EasyChair Preprint

№ 371

Puli - A Problem-Specific OMT Solver

Gergely Kovásznai, Csaba Biró and Balázs Erdélyi

July 23, 2018

Puli – A Problem-Specific OMT solver*

Gergely Kovásznai[1], Csaba Biró[2], and Balázs Erdélyi[1]

[1] IoT Research Institute
[2] Institute of Mathematics and Informatics
Eszterházy Károly University, Eger, Hungary

Abstract

In our previous papers, we investigated several aspects of applying Optimization Modulo Theories (OMT) solvers to Wireless Sensor Networks (WSNs). None of the solvers we used in our experiments scaled enough for WSNs of common size in practice. This is particularly true when investigating additional dependability and security constraints on WSNs of high density.

In this paper, we propose an idea of speeding up the OMT solving process by taking into consideration some resources in the systems and by applying regression analysis on those resource values. For instance, in WSNs, the electrical charge in the batteries of sensor nodes can be considered to be a resource that is being consumed as approaching the maximal lifetime of the network. Another example is the knapsack problem where the remaining capacity of the knapsack can be used as such a resource. We show how to integrate this idea in search algorithms in the OMT framework and introduce a new OMT solver called Puli. We present experiments with Puli on WSN and knapsack benchmarks, which show remarkable improvements in the number of solved instances as well as computation time compared to existing solvers. Furthermore, we show that further significant improvement can be realized on so-called monotonous problems, such as WSN optimization, for which Puli can generate more precise assertions. We present Puli as a work-in-progress prototype that we are planning to upgrade to an official release soon, which we want to make publicly available.

1 Introduction

Considering *WSN (Wireless Sensor Network)* problems, sensor devices are self-powered and, therefore, have limited power supply. Therefore, it is crucial to apply energy efficient protocols to the sensor nodes by synchronizing their operations. To save energy, a sensor node might eventually enter the sleep mode, in which its power consumption is typically the fraction of that in the active mode. However, coverage (and other security constraints) should be maintained during the entire lifetime of the WSN, therefore we typically want to generate a *sleep/wake-up scheduling* which does not violate any of those constraints at any time and provides a *maximal lifetime* for the WSN.

Satisfiability Modulo Theories (SMT) is a powerful tool to solve constraint satisfaction problems in diverse application areas including software and hardware verification, planning, scheduling, etc. A few previous works apply SMT formalization to the aforementioned WSN constraints and use SMT solvers to generate an appropriate sleep/wake-up scheduling for a WSN. [9] focuses only on the coverage problem and reports experiments with the SMT solver Z3 [6]. [7] addresses not only the coverage problem, but also the evasive constraint and the moving target constraint, and utilizes the SMT solver Yices [8] for experiments. Note that

*The research was supported by the grant EFOP-3.6.1-16-2016-00001 "Complex improvement of research capacities and services at Eszterházy Károly University"

both [9] and [7] are only interested in generating a sleep/wake-up scheduling that fulfills the aforementioned constraints, but none of them addresses the maximization problem of the WSN lifetime, which makes them not as practical.

In [10], we generate *Optimization Module Theories (OMT)* benchmarks for WSNs and apply OMT solvers to obtain a sleep/wake-up scheduling for a WSN that maximizes its lifetime. In those experiments, we compare the results by OptiMathSAT [13], Z3 [3], and Symba [12]. We conclude there that "OptiMathSAT provides the most stable performance and scales the best". The follow-up paper [11] investigates further aspects of WSNs, in particular their density, that makes the OMT solving process more challenging. We report on further benchmarks and experiments with OptiMathSAT.

The aim of this paper is to improve the results of existing OMT solvers by taking into consideration some resources in a WSN and then applying *regression analysis* on those resource values. The idea can be generalized to extending an OMT problem with a definition of a *resource function*. Further improvement can be realized on so-called *monotonous* OMT problems. Section 3 shows how to apply our resource-based regression analysis technique to WSN optimization, which is actually a monotonous problem, and to combine it with linear search. We introduce our work-in-progress OMT solver called Puli, and report on experiments with Puli, OptiMathSAT, Z3 and Symba in Section 4. In Section 5, we demonstrate how to apply the same idea to the *knapsack problem*, and then report on experiments. All the experiments show remarkable improvements in the number of solved instances as well as computation time compared to existing OMT solvers.

