Boosting MCSat Modulo Nonlinear Integer Arithmetic via Local Search

Enrico Lipparini[1,2](✉), Thomas Hader[3], Ahmed Irfan[4], and Stéphane Graham-Lengrand[4]

[1] University of Cagliari, Cagliari, Italy
`enrico.lipparini@unica.it`
[2] University of Genoa, Genoa, Italy
[3] TU Wien, Vienna, Austria
`thomas.hader@tuwien.ac.at`
[4] SRI International, Menlo Park, CA, USA
`{ahmed.irfan,stephane.graham-lengrand}@sri.com`

Abstract. The Model Constructing Satisfiability (MCSat) approach to the SMT problem extends the ideas of CDCL from the SAT level to the theory level. Like SAT, its search is driven by incrementally constructing a model by assigning concrete values to theory variables and performing theory-level reasoning to learn lemmas when conflicts arise. Therefore, the selection of values can significantly impact the search process and the solver's performance. In this work, we propose guiding the MCSat search by utilizing assignment values discovered through local search. First, we present a theory-agnostic framework to seamlessly integrate local search techniques within the MCSat framework. Then, we highlight how to use the framework to design a search procedure for (quantifier-free) Nonlinear Integer Arithmetic ($\mathcal{NIA}$), utilizing accelerated hill-climbing and a new operation called *feasible-sets jumping*. We implement the proposed approach in the MCSat engine of the Yices2 solver, and empirically evaluate its performance over the $\mathcal{NIA}$ benchmarks of SMT-LIB.

1 Introduction

Satisfiability Modulo Theory (SMT) is the problem of deciding the satisfiability of a first-order formula with respect to defined background theories. SMT solvers are the core backbone of a vast range of verification and synthesis tools that require reasoning about expressive logical theories such as real/integer arithmetic [3,40]. One of the major state-of-the-art approaches to SMT is the Model Constructing Satisfiability calculus (MCSat) [27,39], which generalizes the ideas of Conflict-Driven Clause Learning (CDCL) to the theory level, and which has been shown to perform particularly well on complex theories such as nonlinear arithmetic. In the MCSat approach, the solver progressively constructs a *theory model*, similarly to how SAT solvers construct Boolean models. Theory reasoning is used to assess the consistency of partial models, provide explanations of infeasibility, decide theory variables, and propagate theory constraints.

When extending the partial model with a new assignment to a theory variable, picking a good value is critical for the overall performance of the solver. Heuristics used by state-of-the-art solvers pick values on the basis of compatibility with the current search state and of computational cheapness. This has a major drawback: these heuristics only consider knowledge of the current search state, neglecting information on how likely a particular assignment is to lead to a satisfying model eventually.

In this work, we address the problem of choosing good values for variable decisions by augmenting the current search state knowledge with insights provided by local search techniques. Following the logic-to-optimization approach [17,35], we associate to the logical formula a cost function that represents the *distance from a model*, and use local search to find assignments that have a small cost. These assignments are then used to guide future MCSat decisions.

Although local search has already been used in the context of SMT, either as a standalone solver [46,33,6] or as a CDCL(T) theory solver [48], our work is the first to propose a tight integration of local search within the MCSat framework, creating a powerful synergy between the reasoning capabilities of MCSat and the intuition provided by local search which boosts performance for both satisfiable and unsatisfiable instances. Our novel approach is flexible enough to allow calls to local search at any point during the MCSat search, seamlessly fitting with the current state. As MCSat progresses through decisions, propagations, and conflicts, the local search problem is instantiated accordingly: (i) the cost function is built upon the simplification of the original formula under current state assumptions, (ii) initial local search assignments are based on the current search state as well as cached values, and (iii) local search moves are enhanced by information on intervals of feasible assignments to theory variables as tracked by the MCSat engine.

While our approach can be applied to any theory supported by MCSat, in this work we showcase its application to the theory of nonlinear integer arithmetic ($\mathcal{NIA}$). In particular, we design a procedure based on a new operation called *feasible-sets jumping*, which allows to move between feasible intervals, and on accelerated hill-climbing [24], to move inside feasible intervals.

Contributions. In this work we: (i) design a theory-agnostic framework to tightly integrate local search techniques within the MCSat approach in order to guide variable decisions, (ii) use the framework to define a local search procedure for the theory of nonlinear integer arithmetic, that makes use of feasible-sets jumping and accelerated hill-climbing, and (iii) show the practical applicability of our method using our implementation in the MCSat engine of Yices2 [15] on the quantifier-free $\mathcal{NIA}$ benchmark set of SMT-LIB [2].

Structure. In Section 2, we provide the necessary background. Section 3 describes a deep integration of local search techniques within the MCSat framework from a general point of view, which is applied in Section 4 to define a local search approach for non-linear integer arithmetic. In Section 5, we show and discuss the results of our experiments before presenting related work in Section 6 and concluding in Section 7.

2 Preliminaries

We assume basic knowledge on the standard first-order quantifier-free logical setting and standard notions of theory, satisfiability, and logical consequence. We write logical *variables* with $x, y, \ldots$, and concrete *values* with $\alpha, \beta, \ldots$ (the domain of concrete values is theory specific, e.g. $\mathbb{Z}$ for integer arithmetic, $\mathbb{R}$ for real arithmetic). An *assignment* μ is a map from variables to values of matching type. If ϕ is a formula, we denote with $\mathit{Vars}(\phi)$ the set of its (free) variables. We use C to denote clauses, and L to denote literals. Nonlinear Integer Arithmetic ($\mathcal{NIA}$) is the theory consisting of arbitrary Boolean combinations of Boolean variables and arithmetic atoms of the form of polynomial equalities and polynomial inequalities over integer variables. It is undecidable by Matiyasevich's theorem [38].

2.1 SMT & MCSat

SMT [3] is the problem of deciding the satisfiability of a first-order formula with respect to some theory or combination of theories. Two of the major approaches for SMT solving are the Conflict-Driven Clause Learning with theory support (CDCL(T)) [42,3] and the MCSat approach. In the former, theory solvers augment a propositional SAT engine with theory reasoning procedures which are capable of deciding a conjunction of literals (i.e. atomic formulas and their negations) in a particular theory. A propositional model (of the Boolean abstraction of the formula) found by the SAT solver is then checked by all theory engines for theory consistency.

The latter, MCSat, applies CDCL-like mechanisms to perform theory reasoning directly. It can be used either as a theory solver for a specific theory (e.g. in Z3 [14] for non-linear arithmetic over the reals and the integers [4]), or as a fully-fledged stand-alone engine able to handle multiple theories (e.g. in Yices2 for non-linear arithmetic over the reals [28] and over the integers [26], bit-vectors [19], arrays [25], and finite fields [23,22]; as well as in SMT-RAT [13] for non-linear real arithmetic [31]). The MCSat architecture consists of a core solver, an assignment trail, and plugins for theory reasoning. Figure 1 illustrates the high level flow of the MCSat framework.

The core solver incrementally constructs a partial model consisting of Boolean and theory assignments (stored in a *trail*), maintaining the invariant that none of the constraints evaluate to false under the partial model. The trail contains three kinds of elements: *propagated literals* (literals implied to be true by the current state), *decided literals* (literals that we assume to be true), and *model assignments* (assignments of first-order variables to concrete values). Propagations, conflict analysis, lemmas generation, and variable decisions are all handled by theory plugins (including a Boolean plugin that is responsible for propositional reasoning). In general, plugins also keep a *feasibility set* for each variable of their competence, containing the values that are consistent with the current trail and are, thus, candidates to be picked for deciding the variable.

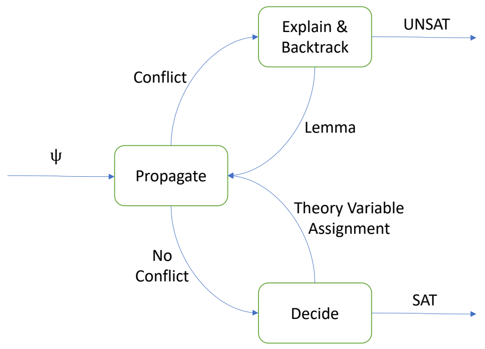

Fig. 1: The MCSat framework consists of the following steps: 1) Propagate the trail. 2) If a conflict is found during propagation, check if there is any decision to backtrack over. If not, return UNSAT. Otherwise, explain the conflict using a lemma, backtrack the trail, and repeat step 1. 3) If no conflict is found during propagation, decide on a variable that is not on the trail. If there is nothing left to decide, return SAT. Otherwise, add the decided variable to the trail and repeat step 1.

