Efficient Local Search for Nonlinear Real Arithmetic

Zhonghan Wang[1,2], Bohua Zhan[1,2](✉), Bohan Li[1,2], and Shaowei Cai[1,2]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
{wangzh,bzhan,libh,caisw}@ios.ac.cn

[2] University of Chinese Academy of Sciences, Beijing, China

Abstract. Local search has recently been applied to SMT problems over various arithmetic theories. Among these, nonlinear real arithmetic poses special challenges due to its uncountable solution space and potential need to solve higher-degree polynomials. As a consequence, existing work on local search only considered fragments of the theory. In this work, we analyze the difficulties and propose ways to address them, resulting in an efficient search algorithm that covers the full theory of nonlinear real arithmetic. In particular, we present two algorithmic improvements: incremental computation of variable scores and temporary relaxation of equality constraints. We also discuss choice of candidate moves and a look-ahead mechanism in case when no critical moves are available. The resulting implementation is competitive on satisfiable problem instances against complete methods such as MCSAT in existing SMT solvers.

Keywords: Local search · Nonlinear arithmetic · SMT

1 Introduction

Satisfiability Modulo Theories (SMT) is the problem of determining the satisfiability of a formula containing both logical operators and functions interpreted in one or more custom theories [6]. Commonly considered theories include equality, arithmetic, bit-vectors, arrays, and strings. After nearly two decades of development, SMT has gained widespread applications in program verification, model checking, planning, and many other areas.

The arithmetic theories can be divided according to the type of numbers involved into integer and real theories, and according to the operations allowed into difference logic, linear, and nonlinear theories. The case of (quantifier-free) nonlinear real arithmetic (NRA) considers satisfiability of equalities and inequalities involving polynomials of degree greater than one, and where the arithmetic variables take on real values. It has applications in the analysis of nonlinear hybrid automata [13], generating ranking functions for termination analysis [22,28], constraint answer set programming [41,42], and even analysis of biological networks [3]. Problem instances from many of these applications are collected in the SMT-LIB benchmarks [5].

R. Dimitrova et al. (Eds.): VMCAI 2024, LNCS 14499, pp. 326–349, 2024.
https://doi.org/10.1007/978-3-031-50524-9_15

Similar to the SAT case, methods for solving SMT problems can be roughly divided into *complete* and *incomplete* methods. Complete methods are usually based on DPLL(T) or close variants. They are able to both find solutions and prove unsatisfiability. Incomplete methods, such as those based on local search [24], explore the solution space heuristically, usually by changing the assignment of one variable at a time, in an attempt to find a satisfying solution. Local search methods are not able to prove unsatisfiability, but can have an advantage over complete methods on some satisfiable instances.

Complete methods for the theory of nonlinear real arithmetic make crucial use of cylindrical algebraic decomposition (CAD) [10], which permits deciding the satisfiability of a conjunction of polynomial (in)equalities over real numbers. This can then serve as the theory solver in DPLL(T) [39]. An innovation over DPLL(T) for arithmetic theories is the MCSAT algorithm [25,34], which constructs models involving both boolean and arithmetic variables at the same time. A nice overview of DPLL(T) and MCSAT for nonlinear real arithmetic can be found in Kremer's thesis [27].

Exploration of applying local search to solve SMT problems over arithmetic theories began only recently. The work [7] applied local search to the theory of linear integer arithmetic. It introduced the concept of *critical moves*, a change in one arithmetic variable that satisfies a previously unsatisfied clause. The algorithm iteratively applies critical moves that most improves the score, a weighted count of unsatisfied clauses. The presence of both boolean and arithmetic variables are dealt with by alternately working in the *integer mode* and the *boolean mode*, when assignments to only integer or boolean variables are changed respectively. Switching between modes are performed after the number of non-improving steps reaches a certain threshold.

Compared to linear integer arithmetic, the problem of nonlinear real arithmetic held additional challenges for local search methods. Unlike integer theories, there is an uncountable number of possible assignments to choose from, including infinite number of choices in any finite interval. Unlike the linear case, there is potential need to solve polynomials of degree greater than one, which is both costly in time and may result in variable assignments that are irrational (i.e. algebraic) numbers, causing a slow-down of the ensuing search process. It is also possible that a nonlinear constraint cannot be satisfied by changing the value of one variable alone, resulting in the lack of critical moves, so that other heuristics are needed in such scenarios.

There has already been some work exploring local search for nonlinear real arithmetic. However, largely due to the challenges listed above, none of the existing work covers the entirety of the theory. The work [29] considers the multilinear case, where each variable appears with degree at most one in each polynomial constraint. The work [31] considers problems where all equality constraints contain at least one variable that is linear. In both of these works, the problem of variable assignments to irrational values is avoided by either assuming linearity in each variable, or by limiting higher-degree constraints to strict inequalities only.

1.1 Contributions

In this paper, we propose several improvements to the local search procedure, aimed at addressing the challenges posed by nonlinear real arithmetic. This results in an efficient local search algorithm for the entire NRA theory.

First, we present efficient data structures for caching and updating variable scores used to determine the next critical move. Existing work on local search for arithmetic theories can be thought of as an extension of the GSAT algorithm [40] with adaptive weighting. It is well-known that efficient implementations of GSAT involve caching and updating of variable scores [24, Section 6.2]. For arithmetic theories, this is complicated by the fact that each variable is associated not with one score, but with different scores for changing its assignment to values in different intervals. Hence, current implementations of local search for arithmetic theories recompute the score information for each variable at every iteration, resulting in potential repeated computations. This is especially serious for nonlinear arithmetic, where computations may involve costly root-finding for higher-degree polynomials. We describe data structures maintaining *boundaries* of score changes for each pair of clause and variable appearing in the clause, which only need to be updated on an as-needed basis, and from which the full score information can be quickly recovered.

Second, we address the problem posed by nonlinear equality constraints between variables, which may force assignment to variables that are irrational numbers. Rather than making such assignments directly, we propose to temporarily *relax* such equalities into inequalities, e.g. changing the constraint $p = 0$ into $p > -\epsilon \wedge p < \epsilon$, and continue the local search process. If an (approximate) solution is found that satisfies the relaxed version of these constraints, we restore the equalities to their original form, and try to find an exact solution near the approximate solution. For this step, we try two different heuristic methods, respectively based on analyzing the structure of equations and local search itself. While neither method is guaranteed theoretically (or in practice) to find an exact nearby solution every time, the use of relaxation means local search mainly works with rational assignments, significantly improving its efficiency in those problem instances where irrational assignments may otherwise be needed.

Finally, we present alternative ways to deal with situations where no critical move is available to satisfy a certain literal (that is, when the literal is *stuck*). We first pick heuristically a variable appearing in the literal, and a set of candidate moves for that variable. For each candidate move, we look ahead to see whether that move will lead to the literal having critical moves on the next step. Such moves are then preferred over the others.