2 Preliminaries

Satisfiability Modulo Theories (SMT) is the decision problem of checking satisfiability of logical formulas with respect to some background theory. *Optimization Modulo Theories (OMT)* is an extension of SMT which allows to find models that optimize given objective functions. Common theories include the theory of integers, reals, fixed-size bit-vectors, etc. The logics that one could use might differ from each other in the linearity or non-linearity of arithmetic, the presence or absence of quantifiers, or in the presence or absence of uninterpreted functions. In this paper, we are using the quantifier-free logic of linear integer arithmetic with uninterpreted functions (QF_UFLIA). The SMT-LIB format [1], as the common input format for SMT solvers, defines the syntax for QF_UFLIA formulas, where the most important features are as follows: (1) No quantifiers $\forall$ and $\exists$ are allowed to be used. (2) Every expression must be of type integer or Boolean. (3) Only the arithmetic operations addition, subtraction, multiplication, division, and comparison can be used. (4) For the sake of linear arithmetic, expressions with multiplication are allowed to be used only in the format $c * t$ where c is a constant. (5) It is allowed to use uninterpreted function symbols, i.e., to specify only the signature for such a function symbol.

Given the number $n \geq 1$ of the sensor nodes, let r_i denote the range of the i^{th} sensor node. The greater the range is, the shorter the lifetime of the sensor node is, denoted by L_i. The objective for the sensor nodes is to cover a set of $m \geq 1$ points of interest. Let $d_{i,j}$ denote the physical distance between the i^{th} sensor node and the j^{th} point. Let $w_{i,t}$ be a Boolean variable that denotes whether the i^{th} sensor node is awake at the t^{th} time interval. T denotes the lifetime of the WSN, thus the *objective function* is $\max : T$.

We define the following constraints on a WSN:

- **Lifetime constraint.** For each sensor node, the number of time intervals at which the

node is awake must not exceed the node's lifetime.

$$\forall i\ (1 \leq i \leq n).\ \sum_{t=1}^{T} w_{i,t} \leq L_i$$

- **Coverage constraint [14, 5, 4].** Every point must be covered by at least $K \geq 1$ sensor nodes at any time.

$$\forall j,t\ (1 \leq j \leq m,\ 1 \leq t \leq T).\ \sum_{i \in S_j} w_{i,t} \geq K$$

where $S_j = \{i \mid d_{i,j} \leq r_i\}$.

- **Evasive constraint [2, 7, 10].** Each sensor node must not stay active for more than $E \geq 1$ consecutive time intervals.

$$\forall i,t\ (1 \leq i \leq n,\ 1 \leq t \leq T - E).\ \sum_{t'=t}^{t+E} w_{i,t'} \leq E$$

- **Moving target constraint [7, 10].** Some of the points are considered to be critical points, which must not be covered by the same sensor for more than $M \geq 1$ consecutive time intervals.

$$\forall j \in CR,\ \forall i \in S_j,\ \forall t\ (1 \leq t \leq T - M).\ \sum_{t'=t}^{t+M} w_{i,t'} \leq M$$

where $CR \subseteq [1, m]$.

Note that all the aforementioned constraints are linear arithmetic constraints. All the variables whose value depends on T are represented as uninterpreted functions. For the sake of the feasibility of the solving process, it is advisable to avoid quantifiers by unrolling them. Therefore, all the aforementioned constraints can be encoded as QF_UFLIA formulas.

3 Search Strategies

3.1 Boosting Linear Search by Regression

How to look for the maximal lifetime for a WSN? We can start with setting the lifetime T to the lowest possible value 1, generating an SMT-LIB instance for the given WSN of lifetime T, and checking if that instance is satisfiable (SAT). If it is, we can increase the lifetime and repeat the process again. This loop stops when we find the first unsatisfiable (UNSAT) instance. The maximal lifetime corresponds to the SAT instance just before the first UNSAT one in the row.

Of course, this is just a naive linear search which is not yet competitive with existing, more sophisticated OMT solving approaches. The problem is that we solve all the SAT instances for each possible T. To boost the search, we should not solve all the SAT instances if certain instances can safely be skipped. For this reason, we use regression analysis, in order to estimate at which T a WSN runs out of its resources and stops working.

