Exploring PDDL-Based Benchmark Planning Problems in ASP

Arif Hasnat

Department of Computing Science
University of Alberta
hasnat@ualberta.ca

Abstract. In this project, we explored some of the benchmark problems from various International Planning Competitions. First, we translated those PDDL-based problems to ASP representations both manually and using an ASP translator, and solved those ASP representations using an ASP solver. Then, we analyzed the performance of the ASP solver compared to some of the state-of-the-art PDDL-based planners to study the effectiveness of ASP solvers on benchmark planning problems.

1 Introduction

AI Planning is one of the oldest sub-areas in AI. The International Planning Competition (IPC) is organized in the context of the International Conference on Planning and Scheduling (ICAPS)[1]. It empirically evaluates state-of-the-art planning systems on a number of benchmark problems. The planning problems are encoded in Planning Domain Definition Language (PDDL) [1]. Compiling those planning problems to Answer Set Programming(ASP) can provide the ASP community access to a wide range of planning problems, and the planning community can benefit from the knowledge representation and reasoning capacities of ASP.

There are some research about compiling planning problems defined in PDDL to it's ASP representation. Among those, Gebser et. al. [2] presented a prototypical system called Plasp which implements planning by compilation to ASP and then Dimopoulos et. al. [3] improved the system to overcome some shortcomings and make it more effective for planning in ASP. These two research provide a PDDL-to-ASP translator[2], which translates PDDL descriptions to ASP facts.

In this project, we experimented with some of the benchmark problems from various International Planning Competitions. At first, we translated those PDDL-based problems to ASP representations both manually and using the Plasp translator. After that, we solved both ASP representations using an ASP solver Clingo[3]. Finally, we analyzed the performance of the ASP solver compared to some of the state-of-the-art PDDL-based planners to study how well the planning problems can be represented and solved in ASP.

[1] http://icaps-conference.org/index.php/Main/Competitions
[2] https://github.com/potassco/plasp
[3] https://potassco.org/clingo/

2 Related Work

Gebser et. al. [2] presented a prototypical system called Plasp for implementing Planning problems by compilation to ASP. It follows the approach of SATPlan [4][5] while keeping the actual compilation simple so that different planning techniques can be applied by meta-programming in ASP. At first, a Parser builds an Astract Syntax Tree (AST) from the PDDL description. Then, the Analyzer gathers information on the particular problem instance. After that, the Preprocessor modifies and improves it for the translation process. Finally, the ASP backend produces an ASP program using the previously gathered data. The final ASP program consists of a set of facts mapped from PDDL description. It along with a meta-program can be solved incrementally [6] by the ASP solver, Clingo. The meta-program can be written to allow sequential or concurrent actions in the output plan.

Dimopoulos et. al. [3] introduced a new version of the PDDL-to-ASP translator, Plasp which accepts a wide range of PDDL features, contains some novel planning encodings inspired by SAT planning, offers advanced planning algorithms borrowed from SAT planning. They redesigned Plasp system with an optional preprocessing of PDDL input by the state-of-the-art planning system Fast Downward. The translator component now provides a spectrum of ASP encodings ranging from adaptions of known SAT encodings to novel encodings taking advantage of ASP-specific concepts and the planner component offers more sophisticated planning algorithms by taking advantage of multishot ASP solving. The new Plasp system is also applicable to dynamic domains beyond PDDL such as ASP encodings of finite model finding instead of planning.

3 Methodology

Plasp translated ASP problem instances together with some ASP meta-encodings[4] available with Plasp tool, can be solved incrementally by Clingo. That means, we don't need to specify the required number of plan steps or plan level. It tries to find the optimal plan with minimum number of steps, incrementally starting from 0.