The above ideas are implemented as a local search algorithm on top of the Z3 prover [33]. The implementation relies on the existing library of polynomials and algebraic numbers in Z3, but is otherwise independent from its implementation of MCSAT for nonlinear real arithmetic. We perform thorough experiments on the SMT-LIB benchmarks, showing the effect of incremental computation of variable scores and relaxation of equalities, and that the resulting local search algorithm is competitive against complete algorithms in existing SMT solvers on the satisfiable instances. Especially, there is a large amount of complementarity between the problems solved by local search and by the complete algorithms, indicating major improvements can be obtained in a portfolio setting.

1.2 Related Work

Our work builds upon existing work applying local search to SMT over arithmetic theories [7,29,31]. They will be reviewed in more detail in Sect. 2. Besides arithmetic theories, there have also been earlier work applying local search to the theory of bit-vectors [16,21,37]. In particular, the work [16] generalized the scoring function to consider operators on bit-vectors, but the moves remain single-bit flips. Later works [37,38] introduced propagation-based move selection and essential inputs to prune the search.

The most commonly used framework for complete algorithms for nonlinear real arithmetic is MCSAT, first proposed by Jovanovic and de Moura [25,34]. It improves upon the use of CAD within the DPLL(T) framework by assigning to both arithmetic and boolean variables during the search process. Recent innovations in SMT solving for nonlinear arithmetic include variants to the application of CAD [1], and alternative heuristics for choosing variable ordering [30]. Besides complete methods based on CAD, alternative methods based on linearization [11], interval constraint propagation [26,44], and subtropical methods [15,35] are also explored. While these approaches alone do not achieve the same level of overall performance as CAD and MCSAT, they may have advantages in specific classes of problems, making them useful in a portfolio setting.

Also closely-related to our work are various methods for determining whether a given region contains a solution to a set of (in)equality constraints, and their applications to SMT solving. The work of Cimatti et al. [12] first uses global optimization methods to find an initial solution, then uses topological methods to determine whether an exact solution exists nearby. The work of Ni et al. [36] also uses optimization to find a candidate solution, followed by general methods for solving equations [32] to isolate an exact solution.

The work of Gao et al. [17,18] introduced the framework of δ-complete decision procedures, implemented in the dReal tool for solving SMT problems over nonlinear formulas [19]. It can handle polynomials as well as trigonometric and exponential functions. The δ-complete framework allows algorithms to either return δ-sat or unsat, where the δ-sat case returns a solution for a δ-weakening of the input formulas. This permits efficient numerical algorithms to be used, as well as showing decidability for a wide range of problems. The concept of relaxation of constraints in our work is similar to δ-weakening, and we also use it to increase efficiency of our algorithm. However, we still aim to return an exact answer, by restoring the constraints to their original form and try to find an exact solution near any approximate solution that is found.

1.3 Structure of the Paper

We begin by defining the SMT problem over nonlinear real arithmetic, and reviewing existing local search algorithms in Sect. 2. Section 3 presents incremental computation of variable scores. Section 4 presents relaxation and restoring of equality constraints. Section 5 discusses implementation choices, including heuristic move selection and look-ahead mechanism for stuck literals. Section 6 compares with existing SMT solvers, and performs ablation study on each of the proposed improvements. Finally, we conclude in Sect. 7 with a discussion of potential future directions.

2 Preliminaries

In this section, we formally define SMT problems over nonlinear real arithmetic, followed by a review of existing local search algorithms for arithmetic theories.

The syntax of a general SMT formula over nonlinear real arithmetic is as follows:

$$\begin{aligned}
p &:= x \mid c \mid p + p \mid p \cdot p && \text{(polynomials)}\\
a &:= b \mid p \geq 0 \mid p \leq 0 \mid p = 0 && \text{(atoms)}\\
f &:= a \mid \neg f \mid f \wedge f \mid f \vee f && \text{(formulas)}
\end{aligned}$$

Here x is an arithmetic variable, c is a constant rational value, b is a boolean variable. A *literal* is either an atom or its negation. A *clause* is a disjunction of literals. In other words, we consider (in)equalities on polynomials with rational coefficients[1]. In practice, we assume that input problem instances are given in conjunctive normal form (CNF), that is as a collection of clauses to be satisfied. Note that strict inequalities $p \neq 0$, $p < 0$ and $p > 0$ can be represented as $\neg(p = 0)$, $\neg(p \geq 0)$ and $\neg(p \leq 0)$, respectively. We allow problem instances to contain boolean and arithmetic variables at the same time. We define *boolean literal* and *arithmetic literal* to mean literal whose atom is a boolean variable and a polynomial inequality, respectively.

A polynomial p is *linear* in some variable x if all terms of the polynomial have degree at most one in x. Alternatively, p can be written in the form $p_1 \cdot x + p_2$, where p_1, p_2 do not contain x. A polynomial is *multilinear* if it is linear in each of its variables.

Given a problem instance containing boolean variables b_i $(1 \leq i \leq m)$ and arithmetic variables x_j $(1 \leq j \leq n)$, a *complete assignment* is a mapping from each b_i to $\{\top, \bot\}$ and each x_j to $\mathbb{R}$. We will only deal with complete assignments in this paper, and hence sometimes call it *assignment* for short. A formula is

[1] Our methods can be extended to handle polynomials with coefficients that are algebraic numbers. Alternatively, a coefficient c_i that is an algebraic number can be encoded as a variable x_i satisfying some polynomial $p(x_i) = 0$ together with interval constraints. Hence we limit the discussion to rational coefficients in this paper.

satisfied under an assignment if it evaluates to true under the standard interpretation of boolean and arithmetic operators. An assignment is a solution to a problem instance if it satisfies all its clauses. The SMT problem for nonlinear real arithmetic is to determine whether a given problem instance is satisfiable by some assignment.

Local search algorithms attempt to determine satisfiability of a problem instance by searching in the space of complete assignments, usually by changing the value of one variable at a time. In the SAT case, each move in local search flips the assignment of one boolean variable. Which flip to make is determined by factors such as the number of clauses that become satisfied/unsatisfied as result of the flip, weight of the clauses, which variables are flipped recently, and so on. For SMT over arithmetic theories, the analogous move changes the value of one arithmetic variable, usually in order to make some clause become satisfied. Such moves, called *critical moves*, are introduced in [7] for linear integer arithmetic.

For nonlinear real arithmetic, the basic procedure used to determine possible moves is *root isolation* for polynomials. Given a polynomial p in a single variable x (where here we allow the coefficients of p to be algebraic numbers), the procedure computes the roots of the equation $p(x) = 0$, together with the sign of the polynomial in each interval separated by the roots. Algebraic numbers in the coefficients of p and in the output of root isolation are represented by their minimal polynomials, together with intervals with rational endpoints that bracket a root of that polynomial.

Given a complete assignment, and an arithmetic literal l involving variable x, we can use root isolation to compute the set of values that assignment of x may be moved to in order for l to be satisfied. This is done by substituting in assignments to other variables in l, resulting in a polynomial containing only variable x, then perform root isolation and check the value of l in each resulting interval. The answer is given in terms of a set of intervals (which may contain $\pm\infty$ as one of the endpoints, and may be either open or closed at each endpoint). We state the definitions precisely as follows.