Our approach works for any OMT problem over QF_UFLIA with a single objective function if the definition of a *resource function* f_{RES} is provided, in order to obtain data points

$\big(T, f_{\text{RES}}(T)\big)$ for regression analysis where T is the current value for the objective function. For a WSN, the charge of the batteries of sensor nodes can be considered to be such a resource, which is continuously decreasing until draining. This is estimated by the following resource function:

$$\sum_{i=1}^{n} L_i - \sum_{i=1}^{n}\sum_{t=1}^{T} w_{i,t}$$

The value of the resource function is obtained by summing the lifetime of all the sensor nodes (which is the theoretical maximum of the network's lifetime), and then subtracting the sum of the time intervals that have been used up so far. In fact, this is an estimation of how long the network will be operational from now on, i.e., how much charge is left in the batteries.

For our approach, it is also necessary to define a *resource target value* for the regression function, which we expect to be taken close to the optimum of the objective function. For a WSN, the resource target value is 0.

We have introduced two solver-specific options to SMT-LIB: `opt-resource-fun` for defining the resource function and `opt-resource-target` for defining the resource target value. Both options can be interpreted by our solver, Puli. For the WSN problem, these options are specified as follows:

```
(define-fun resource-fun () Int
  (-
    (+ L0 L1 ...)
    (+ (w 0 0) (w 0 1) ...)
))
(set-option :opt-resource-fun resource-fun)
(set-option :opt-resource-target 0)
```

Algorithm 1 shows how the boosted linear search works for maximization.[1] One must specify the lower bound lb for T as an input parameter. Different SMT-LIB instances are generated for different T values in a loop, an underlying SMT solver is called with those instances, and the results of those calls are saved in a pool of $(T, result)$ pairs, where $result \in \{\text{SAT}, \text{UNSAT}\}$. The functions SaveResult and LoadResult save and load data into/from this pool, respectively. If the current instance is SAT resp. UNSAT, then T is incremented resp. decremented, and a new iteration is about to start. There are two possible exits from the loop: (1) if the instance for lb is UNSAT, then there exists no optimum; (2) if the instance for T is SAT and that for $T+1$ is UNSAT, then the optimum is T.

Otherwise, if the current instance is SAT, the algorithm makes the SMT solver return the value *resource* of the resource function and saves a new point $(T, resource)$ for regression analysis. If regression provides more than one root, we select the minimum root T' among all $T' > T$. Then we run a check on T and T' to decide (1) if it is worth to do a jump in the value of T instead of simply incrementing it, and (2) to what exact value to jump.

Some additional details about functions used in Algorithm 1:

GenerateSmtlib(T)**:** An assertion on the value of the objective function f_{OBJ} is added. For maximization, this assertion is $f_{\text{OBJ}} \geq T$, by default. Section 3.3 introduces a special class of optimization problems for which this assertion can be more specific.

SmtSolve$(smtlib)$**:** The prototype of Puli uses Z3 as the underlying SMT solver.

RegressionOnResourcePoints$()$**:** The prototype of Puli uses linear regression.

[1] From this, the algorithm for minimization can be obtained easily.

Algorithm 1 PULI's algorithm for maximization

```
1: procedure MAXIMIZATION(lb)
2:     T ← lb
3:     while true do
4:         if LOADRESULT(T) = SAT then
5:             increment T
6:         else if LOADRESULT(T) = UNSAT then
7:             if T = lb then
8:                 return null
9:             end if
10:            decrement T
11:        else
12:            smtlib ← GENERATESMTLIB(T)
13:            (T, result, resource) ← SMTSOLVE(smtlib)
14:            SAVERESULT(T, result)
15:            if result = SAT then
16:                if LOADRESULT(T + 1) = UNSAT then
17:                    return T
18:                end if
19:                SAVERESOURCEPOINT(T, resource)
20:                regression ← REGRESSIONONRESOURCEPOINTS()
21:                T′ ← minimum root of regression where T′ > T
22:                if CONDITIONFORJUMP(T, T′) then
23:                    T ← JUMP(T, T′)
24:                end if
25:            end if
26:        end if
27:    end while
28: end procedure
```

CONDITIONFORJUMP(T, T'), JUMP(T, T'): The prototype of PULI employs a simple jump strategy for the sake of skipping the most possible SAT instances and hitting the least possible UNSAT instances.

