# Constrained Optimization Benchmark for Optimization Modulo Theories: a Cloud Resource Management Problem

Extended Abstract

Mădălina Eraşcu[1][2] Răzvan Meteş[1]

[1] Department of Computer Science, West University of Timişoara, Romania
[2] Institute e-Austria Timişoara, Romania
{madalina.erascu,razvan.metes}@e-uvt.ro

**Abstract**

The development, assessment, and comparison of Optimization Modulo Theories (OMT) algorithms and tools heavily depends on benchmarking. To the best of our knowledge, there is no benchmark environment for OMT, much less one which bears any relation to the number of distinct problem features. This paper proposes a scalable linear, respectively non-linear, constrained optimization problem that is suitable for benchmarking OMT solvers. By comparing two state-of-the-art OMT solvers, the benchmarking environment is demonstrated.

## 1 Introduction

Satisfiability Modulo Theories (SMT) became a popular research area since it has proven its practical application in various domains, for instance in program verification [5] and synthesis [12], security [20], neural networks verification [13]. SMT solvers implement the latest developments in the area (see SMT competition[1]) and provide mature algorithms for reasoning with boolean combinations of constraints over decidable theories in a push-button manner.

Standard decision procedures for SMT have been extended with optimization features, leading to Optimization Modulo Theories (OMT). OMT extends SMT solving with optimization procedures in order to find variable assignments that define an optimal value for objective function(s) under all models of a formula. OMT solvers [16, 6, 19] showed their applicability in task planning [15], Wireless Sensor Networks [14], worst-case execution time computation [11].

In our recent work [17, 10], motivated by the trend that organizations are likely to move their business in the Cloud[2], we studied the following problem. Given the diversity of Cloud Providers (CPs), e.g. Amazon Web Services[3], Google Cloud[4], Azure[5], which CPs can accomodate my component-based application at a fair budget such that my application performance requirements are fulfilled? To answer the question, we solved a *resource management problem*, that is: *(1) mapping* the components to VMs such that the computing, storage, networking requirements of the application are fulfilled, and *(2) selection* of VMs offers which minimize the cost. We formulated it as a linear constrained optimization problem with variables of boolean and integer/real type. The problem is related to the bin-packing problem, however *(1)* the placement of items in bins is limited by constraints, *(2)* the capacity of bins is not fixed (it depends on the VMs offers), *(3)* the number of items is not known (it depends on the constraints

[1] https://smt-comp.github.io
[2] https://ec.europa.eu/digital-single-market/en/cloud
[3] https://aws.amazon.com/
[4] https://cloud.google.com/
[5] https://azure.microsoft.com/en-us/

on the number of instances), *(4)* the smallest price is not necessarily obtained by using the smallest number of bins. We tackled it with approximate methods (evolutionary algorithms) since the exact methods (constrained programming and SMT solving) did not scale [17]. Note that the approximate methods, preferred by the Cloud Computing community when solving optimization problems, do not offer a guarantee that the exact optimum was reached, nor how far from the optimum the provided solution is. In [10], we overcame the scalability issue by proposing a heuristic combining clique detection and symmetries breaking tailored for our problem. As a result the timings improved by at least 2 orders of magnitude.

Motivated by our previous work, in this paper we present our work in progress on an OMT benchmark proposal. We provide two hard constrained optimization problems, linear and non-linear[6], respectively, that are scalable with respect to the problem dimension. The benchmark is constructed on the basis of the problem of component-based applications deployment in the Cloud. The problem dimension is determined by the number of VMs offers, the number of application components instances, and the number of hardware/software constraints. We consider several encodings of the variables involved in the constrained optimization problem which are suitable in the respective context.

The contributions of the papers are as follows: *(1)* additionally to [17, 10], we introduce a non-linear formulation to the constrained optimization problem (Section 2); *(2)* we propose basic benchmarking conventions in order to provide a thorough basis for reproducible and comparable benchmarking tests (Section 4); *(3)* for demonstration purposes, two state-of-the-art OMT solvers are tested on different variable encodings and increasing problem dimension (Section 5).