But, our manually translated ASP instances from PDDL descriptions require to specify the number of plan steps needed to solve it by Clingo. To make it incremental and, thus comparable to Plasp and the PDDL based planners, we adapted a simple naive approach which is stated in the Algorithm 1. The algorithm requires only the ASP encoded domain and problem files as input. It initializes the number of *steps* with 0. After that, it calls the ASP solver Clingo using the given domain file, the problem file and number of steps. Then, it checks the output of Clingo to see if the output contains the text "SATISFIABLE" and not the text "UNSATISFIABLE". If it is satisfiable then the algorithm stops the iteration and returns the final output, the final number of steps required and the total time spent. Otherwise, it increments *steps* by 1 and continues until

[4] https://github.com/potassco/plasp/tree/master/encodings/strips

Algorithm 1 Incremental Solver Algorithm

```
1: procedure INC-SOLVER($domain\_file, problem\_file$)
2:     $start\_time \leftarrow$ Time()
3:     $steps \leftarrow 0$
4:     while $steps \leq MAX\_STEPS$ do
5:         $output \leftarrow$ Clingo($domain\_file, problem\_file, steps$)
6:         if "SATISFIABLE" in $output$ and "UNSATISFIABLE" not in $output$ then
7:             break
8:         end if
9:         $steps++$
10:    end while
11:    $end\_time \leftarrow$ Time()
12:    $spent\_time \leftarrow end\_time - start\_time$
13:    return $output$, $steps$, $spent\_time$
14: end procedure
```

a maximum number of steps to be checked, is exceeded. The function *Clingo* works by calling the ASP solver clingo using the command: "clingo $domain\_file$ $problem\_file$ -c steps=*steps*" and returns the output. Thus the algorithm can be used to solve any ASP encoded planning problems incrementally without the need of specifying the required number of plan steps explicitly.

4 Experiments

We experimented with some benchmark PDDL-based planning problems from various International Planning Competitions by solving them using some of the state-of-the-art planners, and also translating them into ASP encodings and solving by an ASP solver. In the following sections, we will explain these steps in detail. All the codes and resources for the experiments are publicly available[5].

4.1 Benchmark Selection

Plasp does not support some of the features[6] of the latest version, PDDL 3.1 yet. Keeping the limitations of Plasp in mind, we selected three benchmark domains with different level of difficulties from three different International Planning Competitions. The strips domain "Elevator" along with its problem instances is selected from the IPC-2000[7]. The propositional variants of domain "TPP" and its problem instances are selected from the deterministic track of the IPC-2006[8]. The strips domain "Child-Snack" and its problem instances are from the sequential-deterministic track of IPC-2014[9].

[5] https://github.com/hasnat-cse/620_project
[6] https://github.com/potassco/plasp/blob/master/doc/pddl-feature-support.md
[7] http://www.cs.toronto.edu/aips2000
[8] http://idm-lab.org/wiki/icaps/ipc2006/deterministic
[9] https://helios.hud.ac.uk/scommv/IPC-14/domains_sequential.html

4.2 Solving by PDDL-based Planners

We selected three PDDL-based planners- SATPlan[10] (2006), MaxPlan[11] (2006) and SGPlan[12] (5.2.2) to solve the selected benchmark problems. SATplan uses a SAT solver as search backend that leads to shortest plan and its compilation is closely related to Plasp. It took first place for optimal deterministic planning in IPC-2004 and IPC-2006. MaxPlan is an optimal planner for propositional based strips planning using long distance mutual exclusion and backward reduction. It took first place jointly with SATplan for optimal deterministic planning in IPC-2006. SGPlan [7] was the winner of 1st prize for deterministic satisfying track in IPC-2006 and does not guarantee shortest plan lengths.

4.3 Solving by ASP Solver

We translated the PDDL descriptions of the benchmark domains and problems into ASP encodings using plasp[2] (3.1.1), the latest version available at this moment. Then, we ran clingo[3] (5.4.0) to solve all the translated instances together with the meta-encodings[4] available with Plasp in incremental and parallel actions settings.

We also translated all the PDDL domains and problems into ASP encodings manually, allowing parallel actions to occur. To solve those translated instances incrementally, we used the naive Inc-Solver explained in section 3 which also uses clingo (5.4.0) as the core solver.

