From MINIZINC to Optimization Modulo Theories, and Back

Francesco Contaldo, Patrick Trentin, Roberto Sebastiani

DISI, University of Trento, Italy

Abstract. Optimization Modulo Theories (OMT) is an extension of SMT that allows for finding models that optimize objective functions. In this paper we aim at bridging the gap between Constraint Programming (CP) and OMT, *in both directions*. First, we have extended the OMT solver OPTIMATHSAT with a FLATZINC interface – which can also be used as FLATZINC-to-OMT encoder for other OMT solvers. This allows OMT tools to be used in combination with MZN2FZN on the large amount of CP problems coming from the MINIZINC community. Second, we have introduced a tool for translating SMT and OMT problems on the linear arithmetic and bit-vector theories into MINIZINC. This allows MINIZINC solvers to be used on a large amount of SMT/OMT problems.

We have discussed the main issues we had to cope with in either directions. We have performed an extensive empirical evaluation comparing three state-of-the-art OMT-based tools with many state-of-the-art CP tools on (i) CP problems coming from the MINIZINC challenge, and (ii) OMT problems coming mostly from formal verification. This analysis also allowed us to identify some criticalities, in terms of efficiency and correctness, one has to cope with when addressing CP problems with OMT tools, and vice versa.

1 Introduction

The last two decades have witnessed the rise of Satisfiability Modulo Theories (SMT) [11] as efficient tool for dealing with several applications of industrial interest, in particular in the contexts of Formal Verification (FV). SMT is the problem of finding value assignments satisfying some formula in first-order logic wrt. some background theory. Optimization Modulo Theories (OMT) [40,51,34,32,54,15,37,31] is a more-recent extension of SMT searching for the *optimal* value assignment(s) w.r.t. some objective function(s), by means of a combination of SMT and optimization procedures. (Since OMT extends SMT, hereafter we often simply say "OMT" for both SMT and OMT.)

Several distinctive traits of OMT solvers –like, e.g., the efficient combination of Boolean and arithmetical reasoning, incrementality, the availability of decision procedures for infinite-precision arithmetic and the ability to produce conflict explanations– are a direct consequence of their tight relationship with the FV domain and its practical needs. On the whole, it appears that OMT can be a potentially interesting and efficient technology for dealing with Constraint Programming (CP) problems as well. At the same time, modeling CP problems for OMT solvers requires a higher-level of expertise, because the same CP instance can have many possible alternative formulations, s.t. the performance of SMT solvers on each encoding are hardly predictable [27,26,24].

On the other hand, the availability, efficiency and expressiveness of CP tools makes them of potential interest as backend engines also for FV applications (e.g., [21,22,29]), in particular with SW verification, where currently SMT is the dominating backend technology, s.t. a large amount of SMT-encoded FV problems are available [10].

In this paper we aim at bridging the gap between CP and OMT, *in both directions*.

In the CP-to-OMT direction, we have extended the state-of-the-art OMT solver OP-TIMATHSAT [56] with a FLATZINC interface (namely "FZN2OMT"). In combination with the standard MZN2FZN encoder [36], this new interface can be used to either (i) solve CP models with OPTIMATHSAT directly or (ii) generate OMT formulas encoded in the SMT-LIB [23] format with optimization extensions, to be fed to other OMT solvers, such as BCLT [16] and Z3 [15]. This allows state-of-the-art OMT technology to be used on MINIZINC problems coming from the CP community.

In the OMT-to-CP direction, we have introduced a tool for translating SMT and OMT problems on the theories of linear arithmetic over the integers and rational ($\mathcal{LIRA}$) and bit-vector ($\mathcal{BV}$) into MINIZINC models (hereafter "OMT2MZN"). This allows MINIZINC solvers to be used on OMT problems, giving them access to a large amount of OMT problems, mostly coming from formal verification.

With both directions, we first present and discuss the challenges we encountered and the solutions we adopted to address the differences between the two formalisms. Then we present an extensive empirical evaluation comparing three OMT tools with many state-of-the-art CP tools on (i) CP problems coming from the MINIZINC challenge, and (ii) OMT problems coming mostly from formal verification. This analysis allowed us to identify some criticalities, in terms of efficiency and correctness, one has to cope with when addressing CP problems with OMT tools, and vice versa.

Overall, our new compilers FZN2OMT and OMT2MZN in combination with the standard compiler MZN2FZN [36] provide a framework for translating problems encoded in the SMT-LIB or the MINIZINC format in either direction. This framework enables also for a comparison between OMT solvers and CP tools on problems that do not belong to their original application domain. To the best of our knowledge, this is the first time that such a framework has been proposed, and that the OMT and CP technologies have been extensively compared on problems coming from both fields.

Related Work. The tight connection between SMT and Constraint Programming (CP) has been known for a relatively long period of time [41] and it has previously been subject to investigation. Some works considered a direct encoding of CP [27,26] and weighted CP [7] into SMT and MAXSMT, or an automatic framework for translating MINIZINC –a standard CP modeling language [38]– into SMT-LIB –the standard SMT format– [18,17]. Other works explored the integration of typical SAT and SMT techniques within CP solvers [43,25]. Nowadays, several MINIZINC solvers –like, e.g., HAIFACSP [59] and PICAT [60]– are at least partially based on SAT technology.

To this extent, our first contribution FZN2OMT also obviates the loss, due to obsolescence, of the FZN2SMT compiler proposed by Bofill et al. in [18,17]. FZN2SMT is not compatible with the changes that have been introduced to the MINIZINC and FLATZINC standards starting from version 2.0 of the MINIZINC distribution. Since some of these changes are not backward compatible, it is also not possible to use FZN2SMT in

conjunction with an older version of the MZN2FZN compiler when dealing with recent MINIZINC models. Furthermore, FZN2SMT translates satisfaction problems into the Version 1 of the SMT-LIB standard and produces no SMT-LIB output in the case of optimization problems, that are solved directly. However, the optimization interface of modern OMT solvers is based on the Version 2 of the SMT-LIB standard. This makes it difficult to use it together with OMT solvers. Unfortunately, the FZN2SMT compiler is closed source, with only the binaries being freely distributed, and seemingly no longer maintained. This made it necessary to provide a new alternative solution to FZN2SMT. To this extent, our new FLATZINC interface of OPTIMATHSAT, FZN2OMT, translates both satisfaction and optimization problems in the Version 2 of the SMT-LIB standard enriched with the optimization extensions for OMT described in [56].

Content. The rest of the paper is organized as follows. In §2 we provide some background on OMT, MINIZINC and FLATZINC. In §3 we describe the process from MINIZINC to OMT. In §4 we describe the process from OMT to MINIZINC. In §5 we describe an empirical evaluation comparing a OMT-based tool with many state-of-the-art CP tools. Finally, in §6 we conclude and point out some further research directions.

A longer and more detailed version of this paper is publicly available as [23].

2   Background

Satisfiability Modulo Theories (SMT) is the problem of deciding the satisfiability of a first-order formula φ with respect to a combination of decidable first-order theories. Typical theories of SMT interest are (the theory of) linear arithmetic over the rationals ($\mathcal{LRA}$), the integers ($\mathcal{LIA}$) or their combination ($\mathcal{LIRA}$), non-linear arithmetic over the rationals ($\mathcal{NLRA}$) or the integers ($\mathcal{NLIA}$), arrays ($\mathcal{AR}$), bit-vectors ($\mathcal{BV}$), floating-point arithmetic ($\mathcal{FP}$), and their combinations thereof. (See [42,50,11] for an overview.). The last two decades have witnessed the development of very efficient SMT solvers based on the so-called lazy-SMT schema [50,11]. This has brought previously-intractable problems to the reach of state-of-the-art SMT solvers.