3.2 Binary Search

To look for the maximal lifetime for a WSN, we can apply binary search as well, if an upper bound for the objective function is known. For a WSN, the upper bound ub can be defined as $\sum_{i=1}^{n} L_i$.

Our binary search algorithm works as follows: $T = \frac{lb+ub}{2}$ is calculated, for which we call GENERATESMTLIB(T). If, for the resulting instance, the underlying SMT solver returns SAT, we set $lb = T + 1$, otherwise $ub = T - 1$.

3.3 Monotonous Problems

We call an OMT problem *monotonous* if, as incrementing the value of the objective function, all the resulting SMT instances are SAT until exceeding the optimum. That is, there exists no

UNSAT instance below the optimum. For instance, the optimization of WSNs is a monotonous problem. For such problems, the assertion that GenerateSmtlib(T) introduces can assign an exact value to the objective function: $f_{\text{OBJ}} = T$.

Puli is able to deal with monotonous and non-monotonous OMT problems as well. Dealing with WSN optimization as a monotonous problem speeds up Puli's solving significantly, as our experiments show.

4 Experiments and Results

We run experiments on the WSN benchmarks from the paper [11], for four different constraint settings and three different density groups. Experiments were run on 3.6 GHz 8-core CPU with 8 GB memory. The wall clock time limit was set to 1200 seconds and the memory limit to 3 GB. Then we run Puli's linear search boosted by regression, Puli's binary search, and the solvers OptiMathSAT, Z3 and Symba.

Tables 1, 2 and 3 summarize the results of our experiments for the three different density groups, respectively. The columns show the total number of solved SAT/UNSAT instances, the number of timeouts ($\#TO$), the average optimum found for the SAT instances (*Optimum*), and the average runtime resp. memory consumption (*Time* resp. *Space*).

It is clearly visible that Puli provides an remarkably stable performance: it can solve almost all instances with significantly low runtime and less memory. Among the three other solvers, OptiMathSAT has the most stable performance, as it was already shown in [10]. Compared to OptiMathSAT, Z3 and Symba are worth to use on the easier benchmarks, i.e., the ones with lower density and fewer constraints, but their performance significantly declines as benchmarks getting harder.

5 Another Example: the Knapsack Problem

For further experiments, we generated OMT benchmark instances for the *0-1 knapsack problem* as follows. Given the number $n \geq 1$ of items to put in the knapsack. Let w_i and v_i denote the weight and the value of the i^{th} item, respectively. The maximum weight capacity is denoted by $wLimit$. Let x_i be a Boolean variable that denotes if the i^{th} item is put in the knapsack.

A single constraint is needed to be asserted:

$$\sum_{i=1}^{n} w_i x_i \leq wLimit$$

The *objective function* is

$$\max : \sum_{i=1}^{n} v_i x_i$$

We define the *resource function* as the remaining capacity in the knapsack, as follows:

$$wLimit - \sum_{i=1}^{n} w_i x_i$$

In the SMT-LIB instances, our solver-specific options are used as follows:

	Constraint settings	*#SAT/UNSAT*	*#TO*	*Optimum*	*Runtime*	*Space*
Puli's linear search	allon	19/0	1	30.5	63.1	75.1
	evasive off	20/0	0	35.5	3.4	66.4
	moving target off	20/0	0	49.5	6.4	72.5
	evasive moving off	20/0	0	55.7	7.9	63.6
Puli's binary search	allon	19/0	1	30.5	63.7	71.0
	evasive off	20/0	0	35.5	3.4	62.0
	moving target off	20/0	0	49.5	5.8	66.4
	evasive moving off	20/0	0	55.7	3.6	57.0
OptiMathSAT	allon	19/0	1	30.5	173.9	430.6
	evasive off	20/0	0	35.5	51.0	293.3
	moving target off	18/0	2	48.2	285.2	422.6
	evasive moving off	18/0	2	53.0	273.4	374.9
Z3	allon	10/0	10	8.6	605.8	554.7
	evasive off	20/0	0	35.5	64.1	388.3
	moving target off	16/0	4	47.2	292.9	614.5
	evasive moving off	20/0	0	55.7	8.7	138.7
Symba	allon	10/0	10	13.1	654.4	684.4
	evasive off	20/0	0	35.5	152.8	421.4
	moving target off	12/0	8	39.9	632.0	710.5
	evasive moving off	20/0	0	55.7	118.0	187.8