Definition 1. *Given a complete assignment, a literal l and a variable x, the* feasible set *(resp.* infeasible set*) is the collection of intervals that the value of x can be moved to in order for l to be satisfied (resp. unsatisfied). Likewise, we define the feasible set (resp. infeasible set) of a clause with respect to a variable. This is computed by taking the union (resp. intersection) of the feasible set (resp. infeasible set) for each literal in the clause.*

A critical move is defined to be a change in the assignment of some variable x to a value that satisfies some previously unsatisfied clause. The basic local search algorithm then performs critical moves at each iteration, using various scoring metrics to determine which move is chosen next. Such moves can also be interpreted as jumping between CAD cells as in [31]. An innovation in [31] is that when no critical moves are available for some literal, moves that change multiple variables at once along some straight line are also explored.

Scoring of critical moves is usually based on a weighted count of unsatisfied clauses. *Adaptive weighting schemes* assign a weight to each clause, reflecting its importance during the current search. Existing work on local search for arithmetic theories mostly use a probabilistic version of the PAWS weighting scheme [43]. This scheme is parameterized by a smoothing probability sp. Whenever there is no moves available that improves the score, with probability $1 - sp$ the weight of each unsatisfied clause is increased by 1, and with probability sp the weight of each satisfied clause with weight greater than 1 is decreased by 1. Then, the *make-break score* of each critical move equals the total weight of clauses that become satisfied by the move, minus the total weight of clauses that become unsatisfied by the move. A move is *improving* if its score is greater than zero.

One key contribution of [29] is the introduction of *make-break intervals*. The idea is that instead of considering only the (in)feasible intervals of a variable x with respect to some clause, we combine the (in)feasible interval information of x with respect to all clauses. This results in a partition of the real line into intervals, with each interval associated to the make-break score for moving the value of x into that interval. We illustrate this idea with the following example.

Example 1. Consider the set of clauses $x^2 + y^2 \leq 1$, $x + y < 1$ and $x + z > 0$. The current assignment is $x \mapsto 1, y \mapsto 1, z \mapsto 1$, and the current weight of clauses are $1, 3, 2$, respectively. The make-break score for variable x with respect to each of the clauses are:

- $x^2 + y^2 \leq 1$ (unsatisfied): $(-\infty, 0) \mapsto 0$, $[0, 0] \mapsto 1$, $(0, \infty) \mapsto 0$.
- $x + y < 1$ (unsatisfied): $(-\infty, 0) \mapsto 3$, $[0, \infty) \mapsto 0$.
- $x + z > 0$ (satisfied): $(-\infty, -1] \mapsto -2$, $(-1, \infty) \mapsto 0$.

Combining the above information, we obtain the following make-break intervals and scores for x: $(-\infty, -1] \mapsto 1$, $(-1, 0) \mapsto 3$, $[0, 0] \mapsto 1$, $(0, \infty) \mapsto 0$. A preferred move would be to change the value of x into the interval $(-1, 0)$, satisfying the clause $x + y < 1$ and leaving the status of the other clauses unchanged, with a make-break score of 3.

If boolean variables are present, the make-break score of flipping each boolean variable is defined in a similar way, as the total weight of clauses that become satisfied by the flip, minus the total weight of clauses that become unsatisfied.

Algorithm 1 shows the structure of the basic local search procedure. Begin by initializing the assignments to all variables (line 1). At each iteration, first try to find a move with the largest make-break score. If the score is greater than 0 (the variable necessarily comes from an unsatisfied clause), then perform the corresponding move (line 9). If no move has score greater than 0, it indicates that we reached a local minimum. Update the clause weights according to the PAWS scheme (line 11), and then try to make a move that makes a randomly chosen clause satisfied (line 16). If that is also not possible after several tries, randomly change the assignment of some variable in some unsatisfied clause according to some heuristic (line 19). This continues until all clauses are satisfied (line 4) or the time or step limit is reached (line 6).

There are possible variations in the choice between boolean and arithmetic variables on line 7. In [7, 29], the search is separated into modes where only

Algorithm 1: Basic local search algorithm

```
Input  : A set of clauses $F$
Output: An assignment of variables that satisfy $F$, or failure
1  Initialize assignment to variables;
2  while $\top$ do
3      if all clauses satisfied then
4          return success with assignment;
5      if time or step limit reached then
6          return failure;
7      $var, new\_value, score \leftarrow$ best move according to make-break score;
8      if $score > 0$ then
9          Perform move, assigning $var$ to $new\_value$;
10     else
11         Update clause weight according to PAWS scheme;
12         repeat
13             $cls \leftarrow$ random unsatisfied clause;
14             $var, new\_value, score \leftarrow$ critical move making $cls$ satisfied;
15             if $score \neq -\infty$ then
16                 Perform move, assigning $var$ to $new\_value$;
17         until 3 times;
18         if no move performed in previous loop then
19             Change assignment of some variable in some unsatisfied clause;
```

boolean or arithmetic variables are considered. Alternatively, we can combine the lists of moves and decide between them based purely on make-break scores. We take the latter approach in this paper. There are other aspects of the algorithm that are left unspecified, including how the best move is computed on line 7, and the heuristic choice of moves on line 19. These will be specified in more detail in the next sections.

3 Incremental Computation of Variable Scores

One key step in Algorithm 1 is computing the move with the best make-break score. The computation for boolean variables is standard (and is in any case not the bottleneck here), hence we focus on critical moves for arithmetic variables. The default approach is to loop over all variables in all unsatisfied clauses. For each variable, compute its score with respect to each clause and then combine the results (as demonstrated in Example 1). However, it is clear that this may result in repeated computations across iterations. For example, the feasible set of some variable with respect to a clause may be recomputed, even if none of the variables in that clause are changed in the previous step. Following the idea of caching and updating scores in GSAT [40], we propose data structures for caching and updating score information for arithmetic variables.

We define a *boundary* to be a quadruple $\langle val, is\_open, is\_make, cid\rangle$, where *val* is a real number, *is_open* and *is_make* are boolean values, and *cid* is a clause identifier. The boundary indicates that there is a change in make-break score when moving from less than to greater than *val* due to clause *cid*. If *is_make* is $\top$, the score is increased by the weight of clause *cid*, otherwise it is decreased by the weight. If *is_open* is $\top$, the change is not active at *val*, otherwise it is already active at *val*. There is a natural ordering among boundaries, first order by *val* and then by *is_open* (with $\bot < \top$). The make-break score information of each variable with respect to each clause can be characterized by a starting score (indicating the make-break score of large negative values), together with a set of boundaries. The make-break information of each variable with respect to all clauses is formed by summing the starting score and taking the union of the sets of boundaries. We illustrate the computations in the following example.

Example 2. Continuing from Example 1, the starting score and boundary information of variable x with respect to each clause is as follows (we identify the three clauses as 1, 2, 3, respectively).

- $x^2 + y^2 \leq 1$: starting score 0, boundary set $\{(0, \bot, \top, 1), (0, \top, \bot, 1)\}$, indicating no change for large negative values, *make* at boundary $[0, \cdots$, followed by *break* at boundary $(0, \cdots$.
- $x + y < 1$: starting score 3, boundary set $\{(0, \bot, \bot, 2)\}$, indicating *make* at large negative values, and *break* at boundary $[0, \ldots$.
- $x + z > 0$: starting score -2, boundary set $\{(-1, \top, \top, 3)\}$, indicating *break* at large negative values, and *make* at boundary $(-1, \ldots$.