Optimization Modulo Theories (OMT), [40,51,34,32,54,15,37,58], is an extension to SMT that allows for finding a model of a first-order formula φ that is *optimal* with respect to some objective function expressed in some background theory, by means of a combination of SMT and optimization procedures. State-of-the art OMT tools allow optimization in a variety of theories, including linear arithmetic over the rationals (OMT($\mathcal{LRA}$)) [51] and the integers (OMT($\mathcal{LIA}$)) [15,54], bit-vectors (OMT($\mathcal{BV}$)) [37] and floating-point numbers (OMT ($\mathcal{FP}$)) [58].

A relevant strict subcase of OMT($\mathcal{LRA}$) is OMT with Pseudo-Boolean objective functions (OMT($\mathcal{PB}$)) in the form $\sum_i w_i A_i$ s.t. w_i are rational values and A_i are Boolean variables whose values are interpreted as $\{0, 1\}$. Notice that OMT($\mathcal{PB}$) is also equivalent to (partial weighted) MAXSMT, the SMT extension of MAXSAT, and that OMT($\mathcal{PB}$) and MAXSMT can be encoded into OMT($\mathcal{LRA}$) but not vice versa [52]. Encoding OMT($\mathcal{PB}$)/MAXSMT into OMT($\mathcal{LRA}$), however, is not the most efficient way to solve them, so that modern OMT solvers such as BCLT [16], OPTIMATHSAT [56] and Z3 [15] implement specialized OMT($\mathcal{PB}$)/MAXSMT procedures which are much more efficient than general-purpose OMT($\mathcal{LRA}$) ones [15,55,56].

3

We stress the fact that —unlike with purely-combinatorial problems, which are encoded into SAT or MAXSAT and are thus solved by purely-Boolean search– typically OMT problems involve the interleaving of *both* Boolean and arithmetical search: search not only for the best truth-value assignment to the atomic subformulae, but also for the best values to the numerical variables compatible with such truth-value assignment [52].

To this date, few OMT solvers exist, namely BCLT [16], CEGIO [8], HAZEL [37], OPTIMATHSAT [56], PULI [31], SYMBA [34] and Z3 [15]. To this aim, we observe that (i) some of these solvers are quite recent, (ii) most of these solvers focus on different, partially overlapping, niche subsets of Optimization Modulo Theories, and (iii) the lack of an official Input/Output interface for OMT makes it hard to compare some of these tools with one another. OMT finds applications in the context of static analysis [19,30], formal verification and model checking [35,46], scheduling and planning with resources [31,33,48,44], software security and requirements engineering [39], workflow analysis [12], machine learning [57], and quantum computing [13].

A distinctive trait of SMT (and OMT) solvers is the trade-off of speed against the ability to certify the correctness of the result of any computation, which is particularly important in the contexts of Formal Verification (FV) and Model Checking (MC). When dealing with linear arithmetic in particular, SMT solvers employ *infinite-precision arithmetic* software libraries to avoid numerical errors and overflows.

SMT-LIB [23] is the standard input format by SMT solvers, it provides a standardized definition of the most prominent theories supported by SMT solvers and the corresponding language primitives to use these features. At present, there is no standard input format for modeling optimization problems targeting OMT solvers, although there exist only minor syntactical differences between the major OMT solvers. The tools presented in this paper conform to the *extended* SMT-LIB format for OMT presented in [56], that includes language primitives for modeling objectives.

MINIZINC [38,36] is a widely adopted high-level declarative language for modeling Constraint Satisfaction Problems (CSP) and Constraint Optimization Problems (COP). The MINIZINC format defines three scalar types (bool, int and float) and two compound types (sets and fixed-size arrays of some scalar type). The standard provides an extensive list of predefined *global constraints*, a class of high-level language primitives that allows one to encode complex constraints in a compact way.

FLATZINC is a lower-level language whose purpose is to bridge the gap between the high-level modeling in MINIZINC and the need for a fixed, and easy-to-parse, format that can simplify the implementation of the input interface of MINIZINC solvers. A MINIZINC model is typically flattened into a FLATZINC instance using the MZN2FZN compiler [36], and then solved with some MINIZINC tool.

3 From MINIZINC to OMT

We consider the problem of translating MINIZINC models into OMT problems first. Similarly to other MINIZINC solvers, we assume that the MINIZINC model is first translated into FLATZINC using the MZN2FZN standard compiler, as depicted in Figure 1. We describe the main aspects of FZN2OMT, focusing on the challenges we have encountered and on the solutions we have adopted.
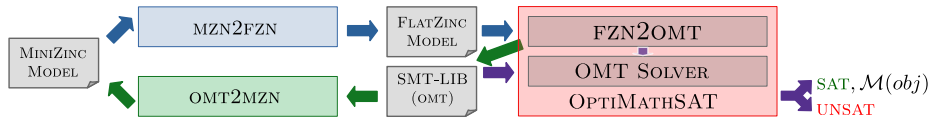
Fig. 1. Circular translation schema from MiniZinc to SMT-LIB and back, resulting from the composition of mzn2fzn, OptiMathSAT and omt2mzn. In this picture, OptiMathSAT acts both as a FlatZinc/OMT solver, and also as a FlatZinc to SMT-LIB compiler.

FlatZinc *data-types.* The first challenge is to find a suitable representation of the data-types supported by FlatZinc in SMT-LIB.

One possible choice for modeling the three basic scalar types of FlatZinc –namely `bool`, `int` and `float`– with SMT-LIB are the Boolean, bit-vector and floating-point theories respectively. However, the decision procedures for the bit-vector and floating-point numbers theories can be significantly more resource demanding than the decision procedure for the linear arithmetic theory ($\mathcal{LIRA}$), in particular when dealing with a substantial amount of arithmetic computations. For this reason, we have opted to model FlatZinc `int` and `float` data-types with the SMT-LIB integer and rational types respectively, by default. For the case in which no substantial linear arithmetic computation is performed, we also optionally allow for encoding the FlatZinc `int` data-type as a SMT-LIB bit-vector.

For what concerns the two compound types of FlatZinc, that is the `set` and `array` data-types, we have chosen to proceed as follows. Given that OptiMathSAT lacks a decision procedure for the theory of finite sets [6], we model a set using the Boolean and integer theories, similarly to what has been done in [17]. The basic idea is to introduce a fresh Boolean variable for each element in the domain of a set, and use such variable as a placeholder for the membership of an integer element to the set instance. Differently from [17], we make an extensive use of cardinality networks [9] to encode constraints over the sets because they are handled more efficiently, for their nice arc-consistency properties. No action is required to encode a FlatZinc `array` into SMT-LIB, because it is used only as a container for other variables.

Floating-Point precision. A consequence of encoding the FlatZinc `int` and `float` data-types with the linear arithmetic theory is that all of our computation is performed with infinite-precision arithmetic. This can result in a performance disadvantage wrt. other MiniZinc solvers using finite-precision arithmetic, due to the increased cost of each operation, but it has the benefit of guaranteeing the correctness of the final result of the computation.