4.4 Experimental Results

We conducted all experiments on a Linux 32-bit PC equipped with 4 cores, Intel 2.4 GHz CPU and 4 GB RAM, imposing 1800 seconds as time limit. Run time results in seconds and shortest plan level required for the problems allowing non conflicting parallel actions are shown in Table 1, 2 and 3; an entry "timeout" indicates that the time limit exceeded, bad_alloc is for std::bad_alloc exception, mem_error is for memory exhaustion and no_result indicates that the planner did not return any plan.

Analysis. All the planners and ASP solvers except SGPlan returned shortest plans with same plan levels for all the problems. SGPlan took a very little amount of time (within 1 second) compared to others to find a plan (not the shortest though) for all the problems in the Elevator and the TPP domains. For the Child-snack domain, it did not return any plans for the problems. One reason is that it was developed around 2007 and does not support some of the features of PDDL version used in Child-snack (2014) domain such as ":constants", ":equality". This is also the case for SATPlan and MaxPlan. MaxPlan perfromed very poor on

[10] https://www.cs.rochester.edu/u/kautz/satplan
[11] https://www.cse.wustl.edu/~ychen/maxplan
[12] https://wah.cse.cuhk.edu.hk/wah/programs/SGPlan/sgplan5.html

Problem	**Plan Level**	**Inc-Solver with Manual Translation**	**Clingo with Plasp**	**SATPlan**	**MaxPlan**	**SGPlan**
Elevator-1-0	4	0.030	0.051	0.010	0.010	0.001
Elevator-2-0	6	0.049	0.074	0.010	0.010	0.001
Elevator-3-0	8	0.093	0.157	0.020	0.040	0.001
Elevator-4-0	12	0.227	1.866	0.070	0.110	0.001
Elevator-5-0	14	0.397	10.754	0.220	0.480	0.001
Elevator-6-0	14	0.492	21.601	0.300	0.920	0.001
Elevator-7-0	18	1.688	157.456	2.570	10.780	0.001
Elevator-8-0	22	6.275	432.804	28.56	86.44	0.001
Elevator-9-0	26	34.512	timeout	211.55	785.09	0.001
Elevator-10-0	27	94.927	timeout	746.37	timeout	0.001
Elevator-11-0	30	478.288	timeout	timeout	timeout	0.001

Table 1. Run time in seconds for Elevator (IPC-2000) problems

Problem	**Plan Level**	**Inc-Solver with Manual Translation**	**Clingo with Plasp**	**SATPlan**	**MaxPlan**	**SGPlan**
TPP-06	9	0.320	2.419	0.040	0.030	0.010
TPP-07	9	0.338	2.840	0.060	0.040	0.020
TPP-08	9	0.395	3.173	0.040	0.060	0.020
TPP-09	11	0.950	8.950	0.130	0.120	0.050
TPP-10	11	0.937	10.172	0.130	0.210	0.090
TPP-18	11	16.940	bad_alloc	8.120	21.760	0.160
TPP-19	10	38.612	bad_alloc	14.15	33.24	0.330
TPP-20	12	91.417	bad_alloc	13.59	33.78	0.380
TPP-21	11	timeout	bad_alloc	timeout	timeout	1.080
TPP-23	11	109.760	bad_alloc	37.64	202.92	0.830

Table 2. Run time in seconds for TPP (IPC-2006) problems

that domain and SATPlan also performed poorly compared to Inc-Solver with manually translated ASP encodings on Child-Snack problems. Although Plasp supports those advanced features, but it did not perform well on the problems of that domain with increasing number of childs. Inc-Solver took significantly less time for Child-snack problems except Childsnack-09 compared to all other solvers. For some unknown reasons, it took very long time for Childsnak-09 problem.

