SMT-based Verification Applied to Non-convex Optimization Problems

Rodrigo Araújo*, Iury Bessa†, Lucas C. Cordeiro†‡, and João Edgar Chaves Filho†.
*Federal Institute of Amazonas, Brazil
†Federal University of Amazonas, Brazil
‡University of Oxford, UK
Email: rodrigo.araujo@ifam.edu.br, {iurybessa,lucascordeiro,jedgarc}@ufam.edu.br

***Abstract*—This paper presents a novel, complete, and flexible optimization algorithm, which relies on recursive executions that re-constrains a model-checking procedure based on Satisfiability Modulo Theories (SMT). This SMT-based optimization technique is able to optimize a wide range of functions, including non-linear and non-convex problems using fixed-point arithmetic. Although SMT-based optimization is not a new technique, this work is the pioneer in solving non-linear and non-convex problems based on SMT; previous applications are only able to solve integer and rational linear problems. The proposed SMT-based optimization algorithm is compared to other traditional optimization techniques. Experimental results show the efficiency and effectiveness of the proposed algorithm, which finds the optimal solution in all evaluated benchmarks, while traditional techniques are usually trapped by local minima.**

***Keywords*—satisfiability modulo theory (SMT), model checking, optimization, global minima, non-convex problems.**

I. Introduction

Optimization is an important research topic in many fields, especially in computer science and engineering [1]. Scientists and engineers have to find parameters (*i.e.*, an optimal solution), which optimize the behavior of a given system or the value of a given function. There are various optimization techniques (*e.g.*, simplex [2], gradient descent [3], and genetic algorithms [4]), which are suitable for different classes of optimization problems (*e.g.*, linear or non-linear, continuous or discrete, convex or non-convex, and single- or multi-objective).

However, the widespread development of embedded systems demand continuous application and deployment of several optimization methods. In particular, due to resource limitation, embedded system design typically includes optimization of cost functions related to power consumption, memory usage, and silicon area [5]. Since optimization problems are commonly solved by computer-based systems, several embedded system applications include optimization problem solving during their execution (*e.g.*, autonomous vehicles navigation systems [6]). As a result, effective and efficient optimization algorithms are needed for leveraging the proliferation on the use and application of embedded systems.

Particularly, a continuous non-convex optimization problem is one of the most complex problems; as a result, several traditional methods (*e.g.*, Newton-Raphson [1] and Gradient Descent [3]) are inefficient to solve that particular class of problems [1]. Therefore, various heuristics are developed for obtaining approximate solutions to those problems. Heuristics methods (*e.g.*, ant colony [7] and genetic algorithms [4]) offer faster solutions for complex problems, but they sacrifice the system's correctness and are easily trapped by local optimal solutions. Thus, they are not suitable for applying to embedded system applications, especially in safety-critical systems whose failure or malfunction may result in serious injury to people or environmental harm.

Recently, Propositional Satisfiability (SAT) and Satisfiability Modulo Theories (SMT) solvers [8] have been applied to solve optimization problems; specialized theories and tools were also developed and evaluated [9]–[11]. The first SMT/SAT solvers applications to optimization problems aim at optimizing cost functions based on Boolean variables and Integer Linear Programming (ILP). Estrada [12] is an example of that type of application, which uses a SAT solver for minimizing the number of gates in digital circuits, thus optimizing the FPGA design. Trindade and Cordeiro apply the Efficient SMT-based Context-Bounded Model Checker (ESBMC) to optimize the hardware/software partitioning in embedded systems [13]–[15]. Recently, Eldib and Wang [16] propose the use of SMT solvers for optimizing fixed-point embedded C software, thus minimizing the minimum bit-width required to run it on a micro-controller.

The potential of SMT-based optimization tools led to the development of solutions for optimizing functions over rationals, which was initially proposed by Sebastiani *et al.* [17], [18]; and the advent of SAT/SMT solvers (*e.g.*, νZ [11] and OptiMathSAT [10]) and theories specialized in optimization problems (*e.g.*, ILP Modulo Theories [9]). However, there is still a lack of SMT tools suitable for non-linear and non-convex optimization problems. To the best of our knowledge, all current SAT/SMT-based optimization studies are limited to linear cost functions, and they cannot be directly applied to solve non-linear (non-convex) problems arising from embedded system applications.

Contributions. This paper presents a novel optimization technique based on SMT solvers, which is suitable for a wide variety of functions, even for non-linear and non-convex functions that are typically found in navigation systems. The function evaluation and the search for optimal solution is performed by means of a recursive execution of successive verifications based on SMT queries. In contrast to other heuristic methods (*e.g.*, genetic algorithms) that are usually employed for optimizing this class of function, the present approach always finds the global optimal point.

Availability of Data and Tools. Our experiments are based on a set of publicly available benchmarks. All tools, benchmarks, and results of our evaluation are available on a supplementary web page `http://esbmc.org/benchmarks`.

Outline. Section II describes fundamentals related to optimization and model-checking techniques. Section III describes our optimization algorithm in further details from a model-

checking perspective. Section IV reports the experimental results for evaluating that optimization algorithm. Section V discusses related studies, while Section VI concludes this work and proposes future studies.

II. Preliminaries

A. Optimization Problems

Let $f : X \rightarrow \mathbb{R}$ be a cost function, such that $X \subset \mathbb{R}^n$ is the decision variables vector $x_1, x_2, ..., x_n$ and $f(x_1, x_2, ..., x_n) \equiv f(\mathbf{x})$. Let $\Omega \subset X$ be a subset settled by a set of constraints.