**Related Work.** The SMT community established and made available to the research community a large library of benchmarks for SMT solvers[7]. Benchmarks for OMT solvers exist; to name a few: *(1)* synthetic benchmarks from [2]: these benchmarks consider the parameterized version of Wordpress application (number of component instances deployed and number of VM offers) in order to address scalability issues. Additionally to these benchmarks, we consider also other component-based applications which imply a wider class of constraints. Moreover, we experiment with different variable encodings for these constraints. *(2)* industrial placement fixer benchmarks coming from a sub-stage of the physical design stage of the Computer-Aided Design as well as the crafted placement fixer benchmarks diversifying the former mentioned instances of the generic problem of [18]. *(3)* CELAR radio frequency assignment problems [7]. However, there is not exist a public benchmark library for OMT, nor specific OMT constructs in the SMT-LIB standard which would allow an initiative similar to SMT competition. Our work, through the benchmark proposed, motivates and proposes the extension of the SMT-LIB standard and the necessity of a benchmark library or sublibrary for OMT.

There also exists related research applying SMT solving for constrained optimization problems related to ours, see e.g. [8, 4, 2], however, for the lack of space we do not detail it here.

## 2 Problem Description

In [17, 10], we proposed the formalization of constrained optimization problems coming from the optimal deployment of component-based applications in the Cloud. We considered component-based applications composed of $N$ components $C_i$ ($i = \overline{1, N}$) imposing various *structural* constraints and having certain *hardware* characteristics, and a set of $M$ VMs, $V_j$ ($j = \overline{1, M}$).

[6]The number of non-linear constraints is, however, reduced compared to the linear ones (see Section 2).

[7]`http://smtlib.cs.uiowa.edu`

The problem we want to solve is to find a mapping (of components to VMs) which: *(1)* satisfies the constraints induced by the interactions between components; *(2)* satisfies the hardware requirements of all components, and *(3)* minimizes the purchasing price. The mapping is encoded like a set of binary variables, $a_{ik} \in \{0, 1\}$ for $i = \overline{1, N}$, $k = \overline{1, M}$, interpreted as follows: $a_{ik}$ is 1 if $C_i$ is assigned to $V_k$, and it is 0 otherwise. The minimization of the leasing price is expressed as the linear constraint $\sum_{k=1}^{M} VP_k$. Note that some $VP_k$ could be 0 if no component is assigned to $V_k$.

For a self-contained presentation, we recall the constraints which might appear in the formulation of the optimization problem. Exemplification of them is in [10].

*Structural constraints* are of two types: *general* and *application-specific*. The *general* constraints are always considered in the deployment model and are related to basic allocation rules. For example, each component $C_i$ must be allocated to at least one VM:

$$\sum_{i=1}^{N} a_{ik} \geq 1 \quad k = \overline{1, M} \tag{1}$$

except those being in *Exclusive Deployment* relation (see below).

We identified two main types of *application-specific constraints* regarding the components: *(1) interactions* (conflict, co-location, exclusive deployment) and *(2) number of instances* (require-provide, full deployment, deployment with a bounded number of instances). The *application-specific* constraints that we consider in the following are inspired by the case studies we consider (*Secure Web Container*, *Secure-Billing Email*, *Oryx2* and *Wordpress*), however, they cover a large set of interactions which can be encountered in most component-based applications.

*Conflict*: two or more components cannot be deployed on the same VM. The information concerning the conflicts between components is stored in the conflict matrix $R_{ij} \in \{0,1\}, i, j = \overline{1, N}$ ($R_{ij} = 1$ if $C_i$ and $C_j$ are conflictual). The constraint written as set of quadratic expressions (with respect to the elements of the assignment matrix) is:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a_{ik} a_{jk} R_{ij} = 0, \quad k = \overline{1, M} \tag{2}$$

This can be rewritten also as a set of linear expressions (w.r.t. the elements of the assignment matrix):

$$a_{ik} + a_{jk} \leq 1, \qquad k = \overline{1, M}, \quad \forall i, j \text{ s.t. } R_{ij} = 1 \tag{3}$$

This second variant leads to a larger number of constraints ($MN^2$ instead of $M$) but it allows the use of linear programming methods, unlike the previous one which requires quadratic programming.

*Co-location*: two or more components should be deployed on the same VM. The information concerning this type of dependence can be stored in a dependency matrix $D_{ij} \in \{0, 1\}, \forall i, j = \overline{1,N}$ ($D_{ij} = 1$ if $C_i$ and $C_j$ should be deployed on the same virtual machine).The constraint can be formulated as:

$$a_{ik} = a_{jk}, \quad k = \overline{1, M}, \quad \forall i, j \text{ s.t. } D_{ij} = 1 \tag{4}$$

*Exclusive deployment*: two or more components cannot be deployed in the same deployment plan:

$$H(\sum_{k=1}^{M} a_{i_1 k}) + H(\sum_{k=1}^{M} a_{i_2 k}) + ... + H(\sum_{k=1}^{M} a_{i_q k}) = 1, \tag{5}$$

where $H$ is the Heaviside-like function defined as: $H(u) = 1$ if $u > 0$ and $H(u) = 0$ if $u = 0$.