Table 1: Results for different constraint settings for WSNs of 40-50% density.

```
(define-fun resource-fun () Int
  (-
    wLimit
    (+ (* w0 x0) (* w1 x1) ...)
))
(set-option :opt-resource-fun resource-fun)
(set-option :opt-resource-target 0)
```

Note that the knapsack problem is a *non-monotonous* OMT problem in general. Therefore, we cannot assign exact values to the target function in each iteration and, thus, we can expect longer runtimes.

5.1 Further Experiments

Table 4 shows the results of the preliminary experiments with the knapsack problem, only against OptiMathSAT. The number of items is 150, the values and the weights of items are random numbers between 1 and 100. The knapsack's capacity is set to the 70% of the total weight of all the items.

The results show that Puli outperforms OptiMathSAT by at least one order of magnitude on those instances.

	Constraint settings	*#SAT/UNSAT*	*#TO*	*Optimum*	*Runtime*	*Space*
Puli's linear search	allon	19/0	1	80.2	79.9	122.3
	evasive off	20/0	0	80.7	14.5	92.6
	moving target off	20/0	0	82.9	44.9	102.7
	evasive moving off	20/0	0	84.3	23.7	69.8
Puli's binary search	allon	19/0	1	80.2	75.5	117.6
	evasive off	20/0	0	80.7	10.9	86.4
	moving target off	20/0	0	82.9	17.9	95.5
	evasive moving off	20/0	0	84.3	6.6	62.8
OptiMathSAT	allon	16/0	4	78.8	327.2	613.0
	evasive off	19/0	1	80.2	190.7	405.0
	moving target off	16/0	4	81.6	397.0	483.4
	evasive moving off	18/0	2	83.9	522.7	625.8
Z3	allon	6/0	14	51.0	858.8	1029.7
	evasive off	16/0	4	75.1	325.1	592.3
	moving target off	13/0	7	78.8	525.5	672.4
	evasive moving off	20/0	0	84.3	11.9	143.3
Symba	allon	7/0	13	54.2	924.5	962.0
	evasive off	15/0	5	77.6	573.4	712.8
	moving target off	9/0	11	71.7	794.7	755.2
	evasive moving off	20/0	0	84.3	127.7	197.2

Table 2: Results for different constraint settings for WSNs of 60-70% density.

6 Conclusion

In this paper, we proposed the idea of speeding up the OMT solving process by taking into consideration a *resource function* in the system to optimize and by applying *regression analysis* on those resource values. Furthermore, we introduced a class of OMT problems, the so-called *monotonous* problems, for which the solving process can be further boosted.

We introduced a *new OMT solver* called Puli and reported experiments on different OMT benchmarks for *WSNs* and for the *knapsack problem*. The results show that Puli significantly outperforms the OMT solvers OptiMathSAT, Z3 and Symba on those benchmarks. In general, Puli can solve any QF_UFLIA problem with a single objective function and can apply its the regression-based boosting if the definition of a *resource function* is provided. For this, we introduced two solver-specific options to SMT-LIB: `opt-resource-fun` for defining the resource function and `opt-resource-target` for defining the resource target value. Both options can be interpreted by Puli.

All the benchmarks and the log files are available at https://iot.uni-eszterhazy.hu/en/research/tools. As future work, we are going to implement Puli in C/C++ as well and to make an official release publicly available. Now that we have an OMT solver scales enough, we are planning to generate benchmarks for WSNs of larger size and of more complex model. We are also planning to formalize other optimization problems, for instance, the problem of finding the minimal spanning tree in a graph, which is of great importance in identifying the backbone of a network. Furthermore, we want to experience with pseudo-Boolean variables and ILP solving.