Definition 1. *A multi-variable optimization problem consists in finding an optimal vector $\boldsymbol{x}$, which minimizes f in Ω.*

According to Definition 1, this type of optimization problem can be written as

$$\begin{array}{ll} \min & f(\mathbf{x}), \\ \text{s.t.} & \mathbf{x} \in \Omega. \end{array} \tag{1}$$

This optimization problem can be classified in different ways w.r.t. cost function f, *e.g.*, it can be linear or non-linear; convex or non-convex; continuous, discontinuous, discrete, integer, rational, or real. Depending on that classification, different optimization techniques can be more suitable to solve that cost function, and some algorithms usually point to suboptimal solutions, *i.e.*, a solution that is not a global minimum of f, but it only locally minimizes f. Global optimal solutions of the function f, aforementioned, can be defined as

Definition 2. *A vector $\boldsymbol{x}^* \in \Omega$ is a global optimal solution of f in Ω iff $f(\boldsymbol{x}^*) \leq f(x), \forall \boldsymbol{x} \in \Omega$.*

Gradient Descent (GD) [3] and Genetic Algorithm (GA) [4] are examples of popular optimization techniques. GD iteratively minimizes a function f by opposite direction of its gradient by means of a step sized by a learning rate. GA optimizes a function with a heuristic search method inspired in natural selection, representing solutions with strings named chromosomes, which are evaluated using a fitness function. The solution is enhanced via computational selection process, re-combinations, and mutations. Both techniques present several limitations, *e.g.*, GD cannot handle function with discontinuities, and might be easily trapped by local minima.

B. Model Checking

Model checking is an automated verification procedure to exhaustively check all (reachable) system's states [19]. The model checking procedure typically consists of three steps: modeling, specification, and verification.

Modeling is the first step, where it converts the system to a formalism that is accepted by a verifier. The modeling step usually requires the use of an abstraction to eliminate irrelevant (or less) important system details [20]. The second step is the specification, which describes the system behavior and the property to be checked. An important issue in the specification is the correctness. Model checking provides ways to check whether a given specification satisfies a system property, but it is difficult to determine whether such specification covers all properties, which the system should satisfy.

Finally, the verification step checks whether a given property is satisfied w.r.t. a given model, *i.e.*, all relevant system states are checked to search for any state that violates the verified property. In case of a property violation, the verifier reports the system execution trace (counterexample), *i.e.*, all steps from the (initial) state to the (bad) state that violates such property. Errors could occur due to incorrect system modeling or inadequate specification, thus generating false results.

C. SMT-Based Bounded Model Checking

The International Competition on Software Verification (SV-COMP) is an annual competition on software verification tools, which evaluates and presents several software testing and verification techniques [21]. One important verification technique, which has presented attractive results over the last years is Bounded Model Checking (BMC). BMC techniques based on SAT [8] or SMT [22], have been successfully applied to verify single- and multi-threaded programs, and also to find subtle bugs in real programs [23], [24]. The idea behind BMC is to check the negation of a given property at a given depth.

Definition 3. *[8] – Given a transition system M, a property ϕ, and a bound k; BMC unrolls the system k times and translates it into a verification condition (VC) ψ, which is satisfiable if and only if ϕ has a counterexample of depth less than or equal to k.*

In this study, the ESBMC tool [25] is used as the verification engine, as it represents one of the most efficient BMC tools that participated in the last software verification competitions [25]; in particular, ESBMC is one of the most efficient tools to reason about programs that make use of bit-vector arithmetic according to the SV-COMP 2016 edition.

ESBMC is an SMT-based bounded model checker for C/C++ programs. ESBMC finds property violations such as pointer safety, array bounds, atomicity, overflows, deadlocks, data race, and memory leaks in single- and multi-threaded software. It also verifies programs that make use of bit-level, pointers, structures, unions, and fixed-point arithmetic. Inside ESBMC, the associated problem is formulated by constructing the following logical formula

$$\psi_k = I(S_0) \wedge \bigvee_{i=0}^{k} \bigwedge_{j=0}^{i-1} \gamma(s_j, s_{j+1}) \wedge \overline{\phi(s_1)} \tag{2}$$

where ϕ is a property and S_0 is a set of initial states of M, and $\gamma(s_j, s_{j+1})$ is the transition relation of M between time steps j and $j+1$. Hence, $I(S_0) \wedge \bigwedge_{j=0}^{i-1} \gamma(s_j, s_{j+1})$ represents the executions of a transition system M of length i. The above VC ψ can be satisfied if and only if, for some $i \leq k$ there exists a reachable state at time step i in which ϕ is violated. If the logical formula (2) is satisfiable (*i.e.*, returns $true$), then the SMT solver provides a satisfying assignment (counterexample).

Definition 4. *A counterexample for a property ϕ is a sequence of states $s_0, s_1, \ldots, s_k$ with $s_0 \in S_0$, $s_k \in S_k$, and $\gamma(s_i, s_{i+1})$ for $0 \leq i < k$ that makes (2) satisfiable. If it is unsatisfiable (i.e., returns false), then one can conclude that there is no error state in k steps or less.*

In addition to embedded software verification, ESBMC has been applied to ensure correctness of discrete-time filters and controllers [26]–[28]. Furthermore, recently ESBMC has been applied to optimizing problems related to hardware/software co-design [13]–[15].

III. SMT-based Verification Applied to Non-convex Optimization Problems

A. Modeling Optimization Problems using a Model Checker

There are two important directives in the C/C++ programming language, which can be used for modeling and controlling a verification process: `ASSUME` and `ASSERT`. The `ASSUME` directive is able to define constraints over (non-deterministic) variables, and the `ASSERT` directive is used to check system's correctness w.r.t. a given property. Using these two statements, any off-the-shelf C/C++ model checker (*e.g.*, ESBMC [25], CBMC [24], and CPAChecker [29]) could be applied to check specific constraints in optimization problems, as described in Eq. (1).