*Require-provides*: this is a special case of interaction between components: when one component requires some functionality offered by other components. Such an interaction induces constraints on the number of instances corresponding to the interacting components as follows: *(1)* $C_i$ requires (consumes) at least $n_{ij}$ instances of $C_j$ and *(2)* $C_j$ can serve (provides) at most $m_{ij}$ instances of $C_i$. This can be written as:

$$n_{ij} \sum_{k=1}^{M} a_{ik} \leq m_{ij} \sum_{k=1}^{M} a_{jk}, \quad n_{ij}, m_{ij} \in \mathbb{N}. \tag{6}$$

A related case is when for each set of $n$ instances of component $C_i$ a new instance of $C_j$ should be deployed. This can be described as:

$$0 \leq n \sum_{k=1}^{M} a_{jk} - \sum_{k=1}^{M} a_{ik} < n, \quad n \in \mathbb{N} \tag{7}$$

This constraint cannot be deduced from (6) because of the following. Taking in (6) $n_{ij} = 1$, we obtain an expression meaning that for $m_{ij}$ instances of $C_j$ one should have at least one instance of $C_i$ (but there can be more). (7) is more specific requiring exactly one instance of $C_j$.
*Full deployment*: a component $C_i$ must be deployed on all VMs (except on those which would induce conflicts:

$$\sum_{k=1}^{M} (a_{ik} + H( \sum_{j, R_{ij}=1} a_{jk})) = M \tag{8}$$

*Deployment with bounded number of instances*: the number of instances corresponding to a set of deployed components, $\overline{C}$, should be equal, greater or less than some values:

$$\sum_{i \in \overline{C}} \sum_{k=1}^{M} a_{ik} \ \langle \text{op} \rangle \ n, \qquad \langle \text{op} \rangle \in \{=, \leq, \geq\}, \ n \in \mathbb{N} \tag{9}$$

The *hardware constraints* specify the amount of resources required by the components assigned to a VM and assure that they do not overpass the VM characteristics as offered by the CPs.

$$\sum_{i=1}^{N} a_{ik} \cdot HR_t^i \leq vm_k^{HR_t}, \quad vm_k^{HR_t}, HR_t^i \in \mathbb{R}_+, \quad k = \overline{1, M}, \quad t = \overline{1, L} \tag{10}$$

where $vm_k^{HR_t}$ is used to store a CPs offer in terms of number of CPUs, memory, storage, while $HR_t^i$ stores the component $i$ requirements in terms of of number of CPUs, memory, storage.

Besides the above constraints, for the correctness of the formalization, we also had to encode the CPs offers and to link them with the components hardware constraints. At this aim, we introduced the variable *vmType* which identifies the CPs offers ($vmType_k \in \{1, \ldots, ON\}$, where $ON$ is the number of CP offers). The link between the *vmType* and the allocation matrix $a$ is done by the following constraints:

$$\bigvee_{h=1}^{ON} vmType_k = h \tag{11}$$

$$\sum_{i=1}^{N} a_{ik} \geq 1 \wedge vmType_k = h \implies VP_k = Price^{Offer_h} \wedge vm_k^{HR_1} = HR_1^{Offer_h} \wedge .. \wedge vm_k^{HR_L} = HR_L^{Offer_h} \tag{12}$$

where $h = \overline{1, ON}$; $HR_t^{Offer_h}$ represents the CPUs number, memory size and storage size for offer $h$ ($t = \overline{1, L}$). For example, for the first offer ($h = 1$) corresponding values for ($Price^{Offer_1}$, $HR_1^{Offer_1}$, $HR_2^{Offer_1}$, $HR_3^{Offer_1}$) are (9.152\$, 64, 488MB, 8GB). The formula above also specifies that the price of a VM contributes to the final price only if the machine is occupied. If the machine is not occupied, then it does not contribute to the final total leasing price. This is expressed by the following constraint:

$$\sum_{i=1}^{N} a_{ik} = 0 \implies VP_k = 0, \quad k = \overline{1, M} \tag{13}$$

Note that the above constraints determine two types of constrained optimization problems: *(1)* linear *(2)* non-linear (when (2) is used instead of (3)).