When the core solver selects a variable for decision, the choice of the value to assign to the variable is handled by the theory plugin responsible for its type. Some solvers (e.g. Yices2) implement a heuristic called *value cache* (a generalization of SAT phase saving [43]), that keeps track of the last value assigned to a variable when the assignment is undone. Then, when a decision has to be made for the variable, the cached value will be used, provided that it is still in the feasibility set; otherwise, it will simply be ignored, and other heuristics will be used (e.g. picking a default value such as 0).

In the following, we will denote with M the trail, with $v[M](x)$ the value of the variable x in the trail (which may be equal to `undef` if the variable is not assigned in the trail), and with $v[M](L)$ the value of the literal L under the assignment in M which may be true ($\top$) or false ($\bot$) if L can be fully evaluated under such assignment, or `undef` otherwise. We denote with $\mathit{feasible}_M(x)$ the feasibility set of x in M, i.e. all values that can be chosen for x in the current search state given by M. For arithmetical theories, we have that $\mathit{feasible}_M(x) \subseteq \mathbb{R}$, and, in particular, that $\mathit{feasible}_M(x)$ is the union of a finite set of *feasible intervals*, i.e. $\mathit{feasible}_M(x) = \bigcup_{i\in[0:m]} I_i$. We assume that theory plugins provide a function $\mathit{pick\_value}(S)$ that returns a value from a set S.

Example 1. Assume a search problem in $\mathbb{Z}$ with variables x, y, and z given by the input formula ϕ.

$$\phi \;=\; (\neg(x \geq 1) \vee (xy = 1)) \;\wedge\; (\neg(xy = 1) \vee (x + 2yz > 0)) \;\wedge\; (z^2 > 1)$$

A possible trail at some point during the search is

$$M = [(z^2 > 1) \mapsto \top, x \mapsto 1, (x \geq 1) \mapsto \top, (xy = 1) \mapsto \top]$$

On M elements are either decided or propagated. The feasibility sets are given by $\mathit{feasible}_M(z) = (-\infty, -1) \cup (1, \infty)$ and $\mathit{feasible}_M(y) = \{1\} = \mathit{feasible}_M(x)$. Since $\mathit{feasible}_M(y)$ is a singleton, we can propagate the assignment $y \mapsto 1$ on M. We further have that $v[M]((x + 2yz > 0)) = \texttt{undef}$ and $v[M](x) = 1$.

2.2 Local Search

We define a local search problem as a triple $(\mu_0, f_c, \mathit{moves})$, where:

- μ_0 is an initial assignment for a set of variables Vars,
- f_c is a cost function from the set of assignments to $\mathbb{R}_{\geq 0}$,
- moves is a neighbor relation between assignments.

A local search algorithm starts from the initial assignment μ_0 and iteratively explores neighboring assignments according to the *moves* relation. We say that μ' is a *move from* μ if $(\mu, \mu') \in \mathit{moves}$. A move μ' is *accepted* if $f_c(\mu') < f_c(\mu)$. When a move is accepted, the new assignment becomes the current assignment and the search continues until either: a zero-cost assignment is found, there are no more possible moves (meaning that the current assignment represents a *local minimum*), or a given stopping criterion is reached (e.g. number of moves).

The problem of finding a solution for an SMT formula ϕ can be encoded as a local search problem, e.g., by following the logic-to-optimization approach [17,35,36], in which a formula ϕ is mapped to a term $\mathcal{L2O}(\phi)$ that represents the *distance from a solution*.

In principle, the $\mathcal{L2O}$ operator can be defined for any theory for which the concept of distance between terms makes sense. Here, we limit ourselves to arithmetic theories. We introduce an arithmetic function symbol d of arity 2 and we assume a fixed interpretation d that satisfies the properties of metric distance, i.e. symmetry, positivity, reflexivity, and triangle inequality. We also assume the existence of a fixed constant term ϵ, such that $\epsilon > 0$. The specific choice of d and ϵ is theory-dependent.

We recursively define $\mathcal{L2O}$ as follows:

$$\begin{aligned}
&\mathcal{L2O}(b) && \overset{\text{def}}{=} ITE(b, 0, 1)\\
&\mathcal{L2O}(\neg b) && \overset{\text{def}}{=} ITE(b, 1, 0)\\
&\mathcal{L2O}(t_1 = t_2) && \overset{\text{def}}{=} d(t_1, t_2)\\
&\mathcal{L2O}(t_1 \leq t_2) && \overset{\text{def}}{=} ITE(t_1 \leq t_2,\ 0,\ d(t_1, t_2))\\
&\mathcal{L2O}(t_1 < t_2) && \overset{\text{def}}{=} ITE(t_1 < t_2,\ 0,\ d(t_1, t_2) + \epsilon)\\
&\mathcal{L2O}(t_1 \neq t_2) && \overset{\text{def}}{=} ITE(t_1 \neq t_2,\ 0,\ 1)\\
&\mathcal{L2O}(\phi_1 \wedge \phi_2) && \overset{\text{def}}{=} \mathcal{L2O}(\phi_1) + \mathcal{L2O}(\phi_2)\\
&\mathcal{L2O}(\phi_1 \vee \phi_2) && \overset{\text{def}}{=} \mathcal{L2O}(\phi_1) \cdot \mathcal{L2O}(\phi_2)\\
&\mathcal{L2O}(ITE(\phi_c, \phi_1, \phi_2) && \overset{\text{def}}{=} ITE(\phi_c,\ \mathcal{L2O}(\phi_1),\ \mathcal{L2O}(\phi_2))\\
&\mathcal{L2O}(\neg ITE(\phi_c, \phi_1, \phi_2)) && \overset{\text{def}}{=} ITE(\phi_c,\ \mathcal{L2O}(\neg\phi_1),\ \mathcal{L2O}(\neg\phi_2))
\end{aligned}$$

It is easy to check that a complete assignment μ satisfies ϕ if and only if $\mathcal{L2O}(\phi)$ evaluates to 0 under μ.

In the following, with a slight abuse of notation, we denote with $\mathcal{L}2\mathcal{O}(\phi)$ also the corresponding arithmetic function determined by the interpretation d and the constant ϵ, and we define the *cost function* associated to ϕ as $f_c \overset{\text{def}}{=} \mathcal{L}2\mathcal{O}(\phi)$.

Example 2. Let $\phi \overset{\text{def}}{=} b \wedge x = y^2$, and $\mathrm{d}(t_1, t_2) \overset{\text{def}}{=} |t_1 - t_2|$. Then, the cost function associated to ϕ is

$$f_c \overset{\text{def}}{=} \mathcal{L}2\mathcal{O}(\phi) = ITE(b, 0, 1) + |x - y^2|$$

Now, let $\mu_0 \overset{\text{def}}{=} \{b \mapsto \bot \ ; \ x \mapsto 4 \ ; \ y \mapsto 1\}$ be a starting assignment. We have that $f_c(\mu_0) = ITE(\bot, 0, 1) + |4 - 1^2| = 1 + 3 = 4$. If we consider the move $\mu_1 \overset{\text{def}}{=} \{b \mapsto \top \ ; \ x \mapsto 4 \ ; \ y \mapsto 1\}$ that flips b, then $f_c(\mu_1) = ITE(\top, 0, 1) + |4 - 1^2| = 0 + 3 = 3$, hence the move is improving and is accepted. Then, if we consider the move $\mu_2 \overset{\text{def}}{=} \{b \mapsto \top \ ; \ x \mapsto 4 \ ; \ y \mapsto 2\}$ that increases the value of y by 1, we have $f_c(\mu_1) = ITE(\top, 0, 1) + |4 - 2^2| = 0 + 0 = 0$. Hence we have found a zero for f_c, i.e. a satisfying assignment for ϕ.

In general, local search is not guaranteed to find a solution of ϕ, if there is any. Nevertheless, it returns a *local minimum/best-effort value* of the cost function in the neighborhood of the initial assignment.

3 Deep combination of Local Search and MCSat

We propose a deep combination of MCSat and local search where: (i) the current state of MCSat is used to instantiate a local search problem and (ii) the results of the local search help guiding future MCSat decisions. Assuming we have a local search procedure LS, we discuss how to instantiate LS (Section 3.1), as well as how to use the result of LS within MCSat (Section 3.2).