Currently, the MiniZinc language does not allow one to express a certain quantity as an infinite-precision fraction between two constant numbers. Instead, the mzn2fzn compiler computes on-the-fly the result of any division operation between two constant integers or floating-point numbers applying the rules corresponding to the type of the operands. However, there are some instances in which we really need to be able to both express quantities and perform computation with infinite-precision arithmetic. One of

such situations is to double-check the correctness of the MINIZINC models generated by the OMT2MZN compiler described in Section §4 (we have done this for the experimental evaluations in Section §5.2). In order to get around this limitation we developed a simple wrapper around the MZN2FZN compiler, called EMZN2FZN [4], that replaces any fraction among two constant `floats` with a fresh variable, and then sets its value using the FLATZINC constraint `float_div()`.

FLATZINC *constraints.* The SMT-LIB encoding of the majority of FLATZINC constraints in OPTIMATHSAT follows their definition in the FLATZINC Standard, with the exception of Pseudo-Boolean constraints, which we examine in detail later on. Several global constraints are also supported in the same way, because the OMT-solver currently lacks ad hoc and efficient decision procedures for dealing with them. Constant values and alias variables (e.g. those arising from the definition of some arrays) are propagated through the formula, so as to keep the set of problem variables as compact as possible. Those constraints requiring non-linear arithmetic –like, e.g., trigonometric, logarithmic and exponential functions– are currently not supported; this situation may change soon due to the recent extension of MATHSAT5 with a procedure for it [20].

Pseudo-Boolean constraints. When dealing with Pseudo-Boolean sums of the form $\sum_{i=1}^{i=N} A_i \cdot w_i$, where A_i is a Boolean variable and w_i is a numerical weight, the MZN2FZN compiler associates a fresh 0/1 variable a_i to each A_i, and encodes the sum as $\sum_{i=1}^{i=N} a_i \cdot w_i$. Notice that the original A_is may not be eliminated from the FLATZINC model, because they typically occur elsewhere in the problem, i.e. as part of a Boolean formula. From our own experience, this situation arises frequently, because Pseudo-Boolean sums are typically used to express cardinality constraints that have a variety of uses. As described in [55], one limitation of this naive approach is that SMT and OMT solvers do not typically handle this encoding efficiently. The main reason is that the pruning power of the conflict clause resulting from a conflicting assignment is typically limited to one specific Boolean assignment at a time, meaning that a large number of conflict clauses (possibly exponential) has to be generated along the search. As shown in [55], SMT and OMT solvers can benefit from encoding Pseudo-Boolean constraints with cardinality networks.

FZN2OMT goes through some effort in order to recognize Pseudo-Boolean sums over the integers, and replace the naive encoding with one based on cardinality networks. We note that using this technique generally results in a trade-off between solving time and the overhead of generating cardinality networks prior to starting the search, especially when dealing with a large number of variables.

Multi-objective optimization. FZN2OMT allows for multiple optimization goals, of heterogeneous type, being defined within the same FLATZINC model. This is a non-standard extension to the FLATZINC format. Multiple objectives can be solved independently from one another, or combined into a Lexicographic or Pareto optimization goal. We refer the reader to [56] for details on the input encoding and the solver configuration.

Functionality. Given a satisfiability or optimization problem encoded in the FLATZINC format, OPTIMATHSAT can be used in the following ways (Figure 1):

- to directly solve the problem, optionally enumerating any sub-optimal solution found during the search or all possible solutions with the same optimal value;
- to produce an OMT problem encoded with the extended SMT-LIB format described in [56]. This problem can be directly solved with OPTIMATHSAT or, with minor transformations[1], fed as input to other OMT solvers such as BCLT and Z3.

4   From OMT to MINIZINC

In this section, we consider the problem of translating OMT formulas, encoded in the optimization-extended SMT-LIB format of [56], into MINIZINC models. Hereafter, we describe the main challenges we have faced and the solutions we have adopted. Further details about this conversion are available in [3].

General Translation Approach. The main challenge is to design an encoding from OMT to MINIZINC that is *correct* (i.e., it preserves in full the semantics of the input OMT problems), *effective* (i.e., it produces as output MINIZINC models which are as compact and easy-to-solve as possible), and *efficient* (i.e. it does it with the least consumption of time and memory). To this extent, one critical design choice is the way in which the internal representation of the input OMT formula is organized and converted in terms of MINIZINC primitives. After a preliminar experimental evaluation we determined that the sweet-spot, in terms of compactness and easiness to solve of the resulting MINIZINC model, is to adopt what we call "$\geq$2-father DAG-ification": a Directed-Acyclic-Graph (DAG) internal representation of the formula where a fresh label is associated to all and only DAG nodes with at least two fathers, inlining all other nodes

Theories Restriction. The SMT-LIB standard describes a wide number of SMT theories, most of which have no direct counterpart in MINIZINC due to the few data-types supported (see §2). Hence, hereafter we deal only with the theory of linear rational and integer arithmetic, and their combination. On this regard, we note that even though OMT2MZN can also handle the theory of bit-vectors, we do not cover it here because it is not used in the experimental evaluation in Section §5 (We cover it in the long version of this paper [23]). We leave the handling of other SMT theories to future work.

Linear Arithmetic Theory. On the surface, encoding linear arithmetic constraints coming from OMT in MINIZINC, using the *int* and *float* data-types, looks like a trivial task. In reality, this poses several challenges and it is subject to several limitations, due to a couple of facts.

   First, in SMT-LIB the linear arithmetic theory requires the capability to perform *infinite-precision* computations. Unfortunately, to the best of our knowledge, no MINIZINC solver provides infinite-precision arithmetic reasoning, and the MZN2FZN compiler itself prevents representing arbitrarily-large and arbitrarily-precise quantities (e.g. the fine-grained decimal weights of the machine learning application in [57]).

   Second, in OMT linear arithmetic variables are not required to be bounded and have quite often no explicit domain (i.e. they lack a lower-bound, an upper-bound or both),

---

[1]To make this step as easy as possible, we collected our scripts into a public repository [1].

because it is not necessary for the problem at hand or it is implied by other constraints. This is in contrast with MINIZINC, whereby linear arithmetic variables are expected to have a finite domain and, when they lack one, their domain appears to be capped with some solver-dependent pair of values.

These restrictions are currently part of the MINIZINC language and the target application domain, and we do not see any obvious work-around solving them. We note that although there exist methods for bounding all variables in a given LP problem (e.g. [45]), these have been deemed too impractical at this stage of our investigation. Nonetheless, we have chosen to translate SMT-LIB linear arithmetic constraints with a corresponding MINIZINC encoding based on the *int* and *float* data-types. Although the encoding is not always applicable, it does still allow one to correctly translate a number of interesting OMT problems into MINIZINC, as witnessed by our experimental evaluation in Section §5.2. More in detail, the translation is done as follows. We declare each integer variable as unbounded, and then extend the MINIZINC model with the appropriate constraints bounding its domain when the input OMT formula contains any such information. Our empirical observation is that MINIZINC models generated in this way are correctly handed by all MINIZINC solvers which we have tried, with the exception of GUROBI, which returns an "unsupported" message. Floating-Point variables, instead, are always declared with a user-defined domain. This is because all of the MINIZINC solvers we have tried, among those that can handle floating-point constraints, require such information.

Other OMT Functionalities. Several problems of OMT interest require the capability of dealing with soft-constraints (i.e. Weighted MAXSMT) and also with multiple objectives, that are either considered independent goals or combined in a Lexicographic or Pareto-like fashion. To the best of our knowledge, the MINIZINC standard does not allow for an explicit encoding of soft-constraints, nor to deal with more than one objective function at a time.