The combined make-break score information is: starting score 1, with the following (ordered) set of boundaries: $\{(-1, \top, \top, 3), (0, \bot, \top, 1), (0, \bot, \bot, 2), (0, \top, \bot, 1)\}$. Make-break score information in terms of intervals can be easily recovered from the above, by traversing the boundaries in order, increasing the score by the weight of the clause when encountering a boundary with $is\_make = \top$, and decreasing the score by the weight otherwise.

During local search, after each move of variable v to a new value, only those variables v' that share a clause with v need to have their make-break score information updated (this is analogous to the concept of dependent variables in the SAT case), and moreover boundary information need to be updated for the shared clauses only. This is summarized in Algorithm 2. The set S collects the set of variables sharing a clause with v. Line 5 recomputes starting score and boundary information. Line 7 recomputes best critical move and score for each updated variable.

Example 3. Continuing from Example 2, suppose the move $y \mapsto -2$ is made, making the clause $x + y < 1$ satisfied. Then the score information for variable z does not need to be updated, as y and z do not share a clause. The score information for variable x need to be updated for the first two clauses. For clause $x^2 + y^2 \leq 1$ there is no longer any boundaries (no assignment of x can

Algorithm 2: Incremental computation of make-break scores

```
    Input   : Variable $v$ that is modified
    Update: Make-break score for all variables
1   $S \leftarrow \{\}$ ;                                    // set of updated variables
2   for clause cls that contains $v$ do
3       for variable $v'$ appearing in cls do
4           add $v'$ to $S$;
5           recompute starting score and boundary of $v'$ with respect to cls;
6   for variable $v'$ in $S$ do
7       recompute best critical move and score in terms of boundary information;
```

make the clause true), and for clause $x + y < 1$ the new starting score and boundary set are 0 and $\{(3, \bot, \bot, 2)\}$, respectively. So the overall starting score and boundary set are -2 and $\{(-1, \top, \top, 3), (3, \bot, \bot, 2)\}$.

Remark 1. Data structures such as arrays, linked lists, or binary trees can be used to maintain set of boundaries. If the total number of boundaries for each variable is small (as is the case for most of the problem instances in SMT-LIB), arrays or linked lists are sufficient. Otherwise the use of binary trees result in better asymptotic performance for the required operations.

Remark 2. A further optimization can be made: it is not necessary to immediately recompute the boundary information for a variable v' that does not appear in any unsatisfied clause, as such variables will never be chosen either on line 7 or line 14 of Algorithm 1. Instead, flags can be used to mark that boundary information for certain clauses need to be updated for v'. When at least one of the clauses containing v' becomes unsatisfied, information for those flagged clauses (as well as other clauses that need to be updated for that step) are updated.

4 Relaxation of Equalities

Equality constraints with degree greater than one pose special difficulty for local search, since it may force assignments of variables to irrational (e.g. algebraic) numbers. For example, for the constraint $x^2 + y^2 + z^2 = 1$, with most rational assignments to x and y, the assignment to z would be forced to be irrational in order for the constraint to be satisfied. While it is possible to represent and compute with algebraic numbers during local search, the time-cost of such computation is significantly increased. Even without considering algebraic numbers, numbers with increasingly large denominators are also problematic for slowing down the search process.

Both [29,31] avoid algebraic numbers by either limiting themselves to the multilinear case, or considering only equality constraints with at least one linear variable (and solving only for those linear variables). The work [29] further incorporated comparison of size of denominators (as well as absolute value) of potential assignments into the scoring heuristic, in order to keep the complexity of assigned values as low as possible.

We propose a novel approach to address the problem of assignments to irrational values and values with large denominators, that allow the algorithm to be applied efficiently to the full set of nonlinear arithmetic problem instances. The approach still relies on comparing complexity of assigned values, hence we first define it below.

Definition 2 (Complexity of values). *We define a preorder $\prec_c$ on algebraic numbers as follows. $x \prec_c y$ if x is rational and y is irrational, or if both x and y are rational numbers, and the denominator of x is less than that of y. We write $x \sim_c y$ if neither $x \prec_c y$ nor $y \prec_c x$.*

The relaxation mechanism can be described simply as follows: whenever some equality (or inequality) constraints force an assignment of some variable to a comparatively complex value, those constraints are relaxed before continuing the local search process, so that such assignments never actually occur. The implementation is parameterized by two thresholds. The parameter ϵ_v (for *variable* threshold) specifies the complexity of assigned values (according to Definition 2) beyond which relaxation of constraints should be applied. The parameter ϵ_p (for *polynomial* threshold) specifies the amount of relaxation of polynomial constraints. Both ϵ_v and ϵ_p are chosen to be 10^{-4} in the implementation.

It should be noted that the constraints $p \geq 0$ and $p \leq 0$ together can also force the assignment of variables to irrational values. These constraints may appear as part of clauses with more than one literal, and hence are not equivalent to $p = 0$. This means in general we consider relaxation of non-strict inequalities, although we will still use the slightly imprecise (but more intuitive) description of relaxing equalities throughout the paper.

The detailed method for determining which constraints to relax is as follows. When computing the best make-break score of a variable v, if that score comes from a one-point interval, the set of clause identifiers in the boundaries contributing to that interval are recorded. If the variable v is chosen, and the new value of v is more complex than both ϵ_v and any other previously assigned value (according to Definition 2), all equalities and non-strict inequalities in the recorded clauses that contribute to the boundary are relaxed. The result of relaxation is as follows.

– If the constraint is of the form $p = 0$, it is relaxed into the pair of inequalities $p < \epsilon_p$ and $p > -\epsilon_p$.
– If the constraint is of the form $p \geq 0$, it is relaxed into $p > -\epsilon_p$. Likewise, if the constraint is of the form $p \leq 0$, it is relaxed into $p < \epsilon_p$.

Note that strict inequality constraints cannot force a variable to a particular value. After relaxation, the local search process proceeds as before, but with all evaluation of literals and computation of make-break scores according to the relaxed interpretation of literals.

After local search finds a "solution" under the relaxation of some constraints, it is only an *approximate* solution. In fact, there is no guarantee that there is an exact solution nearby. We currently try two different ways to find an exact solution near the approximate solution.

The first method performs a heuristic analysis of the structure of the relaxed constraints, in an attempt to find an order of solving for the variables that is likely to produce variable assignments that satisfies all of the equations. Essentially, we are trying to find an exact solution to a set of equality constraints, nearby an existing approximate solution. The analysis performs the following steps:

1. If any variable is currently assigned to zero, this is substituted into the constraints. We found this very helpful in practice for eliminating many terms in the constraints.
2. Eliminate any variable x in an equation of the form $p \cdot x + q = 0$, where the valuation of p under the current assignment is not close to zero. This is more general than eliminating variables during preprocessing, which requires p to be constant (see Sect. 5.2).
3. Finally, we iteratively look for a variable that appear only in one of the equations. We associate this variable to the corresponding equation, and then remove the equation from consideration in the ensuing iterations.