## 3 Satisfiability and Optimization Modulo Theories

Satisfiability Modulo Theories (SMT) is about the satisfiability of first-order formulas with respect to some background theory. It enhances the boolean satisfiability problem (SAT) with theories like real and integer numbers, bitvectors, arrays and so on. The logics that one could use are characterized, for instance, by the linearity/non-linearity of the arithmetic and the presence/absence of quantifiers. In this paper, given the nature of our constraints, we used the following logics: *(1)* `QF_LIA` / `QF_NIA`; *(2)* `QF_LRA` / `QF_NRA`; *(3)* `QF_BV`. The SMT-LIB format [3], the common input format for SMT solvers, defines the syntax of the above mentioned logics.

Optimization Modulo Theories (OMT) seek to find a variable assignment which optimizes a certain objective function (single-criteria optimization) or a combination of multiple objective functions (multi-criteria optimization) under all models of a formula. The existing OMT solvers are mainly built on top of the existing SMT solvers. For example, OptiMathSAT [19] uses an inline architecture in which the SMT solver MathSAT5[8] is run only once and its internal SAT solver is modified to handle the search for the optima. Symba [16] and $\nu Z$ [6][9] both are based on an offline architecture in which the SMT solver Z3 [9] is incrementally called multiple times as a black-box. In our experiments, we did not used Symba since it is not maintained anymore.

Subcases of OMT arise when the constraints or the optimization function are pseudo-boolean or cardinality constraints. Therefore, we have MaxSMT (pseudo-Boolean optimization) where the optimization problem has the form:

$$\textbf{minimize } Cost(x_1, x_2, ..., x_n) \textbf{ subject to } \bigwedge_i \sum_j a_{ij} x_j \geq b_i$$

The cost function is typically defined as a linear function of the pseudo-Boolean variables, that is $Cost(x_1, x_2, ..., x_n) = \sum_j c_j x_j$, $\forall j, c_j \in \mathbb{Z}$.

**Definition 1.** A *pseudo-Boolean constraint* has the following form:

$$\sum_j a_i l_j \rhd b \quad (linear) \qquad\qquad \sum_j a_j \left( \prod_k l_{jk} \right) \rhd b \quad (nonlinear)$$

where $a_i$, $a_j$ and $b$ are integer constants, $l_j$, $l_{jk}$ are literals and $\rhd$ is a relational operator.

**Definition 2.** A cardinality constraint is a constraint on the number of literals which are true among a given set of literals. The following cardinality constraints can be defined: *(1)* `atleast(`$k$`, `$\{x_1, x_2, ..., x_n\}$`)` is true if and only if at least $k$ literals among $x_1, x_2, ..., x_n$ are true. *(2)* `atmost(`$k$`, `$\{x_1, x_2, ..., x_n\}$`)` is true if and only if at most $k$ literals among $x_1, x_2, ..., x_n$ are true. *(3)* `exactly(`$k$`, `$\{x_1, x_2, ..., x_n\}$`)` is true if and only if exactly $k$ literals among $x_1, x_2, ..., x_n$ are true.

Cardinality constraints can be translated to pseudo-boolean constraints and viceversa. Clearly, a cardinality constraint is a special kind of pseudo-Boolean constraint.

## 4 Benchmarking Conventions

In order to use the problem of optimal deployment of component-based applications (Section 2) as a benchmark for OMT comparison, benchmarking principles need to be drawn. Their purpose is to provide a comprehensive benchmarking environment that allows to generate reproducible and comparable test results. However, developers of OMT intending to use the specified benchmark environment are advised to also give algorithmic details of their implementation.

[8] http://mathsat.fbk.eu

[9] $\nu Z$ is the name of the optimization feature of the SMT solver Z3.

The problem dimension can be scaled as follows: *(1)* The number $ON$ of VM offers as available on the Cloud Providers site. In our experiments we used $ON \in \{4, 10, 20, 40, 60, 80, 100\}$. *(2)* The number of components instances in constraints like *deployment with bounded number of instances* and *require-provides* to be deployed. This number depends on the application architect and the imposed application components requirements. *(3)* The number of hardware/software constraints. Currently, we have used CPU, memory and storage, but depending on the application architect and user requirements this list can be extended or restrained. Moreover, the usage of different variable encodings should be taken into consideration.