Here, the verification process is recursively repeated to solve an optimization problem using intrinsic functions available in ESBMC (*e.g.*, *__ESBMC_assume* and *__ESBMC_assert*). Although ESBMC implements BMC, a front-end algorithm executes ESBMC recursively, modifying all verification constraints to efficiently prune the state-space search. Note that completeness is not an issue here (cf. Definitions 1 and 2) since loops are not encoded into the VCs (cf. Definition 2) that are passed to the SMT solver.

B. Illustrative Example

To illustrate the present SMT-based optimization method for non-convex optimization problems, the Himmelblau's function is employed. The Himmelblau's function is represented by a two-variables function with four global minimums (one on each quadrant of x_1x_2 plane) in $f(x_1, x_2) = 0$. Himmelblau's function is defined by Eq. 3; Fig. 1 shows its respective graph [30].

$$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \qquad (3)$$

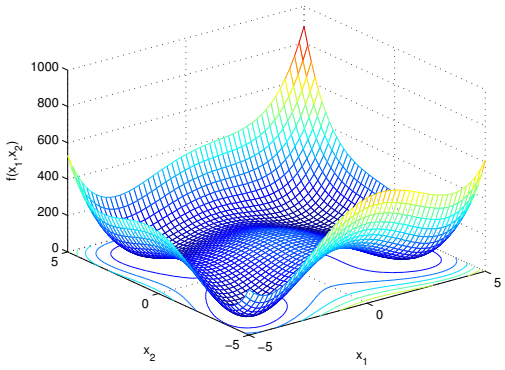


Fig. 1: Himmelblau's function

1) Modeling: The modeling process defines constraints, *i.e.*, Ω boundaries. This step is important for reducing the state-space search and consequently avoiding the state-space explosion by the underlying model-checking procedure. This optimization algorithm is not efficient for unconstrained optimization, and the verification time can be drastically reduced by means of a suitable constraint choice. Note that boundaries have to be chosen based on previous knowledge about the problem over which the optimization is being applied.

Consider the optimization problem given by Eq. (4), which is related to the Himmeblau's function given in Eq. (3):

$$\begin{aligned} \min \quad & f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2, \\ \text{s.t.} \quad & x_1 \leq 0, \\ & x_2 \geq 0. \end{aligned} \qquad (4)$$

Note that inequalities $x_1 \leq 0$ and $x_2 \geq 0$ are pruning the state-space search to the second quadrant; however, even so it produces a (huge) state-space to be explored since x_1 and x_2 can assume values with very high modules. The optimization problem given by Eq. (4) can be properly rewritten as Eq. (5) with new constraints; those new constraints are obtained based on previous knowledge about the function, but an incorrect constraint choice might exclude the global minimum.

$$\begin{aligned} \min \quad & f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2, \\ \text{s.t.} \quad & -7 \leq x_1 \leq 0, \\ & 0 \leq x_2 \leq 7. \end{aligned} \qquad (5)$$

From the optimization problem formal definition given by Eq. (5), the modeling step can be encoded, where decision variables are declared as non-deterministic variables constrained by the `ASSUME`. Fig. 2 shows the respective C code for Eq. (5).

```
1  float nondet_float();
2  int main() {
3    //define decision variables
4    float x1 = nondet_float();
5    float x2 = nondet_float();
6    //constrain the state-space search
7    __ESBMC_assume((x1>=-7) && (x1<=0));
8    __ESBMC_assume((x2>=0) && (x2<=7));
9    //computing Himmelblau's function
10   float fobj = (x1^2+x2-11)*(x1^2+x2-11)
11    +(x1+x2^2-7)*(x1+x2^2-7);
12   return 0;
13 }
```

Fig. 2: C Code for the optimization problem given by Eq. (5).

2) Specification: The next step of the proposed methodology is the specification, where the system behavior and the property to be checked are described. For the Himmelblau's function, the result of the specification step is the C program shown in Fig. 3, which is checked by the underlying verifier.

Indeed, the C program shown in Fig. 2 leads the verifier to produce a considerably large state-space exploration, and consequently it takes a longer verification time, if the decision variables are declared as non-deterministic floating-point datatype. In this study, decision variables are defined as non-deterministic integers, thus discretizing and reducing the state-space exploration. However, this also reduces the optimization process precision.

To trade-off both precision and verification time, also to maintain the convergence to an optimal solution, the underlying model-checking procedure has to be recursively invoked, in order to increase its precision for each successive execution. An integer variable $p = 10^n$ is created and iteratively adjusted, such that n is the amount of decimal places related to the decision variables. Additionally, a new constraint is inserted; in particular, the new value of the objective function $f(\mathbf{x}^{(i)})$

at the i-th must not be greater than the value obtained in the previous iteration $f(\mathbf{x}^{(i-1)})$. Initially, all elements in the state-space search Ω are candidates for optimal points, and this constraint cutoffs several candidates on each iteration.

A property has to be specified to ensure the convergence to the minimum point on each iteration. This property specification is stated by means of an assertion, which checks whether the literal $l_{optimal}$ given in Eq. (6) is satisfiable for every optimal candidate f_c remaining in the state-space search (*i.e.*, traversed from lowest to highest).