We encode (weighted) MAXSMT problems using a standard Pseudo-Boolean encoding, such as the one used in [51]. When dealing with OMT problems that contain N goals $\mathsf{obj}_1, ..., \mathsf{obj}_N$, for $N > 1$, we use the following approach. If these objectives are independent targets, we generate N MINIZINC models, each with a different goal obj_i, and separately solve each model. If instead the multiple objectives belong to a Lexicographic OMT problem, then we generate a unique MINIZINC model that leverages the lexicographic-optimization functionality provided by MINISEARCH [47]. (In all other cases, MINISEARCH is not used). We do not have any encoding for dealing with Pareto-optimization, yet.

5   Experimental Evaluations

In this section we present an extensive empirical evaluation comparing OMT tools with many state-of-the-art CP tools on CP problems coming from the MINIZINC challenge (§5.1), and on OMT problems coming mostly from formal verification (§5.2).

The OMT solvers under evaluation are BCLT, OPTIMATHSAT (v. 1.6.0) and Z3 (v. 4.8.5). These are compared with some of the top-scoring solvers that participated

at recent editions of the MINIZINC challenge, including CHOCO (v. 4.0.4), CHUFFED, G12(FD) (v. 1.6.0), GECODE (v. 6.0.1), GUROBI (v. 8.0.1), HAIFACSP (v. 1.3.0), JA-COP (v. 4.5.0), IZPLUS (v. 3.5.0), OR-TOOLS (v. 6.7.4981) and PICAT (v. 2.4).

Remark 1. We could not include FZN2SMT [18,17] in our experimental evaluation because it is not compatible with the features of MINIZINC that have been added since version 2.0.

We run all these experimental evaluations on two identical *8-core 2.20Ghz Xeon* machines with $64GB$ of RAM and running *Ubuntu Linux*. All the benchmark-sets, the tools and the scripts used to run these experiments, and some of the plots for the results in Tables 1-3 which could not fit into this paper, can be downloaded from [2].

We stress the fact that the goal of these experiments is not to establish a winner among OMT and MINIZINC tools; rather, it is to assess the correctness, effectiveness and efficiency of our OMT-to-CP and CP-to-OMT encoders and, more generally, to investigate the feasibility of solving MINIZINC problems with OMT tools and vice versa, and to identify the criticalities in terms of efficiency and correctness in these processes.

5.1 Evaluation on MINIZINC Benchmark Sets

We consider the benchmark-sets used at the MINIZINC Challenge of 2016 (MC16) and 2019 (MC19), each comprised by 100 instances. For compatibility reasons, the version of MZN2FZN used to convert the problems to the FLATZINC format differs between the two benchmark-sets. We use version 2.2.1 and 2.3.2 (with patches) for the problems in MC16 and MC19 respectively. Due to recent changes in the FLATZINC format that affect the benchmarks in MC19, the version of some MINIZINC tools differs from what described in Section §5 (see Table 1). In some cases, we had to download and compile the latest source available for the tool, i.e. the "nightly" version.

We run each MINIZINC solver with the corresponding directory of global constraints, and we run each MINIZINC and OMT tool with the default options. We consider two OMT encodings of the original FLATZINC problems, LA and BV. The first encodes the FLATZINC *int* type with the theory of *linear integer arithmetic*, whereas the second is based on the theory of *bit-vectors*. We evaluate each OMT solver on both SMT-LIB encodings, except for BCLT that has no support for bit-vector optimization. For uniformity reasons with the other OMT solvers, we evaluate OPTIMATHSAT using its SMT-LIB interface only, using thus its FZN2OMT interface as an external tool, like with the other OMT solvers. We note that the solving time for all OMT solvers includes the time required for translating the formula from the FLATZINC to the SMT-LIB format. Each solver, either OMT or MINIZINC, is given up to $1200s$. to solve each problem, not including the time taken by MZN2FZN to flatten it.

We verify the correctness of the results by automatically checking that all terminating solvers agree on the (possibly optimal) solution and, when this is not the case, we manually investigate the inconsistency.

			terminated						
	inst.	timeout	error	unsup.	incor.	correct	time (s.)	BT1	BT2
tool, configuration & encoding	s | o	s | o	s | o	s | o	s | o	s | o	s | o	s | o	s | o
MINIZINC Challenge 2016									
PICAT(CP)	15 | 85	9 | 70	0 | 0	0 | 0	0 | 0	6 | 15	2281 | 6043	0 | 0	0 | 0
G12(FD)	15 | 85	4 | 71	1 | 3	0 | 0	0 | 0	10 | 11	4436 | 4220	0 | 0	0 | 0
CHOCO()	15 | 85	3 | 50	0 | 0	0 | 0	0 | 0	12 | 35	4256 | 11423	1 | 0	1 | 0
IZPLUS()	15 | 85	6 | 44	0 | 0	0 | 0	0 | 0	9 | 41	999 | 5492	3 | 4	3 | 4
CHUFFED()	15 | 85	2 | 40	0 | 0	5 | 0	0 | 0	8 | 45	635 | 4187	0 | 5	0 | 5
JACOP()	15 | 85	3 | 39	0 | 0	0 | 0	0 | 0	12 | 46	3411 | 12825	0 | 0	0 | 0
GUROBI()	15 | 85	6 | 22	0 | 0	0 | 0	0 | 0	9 | 63	2346 | 3037	0 | 15	0 | 15
HAIFACSP()	15 | 85	4 | 23	0 | 0	0 | 0	0 | 0	11 | 62	591 | 4444	0 | 11	0 | 11
PICAT(SAT)	15 | 85	1 | 26	0 | 0	0 | 0	0 | 0	14 | 59	151 | 7293	10 | 1	10 | 1
OR-TOOLS(SAT)	15 | 85	1 | 15	0 | 0	0 | 0	0 | 0	14 | 70	555 | 1338	1 | 45	1 | 45
VIRTUAL BEST(MINIZINC)	15 | 85	0 | 7	0 | 0	0 | 0	0 | 0	15 | 78	146 | 3514	- | -	- | -
OPTIMATHSAT(INT)	15 | 85	10 | 38	0 | 0	0 | 0	0 | 0	5 | 47	604 | 4856	1 | 20	0 | 0
OPTIMATHSAT(BV)	15 | 85	2 | 42	0 | 0	0 | 0	0 | 0	13 | 43	3664 | 8561	11 | 2	0 | 0
BCLT(INT)	15 | 85	10 | 33	0 | 0	0 | 0	0 | 0	5 | 52	1117 | 5998	0 | 15	0 | 2
Z3(INT)	15 | 85	10 | 32	0 | 0	0 | 0	0 | 0	5 | 53	676 | 10424	0 | 11	0 | 0
Z3(BV)	15 | 85	5 | 28	0 | 0	0 | 0	0 | 0	10 | 57	2938 | 11113	2 | 19	0 | 0
VIRTUAL BEST(OMT)	15 | 85	1 | 21	0 | 0	0 | 0	0 | 0	14 | 64	3842 | 6432	- | -	- | -
VIRTUAL BEST(ALL)	15 | 85	0 | 7	0 | 0	0 | 0	0 | 0	15 | 78	146 | 3514	- | -	- | -
MINIZINC Challenge 2019									
PICAT(CP) [2.7B12]	10 | 90	8 | 67	0 | 11	0 | 5	0 | 0	2 | 7	54 | 1440	1 | 0	1 | 0
IZPLUS()	10 | 90	5 | 71	0 | 4	0 | 0	0 | 0	5 | 15	14 | 3077	1 | 3	1 | 3
G12(FD)	10 | 90	5 | 64	0 | 10	0 | 0	0 | 0	5 | 16	323 | 4010	0 | 0	0 | 0
CHOCO(STD)	10 | 90	4 | 63	0 | 5	0 | 0	0 | 0	6 | 22	415 | 4312	0 | 0	0 | 0
GECODE() [6.2.0]	10 | 90	4 | 63	0 | 0	0 | 0	0 | 0	6 | 27	420 | 5094	0 | 6	0 | 6
JACOP() [4.8]	10 | 90	4 | 55	0 | 6	0 | 0	0 | 0	6 | 29	260 | 5467	0 | 1	0 | 1
HAIFACSP()	10 | 90	0 | 47	0 | 10	0 | 0	2 | 2	8 | 31	2 | 6408	4 | 7	4 | 4
CHUFFED() [NIGHTLY]	10 | 90	0 | 43	0 | 5	10 | 0	0 | 0	5 | 37	1 | 4886	3 | 19	3 | 19
GUROBI() [8.1.1]	10 | 80	0 | 48	0 | 0	0 | 0	0 | 0	10 | 32	705 | 2895	2 | 6	2 | 4
PICAT(SAT) [2.7B12]	10 | 90	0 | 45	0 | 5	0 | 0	0 | 1	10 | 39	275 | 9894	0 | 7	0 | 5
OR-TOOLS(SAT) [NIGHTLY]	10 | 90	5 | 42	0 | 3	0 | 0	0 | 0	5 | 45	8 | 7239	0 | 13	0 | 11
VIRTUAL BEST(MINIZINC)	10 | 90	0 | 29	0 | 0	0 | 0	0 | 0	10 | 61	9 | 5247	- | -	- | -
OPTIMATHSAT(INT) [1.6.4.1]	10 | 90	5 | 62	0 | 0	0 | 0	0 | 0	5 | 28	4 | 3650	2 | 10	0 | 0
OPTIMATHSAT(BV) [1.6.4.1]	10 | 90	4 | 59	0 | 5	0 | 0	0 | 0	6 | 26	484 | 7271	0 | 1	0 | 0
BCLT(INT)	10 | 90	5 | 60	0 | 0	0 | 0	0 | 0	5 | 30	6 | 3369	0 | 6	0 | 5
Z3(INT)	10 | 90	5 | 64	0 | 0	0 | 0	0 | 0	5 | 26	4 | 5358	3 | 6	0 | 1
Z3(BV)	10 | 90	0 | 55	0 | 2	0 | 0	0 | 0	10 | 33	1629 | 7550	5 | 17	0 | 3
VIRTUAL BEST(OMT)	10 | 90	0 | 48	0 | 2	0 | 0	0 | 0	10 | 40	1624 | 5179	- | -	- | -
VIRTUAL BEST(ALL)	10 | 90	0 | 29	0 | 0	0 | 0	0 | 0	10 | 61	9 | 4919	- | -	- | -