3.1 Instantiating the Local Search problem

For the instantiation of LS, we determine the initial assignment and the formula upon which the cost function is constructed. Both choices are of fundamental importance. A good initial assignment is essential to find a good local minimum of the cost function. A *good local minimum* is a local minimum that meets two conditions: (i) it has a smaller cost compared to the cost of the initial assignment and (ii) its assignment values are likely to be accepted by MCSat, i.e., they are consistent with the current trail. Passing a simplified formula that takes the truth value of propagated and decided literals into account is also essential to tailor the search to the current MCSat state and to avoid unnecessary computations.

Initial assignment. For every model assignment $x \mapsto \alpha$ in M, the assigned variable is treated as a constant that takes its respective assigned value (i.e., x is treated as the constant α) and is not allowed to be changed in LS. This reduces the dimension of the LS search space, and avoids moves inconsistent with the

Algorithm 1 Initial assignment for LS

```
Input: a set $Vars(\phi)$, a trail $M$, a value cache $cache$, a feasibility map $feasible_M$
Output: an initial assignment $\mu_0$. a set of fixed variables $VarsFixed \subseteq Vars(\phi)$
1: $VarsFixed \leftarrow \emptyset$
2: for $x \in Vars(\phi)$ do
3:     if $v[M](x) \neq$ undef then                               ▷ check if $x$ has a value in the trail
4:         $\mu_0(x) \leftarrow v[M](x)$                                    ▷ assign trail value
5:         $VarsFixed$.add($x$)
6:     else if $cache(x) \neq$ undef and $cache(x) \in feasible_M(x)$ then
7:         $\mu_0(x) \leftarrow cache(x)$                           ▷ assign feasible cached value
8:     else
9:         $\mu_0(x) \leftarrow pick\_value(feasible_M(x))$              ▷ assign any feasible value
```

current trail. For initial assignment of variables that are unassigned in M, a reasonable choice is to use cached values of previous search states, if present in the value cache. However, cached values are not guaranteed to be in the feasibility set, as they might be the result of a previous decision that eventually led to a conflict. Hence, we first check if the cached value is feasible. If it is not, or there is no cached value, we pick any value from the feasibility set by asking the appropriate theory plugin. The procedure for choosing the initial assignment is shown in Algorithm 1. Note that, for all the variables, the feasibility set cannot be empty. An empty feasibility set indicates an inconsistent trail which is resolved using conflict resolution before starting LS. MCSat maintains the invariant that, for a consistent trail, the feasibility set of all variables is non-empty.

Formula for LS. Every Boolean assignment $L \mapsto \{\top, \bot\}$ in M represents the truth value of the literal L that is assumed to hold at the current search state

Algorithm 2 Formula for LS

```
Input: a formula $\phi$, a trail $M$
Output: a subformula $\phi_{LS}$ of $\phi$
1: $\phi_{LS} \leftarrow \top$                                        ▷ formula to be passed to LS
2: for $C \in \phi$ do
3:     $C_{LS} \leftarrow \bot$
4:     for $L \in C$ do
5:         if $v[M](L) = \top$ then                      ▷ if literal is assigned to true in trail
6:             $C_{LS} \leftarrow \top$                          ▷ substitute the clause with true
7:             $\phi_{LS} \leftarrow \phi_{LS} \wedge L$                                       ▷ store literal
8:             break
9:         else if $v[M](L) = \bot$ then                 ▷ if literal is assigned to false in trail
10:            $\phi_{LS} \leftarrow \phi_{LS} \wedge \neg L$                ▷ store literal (with correct polarity)
11:            continue                                     ▷ ignore literal in the clause
12:        else
13:            $C_{LS} \leftarrow C_{LS} \vee L$                              ▷ keep literal in the clause
14:    $\phi_{LS} \leftarrow \phi_{LS} \wedge C_{LS}$                                  ▷ store simplified clause
```

(either because of a propagation or a decision). We can use this information to simplify the original formula before passing it to LS. For a given clause $C \overset{\text{def}}{=} L \vee L_1 \vee \ldots$, if $v[M](L) = \top$, then, for LS, it suffices to find an assignment that satisfies L, since such an assignment would satisfy C as well. Hence, in this case, we shall pass to LS just L instead of C. On the other hand, if $v[M](L) = \bot$, then there is no incentive for LS to try to find an assignment that makes L true, as any such assignment would be inconsistent with the trail and will be discarded by MCSat immediately. Thus, L is removed from the clause that is passed to LS. Note that, by just removing L from the clause, we still may end up with an assignment that evaluates L to $\top$. Therefore, for literals that are assigned to $\bot$ in the trail, we add, just once, $\neg L$ to the formula that we pass to LS. This procedure is shown in Algorithm 2.

3.2 Guiding MCSat decisions

During the search, we periodically call LS to suggest values for MCSat to choose in subsequent decisions. As a heuristic to decide when to call LS, we are utilizing a polynomially increasing conflict threshold. Initially, this limit is set to 50, and then it is increased according to the polynomial $50 \cdot \text{ls\_calls} \cdot \log_{10}(\text{ls\_calls} + 9)^3$, where ls_calls represents the number of times LS has been called. A similar heuristic is used by SAT solvers to decide when to perform certain cache clearing operations [8]. Once the threshold is reached, we wait until the last conflict has been resolved and all consequences of that conflict are propagated. Then we start LS to guide any further decisions.

The return of LS consists of a complete assignment that contains *suggested values* for future variable decisions. These suggested values are put in the MCSat value cache – recall that the values in the cache are picked first during variable decisions, provided that they are feasible. Note that, during the choice of the initial assignment to pass to LS, we had relied on (feasible) cached values as well. If a cached value was feasible and LS changed its value, it means that such change led to a smaller cost, hence got us closer to a solution. Therefore, replacing the old cached value with the newly found suggestion improves the cache quality. On the other hand, if the cached value was not feasible, then any change to a feasible value improves the cache quality.

Furthermore, LS keeps track of the activity of each variable during its execution. The most active variables have contributed most to the decrease of the cost during LS. We suggest those variables to MCSat as good choices for subsequent decisions. This way, variables that were more active during the local search phase will have a higher impact on the MCSat search.

4 Local Search for Nonlinear Integer Arithmetic

As explained in Section 3.1, LS receives an initial assignment μ_0 and a formula ϕ from MCSat. The formula ϕ is used to construct the cost function f_c using the logic-to-optimization approach (Section 2.2), i.e. $f_c \overset{\text{def}}{=} \mathcal{L}2\mathcal{O}(\phi)$. To apply that, we must first define the distance function d and the strict inequality constant ϵ for integers. For d, we choose a consistent and computationally cheap definition $\mathrm{d}(t_1, t_2) \overset{\text{def}}{=} |t_1 - t_2|$. For ϵ, our choice is $\epsilon \overset{\text{def}}{=} 1$, since $t < 0$ is interchangeable with $t + 1 \leq 0$ for integers.

The building blocks of local search are *moves*. We contemplate three types of moves (or *modes*): one for Boolean variables, and two for integer variables. Given an assignment μ, we have the following types of moves:

– *Boolean flips*: For a Boolean variable b, the assignment $\mu_{\neg b} \overset{\text{def}}{=} \mu[b \mapsto \neg\mu(b)]$ obtained by changing the value of b to the negation of its value assigned by μ is a *flip move* from μ.
– *Hill-climbing moves*: In the basic version of hill-climbing, for an integer variable x, the assignments $\mu_{x+1} \overset{\text{def}}{=} \mu[x \mapsto \mu(x) + 1]$ and $\mu_{x-1} \overset{\text{def}}{=} \mu[x \mapsto \mu(x) - 1]$ obtained by mapping x to the successor and predecessor of its value assigned by μ are moves from μ.
– *Feasible-set-jumps*: For an integer variable x, with feasibility set $\mathit{feasible}(x) = \bigcup_{i\in[0:m]} I_i$, and $x \in I_j$ (for a given $j \in [0 : m]$), the assignments $\mu_{\text{left}} \overset{\text{def}}{=} \mu[x \mapsto \mathit{pick\_value}(I_{j-1})]$, and $\mu_{\text{right}} \overset{\text{def}}{=} \mu[x \mapsto \mathit{pick\_value}(I_{j+1})]$ obtained by picking a value from the left and right feasible intervals of I_j (provided they exist, i.e., respectively, that $j-1 \in [0 : m]$, and $j+1 \in [0 : m]$) are moves from μ.