$$l_{optimal} \iff f(\mathbf{x}) > f_c \tag{6}$$

The verification procedure stops when the literal $l_{optimal}$ is "false", *i.e.*, if there is any $\mathbf{x}^{(i)}$ for which $f(\mathbf{x}^{(i)}) \leq f_c$; a counterexample shows such $\mathbf{x}^i$, converging iteratively $f(\mathbf{x})$ from the optimal $\mathbf{x}^*$. Fig. 3 shows the initial specification for the optimization problem given by Eq. (5). The initial value of the objective function can be randomly initialized. For the example in Fig. 3, $f(\mathbf{x}^{(0)})$ is arbitrarily initialized to 100, but the present optimization algorithm works for any initial state. Note that a previous knowledge about the function allows the reduction of candidate functions to perform the search for $f_c > 0$, once the Himmelblau's function is positive.

```
1  int nondet_int();
2  int main() {
3    int p = 1;  //precision variable
4    //previous objective function value
5    float f_ant = 100;
6    int v = (int) (f_ant*p + 1);
7    int lim_inf_x1 = -7*p;
8    int lim_sup_x1 = 0*p;
9    int lim_inf_x2 = 0*p;
10   int lim_sup_x2 = 7*p;
11   int X1 = nondet_int();
12   int X2 = nondet_int();
13   float x1, x2;
14   __ESBMC_assume( (X1>=lim_inf_x1) &&
15   (X1<=lim_sup_x1) );
16   __ESBMC_assume( (X2>=lim_inf_x2) &&
17   (X2<=lim_sup_x2) );
18   x1 = (float) X1/p;
19   x2 = (float) X2/p;
20   float fobj;
21   fobj = (x1*x1+x2-11)*(x1*x1+x2-11)
22    +(x1+x2*x2-7)*(x1+x2*x2-7);
23   //constrain to exclude fobj>f_ant
24   __ESBMC_assume( fobj < f_ant );
25   float fc;   //fc: candidate for minimum
26   //objective function test
27   for (int i = 0; i <= v; i++){
28     fc = (float) i/p;
29     assert( fobj > fc );
30   }
31   return 0;
32 }
```

Fig. 3: C code after the specification of Eq. (5).

3) Verification: Finally in the verification step, the C program shown in Fig. 3 is checked by the verifier and a counterexample is returned with a set of decision variables $\mathbf{x}$, for which the objective function value is approximated from the optimal value. A specified C program only returns a successful verification result if the previous function value is the optimal point for that specific precision (defined by p), *i.e.*, $f(\mathbf{x}^{(i-1)}) = f(\mathbf{x}^*)$. For that particular example shown in Fig. 3, the verifier shows a counterexample with the following decision variables: $x_1 = 0$ and $x_2 = 2$. Note that $f(0, 2) = 90$, which is less than the initial value (100). Surely, the global minimum is in $[0, 90]$, and this verification can be repeated with the new value of $f(\mathbf{x}^{(i-1)})$, in order to obtain an objective function value close to the optimal point on each iteration.

C. SMT-based Optimization Method

Based on the methodology described in the previous section, an SMT-based verification method for non-convex optimization problems can be proposed, as shown in Alg. 1. In particular, the specification and verification steps are repeated until the optimal solution $\mathbf{x}^*$ is found. The precision of optimal solution defines the desired precision variable ϵ. An unitary value of ϵ results in integer solutions. Solution with one decimal place is obtained for $\epsilon = 10$, two decimal places are achieved for $\epsilon = 100$, *i.e.*, the number of decimal places $D.P.$ for the solution is calculated by means of the equation

$$D.P. = \log \epsilon. \tag{7}$$

Note that Alg. 1 contains two nested loops after the variable initialization and declaration (lines 1-3). In each execution of the outer loop `while` (lines 4-16), a new verification procedure is started for a respecified problem with updated bounds and precision. The inner `for` loop (line 8-14) corresponds to the verification phase, where the objective function interval (constrained during the specification phase) is traversed in $f(\mathbf{x})$ and the candidate functions f_c are analyzed through the satisfiability check of $\neg l_{optimal}$ until a $f(\mathbf{x}) \leq f_c$ violates the `ASSERT` and breaks the `for`-statement, returning to the constraint specification phase (line 7). If the `ASSERT` is not violated inside the `for`-statement, the last f_c is the minimum value with the precision variable p (initially equal to 1), p is multiplied by 10, adding a decimal place to the optimization solution, and the outer loop is repeated. Note that this algorithm uses the manipulation of fixed-point number precision in order to ensure the optimization convergence.

The execution time of this algorithm depends on how the state-space search is restricted and on the number of the solution decimal places. The algorithm presents a fixed-point solution with adjustable precision, *i.e.* the number of decimal places may be defined. Naturally, for integer optimal points, Alg. 1 returns the correct solution quickly (typically in a few seconds). However, in unconstrained problems with non-integer solution, this algorithm might take longer for achieving the optimal solution, depending on the required precision. Although this algorithm frequently produces an execution time, which is longer than other classical solutions, its error rate is typically lower than other existing methods, once it is based on a complete verification procedure.

D. Avoiding the local minima

An important feature of this proposed SMT-based optimization method is the fail-proof global minimum reachability. Many optimization algorithms might be trapped by local minima and they might incorrectly solve optimization problems. However, the present technique ensures the avoidance of those local minima, through the reliable satisfiability checking, which is performed by successive SMT queries. This property is maintained for any class of function and for any initial state.

Figures 4 and 5 show the aforementioned property of this algorithm, comparing its performance to other optimization

input : A cost function $f(\mathbf{x})$, a constraint set Ω, and a desired precision ϵ
output: The optimal decision variable vector $\mathbf{x}^*$, and the optimal value of function $f(\mathbf{x}^*)$

1 *Initialize* $f(\boldsymbol{x}^{(0)})$ *randomly*
2 *Initialize the precision variable with* $p = 1$
3 *Declare the decision variables* ($\boldsymbol{x}$) *as non-deterministic integer variables*
4 **while** $p \leq \epsilon$ **do**
5 *Define upper and lower bounds for* $\boldsymbol{x}$ *with the* `ASSUME` *directive*
6 *Describe a model for* $f(\boldsymbol{x})$
7 *Constraint* $f(\boldsymbol{x}^{(i)}) < f(\boldsymbol{x}^{(i-1)})$ *with the directive* `ASSUME`
8 **for** *Every* $f_c \leq f(\boldsymbol{x}^{(i-1)})$ *in the constrained search state-space* **do**
9 *Verify the satisfiability of* $l_{optimal}$ *given by Eq.* (6)
10 **if** $\neg l_{optimal}$ *is satisfiable* **then**
11 *Update* $\boldsymbol{x}^{(i)}$ *and* $f(\boldsymbol{x}^{(i-1)})$ *based on the counterexample*
12 *Go back to 7*
13 **end**
14 **end**
15 *Update the precision variable* ($p = p \cdot 10$)
16 **end**
17 $\boldsymbol{x}^* = \boldsymbol{x}^{(i)}$ *and* $f(\boldsymbol{x}^*) = f^{(i-1)}(\boldsymbol{x})$
18 **return** $\boldsymbol{x}^*$ *and* $f(\boldsymbol{x}^*)$