If no equations remain at the end of Step 3, we attempt to find an exact solution to the equality constraints by solving for the variables in reverse order of the above process. We begin by considering the association between variables and equations in Step 3 in reverse order, solving for each variable using the associated equation. Then we obtain the values of solved variables in Step 2 in reverse order. The final exact solution to these equations is checked again for satisfaction of all clauses (various numerical inaccuracies may prevent it from being so). If there are unsolved equations remaining after Step 3 of the above analysis, or if the resulting solution fails to satisfy all clauses, we move to the second approach.

The second approach uses a simplified version of local search itself to try to move the approximate solution to an exact solution. First, restore the relaxed constraints to their original forms, and then proceed with local search on the arithmetic variables only, until either an exact solution is found, or no improvement can be made. This is a more limited form of the general local search, as we do not attempt changes to boolean variables, or try random moves in case no improvements can be made. Hence it is likely to terminate quickly with either an exact solution or report of failure.

The above description is summarized as a modification of the overall algorithm, shown in Algorithm 3. The main change to the search process is to relax constraints whenever it forces an assignment to values that are more complex than the variable threshold (line 18). If all clauses are satisfied (indicating an approximate solution is found), we first try finding an exact solution nearby by analyzing the structure of relaxed constraints (line 4). If this fails, we try the limited form of local search described above (line 8–9). If this also fails, the search is restarted with a fresh assignment (line 13).

In practice, we find the two heuristic approaches for finding exact solutions to be useful in different scenarios. The first approach deals nicely with cases where the structure of equality constraints involves solving systems of linear equations, but otherwise poses no difficult choices. The second approach can better handle those cases where there may be choices in which variables to modify, but only some of them is correct in order to satisfy the other inequality constraints. It may also be possible to apply more advanced methods for solving equations or determining existence of solutions in [12,32]. However, the methods we implement easily extends to the non-zero-dimensional case, and returns exact solutions that can be verified independently. We leave the incorporation of more advanced methods for solving equations to future work.

Algorithm 3: Relaxation of equalities

```
   Input  : A set of clauses $F$
   Output: An assignment of variables that satisfy $F$, or failure
1  Initialize assignment to variables;
2  while $\top$ do
3      if all clauses satisfied then
4          success ← find exact solution by analyzing structure of equations;
5          if success then
6              return success with assignment;
7          else
8              Restore relaxed constraints to original form;
9              success ← find exact solution by limited local search;
10             if success then
11                 return success with assignment;
12             else
13                 Perform major restart;
14     if time or step limit reached then
15         return failure;
16     if no improvement for $T_1$ steps then
17         Perform minor restart;
18     Proceed as in line 7-19 of Algorithm 1, except constraints may be relaxed;
```

5 Implementation

In this section, we describe the implementation in more detail. First, we explain our choice of heuristic move selection when encountering a literal without critical moves. Then, we describe some further details on preprocessing, restart mechanism, and other efficiency improvements.

5.1 Heuristic Moves Selection and Look-Ahead

One major difficulty for local search in nonlinear arithmetic is that it is not always possible to find single-variable moves to satisfy a particular constraint. For example, given constraint $x^2 + y^2 < 1$, and the current assignment $x \mapsto 2, y \mapsto 3$, it is not possible to satisfy the constraint by moving only one of x and y. During local search, this is reflected by the situation that no critical move is available for a clause or literal.

Solving this problem in general would likely require complex algorithms such as CAD or polynomial optimization. Indeed, one category in the SMT-LIB benchmarks, Sturm-MBO, coming from analysis of biological networks [3], consists exclusively of problems that require a very complex polynomial to evaluate to zero, subject to positivity constraints of the variables. When the problem has many variables (is high-dimensional), any approach based on heuristically searching for assignments would have difficulty finding the exact combination of assigned values required to satisfy the constraint.

One approach is given in [31], which involves searching in directions other than those parallel to the coordinate axes to look for solutions. The use of gradient information, as well as scoring based on values of polynomials, increase the chance of finding a solution.

In this paper, we propose another approach that still involves moving only one variable at a time. We say a literal is *stuck* if it is currently unsatisfied and has no critical moves to make it satisfied. Given a literal l that is stuck, we first choose a variable x in l whose coefficient is nonzero (according to the current assignment of the other variables), then heuristically pick a set of candidate values to move the assignment of x to. For each candidate value, we compute whether l is still stuck after making that move. We then prefer those moves that result in l no longer being stuck.

Given the current assignment x_0 and the feasible set I of variable x (see Sect. 5.2), the heuristic move selection include the following:

1. rational numbers and integers close to the boundary inside each interval of I. The rational numbers are chosen to be within 10^{-4} of the boundary.
2. the next integer smaller or larger than x_0.
3. three numbers chosen uniformly in the interval $[\frac{x_0}{2}, x_0)$, and three numbers chosen uniformly in the interval $(x_0, 2x_0]$.

The first class reflects what we know about the constraints on x. The second class attempts basic random walk, and prefers (simple) integer values. The third class is the most general, allowing search over large/small values as well as fractions.

The above ideas are summarized in Algorithm 4. The heuristic choice of candidate values are collected into set S (line 2). Then each value in S is tested in turn. If l has critical move after assigning to any value, that value is returned (line 5). Otherwise a randomly chosen value from S is returned (line 7).

Algorithm 4: Heuristic choice of candidate values and look-ahead for critical moves

```
Input  : Literal $l$ without critical moves
Output: Candidate variable $x$ and new value $x_1$
1 $x \leftarrow$ variable in polynomial of $l$ with nonzero coefficient;
2 $S \leftarrow$ heuristic move selection for variable $x$;
3 for value $x_1$ in $S$ do
4     if $l$ has critical move after assigning $x$ to $x_1$ then
5         return $x$, $x_1$
6 $x_1 \leftarrow$ randomly chosen value in $S$;
7 return $x$, $x_1$
```

5.2 Implementation Details

The algorithm is implemented on top of the Z3 prover, and makes use of its library for polynomials and algebraic numbers, as well as data structures for clauses and literals, but otherwise separate from its implementation of the MCSAT algorithm.

Preprocessing. The following preprocessing steps are used. Eliminate clauses with a single boolean variable and propagate assignments. Combine constraints $p \geq 0$ and $p \leq 0$ into equality $p = 0$ (when they appear as clauses on their own; literals $p \geq 0$ and $p \leq 0$ that are parts of larger clauses cannot be combined). Eliminate variable x in an equation of the form $c \cdot x + q = 0$, where c is a constant and q is a polynomial with degree at most 1 and containing at most 2 variables. The conditions on q are designed so that preprocessing does not significantly increase the complexity of the remaining clauses.

Restart Mechanism. We use a two-level restart mechanism with two parameters T_1 and T_2 (both chosen to be 100 in our implementation). Perform a *minor restart* after T_1 moves without improvements, which randomly changes one of the variables in one of the unsatisfied clauses. After T_2 such minor restarts, a *major restart* is performed that resets the value of all variables.

