Journal on Satisfiability, Boolean Modeling and Computation 14 (2023) 17–23

OMTPlan: A Tool for Optimal Planning Modulo Theories

Francesco Leofante f.leofante@imperial.ac.uk
Department of Computing
Imperial College London
London SW7 2AZ
United Kingdom

Abstract

OMTPlan is a Python platform for optimal planning in numeric domains via reductions to Satisfiability Modulo Theories (SMT) and Optimization Modulo Theories (OMT). Currently, OMTPlan supports the expressive power of PDDL2.1 level 2 and features procedures for both satisficing and optimal planning. OMTPlan provides an open, easy to extend, yet efficient implementation framework. These goals are achieved through a modular design and the extensive use of state-of-the-art systems for SMT/OMT solving.

KEYWORDS: *Planning as Satisfiability, Optimisation Modulo Theories*

Submitted 2 February 2022; revised 26 January 2023; accepted 2 May 2023

1. Introduction

AI Planning can be defined as the model-based approach to intelligent behaviour, where a model of the world and of possible actions to be performed is used to decide on a sequence of actions, a *plan*, that brings the world to a desired state [6].

Satisfiability Modulo Theories (SMT) [2] and its extension Optimization Modulo Theories (OMT) [17] are two powerful frameworks that have been used to model, and reason about, expressive planning problems. While SMT technology has long enjoyed popularity within the planning community, the application of OMT to planning has remained unexplored until recently. In [11] we introduced the first domain-independent reduction from optimal planning to OMT solving. More specifically, we presented an OMT-based algorithm to solve optimal numeric planning problems where actions are equipped with unitary (all actions have cost 1), constant (actions have constant costs $\geqslant 1$) or state-dependent action costs (SDAC), where the cost of executing an action is a function of the state where it is executed [9]. Notably, no tool support exists for integrated reasoning on problems of the latter class and state-of-the-art (SoA) planners either do not support SDAC upfront, e.g. [14], or solve them with uninformed search, a general class of search algorithms that explore the solution space of a planning problem ignoring domain specific knowledge such as cost structures, e.g. [12].

The present contribution is intended to fill this gap as we introduce OMTPlan, the first planner that leverages state-of-the-art satisfiability techniques to solve optimal planning problems in the presence of rich cost structures. OMTPlan is implemented in Python and presents a modular architecture that can be easily extended with new functionalities. Besides offering solving capabilities, OMTPlan can also act as a translator from PDDL [5], the standard language of planning, to SMT-LIB [1] thus providing a new source of challenging benchmarks for the automated reasoning community – see, e.g. [3] for such an application of the tool. The planner is open-sourced under a GNU General Public License, version 3 (GPL-3.0). Source

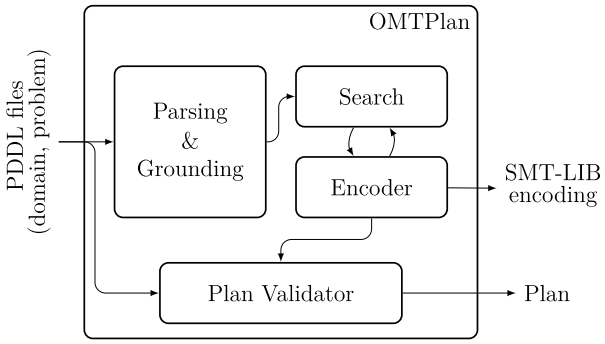

Figure 1. OMTPlan: internal work-flow.

code and the related documentation are available online at: https://github.com/fraleo/OMTPlan.

2. System Architecture

OMTPlan realises its functionalities through the interaction of the components represented in Fig. 1. Each component takes care of different phases of the planning process as detailed in the following.

Input Language. OMTPlan accepts problems specified in the Planning Domain Definition Language (PDDL), the de-facto standard language for planning. Dialects of PDDL exist with different expressive power, here we focus on PDDL 2.1 level 2 [5] as it allows to capture rich numeric structures. PDDL describes a planning task by means of: *(i)* a *domain* file containing the description of general features of the problem (e.g. the available actions) and, *(ii)* a *problem* file defining instance-specific information (e.g. the initial and goal situation, the optimisation metric). We refer to [5] for more details on PDDL.

Parsing and Grounding. We leverage the Python parsing module developed for the Temporal Fast-Downward (TFD) planner.[1] The original implementation does not provide support for the specification of arbitrary metrics for plan quality. Hence, we extended it to support metrics expressed in quantifier-free linear real arithmetic (*QFLRA*), as supported by our approach [11].