## 4.1 Variable Encodings

The problem formulation (see Section 2) leads to the usage of either linear arithmetic, or non-linear arithmetic (due to the constraints of type (2)). Based on the type of the variables from Section 2, we used the encodings from Table 1:

| **Variable name** | **Type** | | | | |
|---|---|---|---|---|---|
| $VP$, $HR_t$, $vm^{HR_t}$, $vmType$ | `Real` | `Int` | `BV` | `Real` | `Int` |
| $a$ | `Real` | `Int` | `BV` | `Bool` | `Bool` |

Table 1: Variables Encodings

**Case** 1**: All variables have type `Real` (`RealReal`).** This requires to add explicitly the constraint that $a$ can only take value 0/1, that is $a_{ik} = 0 \vee a_{ik} = 1$, for all $i = \overline{1, N},\ k = \overline{1, M}$.
**Case** 2**: All variables have type `Int`.** We further considered two subcases ($i$=$\overline{1, N}$, $k$=$\overline{1, M}$): *(1)* $a_{ik} = 0 \vee a_{ik} = 1$ (`IntIntOr`) *(2)* $0 \leq a_{ik} \leq 1$ (`IntIntLessThan`).
**Case** 3**: All variables are bitvectors (`BV`).** The rationale behind using this encoding is that integers can be expressed also as bitvectors. We used bitvectors of size 32. This particular size was required to ensure that big integer numbers and arithmetic operations with them are correctly represented, respectively, computed. In order to work with positive integers, we used the unsigned versions of the relational operators.
**Case** 4**: All variables are `Real` except $a$ which is `Bool`.** There exists many approaches for variables encoding. For example, in order to encode the constraints of the type (10) we had to bring the variables involved to compatible types. This is achieved by transforming $a$ into type `Real` using the `if-the-else` operator `ite` (`RealBool`).

Another approach is determined by the observation that in our formalization the constraints (1)–(5), (8)–(9) are cardinality constraints, while (6)–(7), (10) are pseudo-boolean constraints. However, due to (11)–(13) the problem can not be handled via MaxSMT solely; it containts $ON + (ON \times M) + M$ constraints in which most of the variables must be of type `Real`. Hence, the problem must be handled by a solver over `QF_LRA`/`QF_NRA`.

From personal communication with the OptiMathSAT team, we found out that OptiMathSAT does not support direct encoding of cardinality/pseudo-boolean constraints as introduced in Definition 2. In $\nu Z$, the cardinality constraints can be directly encoded, however the pseudo-boolean constraints (10) can not, since $a_{ik}$ is `Bool` and $HR_t^i$ is `Real`. We came up with two different approaches: *(1)* we used the ternary operator `ite`, transforming the type of $a$ from `Bool` to `Real` in order to have compatible types for the variables involved in the constraints (`RealPBC`); *(2)* we used the equivalent transformation of (10) (`RealPBCMultiObjectives`):

$$\neg a_{ik} \Rightarrow (a_{ik} H_i^{res} = 0) \quad \wedge \quad a_{ik} \Rightarrow (a_{ik} H_i^{res} = H_i^{res}) \quad \wedge \texttt{minimize} \sum_{i=1}^{N} a_{ik} H_i^{res}, \qquad (14)$$

where $i = \overline{1, N}, k = \overline{1, M}$.

The new multi-objective optimization problem is handled by $\nu Z$, by default, using lexicographic combinations, so it is important that the objective $\sum_{k=1}^{M} VP_k$ precedes (14).
**Case 5: All variables are `Int` except $a$ which is Bool.** This case is similar to Case 4 and was not considered in this paper because of lack of space.

## 4.2 Extending the Language of the SMT-LIB standard

The set of benchmarks proposed are aligned with the SMT-LIB standard [3], namely every benchmark uses the command `set-info` to set the following attributes *(1)* `:smt-lib-version` must be the first command in the benchmark), *(2)* `:source`, *(3)* `:license`, and *(4)* `:category`. However, a slight modification was necessary, namely, instead of the attribute `:status`, we used a new one, `:minimum`, which can be either a $\langle numeral \rangle$ or `unknown`, if the minimum could not be found in a time frame of 40 minutes. This timeframe was chosen in order to be compliant with the document SMT-COMP 2019 Rules draft. We do not consider, for now, benchmarks when the optimization problem returns UNSAT.