Shortcut for Linear Equations. Root-isolation is done by calling the existing implementation in Z3, except when the variable to be solved is linear in the polynomial, in which case a direct (and more efficient) solution method is used.

Infeasible Sets of Variables. For each clause involving a single variable, derive infeasible set for that variable implied by the clause. Experience shows that excluding assignments from the infeasible set during local search is beneficial for some problem instances but not others. Hence, we exclude such assignments on alternate turns of minor restarts.

Parameter Settings. Values of tunable parameters used in the implementation are summarized in Table 1.

Table 1. Tunable parameters of the algorithm

Symbol	Explanation	Value
sp	Probability sp for PAWS scheme	0.006
T_1	Number of non-improving steps before minor restart	100
T_2	Number of minor restarts before major restart	100
ϵ_v	Threshold for relaxing equality	10^{-4}
ϵ_p	Amount of relaxation	10^{-4}

6 Evaluation

In this section, we compare the implementation with those of complete procedures in existing SMT solvers Z3 [33], cvc5 [4] and Yices [14], as well as previous work on local search for (fragments of) nonlinear arithmetic. We also perform an ablation study on the two improvements described in Sect. 3 and 4.

The benchmark used in the evaluation comes from SMT-LIB's QF_NRA theory. The benchmark consists mostly of industrial problems from various applications of constraint solving in nonlinear real arithmetic, including analysis of nonlinear hybrid automata (hycomp) [13], and generating ranking functions for termination analysis (LassoRanker) [22,28]. The kissing benchmark contains encoding of the *kissing problem*, which asks how many unit spheres in a given dimension can be placed tangent to a single unit sphere without intersecting each other. Each benchmark is labeled with either sat, unsat or unknown, according to whether it is known to be satisfiable/unsatisfiable at time of submission. Many of the unknown problems are in fact shown to be unsatisfiable by complete algorithms implemented in solvers such as Z3 and cvc5. For the experiments, we choose all problems from the benchmarks that are labeled sat or unknown, but excluding those unknown instances that are found to be unsatisfiable by either Z3 or cvc5. We also note that the SMT-LIB benchmarks contain problems with a wide range of difficulties, but without specified difficulty ratings. For example, many problem instances in the metitarski category, from the MetiTarski tool for proving theorems involving special functions [2], are quite small and do not pose much of a challenge for either local search or complete algorithms.

This yields a total of 6216 instances. The experiments are run on a cluster of machines with Intel Xeon Platinum 8153 processor at 2.00 GHz. Each experiment is run with a time limit of 20 min (as in the SMT competition) and memory limit of 30 GB.

6.1 Overall Result

Results of our implementation are compared against that of other SMT solvers in Table 2. One major advantage of our algorithm is in the Sturm-MBO category, which involves a single complicated polynomial that tripped up other solvers. However, we also showed good result across other categories, and solved most instances overall.

There is significant amount of complementarity between our algorithm and both Z3 and cvc5. As shown in Table 2, there are 148 instances across seven different categories that are solved by local search, but none of Z3, cvc5, and Yices. Moreover, there are 291 instances solved by local search but not Z3, and 378 instances solved by local search but not cvc5. Scatter plots comparing solution times against Z3 and cvc5 are shown in Fig. 1, showing there is significant complementarity in solving times as well.

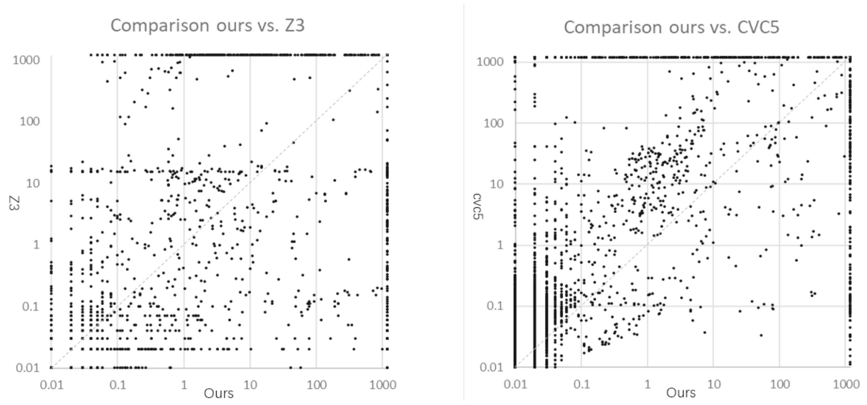

Fig. 1. Scatter plots of running time vs. Z3 and cvc5.

We also note there is a large number of relatively simple problem instances among the SMT-LIB benchmarks for QF_NRA. To put this into quantitative form, we counted the number of instances that are solved within 1 s by all of Z3, cvc5, and our solver. There are 4765 such instances, leaving only 1451 instances that can be considered "challenging" to the solvers. From this view, the overall improvement in the number of solved instances, number of unique solved, and amount of complementarity are quite significant.

Table 2. Comparison with other SMT solvers. Column #inst is the number of instances in the category. Column Z3, cvc5, Yices, and Ours shows number of solved instances by three existing SMT solvers and our implementation. Column Unique is the number of instances solved by our implementation but none of the other three SMT solvers.

Category	#inst	Z3	cvc5	Yices	Ours	Unique
20161105-Sturm-MBO	120	0	0	0	**88**	88
20161105-Sturm-MGC	2	**2**	0	0	0	0
20170501-Heizmann	60	3	1	0	**8**	6
20180501-Economics-Mulligan	93	**93**	89	91	90	0
2019-ezsmt	61	**54**	51	52	19	0
20200911-Pine	237	**235**	201	**235**	224	0
20211101-Geogebra	112	**109**	91	99	101	0
20220314-Uncu	74	73	66	**74**	70	0
LassoRanker	351	155	**304**	122	272	13
UltimateAtomizer	48	**41**	34	39	27	2
hycomp	492	**311**	216	227	304	11
kissing	42	**33**	17	10	**33**	1
meti-tarski	4391	**4391**	4345	4369	4351	0
zankl	133	70	61	58	**100**	27
Total	6216	5570	5476	5376	**5687**	148

6.2 Comparison with Other Work on Local Search

We further compare our results against existing work on local search for fragments of nonlinear real arithmetic. Of the 979 instances that are multilinear considered by [29], our implementation can solve 826 instances, compared to 891 solved instances there. The slightly weaker result is likely due to the more efficient implementation that is possible when only rational numbers need to be considered, and the parameter tuning that is specific to multilinear problems. Of the 2736 instances from SMT-LIB considered by [31], our implementation can solve 2589 instances, compared to 2246 solved instances there. In fact, we solve not only more instances than the local search algorithm given in [31], but also all other SMT solvers used in the comparison. We note that the result given in [31] uses different underlying software (including Maple) and runs on different machines, so this gives only a rough comparison.

6.3 Effect of Incremental Computation of Variable Scores

To show the effect of speedup resulting from incremental computation of variable scores in Sect. 3, we compare three versions of the implementation: with incremental computation (Incremental), without incremental computation (Naive), and without incremental computation, but limiting the number of unsatisfied clauses considered at each turn to 45 (Limit-45). The results are shown in Table 3.