Our specific strategy of the local search algorithm is outlined in Algorithm 3. The algorithm starts with a list of variables *vars* (and an associated *feasible*

Algorithm 3 LS main algorithm

```
Input: a list $vars$, a feasibility map $feasible$, an initial assign. $\mu_0$, a cost function $f_c$
Output: a final assignment $\mu^*$ with $f_c(\mu^*) \leq f_c(\mu_0)$
 1: $\mu^* \leftarrow \mu_0$                                              ▷ best assignment
 2: $cost^* \leftarrow f_c(\mu_0)$                                        ▷ best cost
 3: for $mode \in \{\text{bool-flips}, \text{fs-jumps}, \text{hill-climb}\}$ do
 4:     $n\_vars \leftarrow 0$                      ▷ no. of vars visited since last improvement
 5:     while $n\_vars < \text{len}(vars)$ and $cost^* \neq 0$ do
 6:         $x \leftarrow vars[n\_vars]$                                  ▷ pick next variable
 7:         $\alpha \leftarrow \mu^*(x)$                                  ▷ value assigned to $x$
 8:         while $\alpha_{new} \leftarrow$ Move.choose($x, \alpha, feasible, mode$) do
 9:             $\mu_{new} \leftarrow \mu^*[x \mapsto \alpha_{new}]$      ▷ create new assignment
10:             $cost_{new} \leftarrow f_c(\mu_{new})$
11:             $success \leftarrow cost_{new} < cost^*$       ▷ check if the move has improved
12:             if $success$ then
13:                 $\mu^* \leftarrow \mu_{new}$                          ▷ update best assignment
14:                 $cost^* \leftarrow cost_{new}$                        ▷ update best cost
15:                 $n\_vars \leftarrow 0$                                ▷ reset no. of vars visited
16:                 $vars$.to_front($x$)                      ▷ move $x$ to the front of the list
17:             Move.notify($x, \alpha, \alpha_{new}, feasible, mode, success$)
18:         $n\_vars \leftarrow n\_vars + 1$                             ▷ increas no. of vars visited
```

map), an initial assignment μ_0, and a cost function f_c. The goal of the procedure is to return an assignment μ^* that improves over the initial assignment μ_0 w.r.t. the cost function, i.e. $f_c(\mu^*) < f_c(\mu_0)$.

At the beginning, the best assignment coincides with the initial assignment (Line 1). First, we cycle over modes (Line 3). Then, we enter in a loop over the variables (Line 5). The loop breaks only in two cases: if all the variables have already been visited since the last improvement (in which case, it means we have reached a local minimum w.r.t. the current mode moves), or if the current cost is equal to 0 (in which case it means we have found a solution). At each loop iteration, we pick the next variable (Line 6). Here, for simplicity, we assume that there are no fixed variables (in practice, these variables are just ignored and treated as constants). Then, for the current variable x, we enter in a second loop (Line 8), in which we select new values for x. These values are determined by a move selection module, which we discuss below. The loop breaks only when there are no more moves available. For each value, a new assignment is built by re-assigning x to the new value (Line 9), and the cost of the new assignment is computed (Line 10). We then check whether the current cost is lower than the previous cost (Line 11). If so, then the new assignment becomes the best assignment (Line 13), and x is moved to the front of the list (Line 16). If not, then we try other moves for x, if there are any. In both cases, we notify the move selection module whether the suggested move has led to a success or not (Line 17). The move selection module works as following.

Boolean flips mode. Here, the logic is rather straightforward, as there is only one move possible per variable. Regardless of whether the move has success or not, the cycle over moves terminates, and the algorithm proceeds with the cycles over variables or over modes.

Accelerated hill-climbing mode. The simple hill-climbing moves presented earlier, in which we add or subtract 1 to the current value, can be quite slow in converging toward a local minimum when the search space is huge. For this reason, we *accelerate* hill-climbing by keeping, for each variable, an adaptive *step_size*, which is incremented or decremented according to a fixed acceleration parameter *acc* (in our setting $acc = 1.2$) and on the base of the success of previous moves. At the beginning, *step_size* is set to 1 (i.e., we start with simple hill-climbing moves). At each iteration, we try four moves, corresponding to adding to the current value the product between *step_size* and one of the following: acc, $\frac{1}{acc}$, $\frac{-1}{acc}$, $-acc$. Since we are working with integers, every step value is rounded to the nearest integer. If one of the moves has success, then we set *step_size* to be equal to the best successful step (thus keeping the best velocity). If none of the moves has success, then we stop the moves cycle, and we set *step_size* to $\frac{step\_size}{acc}$ (thus decelerating over this variable for future moves).

Feasible-set-jumping mode. There are two possible versions of fs-jumping: *global* and *local.* In global fs-jumping, given a fixed variable, we try *all* possible jumps over the feasibility set, i.e. we try one jump per feasible interval. While this may give a wide-ranging view over the feasibility set, it can also be very costly, hence we limit global fs-jumping to one time per variable (per LS call). Local fs-jumping, on the contrary, only explores the left and the right feasible intervals w.r.t. to the interval that contains the current value. If one fs-jump is successful, e.g. the one to the left interval, then we continue on that direction and explore the interval further left. As soon as we find that both left and right fs-jumps do not improve, then we stop, hence avoiding to span over all feasible intervals like in the global fs-jumping.

5 Experiments

Implementation. We have implemented our method in the MCSat engine of the YICES2 SMT solver, adding a module for the interaction with LS. We will denote the version of YICES2 that makes use of LS as YICES2$_{LS}$ and the baseline version (without any local search) as YICES2$_{base}$.

Setup. We have run our experiments on a cluster equipped with AMD EPYC 7502 CPUs running at 2.5GHz, using a timeout of 300 seconds, and a memory limit of 8GB. We have compared the base version of YICES2 with the LS-boosted version YICES2$_{LS}$ as well as with the state-of-the-art SMT solvers CVC5 [1] (version 1.2.0), MATHSAT5 [11] (version 5.6.11), and Z3 [14] (version 4.13.3). We have also included in the comparison HYBRIDSMT [48], which runs a portfolio composed by the LS solver LocalSMT [6] (used as a standalone tool) and a LocalSMT-boosted version of Z3's CDCL(T) – see also related work (Section 6).

Benchmarks. We have considered all the SMT-LIB [2] (Version 2024 [44]) benchmarks from the QF_NIA category. This is a class of 25443 benchmarks, among which 14990 and 5183 come with a known status of "sat" and "unsat", respectively, and another 5270 have an "unknown" satisfiability status.

Results. In the presentation of the results, we consider both a short time limit and a long time limit. We set the short time limit to 24s, as in the respective SMT-COMP track [47], and the long time limit to 300s, due to resource constraints. The results are shown in Table 1 and Table 2, respectively. In the columns, we separate per benchmark family; on the rows, for each solver, we report the amount of overall benchmarks solved, and, in parenthesis, the amount of benchmarks solved restricted to sat and unsat instances, respectively. We also include two portfolios between Z3 (resp., HYBRIDSMT) and YICES2$_{LS}$, that work as follows: we run Z3 (resp., HYBRIDSMT) for half of the time limit (i.e., 12s/150s), then, if it has not terminated, we run YICES2$_{LS}$ for the remaining time.

