 1: Coverage for DR, FAMA and HIT for the different observability levels (HIT only deals with complete plans).

Benefits of adding observations - Metrics Analysis. To evaluate the quality of the repairs, we adopt the approach by Aineto, Jiménez, and Onaindia (2018) and assume the ground truth domain to measure syntactic differences between domains using *precision* and *recall* (Davis and Goadrich 2006). *Precision* $= \frac{tp}{tp+fp}$ is the ratio of correctly placed reparations (*true positives*) wrt. the total number of reparations. *Recall* $= \frac{tp}{tp+fn}$ is similar but for *false negatives* (fixes that should appear in actions but are missing). We compute these metrics in two different ways, M_{ap} and M_p:

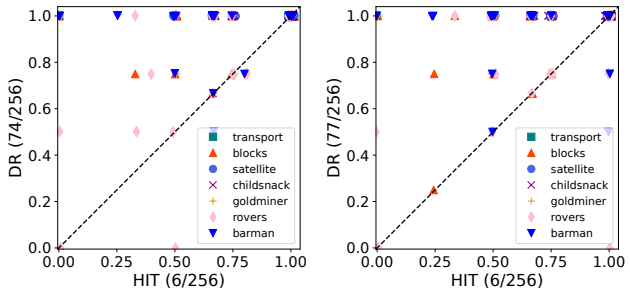
- M_{ap} considers true positive if the action-predicate pair is correct (i.e., if *holding* is added as effect of *pick-up*).
- M_p considers true positive if the predicate is correct (i.e., if *holding* is added as an effect of *any* action).

Table 2 shows the percentage of solved tasks at each observability level where precision and recall are at least 0.75, comparing our DR approach with FAMA.

	DR				FAMA			
	P $\geq$ 0.75		**R $\geq$ 0.75**		**P $\geq$ 0.75**		**R $\geq$ 0.75**	
obs	M_{ap}	M_p	M_{ap}	M_p	M_{ap}	M_p	M_{ap}	M_p
0%	**58.1**	**80.9**	**60.0**	**70.5**	48.9	72.0	38.5	43.3
30%	**65.1**	**83.6**	**65.1**	**74.9**	35.2	62.6	31.9	45.1
70%	74.2	87.9	**75.3**	**86.8**	**83.6**	**91.4**	74.3	82.9
100%	**92.4**	**100.0**	**91.7**	**99.6**	87.1	99.6	80.8	93.7

Table 2: Percentage of solved tasks achieving a precision (P) and recall (R) of at least 0.75 for both metrics defined.

Despite sometimes reducing coverage, adding action observations to domain repair enhances model quality, as precision and recall grow as observability levels increase (with some exception for FAMA). For full observability, our approach solves 68 additional tasks and also increases the number of tasks achieving high-value metrics. The higher scores for M_p indicates that the correct predicate is often repaired in other actions, still providing valuable insight into missing information in the domain. Comparing our approach with FAMA, we solve more tasks and achieve higher quality models, except at 70% observability, where FAMA solves 38 fewer tasks but obtains better values for those it solve. HIT only accepts full observability, so we only compare with it under that configuration. Figure 2 shows the results for the metric M_{ap}. Out of 256 common tasks solved, our approach outperforms in precision on 74 tasks compared to 6 for HIT, and in recall, we are better on 77 tasks versus 6 for HIT.



(a) Precision (b) Recall

Figure 2: Comparison for metric M_{ap} between DR and HIT. Brackets indicate (dominant tasks / common tasks).

Drawbacks of adding observations - Runtime Analysis. Intuitively, the computational cost of repairing a domain is higher than simply computing a plan. At 0% observability, the approach requires reasoning about both the action to apply and potential repairs. With partial observability, it assesses whether to apply observed or new actions and whether they require repair. With full observability, the approach only needs to follow each action in the plan and decide if it requires repair. Figure 3 shows the average time to compute an optimal plan for each configuration, benchmarked against using the ground truth domain. The results confirm the previous analysis, showing that computation times increase as the planner needs to make more decisions.

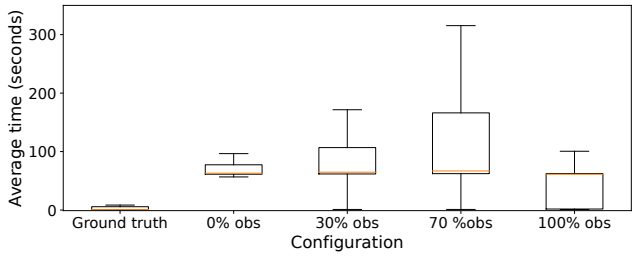


Figure 3: Runtimes for DR vs. ground truth domain.

Conclusions

We presented an extended domain repair approach to incorporate action observations, allowing different levels of observability to inform the repair process. Experiments show that incorporating action observations improves the quality of repaired models, yielding higher precision and recall, though coverage may decrease with partial observability. Overall, our approach outperforms prior methods by solving more tasks and achieving stronger metric values. Importantly, it ensures that generated plans adhere to given observations and reach the problem's goals, offering a robust solution for domains with incomplete or flawed descriptions.

References

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning STRIPS Action Models with Classical Planning. In de Weerdt, M.; Koenig, S.; Röger, G.; and Spaan, M. T. J., eds., *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*, 399–407. AAAI Press.

Behnke, G.; and Bercher, P. 2024. Envisioning a Domain Learning Track for the IPC.

Davis, J.; and Goadrich, M. 2006. The relationship between Precision-Recall and ROC curves. In *Proceedings of ICML 2006, Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148, 233–240. ACM.

Drexler, D.; Gnad, D.; Höft, P.; Seipp, J.; Speck, D.; and Ståhlberg, S. 2023. Ragnarok. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Gragera, A.; Fuentetaja, R.; Olaya, Á. G.; and Fernández, F. 2023. A Planning Approach to Repair Domains with Incomplete Action Effects. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, July 8-13, 2023, Prague, Czech Republic*, 153–161. AAAI Press.

Kambhampati, S. 2007. Model-lite Planning for the Web Age Masses: The Challenges of Planning with Incomplete and Evolving Domain Models. In *Proceedings of AAAI 2007, July 22-26, 2007, Vancouver, British Columbia, Canada*, 1601–1605. AAAI Press.

Lin, S.; Grastien, A.; and Bercher, P. 2023. Towards Automated Modeling Assistance: An Efficient Approach for Repairing Flawed Planning Domains. In Williams, B.; Chen, Y.; and Neville, J., eds., *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, 12022–12031. AAAI Press.

McCluskey, T. L.; Vaquero, T. S.; and Vallati, M. 2017. Engineering Knowledge for Automated Planning: Towards a Notion of Quality. In *Proceedings of K-CAP 2017, Austin, TX, USA, December 4-6, 2017*, 14:1–14:8. ACM.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL–the planning domain definition language–version 1.2. Technical report, Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational ....

Ramírez, M.; and Geffner, H. 2009. Plan Recognition as Planning. In Boutilier, C., ed., *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 1778–1783.

Seipp, J.; Torralba, Á.; and Hoffmann, J. 2022. PDDL Generators. https://doi.org/10.5281/zenodo.6382173.