For the Elevator domain, Clingo on Plasp translated ASP encodings was the slowest than all other solvers. SATPlan found the shortest plan with least amount of time among all solvers for the problems with less complexity (less plan level or steps). But, Inc-Solver performed better overall in terms of run time considering the problem complexity. The same is true for TPP domain, except SATPlan was faster than Inc-Solver on all the problems.

Problem	Plan Level	Inc-Solver with Manual Translation	Clingo with Plasp	SATPlan	MaxPlan	SGPlan
Childsnack-05	4	3.064	12.934	1.220	149.920	no_result
Childsnack-08	4	1.988	54.983	4.160	timeout	no_result
Childsnack-09	4	145.984	82.561	6.540	timeout	no_result
Childsnack-10	4	3.104	bad_alloc	48.400	timeout	no_result
Childsnack-11	4	4.607	bad_alloc	10.760	timeout	no_result
Childsnack-12	4	4.575	bad_alloc	48.050	timeout	no_result
Childsnack-13	4	5.655	bad_alloc	66.910	timeout	no_result
Childsnack-14	4	6.573	bad_alloc	mem_error	timeout	no_result
Childsnack-19	4	23.034	bad_alloc	mem_error	timeout	no_result

Table 3. Run time in seconds for Childsnack (IPC-2014) problems

Overall, it is expected that SATPlan is faster than Clingo with Plasp, because it utilizes a planning-specific frontend [8], while compilations of Plasp are instantiated by a general-purpose ASP grounder. But with Plasp translated ASP instances and meta-encodings, planning can be done with more flexibility such as enabling sequential actions, parallel actions in any sequence, parallel actions in some sequence etc. We can also see that even the run time of the naive approach used in Inc-Solver is very much comparable to SATPlan and much better than Clingo with Plasp. Because, we have more flexibility during translating the domains and problems on our own, whereas the approach of Plasp is more general to support a variety of PDDL features. The run time of Inc-Solver can be further reduced using incremental grounding and other advanced ASP techniques.

5 Conclusion

We explored some problems of three PDDL-based benchmark domains - Elevator (IPC-2000), TPP (IPC-2006) and Child-Snack(2014). First, we translated them into ASP representations both manually and using an ASP translator, Plasp.Then, we analyzed the performance of the ASP solver, Clingo on solving those ASP translations, compared to three best performing PDDL-based planners of IPC-2006 - SATPlan, MaxPlan and SGPlan. Although solving Plasp translated ASP encodings by Clingo is slower than PDDL-based planners, but it gives a great amount of flexibility in applying different planning techniques through ASP meta-encodings. Furthermore, we found that solving planning problems by translating them manually into ASP encodings is very much comparable to directly solving those problems using PDDL-based planners. Even it was better than all other solvers for complex problem instances. So, with further improvement, the translation of PDDL-to-ASP using Plasp can be made more efficient and effective so that ASP solvers can solve planning problems with less time and greater flexibility.

References

1. McDermott, D.: Pddl - the planning domain definition language. In: Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control (1998)
2. Gebser, M., Kaminski, R., Knecht, M., Schaub, T.: plasp: A prototype for pddl-based planning in asp. In: Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11). (2011) 358–363
3. Dimopoulos, Y., Gebser, M., Luhne, P., Romero, J., Schaub, T.: plasp 3: Towards effective asp planning. Theory and Practice of Logic Programming **19** (2019) 477–504
4. Kautz, H., Selman, B.: Planning as satisfiability. In: Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92), Wiley (1992) 359–363
5. Kautz, H., Selman, B.: Pushing the envelope: Planning, propositional logic, and stochastic search. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2. AAAI'96, AAAI Press (1996) 1194–1201
6. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: Engineering an incremental ASP solver. In: ICLP. Volume 5366 of Lecture Notes in Computer Science., Springer (2008) 190–205
7. Hsu, C.W., Wah, B., Huang, R.: Constraint partitioning for solving planning problems with trajectory constraints and goal preferences. (2007) 1924–1929
8. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artificial Intelligence **90** (1997) 281 – 300