Table 1. MINIZINC Challenge formulas. The columns list the total number of instances (inst.), of timeouts (timeout), of run-time errors (error), of unsupported problems (unsup.), of incorrectly solved instances (incor.), of correctly solved instances (correct), the total solving time for all solved instances (time), the number of instances solved in the shortest time within the same category (BT1) and those solved in the shortest time considering all tools (BT2).

Experiment Results. The results of this experiment are shown in Table 1, with separate numbers for satisfiability (**s**) and optimization (**o**) instances in each benchmark-set. Using the experimental data, we separately computed the virtual best configuration among all MINIZINC solvers (i.e. VIRTUAL BEST(MINIZINC)), all OMT solvers (i.e. VIRTUAL BEST(OMT)), and also the virtual best among all tools considered in the experiment (i.e. VIRTUAL BEST(ALL)). The last two columns in the table list the number of prob-

lems solved by the given configuration in the same amount of time as the VIRTUAL BEST() of each group (col. *BT1*) and as the VIRTUAL BEST(ALL) (col. *BT2*).

We start by looking at the MINIZINC solvers in Table 1. The performance ladder is dominated by OR-TOOLS(SAT) and PICAT(SAT), closely followed by GUROBI, HAIFACSP and CHUFFED (in MC19). By looking at column *BT1*, we observe that the top-performing MINIZINC solvers tend to dominate over all the others. Looking at the results of the MC19 experiment, we notice a significant increase in the number of errors with respect to the benchmark-set of the MC16 edition, as well as a handful of problems solved incorrectly. In the case of GUROBI and PICAT(SAT), the MZN2FZN compiler encountered an error over a few instances. As a consequence, the total number of problems is smaller than 100 for both tools. After taking a closer look, we ascribe this phenomenon to the recent changes in the MINIZINC/FLATZINC format, that has created some minor issues with some tools that have not been adequately updated.

Looking at the OMT tools only, we observe that Z3 has leading performance over the other solvers. When compared to the MINIZINC solvers, the OMT solvers place themselves in the middle of the rank on both benchmark-sets. Given the fact that none of the OMT solvers has specialized procedures or encodings for dealing with global constraints, we consider this an interesting result.

5.2 Evaluation on OMT Benchmark Sets

In this experimental evaluation we use OMT formulas taken from well-known, publicly available, repositories. We characterize these benchmark-sets as follows:

– *SAL* [integers]: 66 SMT-based Bounded Model Checking and K-Induction parametric problems created with the SAL model checker [5];
– *SAL* [rationals]: as above, with problems on the rationals;
– *Symba* [rationals]: 2632 *bounded*[2] software verification instances derived from a set of C programs used in the Software Verification Competition of 2013 [34];
– *Jobshop and Strip Packing* [rationals]: 190 problems taken from [49,51];
– *Machine Learning* [rationals]: 510 OMT instances generated with the PYLMT tool based on Machine Learning Modulo Theories [57].

The first benchmark-set is on the integers, whereas the other four are on the rationals. We stress the fact that all formulas contained in all benchmark-sets are *satisfiable*.

Remark 2. Although there exists a repository of multi-objective OMT formulas (e.g. [34,54]), we have chosen to not include these in our experimental evaluation. The reason for this is twofold. First, such comparison would likely be unfair wrt. CP tools because that the workaround for dealing with multi-independent OMT formulas described in Section §4 is not competitive with the integrated optimization schema provided by OMT solvers [34,54]. In fact, the experimental evidence in [34,54] collected on a group of OMT solvers indicates that the latter approach can be an order of magnitude faster than the former one. Second, the workaround for dealing with lexicographic-optimization is limited by the fact that MINISEARCH is not fully compatible with recent versions of MINIZINC, and it only works with a restricted set of tools.

---

[2] We discarded any *unbounded* instance in the original benchmark-set in [34].

We have used the OMT2MZN tool described in Section §4 to translate each OMT formula to the MINIZINC format. OMT2MZN is written in Python and it is built on top of PYSMT [28], a general-purpose Python library for solving SMT problems, and it is available at [3]. During this step, it has been necessary to impose a finite domain to any unconstrained SMT-LIB rational variable, because otherwise none of the MINIZINC solvers would have been able to deal with them. We have experimented with two different domains: the largest feasible domain for floating-point variables of 32 bits (i.e. $\pm 3.402823e + 38$) for the first two benchmark-sets, and the largest feasible domain for integer variables (i.e. $\pm 2^{31}$) for the last two.