Besides parsing operations, this module is also responsible for grounding the first-order representation used in PDDL. The grounding algorithm of TFD makes use of a compilation to a logic program in order to perform reachability analysis and grounding all in one [8]. This compilation is specific to both the set of action schemas and the initial state of the search. The reachability analysis is able to infer if certain ground actions will never be applicable when starting from the given initial state, and that some variables will never actually change their value, i.e. they are seen as static facts. As a result, ground actions that are not applicable are pruned and static variables are compiled away and do not need to be represented with variables in the planning formula.

[1] Available at http://gki.informatik.uni-freiburg.de/tools/tfd.

Search. OMTPlan computes plans by progressively unrolling the transition system induced by the planning problem, as commonly done in Planning as SAT [15]. Given a planning problem Π, we build formulas Π_n for unrolling of increasing number of steps n until a plan is found, or a user-defined upper bound is reached. This module is responsible for implementing the logics according to which the unrolling is performed. OMTPlan implements common strategies such as linear ($n = 1, 2, 3, \ldots$) and exponential ($n = 1, 2, 4, \ldots$) increment; however custom strategies can be easily implemented.

Once a search strategy has been selected, this module schedules calls to the encoder to produce unrollings of different length. These are dispatched and tested sequentially, although other strategies could be added. The search module is also responsible for feeding the planning formula to the underlying solver, fetching the result of the satisfiability check and act accordingly.

Encoder. The main task of this module is to traverse the parse tree produced by the parsing module and build the planning formula encoding the unrolling for the length selected by the search module. In the current implementation the user can choose between:

(1) SMT encodings for satisficing planning;
(2) SMT encodings for optimal planning with unitary costs;
(3) OMT encodings for optimal planning with constant costs and SDACs.

Encodings (1) are based on the classical state-based representation of Planning as SAT [15], here extended to numeric variables. Both the *parallel* and *serial execution semantics* for actions are supported. The former allows for multiple actions to be executed in parallel provided they act on different subsets of variables; the latter only allows for serial plans where at most one action can be executed per step.

Encodings (2) combine serial encodings with a linear increment strategy to enable optimal planning with unitary costs. An optimal plan in this setting is one which contains the minimum number of actions. If $\Pi_0, \ldots, \Pi_{n-1}$ are unsatisfiable and Π_n has a solution, then an optimal plan is found and has cost n.

The OMT encodings (3) instead are the most general and extend OMTPlan's capabilities to perform optimal planning under constant and state-dependent action costs alike. Optimisation objectives are defined as pseudo-boolean expressions for problems with unitary costs and *QFLRA* expressions for the other cases.

All the encodings above combine elements of the standard Planning as SAT encoding with novel abstraction techniques we presented in [11]. Building on these results, OMTPlan is able to *(i)* detect efficiently if a goal is not reachable without requiring any unrolling the transition system and *(ii)* provide a new sufficient condition for determining whether a plan is a global optimum for the planning problem. The latter feature is crucial when dealing with SDAC, as plans with the least number of actions are not necessarily cost-optimal. When a locally optimal solution is found for Π_n, proving that this is also a global optimum would require checking that longer unrollings do not yield better plans, for all unrollings of length greater than n. We overcome this challenge by extending the classical encoding with *(i)* a boolean abstraction that disregards concrete numeric effects and allows to reason on unbounded executions and *(ii)* loop formulas [13] which enforce a notion of optimality in the abstracted boolean space. Due to space constraint we cannot report all the details of our encoding here; we kindly refer the reader to [11] for that.

Finally, this module also allows to export SMT-LIB encodings [1] of planning formulas. This functionality serves two different purposes:

- **debugging**: SMT-LIB encodings can be used to detect bugs in the logic of the encoder;
- **benchmarking**: benchmarks developed by the planning community can be used to test OMT solvers, as done, e.g. in [3].

Validation. When a plan is found, the validation module is called to check its correctness against the PDDL domain and problem files. To this end, a plan is extracted from the model of the planning formula and converted back into PDDL syntax. The validation task is then performed by the plan validator VAL,[2] which checks whether the plan complies with the PDDL description of the problem. If a plan is deemed valid, it is passed on to the main routine for subsequent operations. Otherwise, OMTPlan reports a failure.

3. Experimental Evaluation

We demonstrate the capabilities of OMTPlan on optimal numeric planning problems taken from the literature [11,12,16]. The benchmarks include *simple* domains, where numeric effects of actions are restricted to assignments to constant values, and *linear* ones, where effects can be described by linear expressions. All instances are satisfiable, i.e. they admit a valid plan.

The current implementation of OMTPlan relies on Z3/νZ [4] to construct and solve planning formulas.[3] However other OMT solvers could be used for the latter step via a compilation to SMT-LIB.