Discussion. First, we observe that, with both time limits, YICES2$_{LS}$ solves a significant number of benchmarks more than YICES2$_{base}$. In particular, on both satisfiable and unsatisfiable instances, it improves (or matches) YICES2$_{base}$ results over all families, except one. Improving on unsatisfiable benchmarks is noteworthy: indeed, while, in general, local search is geared toward proving satisfiability, integrating it within MCSat enables to generate better lemmas. This is witnessed not only by the higher amount of benchmarks solved overall, but also by the lower amount of conflicts and theory variables assignments occurred. On unsatisfiable benchmarks solved by both tools, on average (resp. median),

Table 1: Summary of results for $\mathcal{NIA}$ benchmarks with a timeout of 24s.

	cvc5	HybridSMT	MathSAT5	Yices2$_{base}$	Yices2$_{LS}$	Z3	Portfolio (Z3 + Yices2$_{LS}$)	Portfolio (HybridSMT + Yices2$_{LS}$)
VeryMax	7799 (5465) (2334)	13555 (10427) (3128)	11233 (7615) (3618)	13695 (9461) (4234)	**14269** (10019) (4250)	13975 (9599) (4376)	14848 (10271) (4577)	15428 (11105) (4323)
calypto	171 (79) (92)	174 (78) (96)	168 (79) (89)	174 (79) (95)	174 (79) (95)	**176** (80) (96)	176 (80) (96)	176 (80) (96)
ezsmt	**8** (8) (0)	**8** (8) (0)	**8** (8) (0)	**8** (8) (0)	**8** (8) (0)	**8** (8) (0)	8 (8) (0)	8 (8) (0)
LassoRank	97 (4) (93)	**105** (4) (101)	**105** (4) (101)	91 (4) (87)	96 (4) (92)	104 (4) (100)	102 (4) (98)	104 (4) (100)
Dartagnan	320 (11) (309)	**354** (10) (344)	327 (12) (315)	142 (1) (141)	87 (0) (87)	352 (9) (343)	344 (9) (335)	347 (9) (338)
LCTES	**1** (0) (1)	**1** (0) (1)	**1** (0) (1)	0 (0) (0)	0 (0) (0)	**1** (0) (1)	1 (0) (1)	1 (0) (1)
MathProbl	107 (100) (7)	93 (86) (7)	141 (134) (7)	122 (115) (7)	**303** (296) (7)	118 (111) (7)	300 (293) (7)	305 (298) (7)
leipgiz	72 (70) (2)	**151** (150) (1)	114 (112) (2)	102 (101) (1)	111 (110) (1)	120 (119) (1)	119 (118) (1)	148 (147) (1)
UltAut23	16 (8) (8)	10 (7) (3)	17 (7) (10)	7 (7) (0)	7 (7) (0)	**21** (8) (13)	20 (8) (12)	9 (7) (2)
mcm	9 (9) (0)	**56** (56) (0)	3 (3) (0)	6 (6) (0)	6 (6) (0)	5 (5) (0)	5 (5) (0)	46 (46) (0)
sqrtmodinv	2 (0) (2)	7 (0) (7)	0 (0) (0)	0 (0) (0)	0 (0) (0)	**17** (0) (17)	17 (0) (17)	6 (0) (6)
UltAut	**7** (0) (7)	**7** (0) (7)	**7** (0) (7)	**7** (0) (7)	**7** (0) (7)	**7** (0) (7)	7 (0) (7)	7 (0) (7)
AProVE	1816 (1251) (565)	2212 (1572) (640)	2085 (1556) (529)	2328 (1615) (713)	**2356** (1642) (714)	2289 (1612) (677)	2380 (1657) (723)	2371 (1649) (722)
UltLasso	**32** (6) (26)	31 (6) (25)	**32** (6) (26)	**32** (6) (26)	**32** (6) (26)	**32** (6) (26)	32 (6) (26)	32 (6) (26)
Total	10457 (7011) (3446)	16764 (12404) (4360)	14241 (9536) (4705)	16714 (11403) (5311)	**17456** (12177) (5279)	17225 (11561) (5664)	18359 (12459) (5900)	18988 (13359) (5629)

Yices2$_{LS}$ encountered 225 (resp. 4) fewer conflicts and 27826 (resp. 36) fewer theory variables assignments than Yices2$_{base}$. Note that, on average (resp. median), Yices2$_{base}$ encountered 1669 (resp. 458) conflicts and 79842 (resp. 10537) theory variable assignments. These numbers show that there is a considerable amount of benchmarks for which the number of conflicts and theory variable

Table 2: Summary of results for $\mathcal{NIA}$ benchmarks with a timeout of 300s.

	cvc5	HybridSMT	MathSAT5	Yices2$_{base}$	Yices2$_{LS}$	Z3	Portfolio (Z3 + Yices2$_{LS}$)	Portfolio (HybridSMT + Yices2$_{LS}$)
VeryMax	10489 (7122) (3367)	$\underline{\mathbf{17060}}$ ($\underline{12093}$) (4967)	13651 (9509) (4142)	14549 (10164) (4385)	15160 (10719) (4441)	16304 (11044) ($\underline{5260}$)	16698 (11366) (5332)	17303 (12156) (5147)
calypto	173 (79) (94)	175 (78) ($\underline{97}$)	169 (79) (90)	174 (79) (95)	175 (79) (96)	$\underline{\mathbf{177}}$ ($\underline{80}$) ($\underline{97}$)	177 (80) (97)	177 (80) (97)
ezsmt	$\underline{\mathbf{8}}$ ($\underline{8}$) ($\underline{0}$)	$\underline{\mathbf{8}}$ ($\underline{8}$) ($\underline{0}$)	$\underline{\mathbf{8}}$ ($\underline{8}$) ($\underline{0}$)	$\underline{\mathbf{8}}$ ($\underline{8}$) ($\underline{0}$)	$\underline{\mathbf{8}}$ ($\underline{8}$) ($\underline{0}$)	$\underline{\mathbf{8}}$ ($\underline{8}$) ($\underline{0}$)	8 (8) (0)	8 (8) (0)
LassoRank	98 ($\underline{4}$) (94)	$\underline{\mathbf{106}}$ ($\underline{4}$) ($\underline{102}$)	105 ($\underline{4}$) (101)	93 ($\underline{4}$) (89)	97 ($\underline{4}$) (93)	$\underline{\mathbf{106}}$ ($\underline{4}$) ($\underline{102}$)	104 (4) (100)	105 (4) (101)
Dartagnan	350 (17) (333)	$\underline{\mathbf{369}}$ (14) ($\underline{355}$)	347 ($\underline{18}$) (329)	311 (7) (304)	288 (3) (285)	368 (14) (354)	363 (13) (350)	367 (13) (354)
LCTES	1 ($\underline{0}$) (1)	$\underline{\mathbf{2}}$ ($\underline{0}$) ($\underline{2}$)	1 ($\underline{0}$) (1)	0 ($\underline{0}$) (0)	0 ($\underline{0}$) (0)	$\underline{\mathbf{2}}$ ($\underline{0}$) ($\underline{2}$)	1 (0) (1)	1 (0) (1)
MathProbl	177 (170) ($\underline{7}$)	107 (100) ($\underline{7}$)	183 (176) ($\underline{7}$)	124 (117) ($\underline{7}$)	$\underline{\mathbf{311}}$ ($\underline{304}$) ($\underline{7}$)	118 (111) ($\underline{7}$)	314 (307) (7)	327 (320) (7)
leipgiz	89 (87) ($\underline{2}$)	$\underline{\mathbf{156}}$ ($\underline{155}$) (1)	126 (124) ($\underline{2}$)	104 (103) (1)	111 (110) (1)	135 (134) (1)	134 (133) (1)	154 (153) (1)
UltAut23	16 ($\underline{8}$) (0)	10 (7) (3)	17 (7) (10)	7 (7) (0)	7 (7) (0)	$\underline{\mathbf{21}}$ ($\underline{8}$) ($\underline{13}$)	21 (8) (13)	10 (7) (3)
mcm	17 (17) ($\underline{0}$)	$\underline{\mathbf{69}}$ ($\underline{69}$) ($\underline{0}$)	10 (10) ($\underline{0}$)	9 (9) ($\underline{0}$)	9 (9) ($\underline{0}$)	10 (10) ($\underline{0}$)	11 (11) (0)	64 (64) (0)
sqrtmodinv	2 ($\underline{0}$) (2)	10 ($\underline{0}$) (10)	0 ($\underline{0}$) (0)	0 ($\underline{0}$) (0)	0 ($\underline{0}$) (0)	$\underline{\mathbf{17}}$ ($\underline{0}$) ($\underline{17}$)	17 (0) (17)	8 (0) (8)
UltAut	$\underline{\mathbf{7}}$ ($\underline{0}$) ($\underline{7}$)	$\underline{\mathbf{7}}$ ($\underline{0}$) ($\underline{7}$)	$\underline{\mathbf{7}}$ ($\underline{0}$) ($\underline{7}$)	$\underline{\mathbf{7}}$ ($\underline{0}$) ($\underline{7}$)	$\underline{\mathbf{7}}$ ($\underline{0}$) ($\underline{7}$)	$\underline{\mathbf{7}}$ ($\underline{0}$) ($\underline{7}$)	7 (0) (7)	7 (0) (7)
AProVE	1939 (1330) (609)	2352 (1640) (685)	2180 (1622) (558)	2337 (1621) (716)	$\underline{\mathbf{2367}}$ ($\underline{1647}$) ($\underline{720}$)	2339 (1640) (699)	2394 (1661) (733)	2387 (1656) (731)
UltLasso	$\underline{\mathbf{32}}$ ($\underline{6}$) ($\underline{26}$)	31 ($\underline{6}$) (25)	$\underline{\mathbf{32}}$ ($\underline{6}$) ($\underline{26}$)	$\underline{\mathbf{32}}$ ($\underline{6}$) ($\underline{26}$)	$\underline{\mathbf{32}}$ ($\underline{6}$) ($\underline{26}$)	$\underline{\mathbf{32}}$ ($\underline{6}$) ($\underline{26}$)	32 (6) (26)	32 (6) (26)
Total	13398 (8848) (4550)	$\underline{\mathbf{20435}}$ ($\underline{14174}$) (6261)	16836 (11563) (5273)	17755 (12125) (5630)	18572 (12896) (5676)	19644 (13059) ($\underline{6585}$)	20281 (13597) (6684)	20950 (14467) (6483)

assignments is significantly lower (note that such a lower median w.r.t. average implies a pronounced right-skewness).