We see that while the difference in total number of problem instance solved is not large, a noticeable effect is still present in the LassoRanker category, whose instances usually require a long time to solve. A closer look at the running time shows that it usually takes 2–10 times longer to solve a particular instance using either (Naive) or (Limit-45) compared to (Incremental), with the exact ratio depending strongly on the specific instance. For a time limit of 20 min the resulting difference in the number of solved problems is not large, but we expect a larger difference with shorter time limits, and especially when local search is incorporated into other methods such as DPLL [8,9].

6.4 Effect of Relaxation of Equalities

We demonstrate the effect of relaxation of constraints by comparing three possible implementations: with relaxation of constraints (Relaxation), without relaxation of constraints, but preferring variable assignments that are less complex than ϵ_v (Threshold), and without relaxation of constraints, with choosing variable assignments without considering complexity order (NoOrder). The results are shown in Table 4.

The results indicate that while taking complexity of assigned values into consideration does have an effect in keeping the search efficient for most categories of problem instances, it is not sufficient for the hycomp category, which involves a large number of nonlinear equalities. In that category using relaxation of constraints have a significant effect, while also performing well in other categories.

Table 3. Comparison showing effect of incremental computation

Category	#inst	Incremental	Naive	Limit-45
20161105-Sturm-MBO	120	88	85	85
20161105-Sturm-MGC	2	0	0	0
20170501-Heizmann	60	8	5	5
20180501-Economics-Mulligan	93	90	89	89
2019-ezsmt	61	19	19	15
20200911-Pine	237	224	222	222
20211101-Geogebra	112	101	101	101
20220314-Uncu	74	70	70	70
LassoRanker	351	272	264	269
UltimateAtomizer	48	27	26	26
hycomp	492	304	298	298
kissing	42	33	32	33
meti-tarski	4391	4351	4352	4352
zankl	133	100	100	100
Total	6216	5687	5663	5665

Table 4. Comparison showing effect of temporary relaxation of constraints

Category	#inst	Relaxation	Threshold	NoOrder
20161105-Sturm-MBO	120	88	100	99
20161105-Sturm-MGC	2	0	0	0
20170501-Heizmann	60	8	9	3
20180501-Economics-Mulligan	93	90	89	86
2019-ezsmt	61	19	19	19
20200911-Pine	237	224	223	222
20211101-Geogebra	112	101	98	92
20220314-Uncu	74	70	70	70
LassoRanker	351	272	277	278
UltimateAtomizer	48	27	26	20
hycomp	492	304	211	164
kissing	42	33	31	27
meti-tarski	4391	4351	4353	4360
zankl	133	100	100	100
Total	6216	5687	5606	5540

6.5 Other Techniques

We also tried other techniques commonly used in works on local search, including tabu search [20], switching between phases for adjusting boolean and arithmetic variables (as applied in [7]), and incorporating random walk (such as variants of WalkSAT [23]). Unlike their applications in earlier work, the use of such methods did not result in noticeable improvements on the current benchmark. However, it remains to investigate whether they will be useful on other types of problems, or in combination with other improvements to the algorithm.

7 Conclusion

In this paper, we presented improvements to the local search algorithm for solving SMT problems in nonlinear real arithmetic. Building upon the basic structure of local search, we presented incremental computation of variable scores and temporary relaxation of constraints. We also described heuristic move selection with look-ahead for dealing with literals without critical moves, and implementation details to improve efficiency. The resulting implementation is competitive against complete algorithms based on DPLL(T) and MCSAT on satisfiable problem instances, as implemented in other SMT solvers. It is the first local search algorithm designed for the entirety of nonlinear real arithmetic, covering a wider range of problems than existing work [29,31].

While the methods proposed in this paper made progress in addressing challenges of local search for nonlinear real arithmetic, there are remaining problems that represent interesting directions of future work.

- Look-ahead for critical moves presents another way to improve upon random search in cases when no critical move is available. On the other hand, methods based on CAD or polynomial optimization would give a more complete way to determine assignments that satisfy a certain literal. A major challenge is how to incorporate such algorithms into local search in an efficient way.
- In the current work, after an approximate solution is found that satisfies the relaxed version of equalities, we use various heuristic methods to attempt to find an exact solution nearby. Designing or incorporating more general algorithms for finding exact solutions near approximate solutions (or determining that none exist) is an interesting problem that we leave to future work.
- Finally, local search can be incorporated into complete methods such as DPLL, improving its performance even on unsatisfiable instances, as shown by the works [8,9]. It is interesting to investigate this possibility for SMT problems over nonlinear real arithmetic. The improvements in efficiency in our work would be very helpful for such combination, as in those cases local search is only given very short running times.

Acknowledgements. This work was partially supported by the National Natural Science Foundation of China under Grant Nos. 62032024 and 62002351.

References

1. Ábrahám, E., Davenport, J.H., England, M., Kremer, G.: Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. J. Log. Algebraic Methods Program. **119**, 100633 (2021). https://doi.org/10.1016/j.jlamp.2020.100633
2. Akbarpour, B., Paulson, L.C.: MetiTarski: an automatic theorem prover for real-valued special functions. J. Autom. Reason. **44**(3), 175–205 (2010). https://doi.org/10.1007/s10817-009-9149-2
3. Akutsu, T., Hayashida, M., Tamura, T.: Algorithms for inference, analysis and control of Boolean networks. In: Horimoto, K., Regensburger, G., Rosenkranz, M., Yoshida, H. (eds.) AB 2008. LNCS, vol. 5147, pp. 1–15. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85101-1_1
4. Barbosa, H., et al.: cvc5: a versatile and industrial-strength SMT solver. In: TACAS 2022. LNCS, vol. 13243, pp. 415–442. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_24
5. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB) (2016). www.SMT-LIB.org
6. Barrett, C., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Model Checking, pp. 305–343. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_11
7. Cai, S., Li, B., Zhang, X.: Local search for SMT on linear integer arithmetic. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, 7–10 August 2022, Proceedings, Part II. LNCS, vol. 13372, pp. 227–248. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-13188-2_12