The benchmarks are publicly available at [1]. We have split the benchmarks into linear and non-linear, and in each case, we grouped them with respect to the background theory of the constraints: `QF_LIA`, `QF_LRA`, `QF_BV`, respectively `QF_NIA`, `QF_NRA`, `QF_BV`. Each file in the benchmark is named by concatenating: the application name, the number of VM offers, the encoding name.

## 5 Algorithm assessment and comparison

This section is concerned with the evaluation of OMT solvers $\nu Z$ and OptiMathSAT. The timing results were obtained by taking the mean value time of 5 independent runs on use cases deriving the general problem formulation from Section 2. The use-cases were described in detail in [17, 10], here we mention, for each of them, their specific constraints: *(1) Secure-Billing E-mail Service*: conflicts (2)/(3), equal bound (9); *(2) Secure Web Container*: conflicts (2)/(3), equal and lower bound (9), full deployment (8), require provides (7); *(3) Oryx2*: conflicts (2)/(3), equal and lower bound (9), full deployment (8), require provides (6); *(4) Wordpress*: conflicts (2)/(3), upper and lower bound (9), alternative components (5), require provides (6). As an observation, *Secure-Billing E-mail Service* application involves application-specific constraints with not massive arithmetic as those involving, for example, require-provides constraints.

The scalability of the OMT tools for the case studies above was studied from two perspectives: number of VMs offers, respectively number of deployed components. For *Secure Web Container*, *Secure-Billing Email Service* and *Oryx2* applications, we considered up to 100 CPs offers. Additionally, for the *Wordpress* application, we considered the number of instances of Wordpress component to be deployed. For each application, we considered both formulations (linear and non-linear) and for each formulation the encodings from Table 1, counting 7.

All tests in this paper were done on an MacBook Pro with the following configuration: 3.1 GHz dual-core Intel Core i5, Turbo Boost up to 3.5GHz. All timings are expressed in seconds.

**Remark 1.** OptiMathSAT does not have support for nonlinear constraints. It is not clear if $\nu Z$ does, but the results for the minimum obtained in nonlinear case is the same as for the linear one and we obtained no warning/error during running that nonlinearity is not supported.

**Remark 2.** OptiMathSAT does not have support for the `RealPBC` encoding using the non-standard constructs like `atmost`, `atleast`, `exactly`, however they can be translated into `assert-soft` constraints. The automatic translation is work in progress.

For the lack of space, we do not list here all the timings, but we present an overview of the results. The timings can be consulted online at `https://github.com/merascu/Dissemination/tree/master/SMT2019`.

OptiMathSAT runs faster, but not significantly, than $\nu Z$ only in three cases: for the *Secure-Billing Email* application, `BV` encoding, when the number of offers is 10 and 60 (see Table 2 and Figure 1).

| #offers=4 | | #offers =10 | | #offers =20 | | #offers =40 | | #offers =60 | | #offers =80 | | #offers =100 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\nu Z$ | Opti MathSAT | $\nu Z$ | Opti MathSAT | $\nu Z$ | Opti MathSAT | $\nu Z$ | Opti MathSAT | $\nu Z$ | Opti MathSAT | $\nu Z$ | Opti MathSAT | $\nu Z$ | Opti MathSAT |
| 0.86 | 0.98 | 1.7 | 1.56 | 1.34 | 2.23 | 1.76 | 3.27 | 4.99 | 3.85 | 2.34 | 3.67 | 3.06 | 3.3 |

Table 2: Secure-Billing Email (`BV` encoding)

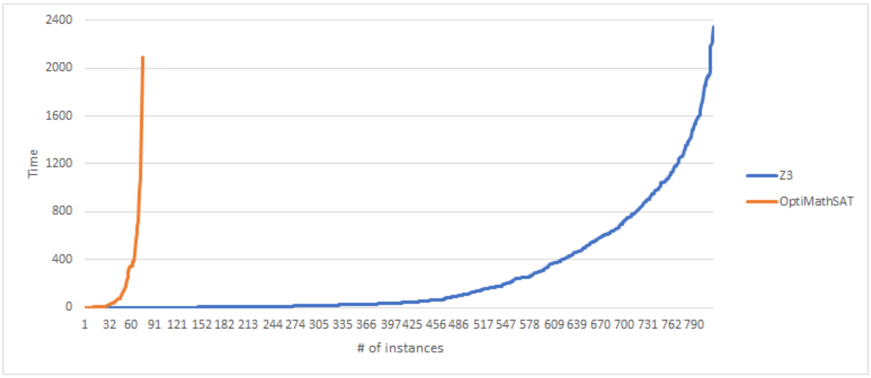

Figure 1: Comparison between $\nu Z$ and OptiMathSAT (linear formulation)