Overall, we see that, in the 24s track, Yices2$_{LS}$ solves more benchmarks than any other solver, while, in the 300s track, it comes third, after HybridSMT and Z3. The complementarity of Yices2$_{LS}$ w.r.t. both tools can be witnessed by the scatter plots in Figure 3, and by the results of the portfolios in Tables 1 and 2. Note that Z3 internally utilizes portfolio tactics that combine multiple solving techniques sequentially (clearly observable in Figures 2 and 3). HybridSMT runs a higher-level portfolio that combines the LS-based LocalSMT and Z3. The results of the portfolios that include Yices2$_{LS}$ show that our approach brings significant diversity to the strategies already used in state-of-the-art portfolio approaches.

Since HybridSMT is the only other solver that – to the best of our knowledge – leverages local search techniques for $\mathcal{NIA}$, it is interesting to compare the improvements it brings to Z3 with the improvements that Yices2$_{LS}$ brings over Yices2$_{base}$. We can see that, with a 300s time limit, the improvements are comparable, as both tools solve around 800 benchmarks more than their base solvers. With a 24s time limit, however, we see that Yices2$_{LS}$ is able to solve around 700 benchmarks more than Yices2$_{base}$, while, on the contrary, HybridSMT loses around 450 benchmarks compared to Z3. Figure 2 shows that the point at which using local search pays off is much earlier for Yices2$_{LS}$ ($< 10s$) than for HybridSMT (just below $100s$).

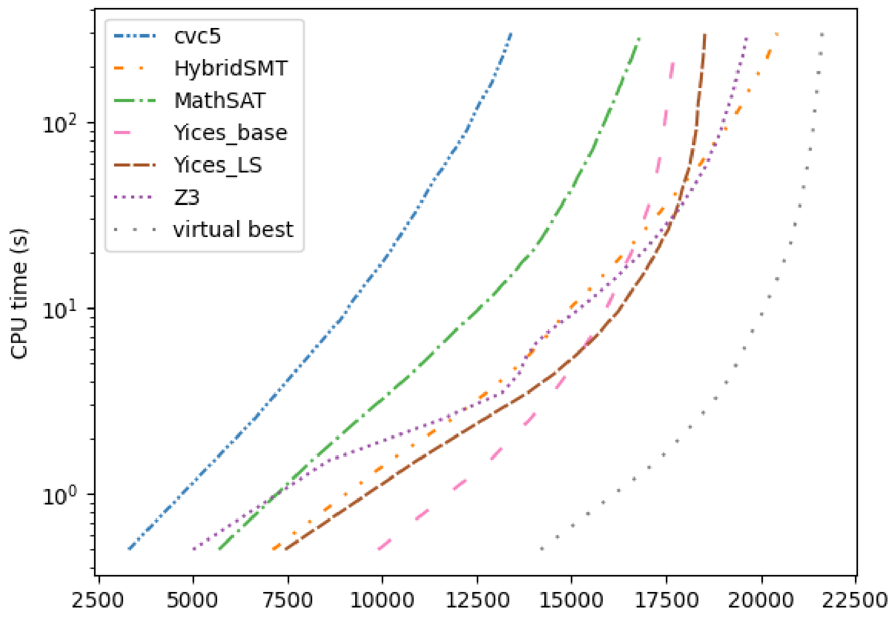

Fig. 2: Plots showing the number of instances solved (x axis) within given time in seconds (y axis) in log scale.

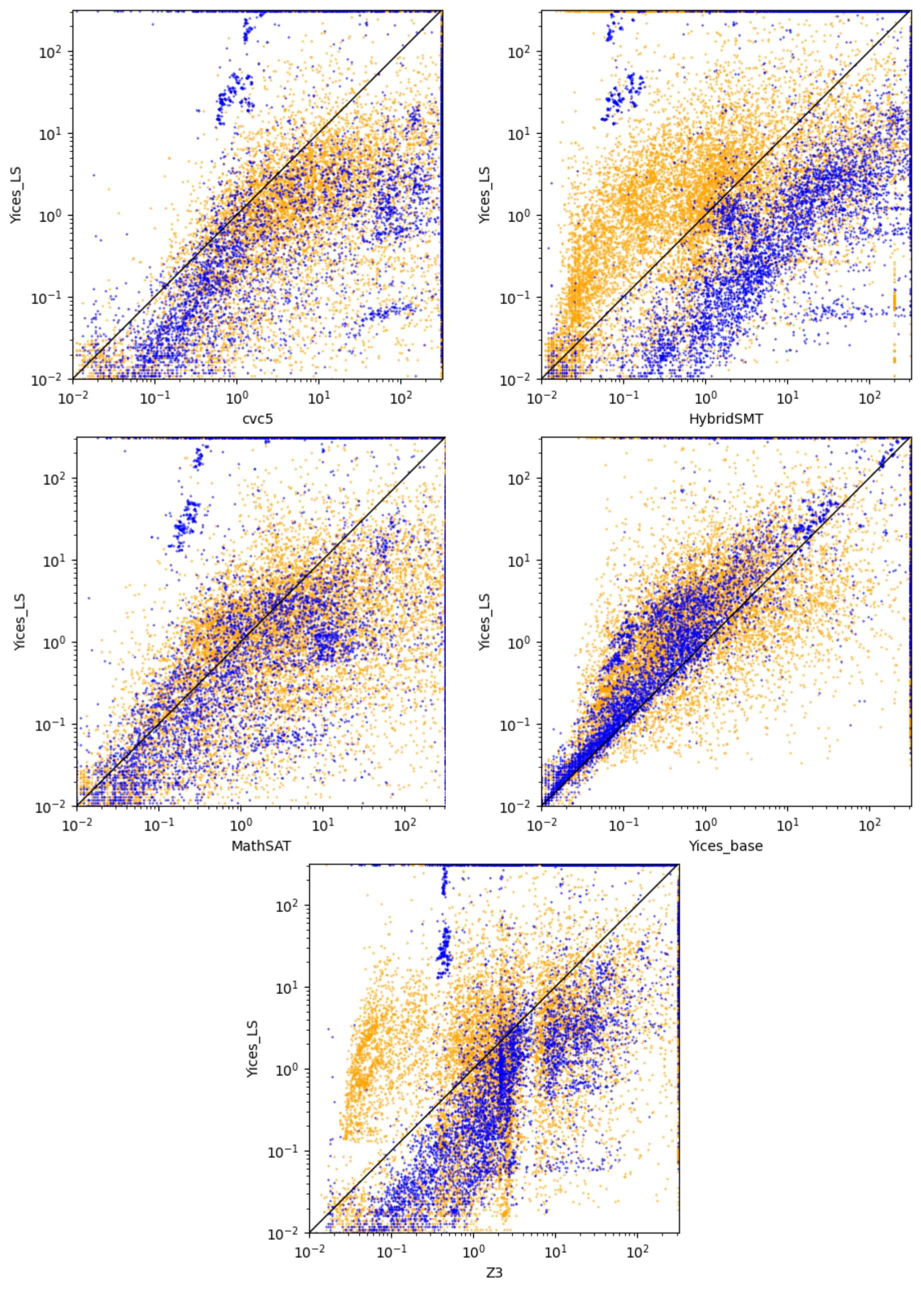

Fig. 3: Scatter plots comparing YICES2$_{LS}$ to CVC5, HYBRIDSMT, MATHSAT5, YICES2$_{base}$, and Z3, respectively; on sat. (orange) and unsat. (blue) instances.