Algorithm 1: SMT-based verification algorithm for non-convex optimization.

algorithms: GA and GD. In those figures, the Himmelblau's function is adapted for a simple single-variable problem over x_1, *i.e.*, the x_2 is considered fixed and equals to 3.131, and the function is reduced to a plane crossing the global optimum in $x_1 = -2.805$ and in a local minimum with positive x_1. The partial results after each iteration are illustrated by the various marks in these graphs. Note that the SMT-based optimization method does not present continuous trajectory from the initial point to the optimal point; however, it always achieves the correct solution. Fig. 4 shows that all techniques (GA, GD, and SMT) achieve the global optimum if they are initialized in points closer to the global minimum than local minimum. However, Fig. 5 shows that both GA and GD might be trapped by the local minimum for a different initial point, close to the local minimum. In contrast, the proposed SMT-based optimization method can be initialized further away from the global minimum and as a result it can find the global minimum after some iterations as shown in Figures 4 and 5.

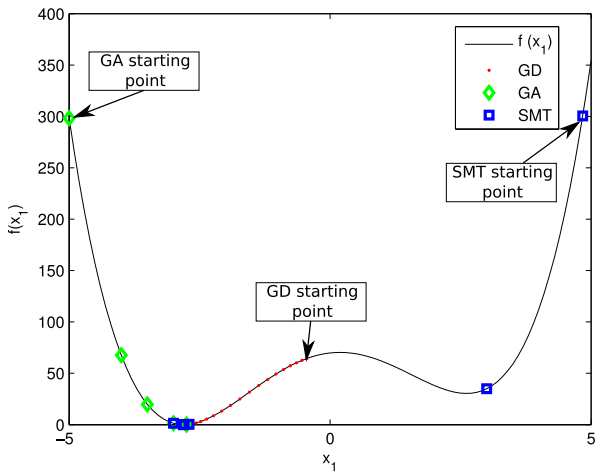


Fig. 4: Optimization trajectory of GA, GD, and SMT for a Himmelblau's plane in $x_2 = 3.131$. All methods obtain the correct answer.

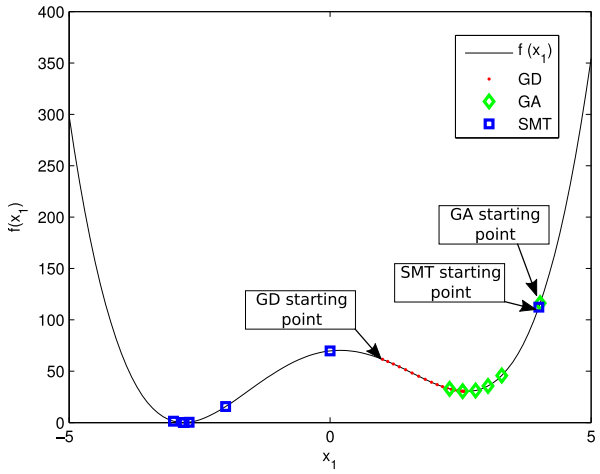


Fig. 5: Optimization trajectory of GA, GD, and SMT for a Himmelblau's plane in $x_2 = 3.131$. GA and GD are trapped by an local minimum, but SMT obtains the correct answer.

IV. Experimental Evaluation

A. Experimental Objectives and Setup

All conducted experiments evaluate the proposed SMT-based optimization algorithm and compare its performance to well-known optimization algorithms (GD and GA). The following functions are used for evaluating the present approach: Himmelblau, Styblinski-Tang, and Goldstein-Price. Note that these functions are typically employed for evaluating optimization algorithms [30]. The GA was used with a population of 10 individuals and 50 generations; the other options were defined as default for the `ga` function in the Matlab Optimization Toolbox. For GD method we used the learning rate of 0.01 to Himmelblau's and Styblinski-Tang's functions; for the Goldstein-Price's function 0.00005 was used, since there are function regions that grow too quickly and it has several local minima nearby, the GD algorithm could not converge for greater learning rates. The GD algorithm is stopped when 100 iterations are achieved or when the gradient is less than 10^{-4}.

ESBMC 3.0.0 64-bit [25] is used for this experimental evaluation. Boolector v2.1.1 [31] is employed as a back-end SMT solver for discharging the VCs (cf. Eq. (2)). All non-convex optimization benchmarks[1] are written as C programs. Matlab Optimization Toolbox [32] is used to perform gradient descent and genetic algorithm methods. A DELL Inspiron 5000 with Linux Fedora 21 64-bit Workstation, 16GB of RAM, and i7 processor Intel i7-5500U 3.0GHz clock is used throughout the experimental evaluation. The execution time is measured for gradient descent and genetic methods using the MATLAB time function, while for ESBMC all times given are wall clock time in seconds as measured by UNIX time command.

[1] Available at http://esbmc.org/benchmarks/sbesc2016_benchmarks.zip

B. Description of the Benchmarks

Table I shows equations, domains, and optimal points for each representative function employed in the following experiments. First, the proposed approach is applied to minimize the Himmelblau's function, as described in Section III. Two tests are performed with Himmelblau's function considering different domains, one restricted to the second quadrant, as given by Eq. (5) (named Himmelblau 1), and another one with four quadrants (named Himmelblau 2).