It is not the number of VMs offers, that is number of variables and constraints, which decides the complexity of the problem, but rather the arithmetic of the constraints. For example, both for $\nu Z$ and OptiMathSAT the time does not increase with the number of offers (see e.g. Table 2). We also observed, in case of $\nu Z$, that the timings at different runs can differ very much especially for constraints involving massive arithmetic (see Table 3). We can not comment if OptiMathSAT has simmilar behavior since it does not scale for such problems.

| | #offers=4 | #offers=10 | #offers=20 | #offers=40 |
|---|---|---|---|---|
| Run 1 | 49.55 | 128.54 | 45.23 | 983.39 |
| Run 2 | 59.85 | 120.37 | 133.11 | 2368.54 |
| Run 3 | 62.33 | 45.38 | 280.34 | 2484.58 |
| Run 4 | 58.71 | 81.65 | 125.51 | 1790.21 |
| Run 5 | 44.31 | 131.43 | 105.61 | 1955.71 |
| Average | 54.95 | 101.47 | 137.96 | 1916.48 |

Table 3: Wordpress application: 4 instances of Wordpress to be deployed. The timings are for the `RealPBC` encoding. For more offers, the average timings is more than 40 minutes and are not listed.

For the linear formulation, the `RealPBC` encoding gives always the best timings for the applications *Secure Web Container* and *Secure-Billing Email*; these applications do have complicated arithmetic. Overall, this encoding scales the best (see Figure 2): it solves a problem involving spprox. 500 variables of type `Real` and `Bool`, and around 3000 linear constraints of all

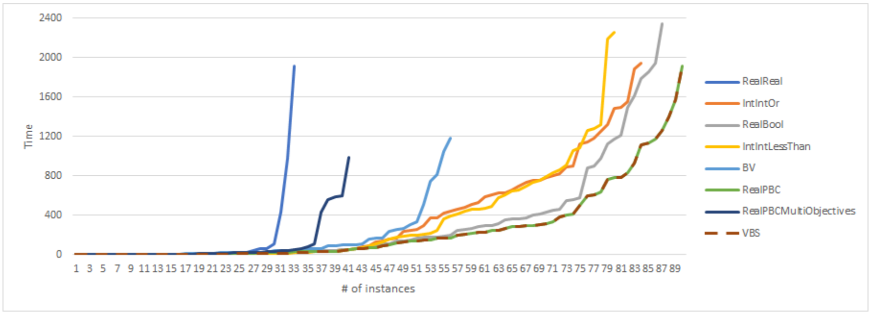

Figure 2: Scalability of $\nu Z$ for different linear encodings

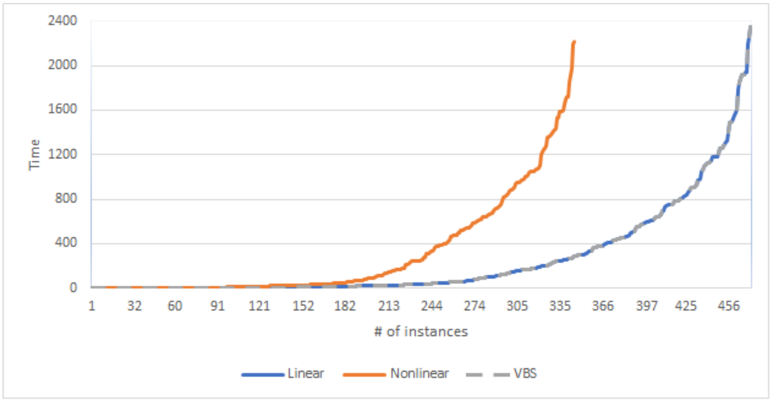

Figure 3: Comparison between linear/non-linear formalizations for $\nu Z$

types enumerated in Section 2. Regarding the timings comparison between linear and nonlinear formulations, the outcome is that the linear case runs best (see Figure 3).

The developmental stage of a Python implementation of the introduced benchmarking environment is made publicly available in a Github repository [1].