8. Cai, S., Zhang, X.: Deep cooperation of CDCL and local search for SAT. In: Li, C.-M., Manyà, F. (eds.) SAT 2021. LNCS, vol. 12831, pp. 64–81. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-80223-3_6
9. Cai, S., Zhang, X., Fleury, M., Biere, A.: Better decision heuristics in CDCL through local search and target phases. J. Artif. Intell. Res. **74**, 1515–1563 (2022). https://doi.org/10.1613/jair.1.13666
10. Caviness, B.F., Johnson, J.R.: Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts and Monographs in Symbolic Computation. Springer, Vienna (2004). https://doi.org/10.1007/978-3-7091-9459-1
11. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. ACM Trans. Comput. Log. **19**(3), 19:1–19:52 (2018). https://doi.org/10.1145/3230639
12. Cimatti, A., Griggio, A., Lipparini, E., Sebastiani, R.: Handling polynomial and transcendental functions in SMT via unconstrained optimisation and topological degree test. In: Bouajjani, A., Holík, L., Wu, Z. (eds.) Automated Technology for Verification and Analysis - 20th International Symposium, ATVA 2022, Virtual Event, 25–28 October 2022, Proceedings. LNCS, vol. 13505, pp. 137–153. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-19992-9_9
13. Cimatti, A., Mover, S., Tonetta, S.: A quantifier-free SMT encoding of non-linear hybrid automata. In: Cabodi, G., Singh, S. (eds.) Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK, 22–25 October 2012, pp. 187–195. IEEE (2012). https://ieeexplore.ieee.org/document/6462573/
14. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 737–744. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_49
15. Fontaine, P., Ogawa, M., Sturm, T., Vu, X.T.: Subtropical satisfiability. In: Dixon, C., Finger, M. (eds.) FroCoS 2017. LNCS (LNAI), vol. 10483, pp. 189–206. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66167-4_11
16. Fröhlich, A., Biere, A., Wintersteiger, C.M., Hamadi, Y.: Stochastic local search for satisfiability modulo theories. In: Bonet, B., Koenig, S. (eds.) Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 25–30 January 2015, Austin, Texas, USA, pp. 1136–1143. AAAI Press (2015). http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9896
17. Gao, S., Avigad, J., Clarke, E.M.: δ-complete decision procedures for satisfiability over the reals. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 286–300. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31365-3_23
18. Gao, S., Avigad, J., Clarke, E.M.: Delta-decidability over the reals. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, 25–28 June 2012, pp. 305–314. IEEE Computer Society (2012). https://doi.org/10.1109/LICS.2012.41
19. Gao, S., Kong, S., Clarke, E.M.: dReal: an SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 208–214. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38574-2_14
20. Glover, F.W., Laguna, M.: Tabu Search. Kluwer (1997). https://doi.org/10.1007/978-1-4615-6089-0
21. Griggio, A., Phan, Q.-S., Sebastiani, R., Tomasi, S.: Stochastic local search for SMT: combining theory solvers with WalkSAT. In: Tinelli, C., Sofronie-Stokkermans, V. (eds.) FroCoS 2011. LNCS (LNAI), vol. 6989, pp. 163–178. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24364-6_12

22. Heizmann, M., Hoenicke, J., Leike, J., Podelski, A.: Linear ranking for linear lasso programs. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 365–380. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-02444-8_26
23. Hoos, H.H., Stützle, T.: Local search algorithms for SAT: an empirical evaluation. J. Autom. Reason. **24**(4), 421–481 (2000). https://doi.org/10.1023/A:1006350622830
24. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations & Applications. Elsevier/Morgan Kaufmann (2004)
25. Jovanović, D., de Moura, L.: Solving non-linear arithmetic. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 339–354. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31365-3_27
26. Khanh, T.V., Ogawa, M.: SMT for polynomial constraints on real numbers. In: Jeannet, B. (ed.) Third Workshop on Tools for Automatic Program Analysis, TAPAS 2012, Deauville, France, 14 September 2012. ENTCS, vol. 289, pp. 27–40. Elsevier (2012). https://doi.org/10.1016/j.entcs.2012.11.004
27. Kremer, G.: Cylindrical algebraic decomposition for nonlinear arithmetic problems. Ph.D. thesis, RWTH Aachen University, Germany (2020). https://publications.rwth-aachen.de/record/792185
28. Leike, J., Heizmann, M.: Ranking templates for linear loops. Log. Methods Comput. Sci. **11**(1) (2015). https://doi.org/10.2168/LMCS-11(1:16)2015
29. Li, B., Cai, S.: Local search for SMT on linear and multilinear real arithmetic. CoRR abs/2303.06676 (2023). https://doi.org/10.48550/arXiv.2303.06676. Accepted for FMCAD
30. Li, H., Xia, B., Zhang, H., Zheng, T.: Choosing better variable orderings for cylindrical algebraic decomposition via exploiting chordal structure. J. Symb. Comput. **116**, 324–344 (2023). https://doi.org/10.1016/j.jsc.2022.10.009
31. Li, H., Xia, B., Zhao, T.: Local search for solving satisfiability of polynomial formulas. In: Enea, C., Lal, A. (eds.) Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, 17–22 July 2023, Proceedings, Part II. LNCS, vol. 13965, pp. 87–109. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-37703-7_5
32. Li, H., Xia, B., Zhao, T.: Square-free pure triangular decomposition of zero-dimensional polynomial systems. J. Syst. Sci. Complex. (2023)
33. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
34. de Moura, L., Jovanović, D.: A model-constructing satisfiability calculus. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) VMCAI 2013. LNCS, vol. 7737, pp. 1–12. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35873-9_1
35. Nalbach, J., Ábrahám, E.: Subtropical satisfiability for SMT solving. In: Rozier, K.Y., Chaudhuri, S. (eds.) NASA Formal Methods - 15th International Symposium, NFM 2023, Houston, TX, USA, 16–18 May 2023, Proceedings. LNCS, vol. 13903, pp. 430–446. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-33170-1_26
36. Ni, X., Wu, Y., Xia, B.: Solving SMT over non-linear real arithmetic via numerical sampling and symbolic verification. In: SETTA 2023 (2023)
37. Niemetz, A., Preiner, M., Biere, A.: Precise and complete propagation based local search for satisfiability modulo theories. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 199–217. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41528-4_11

38. Niemetz, A., Preiner, M., Fröhlich, A., Biere, A.: Improving local search for bit-vector logics in SMT with path propagation (2015)
39. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: from an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). J. ACM **53**(6), 937–977 (2006). https://doi.org/10.1145/1217856.1217859
40. Selman, B., Levesque, H.J., Mitchell, D.G.: A new method for solving hard satisfiability problems. In: Swartout, W.R. (ed.) Proceedings of the 10th National Conference on Artificial Intelligence, San Jose, CA, USA, 12–16 July 1992, pp. 440–446. AAAI Press/The MIT Press (1992). http://www.aaai.org/Library/AAAI/1992/aaai92-068.php
41. Shen, D., Lierler, Y.: SMT-based constraint answer set solver EZSMT+ for non-tight programs. In: Thielscher, M., Toni, F., Wolter, F. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October–2 November 2018, pp. 67–71. AAAI Press (2018). https://aaai.org/ocs/index.php/KR/KR18/paper/view/18049
42. Susman, B., Lierler, Y.: SMT-based constraint answer set solver EZSMT (system description). In: Carro, M., King, A., Saeedloei, N., Vos, M.D. (eds.) Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, New York City, USA, 16–21 October 2016. OASIcs, vol. 52, pp. 1:1–1:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/OASIcs.ICLP.2016.1
43. Thornton, J., Pham, D., Bain, S., Ferreira Jr., V.: Additive versus multiplicative clause weighting for SAT. In: McGuinness, D.L., Ferguson, G. (eds.) Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, 25–29 July 2004, San Jose, California, USA, pp. 191–196. AAAI Press/The MIT Press (2004). http://www.aaai.org/Library/AAAI/2004/aaai04-031.php
44. Tung, V.X., Khanh, T.V., Ogawa, M.: raSAT: an SMT solver for polynomial constraints. Formal Methods Syst. Des. **51**(3), 462–499 (2017). https://doi.org/10.1007/s10703-017-0284-9