We consider two OPTIMATHSAT configurations: OPTIMATHSAT(SMT), solving the original OMT formulas, and OPTIMATHSAT(FZN), executed on the generated MINIZINC instances. The benefits of this choice is two-fold. First, we can double-check the correctness of such encoding, by comparing the optimum models generated in the two cases. Second, we can verify whether there is any performance loss caused by the encoding of the formula.

Only four of the MINIZINC solvers listed in Section §5 support floating-point reasoning. This limited the number of tools that could be used with some OMT benchmark-sets. The running-time of each MINIZINC solver reported in these experiments (including OPTIMATHSAT(FZN)) is comprehensive of the time taken by the MZN2FZN compiler, because the latter can sometime solve the input formulas on its own. The overall timeout is set to $600s$.

Notice that the optimal solutions found by OPTIMATHSAT(SMT) have been previously independently verified with a third-party SMT tool as reported in previous publications [53,54,55].[3] Therefore, we verify the correctness of the results found by any other configuration by comparing them with those found by OPTIMATHSAT(SMT), and otherwise mark the result as "unverified".

Experimental Results over the Integers. In this experiment, we evaluate the *SAL (over integers)* benchmark-set. The results are collected in Table 2.

We notice first that OPTIMATHSAT(FZN) always produces correct results and it shows comparable performances in terms on number of problems solved wrt. the baseline OPTIMATHSAT(SMT), solving even 4 problems more. (We conjecture that the latter fact should be attributed to the limited, but effective, deduction capabilities of the MZN2FZN compiler, that may have helped OPTIMATHSAT in solving the input formulas.) This suggests that, at least on problems on the integers, OMT2MZN is efficient and effective and does not affect correctness.

In general, MINIZINC solvers do not seem to deal efficiently with this benchmark-set. Some tools have experienced some internal error (e.g. dumped-core, segmentation fault), some others have been killed to to a high memory consumption (over 32GB), whereas the majority of the remaining tools had a timeout.

We explain this behavior with the fact that the given benchmark set is characterized by the presence of a heavy Boolean structure combined with arithmetical con-

_____

[3] For every OMT problem $\langle \varphi, \text{obj} \rangle$ s.t. OPTIMATHSAT(SMT) returns a minimum value min for obj on the formula φ, we say min is correct iff $\varphi \wedge (\text{obj} = min)$ is satisfiable and $\varphi \wedge (\text{obj} < min)$ is unsatisfiable. (Dual for maximization.)

Tools & Configuration	inst.	timeout	tool-er.	unsupp.	incor.	Terminated correct	time(s)
GUROBI()	66	0	0	66	0	0	0
G12(FD)	66	0	66	0	0	0	0
IZPLUS()	66	0	66	0	0	0	0
JACOP()	66	0	66	0	0	0	0
CHUFFED()	66	19	47	0	0	0	0
OR-TOOLS(SAT)	66	57	9	0	0	0	0
CHOCO()	66	66	0	0	0	0	0
HAIFACSP()	66	66	0	0	0	0	0
PICAT(CP)	66	66	0	0	0	0	0
GECODE()	66	66	0	0	0	0	0
GUROBI(L)	66	63	0	0	0	3	166
PICAT(SAT)	66	62	0	0	0	4	1667
VIRTUAL BEST(MINIZINC)	66	62	0	0	0	4	718
OPTIMATHSAT(FZN)	66	18	0	0	0	48	7113
VIRTUAL BEST(FZN)	66	18	0	0	0	48	7113
OPTIMATHSAT(SMT)	66	22	0	0	0	44	2657
VIRTUAL BEST(ALL)	66	16	0	0	0	50	5037

Table 2. SAL over integers. A SAT result is marked as *correct* when the objective value matches the reference solution provided by OPTIMATHSAT(SMT) (when run without a timeout), as *incorrect* otherwise.

Tools & Configuration	Instances	Timeout	Tool-errors	Terminated Incorrect	Verified	Unverified	Time(s)	Incorrect Results unsat	$\Delta \geq 10^{-6}$	$\Delta \geq 10^{-3}$	$\Delta \geq 10^{-1}$	$\Delta \geq 10^{0}$	$\Delta \geq 10^{1}$
SAL, Symba, Jobshop and Strippacking													
G12(MIP)	2888	10	0	2855	0	23	317	2765	90	90	86	39	0
GUROBI()	2888	48	0	2728	104	8	3961	2684	44	32	32	1	0
GECODE()	2888	2733	0	0	155	0	10800	0	0	0	0	0	0
VIRTUAL BEST(MINIZINC)	2888	0	0	2628	237	23	13800	-	-	-	-	-	-
OPTIMATHSAT(FZN)	2888	31	0	0	2854	3	22320	0	0	0	0	0	0
VIRTUAL BEST(FZN)	2888	0	0	11	2854	23	20673	-	-	-	-	-	-
OPTIMATHSAT(SMT)	2888	23	0	0	2865	0	15675	0	-	-	-	-	-
VIRTUAL BEST(ALL)	2888	0	0	0	2865	23	15182	-	-	-	-	-	-
Machine Learning													
G12(MIP)	510	108	0	400	2	0	224	400	0	0	0	0	0
GUROBI()	510	9	0	476	24	1	135	468	8	8	6	4	0
GECODE()	510	322	0	164	24	0	10	147	17	2	0	0	0
VIRTUAL BEST(MINIZINC)	510	9	0	468	32	1	317	-	-	-	-	-	-
OPTIMATHSAT(FZN)	510	298	0	177	35	0	32	177	0	0	0	0	0
OPTIMATHSAT(FZN+E)	510	442	0	0	68	0	80	0	0	0	0	0	0
VIRTUAL BEST(FZN)	510	9	0	428	72	1	359	-	-	-	-	-	-
OPTIMATHSAT(SMT)	510	10	0	0	500	0	7707	0	-	-	-	-	-
VIRTUAL BEST(ALL)	510	9	0	0	500	1	7737	-	-	-	-	-	-

Table 3. OMT Problems defined over the rationals. A SAT result is marked as *correct* when the objective value matches the reference solution provided by OPTIMATH-SAT(SMT) with an absolute error $\Delta < 10^{-6}$. A result is marked as *unverified* when we have no reference solution and *incorrect* if neither of the previous two conditions apply.

straints, which requires the efficient combination of strong Boolean-reasoning capabilities (e.g., efficiently handling chains of unit propagations) with strong arithmetical-solving&optimization capabilities, which is a typical feature of OMT solvers.

None of the input formulas was initially supported by GUROBI. After restricting the bound of every integer variable to $\pm 10^6$, GUROBI(L) was able to solve 3 instances within the timeout. Among the MINIZINC solvers, the best result is obtained by PICAT(SAT), that solved 4 problems out of 66.

Experimental Results Over the Rationals. We consider first the first three benchmark-sets over the rationals: SAL over rationals, Symba, JobShop&Strip-Packing. (Separate tables for the four benchmarks are reported in the extended version of this paper [23].) Of all MINIZINC solvers we have tried, only three are able to deal with floating-point constraints. The results are shown in Table 3. Since each of the input formulas is satisfiable, we consider a result incorrect either when it is equal to UNSAT, or when the relative error Δ exceeds 10^{-6}, s.t.: $\Delta \overset{\text{def}}{=} \frac{|o_{smt} - o_{fzn}|}{|o_{smt}|}$, o_{smt} and o_{fzn} being the optimal value found by OPTIMATHSAT(SMT) and the optimal value found by the MINIZINC solver under test respectively. (Recall that the former was previously checked to be correct.)