## References

[1] OMT Benchmarks derived from Cloud Deployment of Component-based Applications. `https://github.com/merascu/Optimization-Modulo-Theory/`. Accessed: 2019-06-02.

[2] E. Ábrahám, F. Corzilius, E. B. Johnsen, G. Kremer, and J. Mauro. Zephyrus2: On the Fly Deployment Optimization Using SMT and CP Technologies. In *Dependable Software Engineering: Theories, Tools, and Applications - Second International Symposium, SETTA 2016, Beijing, China, November 9-11, 2016, Proceedings*, pages 229–245, 2016.

[3] C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at `www.SMT-LIB.org`.

[4] S. Bayless, N. Kodirov, I. Beschastnikh, H. H. Hoos, and A. J. Hu. Scalable Constraint-based Virtual Data Center Allocation. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 546–554, 2017.

[5] N. Bjørner. SMT in Verification, Modeling, and Testing at Microsoft. In *Proceedings of the 8th International Conference on Hardware and Software: Verification and Testing*, HVC'12, pages 3–3, Berlin, Heidelberg, 2013. Springer-Verlag.

[6] N. Bjørner, A. Phan, and L. Fleckenstein. *$\nu Z$ - An Optimizing SMT Solver*. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 194–199, 2015.

[7] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. Warners. Radio Link Frequency Assignment. *Constraints*, 4(1):79–89, Feb 1999.

[8] C. Chen, S. Yan, G. Zhao, B. Lee, and S. Singhal. A Systematic Framework Enabling Automatic Conflict Detection and Explanation in Cloud Service Selection for Enterprises. In *2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, June 24-29, 2012*, pages 883–890, 2012.

[9] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[10] M. Erascu, F. Micota, and D. Zaharie. Influence of Variables Encoding and Symmetry Breaking on the Performance of Optimization Modulo Theories Tools Applied to Cloud Resource Selection. In G. Barthe, K. Korovin, S. Schulz, M. Suda, G. Sutcliffe, and M. Veanes, editors, *LPAR-22 Workshop and Short Paper Proceedings*, volume 9 of *Kalpa Publications in Computing*, pages 1–14. EasyChair, 2018.

[11] J. Henry, M. Asavoae, D. Monniaux, and C. Maïza. How to Compute Worst-case Execution Time by Optimization Modulo Theory and a Clever Encoding of Program Semantics. In *Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*, LCTES '14, pages 43–52, New York, NY, USA, 2014. ACM.

[12] S. Jha, S. Gulwani, S. A. Seshia, and A. Tiwari. Oracle-guided Component-based Program Synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, pages 215–224, 2010.

[13] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings*, pages 97–117, 2017.

[14] G. Kovásznai, C. Biró, and B. Erdélyi. Puli – A Problem-Specific OMT Solver. In *Proceedings of the 16th International Workshop on Satisfiability Modulo Theories, affiliated with IJCAR 2018, part of FLoC 2018*, 2018.

[15] F. Leofante, E. Ábrahám, and A. Tacchella. Task Planning with OMT: An Application to Production Logistics. In *Integrated Formal Methods - 14th International Conference, IFM 2018, Maynooth, Ireland, September 5-7, 2018, Proceedings*, pages 316–325, 2018.

[16] Y. Li, A. Albarghouthi, Z. Kincaid, A. Gurfinkel, and M. Chechik. Symbolic Optimization with SMT Solvers. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'14, San Diego, CA, USA, January 20-21, 2014*, pages 607–618, 2014.

[17] F. Micota, M. Erascu, and D. Zaharie. Constraint Satisfaction Approaches in Cloud Resource Selection for Component Based Applications. In *14th IEEE International Conference on Intelligent Computer Communication and Processing, ICCP 2018, Cluj-Napoca, Romania, September 6-8, 2018*, pages 443–450, 2018.

[18] A. Nadel and V. Ryvchin. Bit-Vector Optimization. In M. Chechik and J.-F. Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 851–867, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[19] R. Sebastiani and P. Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories. In D. Kroening and C. S. Păsăreanu, editors, *Computer Aided Verification*, pages 447–454, Cham, 2015. Springer International Publishing.

[20] J. Vanegue and S. Heelan. SMT Solvers in Software Security. In *6th USENIX Workshop on Offensive Technologies, WOOT'12, August 6-7, 2012, Bellevue, WA, USA, Proceedings*, pages 85–96, 2012.