We test the three configurations of our planner that support optimal planning, namely:

- **SMT-s**: SMT encoding with serial execution and linear increment;
- **OMT-s**: OMT encoding with serial execution and exponential increment;
- **OMT-p**: OMT encoding with parallel execution and exponential increment.

We compare[4] OMTPlan against two state-of-the-art numeric planners: ENHSP [16], based on A^* search [7], and C^{SC}, based on a reduction to MILP [14]. We report the number of instances solved for each domain (coverage) and the corresponding solving time.

We run our experiments using a 30 minute timeout and 4 GB memory limits on a machine running Debian 3.16 with processor Intel(R) Xeon(R) CPU E5- 2640 v4 @ 2.40 GHz.

Table 1 shows coverage and total solving time for simple numeric domains with unitary costs. C^{SC} appears to be the most effective tool at solving this class of problems, being able to solve the largest number of instances in the least total time. ENHSP and SMT-s follow, with the former yielding better performance on domains that feature long plans (i.e. requiring many actions) that force OMTPlan to construct large encodings (e.g. Sailing). On the other hand, SMT-s outperforms ENHSP on domains admitting shorter solutions. OMT encodings do not show their full potential here: simple numeric problems with unitary costs require limited optimisation capabilities; however, this is not detected by the OMT solver which resorts to expensive optimisation procedures that may result in an unnecessary overhead.

Table 2 shows the results obtained for linear numeric domains with unitary, constant (-Metric) and state-dependent (-SDAC) costs. These experiments show a completely different picture from what is observed for simple domains. The increased expressivity of these problems poses a considerable challenge to C^{SC} and ENHSP. OMTPlan-p, on the other hand,

[2] Available at https://github.com/KCL-Planning/VAL.
[3] Available at https://github.com/Z3Prover/z3.
[4] Results for ENHSP and C^{SC} are taken from [11] and reported here for comparison. The exact same hardware has been used for both sets of experiments.

Table 1. Total number of instances (#), coverage (C) and total solving time (T) for simple numeric domains with unitary costs

Domain	#	SMT-s		OMT-s		OMT-p		ENHSP		**C^{SC}**	
		C	T (s)	C	T (s)	C	T (s)	C	T (s)	C	T (s)
Counters	15	5	1678.88	4	14.95	7	150.96	6	28.22	**15**	1.36
Depots	20	2	81.36	1	22.72	1	58.09	**3**	1050.22	1	4.9
Gardening	63	46	2893.72	24	3843.95	25	2066.70	**63**	599.85	**63**	887.33
Sailing	20	5	64.46	5	167.07	4	354.36	16	2101.13	**17**	2813.55
Satellite	20	**4**	191.20	0	–	1	21.70	2	293.10	**4**	459.80
Rover (1-10)	10	**4**	27.99	**4**	380.24	**4**	82.84	**4**	25.91	**4**	10.93
Zenotravel (1-10)	10	**8**	323.55	4	175.69	4	146.09	6	579.30	7	699.65

Table 2. Total number of instances (#), coverage (C) and total solving time (T) for linear domains with unitary, constant (-Metric) and state-dependent (-SDAC) costs. **n.s.**: tool does not support optimal reasoning on the problem; **s.f.**: a segmentation fault occurred on all instances

Domain	#	SMT-s		OMT-s		OMT-p		ENHSP		**C^{SC}**	
		C	T (s)	C	T (s)	C	T (s)	C	T (s)	C	T (s)
FO-Counters (1-10)	10	3	4.64	3	101.83	**9**	1989.84	4	339.84	3	223.83
FO-Counters-Inv (1-10)	10	2	2.02	2	13.89	**6**	1051.67	3	77.29	2	48.82
FO-Counters-Rnd (1-30)	30	13	1163.94	11	185.37	**23**	2917.34	14	1411.79	10	520.29
FO-Farmland	20	3	232.52	1	18.38	2	754.41	**13**	1035.09	2	47.07
Rover-Metric (1-10)	10	n.s.	n.s.	4	957.01	**5**	291.89	4	151.69	4	14.02
TPP-Metric (1-10)	10	n.s.	n.s.	0	–	3	484.81	**5**	20.51	s.f.	s.f.
Zenotravel-Metric (1-10)	10	**8**	1799.74	4	338.50	4	1088.98	4	145.55	2	1.55
SecurityClearance-SDAC	30	n.s.	n.s.	8	1427.47	**26**	2115.61	16	952.36	n.s.	n.s.

is able to harness the power of the underlying SMT/OMT technology to handle this added expressivity and proves to be effective. This is particularly evident in domains that feature a high-degree of parallelism, which OMTPlan can exploit to build more succinct encodings. Similar results have also been reported by [10], where OMTPlan outperformed SoA planners on linear domains. We highlight that C^{SC} does not support planning with state-dependent costs; ENHSP does not provide admissible heuristics for linear numeric planning. As a result, the planner had to be run using uninformed search that disregards cost structures.