Similarly to the previous experiment on the integers, OPTIMATHSAT(FZN) always produces correct results, and display comparable performance wrt. OPTIMATHSAT(OMT) in terms of number of instances being solved, solving somewhat fewer problems. This is not the case of the other three MINIZINC solvers. Among these, GECODE experienced a timeout on the majority of the formulas being considered, G12(MIP) returned mostly incorrect answers, whereas GUROBI seems to have the best performance, in particular on the third benchmark-set.

We attribute the large number of incorrect results returned by all three MINIZINC solvers to the fact that these tools use finite-precision floating-point arithmetic internally. The incorrect behavior of some of these solvers (e.g. GUROBI) can also be partially explained with the large domain of floating-point variables in these problems. However, given the nature of these input instances, it was not possible for us to assign a smaller domain to each variable in the problem *a priori*.

We analyze separately the results for the last benchmark-set reported in Table 3. The peculiar aspect of the Machine Learning benchmark-set [57] is that it is characterized by Pseudo-Boolean sums over rational weights, and by very fine-grained rational values[4]. Unfortunately, these fine-grained rational values are rounded by the standard MZN2FZN compiler, which causes the incorrect results even of OPTIMATHSAT(FZN) in Table 3, despite the fact that OPTIMATHSAT uses infinite-precision arithmetic.

In order to overcome this issue, we leverage the EMZN2FZN compiler described in Section §3 to preserve the original fractional values, and show that with this approach OPTIMATHSAT does not produce incorrect results any longer (configuration OPTIMATHSAT(FZN+E) in Table 3), solving correctly 33 problems more than OPTIMATHSAT(FZN).

Despite this improvement, however, OPTIMATHSAT(FZN+E) still solves much less problems than OPTIMATHSAT(SMT). We have analyzed this fact, and we noticed that the performance gap is due to the structural changes introduced by the MZN2FZN com-

---

[4]For example, $\frac{1799972218749879}{2251799813685248}$ is a sample weight value from problems in [57].

piler, which prevents OPTIMATHSAT(FZN+E) to use some efficient solving techniques when dealing with the same problem given to OPTIMATHSAT(SMT). [5]

Overall, since there are at least 177 formulas affected by the above issue with the MZN2FZN compiler, we avoid an in-depth discussion of the results obtained by the other MINIZINC solvers. However, at a first glance the situation does not seem to differ from the other benchmark-sets over the rationals.

5.3 Discussion

On the whole, from our experiments, OMT tools appear to be still at some disadvantage when dealing with MINIZINC problems wrt. specific tools, and vice versa.

On the one hand, OMT solvers seem to be penalized by their lack of efficient ad hoc decision procedures for dealing with global constraints. Moreover, the approach taken by the MZN2FZN compiler, that creates lots of alias Boolean, integer and floating-point variables for dealing with Pseudo-Boolean constraints, is particularly challenging to deal with efficiently by an OMT solver.

On the other hand, MINIZINC solvers seem to suffer with problems needing an arithmetic-reasoning component combined with heavy Boolean-reasoning component. Even more importantly, the lack of infinite-precision linear arithmetic procedures causes a number of incorrect results when dealing with OMT problems over the rationals. Both of these points need to be addressed in order to deal with the vast number of Formal Verification and Model Checking applications in the SMT/OMT domain.

6 Conclusions & Future Work

In this paper we have taken a first step forward towards bridging the MINIZINC and the OMT communities. The ultimate goal is to obtain a correct, effective and efficient fully-automated system for translating problems from one community to the other, so as to extend the application domain of both communities. With our experimental evaluation, we have identified some criticalities that need to be addressed by each community in order to solidify this union.

We plan to push this investigation forward as follows. In the short term, we plan to address the inefficient handling of Pseudo-Boolean constraints over the rationals revealed by the experimental evaluation in Section §5.2. In order to deal with those FLATZINC constraints that require non-linear arithmetic, we envisage an opportunity to either extend OPTIMATHSAT with proper handling of the non-linear arithmetic theory [20] or to experiment with an encoding based on the floating-point theory [58]. This objective goes hand in hand with the extension of OMT2MZN to deal with other SMT theories. In the long term, OMT solving may also benefit from adopting efficient ad hoc decision procedures for frequently used global constraints. Finally, we plan to broaden the scope of our investigation and include other OMT solvers in our study.

---

[5]For example, an equality like $(t = \sum_i F_i \cdot w_i)$ where w_i are float values, t in an unbounded float variable and $F_i \in \{0.0, 1.0\}$ are float variables, is rewritten by MZN2FZN into the equality $(0.0 = \sum_i F_i \cdot w_i + t \cdot -1.0)$, where the Pseudo-Boolean sum is added to an arithmetic term, preventing OPTIMATHSAT(FZN+E) from recognizing it as the definition of a Pseudo-Boolean sum, and hence from activating the specialized solving technique for Pseudo-Boolean equalities.

References

1. FZN2OMT. https://github.com/PatrickTrentin88/fzn2omt.
2. Benchmarks, Tools and Data. http://disi.unitn.it/trentin/resources/cpaior2020.tar.xz.
3. omt2mzn. https://github.com/cespio/omt2mzn.
4. EMZN2FZN Repository. https://github.com/PatrickTrentin88/emzn2fzn.
5. Sal, symbolic analysis laboratory. http://sal.csl.sri.com.
6. SMT-LIB Format for Finite Lists, Sets and Maps. https://www.cprover.org/SMT-LIB-LSM/.
7. C. Ansótegui, M. Bofill, M. Palahí, J. Suy, and M. Villaret. Solving weighted CSPs with meta-constraints by reformulation into Satisfiability Modulo Theories. *Constraints*, 18(2), 2013.
8. R. Araújo, I. Bessa, L. C. Cordeiro, and J. E. C. Filho. SMT-based Verification Applied to Non-convex Optimization Problems. In *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*, Nov 2016.
9. R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality Networks: a theoretical and empirical study. *Constraints*, 16(2), 2011.
10. C. Barrett, S. Ranise, A. Stump, and C. Tinelli. The satisfiability modulo theories library (smt-lib). http://www.smtlib.org, 2010.
11. C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. *Satisfiability Modulo Theories*, chapter 26, pages 825–885. Volume 185 of Biere et al. [14], February 2009.
12. C. Bertolissi, D. R. dos Santos, and S. Ranise. Solving Multi-Objective Workflow Satisfiability Problems with Optimization Modulo Theories Techniques. In *SACMAT*. ACM, 2018.
13. Z. Bian, F. Chudak, W. Macready, A. Roy, R. Sebastiani, and S. Varotti. Solving SAT and MaxSAT with a Quantum Annealer: Foundations and a Preliminary Report. In *Frontiers of Combining Systems*, volume 10483 of *LNCS*. Springer, 2017.
14. A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*. IOS Press, February 2009.
15. N. Bjorner, A.-D. Phan, and L. Fleckenstein. νZ - An Optimizing SMT Solver. In *Proc. TACAS*, volume 9035 of *LNCS*. Springer, 2015.
16. M. Bofill, R. Nieuwenhuis, A. Oliveras, E. R. guez Carbonell, and A. Rubio. The Barcelogic SMT Solver. In *20th International Conference on Computer Aided Verification, CAV'08*, volume 5123 of *LNCS*. Springer, 2008.
17. M. Bofill, M. Palahí, J. Suy, and M. Villaret. Solving constraint satisfaction problems with SAT modulo theories. *Constraints*, 17(3), 2012.
18. M. Bofill, J. Suy, and M. Villaret. A System for solving constraint satisfaction problems with SMT. *Theory and Applications of Satisfiability Testing–SAT 2010*, 2010.
19. L. Candeago, D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. Speeding up the Constraint-Based Method in Difference Logic. In *SAT*, volume 9710 of *LNCS*. Springer, 2016.
20. A. Cimatti, A. Griggio, A. Irfan, M. Roveri, and R. Sebastiani. Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. *ACM Trans. Comput. Logic*, 19(3), Aug. 2018.
21. H. Collavizza and M. Rueher. Exploration of the capabilities of constraint programming for software verification. 03 2006.
22. H. Collavizza, M. Rueher, and P. Van Hentenryck. Cpbpv: a constraint-programming framework for bounded program verification. *Constraints*, 15(2), Apr 2010.
23. F. Contaldo, P. Trentin, and R. Sebastiani. From minizinc to optimization modulo theories, and back (extended version), 2019. https://arxiv.org/abs/1912.01476.