	Constraint settings	*#SAT/UNSAT*	*#TO*	*Optimum*	*Runtime*	*Space*
Puli's linear search	allon	20/0	0	119.3	93.4	184.2
	evasive off	20/0	0	119.3	53.5	126.5
	moving target off	20/0	0	119.3	141.1	135.9
	evasive moving off	20/0	0	119.3	40.5	75.0
Puli's binary search	allon	20/0	0	119.3	42.3	176.0
	evasive off	20/0	0	119.3	23.3	119.2
	moving target off	20/0	0	119.3	35.8	127.8
	evasive moving off	20/0	0	119.3	11.4	67.5
OptiMathSAT	allon	20/0	0	119.3	192.8	678.9
	evasive off	19/0	1	120.8	190.5	457.9
	moving target off	18/0	2	117.6	619.5	631.7
	evasive moving off	14/0	6	116.8	794.5	847.2
Z3	allon	9/0	11	117.8	774.7	1198.8
	evasive off	17/0	3	119.5	441.8	716.4
	moving target off	12/0	8	119.3	589.5	699.6
	evasive moving off	20/0	0	119.3	13.3	126.1
Symba	allon	4/0	16	125.0	1098.8	1238.5
	evasive off	17/0	3	117.6	583.5	727.7
	moving target off	11/0	9	123.2	820.2	763.5
	evasive moving off	20/0	0	119.3	155.0	171.4

Table 3: Results for different constraint settings for WSNs of 80-90% density.

Solver	*#SAT/UNSAT*	*#TO*	*Optimum*	*Runtime*	*Space*
Puli's linear search	30/0	0	6996.3	32.3	45.4
Puli's binary search	30/0	0	6996.3	3.3	45.9
OptiMathSAT	30/0	0	6996.3	103.6	110.7

Table 4: Results for the knapsack problem.

References

[1] Clark Barrett, Pascal Fontaine, and Cesare 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`.

[2] Zinaida Benenson, Felix C. Freiling, and Peter M. Cholewinski. Advanced evasive data storage in sensor networks. In *Proc. Int. Conf. on Mobile Data Management*, MDM'07, pages 146–151. IEEE Computer Society, 2007.

[3] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. μZ - an optimizing SMT solver. In *Proc. Int. Conf. for Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 9206 of *LNCS*, pages 194–199. Springer, 2015.

[4] Mihaela Cardei. Coverage problems in sensor networks. In *Handbook of Combinatorial Optimization*, pages 899–927. Springer, 2013.

[5] Mihaela Cardei and Ding-Zhu Du. Improving wireless sensor network lifetime through power aware

organization. *Wireless Networks*, 11(3):333–340, 2005.

[6] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proc. Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, TACAS'08/ETAPS'08, pages 337–340. Springer-Verlag, 2008.

[7] Qi Duan, Saeed Al-Haj, and Ehab Al-Shaer. Provable configuration planning for wireless sensor networks. In *Proc. 8th Int. Conf. on Network and Service Management (CNSM) and Workshop on Systems Virtualization Management (SVM)*, pages 316–321, 2012.

[8] Bruno Dutertre. Yices 2.2. In Armin Biere and Roderick Bloem, editors, *Proc. Int. Conf. on Computer-Aided Verification (CAV)*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, 2014.

[9] Weiqiang Kong, Ming Li, Long Han, and Akira Fukuda. An SMT-based accurate algorithm for the K-coverage problem in sensor network. In *Proc. 8th Int. Conf. on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM)*, pages 240–245, 2014.

[10] G. Kovásznai, Cs. Biró, and B. Erdélyi. Generating optimal scheduling for wireless sensor networks by using optimization modulo theories solvers. In *Proc. Int. Workshop on Satisfiability Modulo Theories (SMT)*, volume 1889 of *CEUR*, pages 15–27, 2017.

[11] 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)*, pages 1–8. IEEE, 2018.

[12] Yi Li, Aws Albarghouthi, Zachary Kincaid, Arie Gurfinkel, and Marsha Chechik. Symbolic optimization with SMT solvers. In *Proc. ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 607–618. ACM, 2014.

[13] Roberto Sebastiani and Patrick Trentin. OptiMathSAT: A tool for optimization modulo theories. In *Proc. Int. Conf. on Computer-Aided Verification (CAV)*, volume 9206 of *LNCS*, pages 447–454. Springer, 2015.

[14] Di Tian and Nicolas D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *Proc. Int. Workshop on Wireless Sensor Networks and Applications*, WSNA'02, pages 32–41. ACM, 2002.