Second, the present approach is applied to minimize the Styblinski-Tang's function, as given in Table I; Fig. 6 shows its respective graph. The Styblinski-Tang's function for $d = 2$ and for the given domain is a 3-dimensional function with only one global minimum and three local minima.

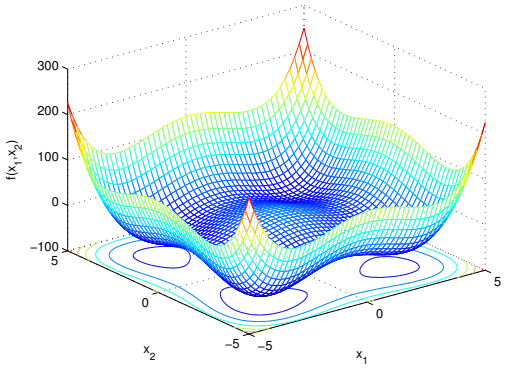


Fig. 6: Styblinski-Tang's function

Finally, the SMT-based optimization method is used to minimize the Goldstein-Price's function. Both equation and information domain are given in Table I; Fig. 7 shows its respective graph. The Goldstein-Price's function is also a 3-dimensional function with only one global minimum, but this function presents a region with several local minima.

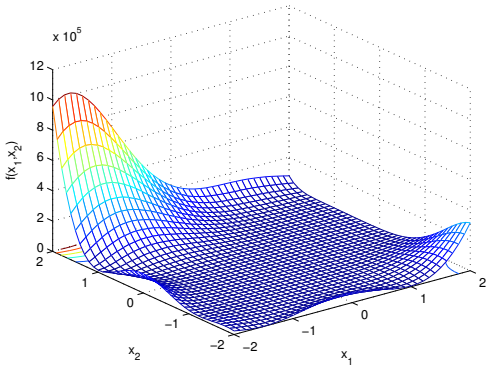


Fig. 7: Goldstein-Price's function

C. Experimental Results

Each function previously described is used 100 times in the optimization experiments for each method (GA, GD, and SMT), resulting in a total of 400 experiments. The execution time for each method is obtained from the arithmetic mean of 100 executions. The correct answers are considered only if the respective algorithm returns an error less than or equal to 10^{-3} for the function optimal value.

Table II shows all experimental results. The first column indicates the test function and the second column indicates the optimization technique (GA, GD, or SMT). The accuracy rate and execution time are presented in third and fourth columns, respectively.

For Himmelblau 1, the genetic algorithm presents satisfactory performance; however, the gradient descent method produced the correct answer only in 55% of the benchmarks. This happens due to the random initialization; additionally, the function has saddle points next to the given domain limit. The SMT-based optimization method finds the correct answer, but its execution time takes longer than other approaches.

For Himmelblau 2, the SMT-based optimization method produces satisfactory performance with much lower verification time; this happens because one of the minimum is in the integer domain. Since Alg. 1 begins its execution discretizing the state-space and continues increasing its accuracy, the solution is always the same, *i.e.*, it is found with low precision in the first loop and is only confirmed when the accuracy is increased. Fig. 8a shows the trajectories of the solutions absolute error (*i.e.*, step-by-step optimization results) of the three algorithms for the Himmelblau's 2 function. Note that three algorithms converge for the correct answer (error equals to zero) after a few iterations.

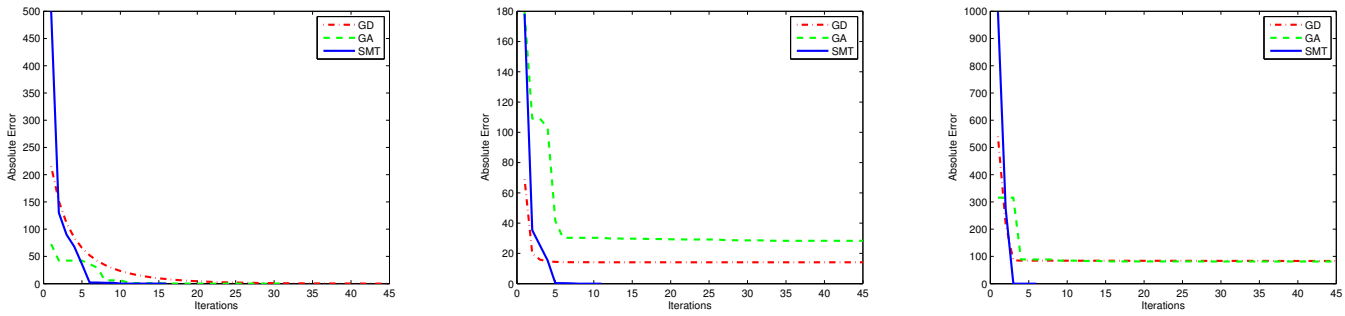
For the Styblinski-Tang's function, the genetic algorithm presents the worst results, with only 9% accuracy and gradient descent further decreasing its performance. This function has a local minimum in each quadrant of the x_1x_2 plane and the global minimum in the 4th quarter, thus it is expected that at least 75% of the gradient descent attempts are trapped in a local minimum, because it is highly dependent on the initial point. As observed in the experimental evaluation, the SMT-based optimization method properly ignores all local minima as optima solution. Fig. 8b shows the trajectories of solutions absolute error of the three algorithms for the Styblinski-Tang's function. Note that only the SMT-based optimization method converges to the optimal result; GA and GD are trapped by the local minimum and cannot achieve the error equals to zero.