24. H. A. Elgabou and A. M. Frisch. Encoding The Lexicographic Ordering Constraint in SAT Modulo Theories. In *In Proc. of Thirteenth International Workshop on Constraint Modelling and Reformulation*, 09 2014.

25. T. Feydy and P. J. Stuckey. Lazy Clause Generation Reengineered. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, CP'09, Berlin, Heidelberg, 2009. Springer-Verlag.

26. A. M. Frisch and P. A. Giannaros. SAT Encodings of the At-Most-k Constraint Some Old , Some New , Some Fast , Some Slow. 2010.

27. A. M. Frisch and M. Palahí. Anomalies in SMT Solving: Difficulties in Modelling Combinatorial Problems. In *In Proc. of Thirteenth International Workshop on Constraint Modelling and Reformulation*, 09 2014.

28. M. Gario and A. Micheli. Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In *SMT workshop 2015, 13th International Workshop on Satisfiability Modulo Theories*, 2015.

29. P. J. Grinchtein O., Carlsson M. A constraint optimisation model for analysis of telecommunication protocol logs. In *Tests and Proofs. TAP 2015.*, volume 9154 of *LNCS*. Springer, 2015.

30. G. E. Karpenkov. *Finding inductive invariants using satisfiability modulo theories and convex optimization*. Theses, Université Grenoble Alpes, Mar. 2017.

31. G. Kovásznai, B. Erdélyi, and C. Biró. Investigations of graph properties in terms of wireless sensor network optimization. In *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*, Jan 2018.

32. D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. Minimal-Model-Guided Approaches to Solving Polynomial Constraints and Extensions. In *SAT*, 2014.

33. F. Leofante, E. Abraham, T. Niemueller, G. Lakemeyer, and A. Tacchella. Integrated Synthesis and Execution of Optimal Plans for Multi-Robot Systems in Logistics. *Information Systems Frontiers*, May 2018.

34. Y. Li, A. Albarghouthi, Z. Kincad, A. Gurfinkel, and M. Chechik. Symbolic Optimization with SMT Solvers. In *POPL*, 2014.

35. T. Liu, S. S. Tyszberowicz, B. Beckert, and M. Taghdiri. Computing Exact Loop Bounds for Bounded Program Verification. In *SETTA*, volume 10606 of *LNCS*. Springer, 2017.

36. MiniZinc. www.minizinc.org.

37. A. Nadel and V. Ryvchin. Bit-Vector Optimization. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2016*, volume 9636 of *LNCS*. Springer, 2016.

38. N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. MiniZinc: Towards a Standard CPModelling Language. In *Principles and Practice of Constraint Programming – CP 2007*, volume 4741 of *LNCS*. Springer Berlin Heidelberg, 2007.

39. C. M. Nguyen, R. Sebastiani, P. Giorgini, and J. Mylopoulos. Multi-objective reasoning with constrained goal models. *Requirements Engineering*, 2016. In print. Published online 24 December 2016. DOI: http://dx.doi.org/10.1007/s00766-016-0263-5.

40. R. Nieuwenhuis and A. Oliveras. On SAT Modulo Theories and Optimization Problems. In *Proc. Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *LNCS*. Springer, 2006.

41. R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. Challenges in Satisfiability Modulo Theories. In *Proc. RTA'07*, volume 4533 of *LNCS*. Springer, 2007.

42. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6), November 2006.

43. O. Ohrimenko, P. J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints*, 14(3), 2009.

44. R. S. Oliver, S. S. Craciunas, and W. Steiner. IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2018.

45. C. H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4), 1981.

46. S. Ratschan. Simulation Based Computation of Certificates for Safety of Dynamical Systems. In *Formal Modeling and Analysis of Timed Systems - 15th International Conference, FORMATS 2017, Berlin, Germany, September 5-7, 2017, Proceedings*, volume 10419 of *LNCS*. Springer, 2017.

47. A. Rendl, T. Guns, P. J. Stuckey, and G. Tack. Minisearch: A solver-independent meta-search language for minizinc. In *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, 2015.

48. S. F. Roselli, K. Bengtsson, and K. Åkesson. SMT Solvers for Job-Shop Scheduling Problems: Models Comparison and Performance Evaluation. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, Aug 2018.

49. N. W. Sawaya and I. E. Grossmann. A cutting plane method for solving linear generalized disjunctive programming problems. *Computers & Chemical Engineering*, 29(9), 2005.

50. R. Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation, JSAT*, 3(3-4), 2007.

51. R. Sebastiani and S. Tomasi. Optimization in SMT with LA(Q) Cost Functions. In *IJCAR*, volume 7364 of *LNAI*. Springer, July 2012.

52. R. Sebastiani and S. Tomasi. Optimization Modulo Theories with Linear Rational Costs. *ACM Transactions on Computational Logics*, 16(2), March 2015.

53. R. Sebastiani and P. Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories. In *Proc. International Conference on Computer-Aided Verification, CAV 2015*, volume 9206 of *LNCS*. Springer, 2015.

54. R. Sebastiani and P. Trentin. Pushing the Envelope of Optimization Modulo Theories with Linear-Arithmetic Cost Functions. In *Proc. Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'15*, volume 9035 of *LNCS*. Springer, 2015.

55. R. Sebastiani and P. Trentin. On Optimization Modulo Theories, MaxSMT and Sorting Networks. In *Proc. Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'17*, volume 10205 of *LNCS*. Springer, 2017.

56. R. Sebastiani and P. Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories. *Journal of Automated Reasoning*, Dec 2018.

57. S. Teso, R. Sebastiani, and A. Passerini. Structured learning modulo theories. *Artif. Intell.*, 244, 2017.

58. P. Trentin and R. Sebastiani. Optimization Modulo the Theory of Floating-Point Numbers. In *27th International Conference on Automated Deduction - CADE-27*, volume 11716 of *LNCS*. Springer, 2019.

59. M. Veksler and O. Strichman. Learning general constraints in CSP. *Artif. Intell.*, 238, 2016.

60. N.-F. Zhou and H. Kjellerstrand. Optimizing SAT Encodings for Arithmetic Constraints. 08 2017.