4. Conclusions

We presented OMTPlan, the first domain independent planner based on Optimisation Modulo Theories. Building upon the theoretical results of [11], OMTPlan implements reductions to SMT and OMT to solve expressive planning problems for which tool support was lacking. We discussed the internal work-flow of OMTPlan and explained its main functionalities. Notably, OMTPlan can be used to convert PDDL problems into SMT-LIB format, thus enabling fruitful interactions between the planning and SMT/OMT communities [3]. We

reported numeric experiments that demonstrate the capabilities of OMTPlan. Our results, in combination with those already presented in [11], show that OMTPlan offers an efficient solution to solve planning problems that require complex theory reasoning.

Acknowledgements

This work was partly supported by the Imperial College Research Fellowship scheme.

References

[1] C. Barrett, P. Fontaine and C. Tinelli, *The Satisfiability Modulo Theories Library*, 2016.

[2] C.W. Barrett, R. Sebastiani, S.A. Seshia and C. Tinelli, Satisfiability modulo theories, in: *Handbook of Satisfiability*, Vol. 185, IOS Press, 2009, pp. 825–885.

[3] F. Bigarella, A. Cimatti, A. Griggio, A. Irfan, M. Jonás, M. Roveri, R. Sebastiani and P. Trentin, Optimization modulo non-linear arithmetic via incremental linearization, in: *Proceedings of FroCoS'21*, LNCS, Vol. 12941, Springer, 2021, pp. 213–231.

[4] N. Bjørner, A. Phan and L. Fleckenstein, νZ – an optimizing SMT solver, in: *Proceedings of TACAS'15*, LNCS, Vol. 9035, Springer, 2015, pp. 194–199.

[5] M. Fox and D. Long, PDDL2.1: An extension to PDDL for expressing temporal planning domains, *J. Artif. Intell. Res.* **20** (2003), 61–124. doi:10.1613/jair.1129.

[6] H. Geffner and B. Bonet, *A Concise Introduction to Models and Methods for Automated Planning, Synthesis Lectures on Artificial Intelligence and Machine Learning*, Morgan & Claypool Publishers, 2013.

[7] P.E. Hart, N.J. Nilsson and B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Syst. Sci. Cybern.* **4**(2) (1968), 100–107. doi:10.1109/TSSC.1968.300136.

[8] M. Helmert, Concise finite-domain representations for PDDL planning tasks, *Artif. Intell.* **173**(5–6) (2009), 503–535. doi:10.1016/j.artint.2008.10.013.

[9] F. Ivankovic, P. Haslum, S. Thiébaux, V. Shivashankar and D.S. Nau, Optimal planning with global numerical state constraints, in: *Proceedings ICAPS'14*, 2014.

[10] R. Kuroiwa and C. Beck, A branch-and-cut approach for a mixed integer linear programming compilation of optimal numeric planning, in: *HSDP@ICAPS*, 2021.

[11] F. Leofante, E. Giunchiglia, E. Ábrahám and A. Tacchella, Optimal planning modulo theories, in: *Proceedings of IJCAI'20*, 2020, pp. 4128–4134.

[12] D. Li, E. Scala, P. Haslum and S. Bogomolov, Effect-abstraction based relaxation for linear numeric planning, in: *Proceedings of IJCAI'18*, 2018, pp. 4787–4793.

[13] F. Lin and Y. Zhao, ASSAT: Computing answer sets of a logic program by SAT solvers, in: *Proceedings of AAAI'02*, 2002, pp. 112–118.

[14] C. Piacentini, M. Castro, A. Ciré and C. Beck, Compiling optimal numeric planning to mixed integer linear programming, in: *Proceedings of ICAPS'18*, 2018, pp. 383–387.

[15] J. Rintanen, Planning and SAT, in: *Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications*, Vol. 185, IOS Press, 2009, pp. 483–504.

[16] E. Scala, P. Haslum, S. Thiébaux and M. Ramírez, Interval-based relaxation for general numeric planning, in: *Proceedings of ECAI'16*, Vol. 285, IOS Press, 2016, pp. 655–663.

[17] R. Sebastiani and S. Tomasi, Optimization modulo theories with linear rational costs, *ACM Trans. Comput. Log.* **16**(2) (2015), 12:1–12:43. doi:10.1145/2699915.