Finally, for the Goldstein-Price's function that has several local minima, the gradient descent failed to achieve the optimal value in all simulations, as the gradient tends to 0 rapidly in the region of several minimum. However, the genetic algorithm improved its performance. Although there are many local minima, these are not separated by saddle points as the Styblinski-Tang function is more suitable for GA. Again, the SMT-based optimization method produced a satisfactory performance. In addition to that function, the execution time fell sharply, it occurs because the minimum values are integers, as Himmelblau 2. Consequently, the solution is found in the first loop and it is only confirmed when the accuracy is increased. Fig. 8c shows the trajectories of solutions absolute error of the three algorithms for the Goldstein-Price's function. Similarly to the previous function, only SMT shows a correct convergence, GD and GA present similar absolute errors after the convergence.

These experimental results show that the SMT-based optimization algorithm is extremely useful since it is able to find the optimal solution for any class of function. Its performance does not depend on initial values or function shape, and the correct answer is found in all benchmarks. However, the

TABLE I: Non-convex Optimization Benchmarks

Function	Equation	Domain	Optimum Point
Himmelblau 1	$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$	$-7 \leq x_1 \leq 0$ $0 \leq x_2 \leq 7$	$x^* = (-2.805, 3.131)$ $f(x^*) = 0$
Himmelblau 2	$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$	$-7 \leq x_1 \leq 7$ $-7 \leq x_2 \leq 7$	$x^* = (3, 2)$ $x^* = (-2.805, 3.131)$ $x^* = (-3.779, -3.283)$ $x^* = (3.584, -1.848)$ $f(x^*) = 0$
Styblinski-Tang	$f(x_1, ..., x_i) = \frac{1}{2}\sum_{i=1}^{d} x_i^4 - 16x_i^2 + 5x_i$	$-5 \leq x_i \leq 5$ $i = 1, 2$ $d = 2$	$x^* = (-2.903, -2.903)$ $f(x^*) = -78.332$
Goldstein-Price	$f(x_1, x_2) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$ x$[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	$-2 \leq x_i \leq 2$ $i = 1, 2$	$x^* = (0, -1)$ $f(x^*) = 3$



(a) Himmelblau's Function (b) Styblinski-Tang's Function (c) Goldstein-Price's Function

Fig. 8: Absolute error of optimization process per iteration for (a) Himmelblau's Function, (b) Styblinski-Tang's Function, and (c) Goldstein-Price's Function.

TABLE II: Experimental results for Gradient Descent, Genetic Algorithm, and SMT-based Optimization.

Function	Method	Correct Answer (%)	Execution Time (s)
Himmelblau 1	Gradient Descent	55	< 1
	Genetic Algorithm	100	< 1
	SMT-based Optimization	100	1622
Himmelblau 2	Gradient Descent	100	< 1
	Genetic Algorithm	100	< 1
	SMT-based Optimization	100	4
Styblinski-Tang	Gradient Descent	21	< 1
	Genetic Algorithm	9	< 1
	SMT-based Optimization	100	1045
Goldstein-Price	Gradient Descent	0	1
	Genetic Algorithm	69	< 1
	SMT-based Optimization	100	14

execution time takes longer than other algorithms since the present algorithm does an exhaustive search in the state space. This effect is reduced by the previously reported discretization of the state-space; otherwise, the number of iterations is less than all other algorithms. Performance problems can be solved with further abstraction of the state space (*e.g.*, intervals analysis) [33], or restricting it by imposing new restrictions on the variables, changing the intervals at each iteration.

V. Related Work

SAT/SMT solvers have been widely applied to solve different optimization problems, *e.g.*, minimize errors in the linear fixed-point arithmetic computations in embedded control software [16], reduce the number of gates in FPGA digital circuits [12], and partition hardware/software components in embedded systems by means of a Boolean domain to decide the most efficient system implementation given a set of design's metrics [13]–[15].

Recently, νZ [11] extends the SMT solver Z3 [34] for linear optimization problems; Sebastiani and Trentin [10] present OptiMathSat, which is an optimization tool that extends MathSAT5 SMT solver to allow one to solve linear functions in the Boolean, rational, and integer domains or a combination of them; in Sebastiani and Tomasi [18], the authors used a combination of SMT and LP techniques to minimize rational functions; the related study in [17] extends their work with linear arithmetic on the mixed integer/rational domain, thus combining SMT, LP, and ILP techniques.

As an application example, Pavlinovic *et al.* [35] propose an approach which considers all possible compiler error sources for statically typed functional programming languages and reports the most useful one subject to some usefulness criterion. The authors formulate this approach as an optimization problem related to SMT and use νZ to compute an optimal error source in a given ill-typed program. The approach described by Pavlinovic *et al.*, which uses MaxSMT solver νZ, shows a significant performance improvement if compared to previous SMT encodings and localization algorithms.

All previous related studies with SMT-based optimization can solve linear problems over integer, rational, or Boolean domains. In contrast, this paper introduces a new SMT-based

optimization method to minimize functions, linear or non-linear, convex or non-convex, continuous or discontinuous with fixed-point solutions. As a result, the present method is able to solve optimization problems directly on the rational domain with adjustable precision, without using any other technique to assist the manipulation of the state-space.

VI. Conclusions

This paper presented a novel method for optimizing functions using SMT-based verification techniques. In particular, the proposed optimization algorithm, which relies on recursive executions that re-constrains a model-checking procedure based on SMT, is described and evaluated using standard non-convex optimization benchmarks. Experimental results show that the proposed method ensures the optimal solution, although it takes longer than other traditional mathematical methods (*e.g.*, decent gradient) or those based on heuristics (*e.g.*, genetic algorithms). The presented method is complete, and provides an improved accuracy compared to other existing techniques. Additionally, the present approach is suitable for every class of function, differently from previous SMT/SAT-based optimization techniques.

However, the verification time of the present approach could be reduced with appropriate problem constraints and with the use of state-of-art verification techniques, *e.g.*, abstraction interpretation [33]. Thus, this paper represents an important advance in SMT/SAT optimization techniques, paving the way for several future improvements. Future studies include the application of the present approach to autonomous vehicles navigation systems and enhancements in the model-checking procedure for reducing the verification time by means of multi-core verification [13] and invariant generation [33], [36], in addition to compare the present approach to other optimization methods [37], [38].