6 Related work

In propositional SAT solving, local search techniques have been successfully used to solve difficult satisfiable problems [29] as well as unsatisfiable instances [45]. Recently, their tight integration in the propositional CDCL framework has been shown to improve performance [7,8] and are now considered a key component of state-of-the-art SAT solvers. In the context of SMT, on the other hand, the adoption of LS is a lot less widespread.

In [20], the LS-based SAT solver WalkSAT has been used in combination with a theory solver as an alternative to the classic CDCL(T) approach; however, the use of local search remained limited to the Boolean level. For the theory of bit-vectors, the idea of Boolean flips in SAT solving has been transposed to the bit level by introducing *bit-flips* [16], possibly augmented with propagations [41].

The adoption of LS for arithmetic theories is more recent. For the theories of Linear Integer Arithmetic ($\mathcal{LIA}$) [5] and Multi-linear Real Arithmetic [32] a *critical move* operation is used to change the value of a variable that appears in a literal violated by the current assignment in order to make the literal satisfied. To deal with the nonlinear arithmetic constraints, the *cell-jumping* technique is used, which first isolates the roots of a falsified polynomial w.r.t. to a variable (by fixing the value of the other variables), thus decomposing the real space into finitely many intervals (*cells*, in the CAD [12] terminology), and then tries to satisfy the polynomial by changing the value of the variable by jumping around these cells. This technique has been implemented in LocalSMT for $\mathcal{NIA}$ [6], and as a tool in Maple [33] and on top of Z3 [46] for $\mathcal{NRA}$.

Local search has also been used as a sub-routine for *global search* techniques, as in the case of floating points [17], and of $\mathcal{NRA}$ possibly augmented with transcendental functions ($\mathcal{NTA}$) [35,37,34]. In these works, numerical optimization algorithms, e.g. the gradient-descent, are used to find local minima, while stochastic jumping is used to move away from a local minimum in order to explore other regions in search for a global minimum.

All the methods discussed so far for arithmetic theories are only able to prove satisfiability; if they fail, then all the knowledge that has been acquired by the search is lost. HybridSMT [48] addresses this issue, for the case of $\mathcal{NIA}$, by integrating LocalSMT within Z3's CDCL(T). In particular, LocalSMT takes as input a subformula corresponding to a Boolean skeleton solution, and, if it does not find an integer solution for the subformula, it returns the best assignment found and the conflict frequency for atoms. This information is used to improve phase selection (i.e. Boolean assignments) and variable ordering. Although both HybridSMT and our method share the idea of integrating LS within a reasoning calculus (CDCL(T) and MCSat, respectively), there are some substantial difference. First, in HybridSMT, LS takes into account complete Boolean variable assignments. In our framework, LS can take as input both Boolean and theory variable assignments, either partial or complete. Additionally, while in HybridSMT LS can only suggest assignments for (and ordering of) Boolean literals, we extend that to theory variables as well. Moreover, there is a theory-specific difference in our approach. LocalSMT relies on cell-jumps, which require to perform potentially very expensive root isolation sub-routines at every step. In contrast, our method uses fs-jumps that rely on feasibility intervals already maintained by the theory plugin in the MCSat framework. This eliminates the need for additional computation and can be viewed as a lazy version of cells, progressively refined on-demand. Furthermore, we pair fs-jumps with hill-climbing to move inside feasible intervals.

Most state-of-the-art solvers do not use local search for $\mathcal{NIA}$ problems. Bit-blasting [18] aims at proving satisfiability by iteratively imposing bounds on the variables and then encoding the obtained sub-formula into an equi-satisfiable Boolean formula, which is then handled by a SAT solver. In the branch-and-bound approach [30,26] the integer domain is relaxed by allowing variables to range over real numbers. Incremental Linearization [9,10] leverages decision procedures for $\mathcal{LIA}$ by abstracting non-linear multiplications with uninterpreted functions and then incrementally axiomatize them.

7 Conclusion

In this work, we have introduced a theory-independent framework for integrating local search into the MCSat calculus. By combining local search intuition with MCSat reasoning capabilities, our approach leverages logic-to-optimization formalization to provide guidance to the core MCSat solver. Specifically, we addressed the theory of nonlinear integer arithmetic by proposing a local search procedure based on feasibility-set jumping and hill-climbing.

We implemented our approach in the Yices2 SMT solver, and empirically demonstrated its improvements for both satisfiable and unsatisfiable instances. Our results show that the new Yices2 solver with local search compares favorably and often outperforms other SMT solvers; in particular, it manages to solve a significant amount of benchmarks not solved by other state-of-the-art tools. Moreover, our results show that the new Yices2 is able to reach solutions or proofs more efficiently, compared to the baseline Yices2, in terms of the number of decisions and conflicts. Our findings indicate that this new approach complements existing techniques used by other solvers, as evident in the experimental results (see virtual best and portfolio solvers in the plots and tables).

In the future, we aim to extend our approach to other theories such as finite fields and bit-vectors, and conduct more comprehensive experimental evaluations. Additionally, we plan to integrate this approach with different caching schemes, including value and target caches, along with periodic recaching in MCSat [21].

Acknowledgments. This material is based upon work supported in part by NSF grant 2016597. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the US Government or NSF.
The work of Enrico Lipparini was partially supported by project PRIN 2022 DeLiCE (F53D23009130001) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength SMT solver. In: Fisman, D., Rosu, G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13243, pp. 415–442. Springer (2022). `https://doi.org/10.1007/978-3-030-99524-9_24`, `https://doi.org/10.1007/978-3-030-99524-9_24`
2. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org` (2016)
3. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability Modulo Theories. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 1267–1329. IOS Press (2021). `https://doi.org/10.3233/FAIA201017`, `https://doi.org/10.3233/FAIA201017`
4. Bjørner, N., Nachmanson, L.: Arithmetic solving in z3. In: Gurfinkel, A., Ganesh, V. (eds.) Computer Aided Verification. pp. 26–41. Springer Nature Switzerland, Cham (2024). `https://doi.org/10.1007/978-3-031-65627-9_2`
5. Cai, S., Li, B., Zhang, X.: Local Search for SMT on Linear Integer Arithmetic. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification. pp. 227–248. Springer International Publishing, Cham (2022). `https://doi.org/10.1007/978-3-031-13188-2_12`
6. Cai, S., Li, B., Zhang, X.: Local search for satisfiability modulo integer arithmetic theories. ACM Trans. Comput. Logic **24**(4) (Jul 2023). `https://doi.org/10.1145/3597495`, `https://doi.org/10.1145/3597495`
7. Cai, S., Zhang, X.: Deep cooperation of CDCL and local search for SAT. In: Li, C.M., Manyà, F. (eds.) Theory and Applications of Satisfiability Testing – SAT 2021. pp. 64–81. Springer International Publishing, Cham (2021). `https://doi.org/10.1007/978-3-030-80223-3_6`
8. Cai, S., Zhang, X., Fleury, M., Biere, A.: Better decision heuristics in CDCL through local search and target phases. J. Artif. Int. Res. **74** (Sep 2022). `https://doi.org/10.1613/jair.1.13666`, `https://doi.org/10.1613/jair.1.13666`
9. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Experimenting on solving nonlinear integer arithmetic with incremental linearization. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) Theory and Applications of Satisfiability Testing – SAT 2018. pp. 383–398. Springer International Publishing, Cham (2018). `https://doi.org/10.1007/978-3-319-94144-8_23`
10. 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. Logic **19**(3) (aug 2018). `https://doi.org/10.1145/3230639`, `https://doi.org/10.1145/3230639`

11. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT Solver. In: Piterman, N., Smolka, S. (eds.) Proceedings of TACAS. LNCS, vol. 7795. Springer (2013). `https://doi.org/10.1007/978-3-642-36742-7_7`
12. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In: Brakhage, H. (ed.) Automata Theory and Formal Languages. pp. 134–183. Springer Berlin Heidelberg, Berlin, Heidelberg (1975)
13. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Ábrahám, E.: SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving. In: SAT (09 2015). `https://doi.org/10.1007/978-3-319-24318-4_26`
14. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. p. 337–340. TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg (2008). `https://doi.org/10.5555/1792734.1792766`
15. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Computer-Aided Verification (CAV'2014). Lecture Notes in Computer Science, vol. 8559, pp. 737–744. Springer (July 2014)
16. Fröhlich, A., Biere, A., Wintersteiger, C., Hamadi, Y.: Stochastic local search for Satisfiability Modulo Theories. Proceedings of the AAAI Conference on Artificial Intelligence **29**(1) (Feb 2015). `https://doi.org/10.1609/aaai.v29i1.9372`, `https://ojs.aaai.org/index.php/AAAI/article/view/9372`
17. Fu, Z., Su, Z.: XSat: A fast floating-point satisfiability solver. In: CAV. Lecture Notes in Computer Science, vol. 9780, pp. 187–209. Springer (2016). `https://doi.org/10.1007/978-3-319-41540-6_11`
18. Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: SAT solving for termination analysis with polynomial interpretations. In: Marques-Silva, J., Sakallah, K.A. (eds.) Theory and Applications of Satisfiability Testing – SAT 2007. pp. 340–354. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). `https://doi.org/10.1007/978-3-540-72788-0_33`
19. Graham-Lengrand, S., Jovanovic, D., Dutertre, B.: Solving bitvectors with MC-SAT: explanations from bits and pieces. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) Intl. Joint Conf. on Automated Reasoning (IJCAR), Part I. LNCS, vol. 12166, pp. 103–121. Springer (2020). `https://doi.org/10.1007/978-3-030-51074-9_7`, `https://doi.org/10.1007/978-3-030-51074-9_7`
20. 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.) Frontiers of Combining Systems. pp. 163–178. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). `https://doi.org/10.1007/978-3-642-24364-6_12`
21. Hader, T., Irfan, A., Graham-Lengrand, S.: Decision heuristics in MCSat *(to appear)*. In: Computer Aided Verification (2025)
22. Hader, T., Kaufmann, D., Irfan, A., Graham-Lengrand, S., Kovács, L.: MCSat-based finite field reasoning in the Yices2 SMT solver (short paper). In: IJCAR (1). Lecture Notes in Computer Science, vol. 14739, pp. 386–395. Springer (2024)
23. Hader, T., Kaufmann, D., Kovács, L.: SMT solving over finite field arithmetic. In: LPAR. EPiC Series in Computing, vol. 94, pp. 238–256. EasyChair (2023)
24. Hernando, L., Mendiburu, A., Lozano, J.: Hill-climbing algorithm: Let's go for a walk before finding the optimum. pp. 1–7 (07 2018). `https://doi.org/10.1109/CEC.2018.8477836`
25. Irfan, A., Graham-Lengrand, S.: Arrays reasoning in MCSat. In: SMT@CAV. CEUR Workshop Proceedings, vol. 3725, pp. 24–35. CEUR-WS.org (2024)

26. Jovanović, D.: Solving nonlinear integer arithmetic with MCSAT. In: Bouajjani, A., Monniaux, D. (eds.) Verification, Model Checking, and Abstract Interpretation. pp. 330–346. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-52234-0_18
27. Jovanovic, D., Barrett, C., de Moura, L.: The design and implementation of the model constructing satisfiability calculus. In: Intl. Conf on Formal Methods in Computer-Aided Design (FMCAD). pp. 173–180. IEEE (2013). https://doi.org/10.1109/FMCAD.2013.7027033
28. Jovanović, D., de Moura, L.: Solving non-linear arithmetic. ACM Commun. Comput. Algebra **46**(3/4), 104–105 (jan 2013). https://doi.org/10.1145/2429135.2429155, https://doi.org/10.1145/2429135.2429155
29. Kautz, H.A., Sabharwal, A., Selman, B.: Incomplete algorithms. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 213–232. IOS Press (2021). https://doi.org/10.3233/FAIA200989, https://doi.org/10.3233/FAIA200989
30. Kremer, G., Corzilius, F., Ábrahám, E.: A generalised branch-and-bound approach and its application in SAT modulo nonlinear integer arithmetic. In: Gerdt, V.P., Koepf, W., Seiler, W.M., Vorozhtsov, E.V. (eds.) Computer Algebra in Scientific Computing - 18th International Workshop, CASC 2016, Bucharest, Romania, September 19-23, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9890, pp. 315–335. Springer (2016). https://doi.org/10.1007/978-3-319-45641-6_21, https://doi.org/10.1007/978-3-319-45641-6_21
31. Kremer, G., Ábrahám, E.: Modular strategic SMT solving with SMT-RAT. Acta Universitatis Sapientiae, Informatica **10**(1), 5–25 (2018)
32. Li, B., Cai, S.: Local search for SMT on linear and multi-linear real arithmetic. In: 2023 Formal Methods in Computer-Aided Design (FMCAD). pp. 1–10 (2023). https://doi.org/10.34727/2023/isbn.978-3-85448-060-0_25
33. Li, H., Xia, B., Zhao, T.: Local search for solving satisfiability of polynomial formulas. In: Enea, C., Lal, A. (eds.) Computer Aided Verification. pp. 87–109. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-37703-7_5
34. Lipparini, E.: Satisfiability modulo Nonlinear Arithmetic and Transcendental Functions via Numerical and Topological methods. Ph.D. thesis (2024). https://doi.org/10.15167/lipparini-enrico_phd2024-12-10
35. Lipparini, E., Cimatti, A., Griggio, A., 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. pp. 137–153. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-19992-9_9
36. Lipparini, E., Ratschan, S.: Satisfiability of non-linear transcendental arithmetic as a certificate search problem. In: Rozier, K.Y., Chaudhuri, S. (eds.) NASA Formal Methods. pp. 472–488. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-33170-1_29
37. Lipparini, E., Ratschan, S.: Satisfiability of non-linear transcendental arithmetic as a certificate search problem. J. Autom. Reason. **69**(1) (Jan 2025). https://doi.org/10.1007/s10817-024-09716-3, https://doi.org/10.1007/s10817-024-09716-3
38. Matiyasevich, Y.V.: Hilbert's tenth problem. MIT Press, Cambridge, MA, USA (1993). https://doi.org/10.5555/164759

39. de Moura, L., Jovanovic, D.: A model-constructing satisfiability calculus. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) Intl. Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI). LNCS, vol. 7737, pp. 1–12. Springer (2013). https://doi.org/10.1007/978-3-642-35873-9_1, https://doi.org/10.1007/978-3-642-35873-9_1
40. de Moura, L.M., Bjørner, N.S.: Satisfiability modulo theories: introduction and applications. Commun. ACM **54**(9), 69–77 (2011)
41. Niemetz, A., Preiner, M., Biere, A.: Propagation based local search for bit-precise reasoning. Formal Methods in System Design **51**(3), 608–636 (Dec 2017). https://doi.org/10.1007/s10703-017-0295-6, https://doi.org/10.1007/s10703-017-0295-6
42. 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 (Nov 2006). https://doi.org/10.1145/1217856.1217859, https://doi.org/10.1145/1217856.1217859
43. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In: Theory and Applications of Satisfiability Testing–SAT 2007: 10th International Conference, Lisbon, Portugal, May 28-31, 2007. Proceedings 10. pp. 294–299. Springer (2007)
44. Preiner, M., Schurr, H.J., Barrett, C., Fontaine, P., Niemetz, A., Tinelli, C.: SMT-LIB release 2024 (non-incremental benchmarks) (Apr 2024). https://doi.org/10.5281/zenodo.11061097, https://doi.org/10.5281/zenodo.11061097
45. Prestwich, S., Lynce, I.: Local search for unsatisfiability. In: Biere, A., Gomes, C.P. (eds.) Theory and Applications of Satisfiability Testing - SAT 2006. pp. 283–296. Springer Berlin Heidelberg, Berlin, Heidelberg (2006). https://doi.org/10.1007/11814948_28
46. Wang, Z., Zhan, B., Li, B., Cai, S.: Efficient local search for nonlinear real arithmetic. In: Dimitrova, R., Lahav, O., Wolff, S. (eds.) Verification, Model Checking, and Abstract Interpretation. pp. 326–349. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-50524-9_15
47. Weber, T., Conchon, S., Déharbe, D., Heizmann, M., Niemetz, A., Reger, G.: The SMT competition 2015-2018. J. Satisf. Boolean Model. Comput. **11**(1), 221–259 (2019), https://doi.org/10.3233/SAT190123
48. Zhang, X., Li, B., Cai, S.: Deep combination of CDCL(T) and local search for satisfiability modulo non-linear integer arithmetic theory. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. ICSE '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3597503.3639105, https://doi.org/10.1145/3597503.3639105