Acknowledgment

The authors thank Nikolaj Bjorner for reviewing a draft version of this paper.

References

[1] K. Deb, *Optimization for Engineering Design: Algorithms and Examples*. Prentice-Hall of India, 2004.

[2] R. Garfinkel and G. Nemhauser, *Integer programming*, ser. Series in decision and control. Wiley, 1972.

[3] M. Bartholomew-Biggs, *The Steepest Descent Method*. Boston, MA: Springer US, 2008, pp. 1–8.

[4] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, ser. Artificial Intelligence. Addison-Wesley Publishing Company, 1989.

[5] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg, "Data and memory optimization techniques for embedded systems," *ACM TODAES*, vol. 6, no. 2, pp. 149–206, Apr. 2001.

[6] L. Adouane, *Autonomous Vehicle Navigation: From Behavioral to Hybrid Multi-Controller Architectures*. CRC Press, 2016.

[7] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Computat. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov 2006.

[8] A. Biere, "Bounded model checking," in *Handbook of Satisfiability*. IOS Press, 2009, pp. 457–481.

[9] P. Manolios and V. Papavasileiou, "ILP modulo theories," in *CAV*. Springer Berlin Heidelberg, 2013, pp. 662–677.

[10] R. Sebastiani and P. Trentin, "OptiMathSAT: A tool for optimization modulo theories," in *CAV*. Springer International Publishing, 2015, pp. 447–454.

[11] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νZ - an optimizing SMT solver," in *TACAS*. Springer Berlin Heidelberg, 2015, pp. 194–199.

[12] G. G. Estrada, "A note on designing logical circuits using SAT," in *ICES*. Springer Berlin Heidelberg, 2003, pp. 410–421.

[13] A. Trindade, H. Ismail, and L. Cordeiro, "Applying multi-core model checking to hardware-software partitioning in embedded systems," in *SBESC*, 2015, pp. 102–105.

[14] A. Trindade and L. Cordeiro, "Aplicando verificação de modelos para o particionamento de hardware/software," in *SBESC*, 2014, p. 6. [Online]. Available: http://sbesc.lisha.ufsc.br/sbesc2014/dl185

[15] ——, "Applying SMT-based verification to hardware/software partitioning in embedded systems," *DES AUTOM EMBED SYST*, vol. 20, no. 1, pp. 1–19, 2016.

[16] H. Eldib and C. Wang, "An SMT based method for optimizing arithmetic computations in embedded software code," *IEEE CAD*, vol. 33, no. 11, pp. 1611–1622, 2014.

[17] R. Sebastiani and P. Trentin, "Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions," in *TACAS*. Springer Berlin Heidelberg, 2015, pp. 335–349.

[18] R. Sebastiani and S. Tomasi, "Optimization modulo theories with linear rational costs," *ACM TOCL*, vol. 16, no. 2, pp. 12:1–12:43, Feb. 2015.

[19] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[20] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[21] D. Beyer, *Reliable and Reproducible Competition Results with BenchExec and Witnesses (Report on SV-COMP 2016)*. Springer Berlin Heidelberg, 2016, pp. 887–904.

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

[23] L. Cordeiro, B. Fischer, and J. Marques-Silva, "SMT-based bounded model checking for embedded ANSI-C software," *IEEE TSE*, vol. 38, no. 4, pp. 957–974, 2012.

[24] D. Kroening and M. Tautschnig, "CBMC – c bounded model checker," pp. 389–391, 2014.

[25] J. Morse, M. Ramalho, L. Cordeiro, D. Nicole, and B. Fischer, "ESBMC 1.22 - (competition contribution)," in *TACAS*, 2014, pp. 405–407.

[26] B. R. Abreu, Y. M. R. Gadelha, C. L. Cordeiro, B. E. de Lima Filho, and S. W. da Silva, "Bounded model checking for fixed-point digital filters," *JBCS*, vol. 22, no. 1, pp. 1–20, 2016.

[27] I. V. Bessa, H. I. Ismail, L. C. Cordeiro, and J. E. C. Filho, "Verification of fixed-point digital controllers using direct and delta forms realizations," *DES AUTOM EMBED SYST*, vol. 20, no. 2, pp. 95–126, 2016.

[28] I. Bessa, H. Ibrahim, L. Cordeiro, and J. E. Filho, "Verification of Delta Form Realization in Fixed-Point Digital Controllers Using Bounded Model Checking," in *SBESC*, 2014.

[29] D. Beyer and M. E. Keremoglu, "CPAchecker: A tool for configurable software verification," in *CAV*. Springer Berlin Heidelberg, 2011, pp. 184–190.

[30] M. Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimisation problems," *IJMMNO*, vol. 4, no. 2, pp. 150–194, 2013.

[31] R. Brummayer and A. Biere, "Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays," in *TACAS*, 2009, pp. 174–177.

[32] *Matlab Optimization Toolbox User's Guide*, The Mathworks, Inc., 2016.

[33] H. Rocha, H. Ismail, L. Cordeiro, and R. Barreto, "Model checking embedded c software using k-induction and invariants," in *SBESC*, Nov 2015, pp. 90–95.

[34] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *TACAS*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340.

[35] Z. Pavlinovic, T. King, and T. Wies, "Practical SMT-based type error localization," in *ICFP*, 2015, pp. 412–423.

[36] M. Y. R. Gadelha, H. I. Ismail, and L. C. Cordeiro, "Handling loops in bounded model checking of c programs via k-induction," *STTT*, pp. 1–18, 2015.

[37] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, 1995, pp. 1942–1948.

[38] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B*, vol. 26, no. 1, pp. 29–41, 1996.