Stochastic Local Search for SMT: a Preliminary Report (Extended Abstract)

Alberto Griggio, Roberto Sebastiani, and Silvia Tomasi

DISI, Università di Trento, Italy.
1. Motivations and goals
2. Background
3. Stochastic Local Search for SMT
   - WALKSMT: basic schema
   - Enhancements
4. Preliminary Experimental Evaluation
   - Experiments on SMT-LIB Instances
   - Experiments on Random Instances
5. Conclusions and potential research directions
Outline

1 Motivations and goals

2 Background

3 Stochastic Local Search for SMT
   - WALKSMT: basic schema
   - Enhancements

4 Preliminary Experimental Evaluation
   - Experiments on SMT-LIB Instances
   - Experiments on Random Instances

5 Conclusions and potential research directions
Motivations and Goals

Motivations

- “Lazy” approach to SMT: integration of DPLL and a $T$-solver.
- SAT: stochastic local-search (SLS) procedures sometimes outperform DPLL on satisfiable instances (unstructured problems).
- Therefore, it is a natural research question to wonder whether SLS can be exploited successfully also inside SMT tools.

Goal

To start investigating the issue of using SLS in SMT.
Motivations and Goals

Motivations

- “Lazy” approach to SMT: integration of DPLL and a $T$-solver.
- SAT: stochastic local-search (SLS) procedures sometimes outperform DPLL on satisfiable instances (unstructured problems).
- Therefore, it is a natural research question to wonder whether SLS can be exploited successfully also inside SMT tools.

Goal

To start investigating the issue of using SLS in SMT.
Outline

1 Motivations and goals

2 Background

3 Stochastic Local Search for SMT
   - WALKSMT: basic schema
   - Enhancements

4 Preliminary Experimental Evaluation
   - Experiments on SMT-LIB Instances
   - Experiments on Random Instances

5 Conclusions and potential research directions
Stochastic Local Search for SAT

**Local Search (LS) algorithms**
- typically incomplete
- start at some location of the search space
- iteratively move from the current to a neighbouring location taking decisions on the base of local information

**Stochastic Local Search (SLS) algorithms**
- LS algorithms make randomized choices during the search
- successfully applied to $\mathcal{NP}$-complete decision problems
- for SAT, WalkSAT is a popular (family of) SLS-based algorithm(s)
Local Search (LS) algorithms
- typically incomplete
- start at some location of the search space
- iteratively move from the current to a neighbouring location taking decisions on the base of local information

Stochastic Local Search (SLS) algorithms
- LS algorithms make randomized choices during the search
- successfully applied to \( \mathcal{NP} \)-complete decision problems
- for SAT, WalkSAT is a popular (family of) SLS-based algorithm(s)
The WalkSAT Family Schema

Require: CNF formula \( \varphi \), MAX_TRIES, MAX_FLIPS

1: for \( i = 1 \) to MAX TRIES do
2: \( \mu \leftarrow \text{INITIALTRUTHASSIGNMENT}(\varphi) \)
3: for \( j = 1 \) to MAX_FLIPS do
4: if \( (\mu \models \varphi) \) then
5: return SAT
6: else
7: \( c \leftarrow \text{CHOOSE_UNSATISFIEDCLAUSE}(\varphi, \mu) \)
8: \( \mu \leftarrow \text{NEXTTRUTHASSIGNMENT}(\varphi, c, \mu) \)
9: end if
10: end for
11: end for
12: return UNKNOWN

- differ for different techniques for \text{INITIALTRUTHASSIGNMENT}, \text{CHOOSE_UNSATISFIEDCLAUSE} and \text{NEXTTRUTHASSIGNMENT}
- different degrees and forms of greediness and randomness
The WalkSAT Family Schema

Require: CNF formula $\varphi$, MAX_TRIES, MAX_FLIPS

1: $\text{for } i = 1 \text{ to } \text{MAX\_TRIES} \text{ do}$
2: $\mu \leftarrow \text{INITIAL\_TRUTH\_ASSIGNMENT}(\varphi)$
3: $\text{for } j = 1 \text{ to } \text{MAX\_FLIPS} \text{ do}$
4: $\text{if } (\mu \models \varphi) \text{ then}$
5: $\text{return SAT}$
6: $\text{else}$
7: $c \leftarrow \text{CHOOSE\_UNSATISFIED\_CLAUSE}(\varphi, \mu)$
8: $\mu \leftarrow \text{NEXT\_TRUTH\_ASSIGNMENT}(\varphi, c, \mu)$
9: $\text{end if}$
10: $\text{end for}$
11: $\text{end for}$
12: $\text{return UNKNOWN}$

- differ for different techniques for \text{INITIAL\_TRUTH\_ASSIGNMENT}, \text{CHOOSE\_UNSATISFIED\_CLAUSE} and \text{NEXT\_TRUTH\_ASSIGNMENT}
- different degrees and forms of greediness and randomness
Outline

1. Motivations and goals
2. Background
3. Stochastic Local Search for SMT
   - \textit{WALKSMT}: basic schema
   - Enhancements
4. Preliminary Experimental Evaluation
   - Experiments on SMT-LIB Instances
   - Experiments on Random Instances
5. Conclusions and potential research directions
SMT from a SAT perspective

SMT on $\varphi$: solve a *partially-invisible* SAT formula $\varphi^p \land \tau^p$ s.t.
- $\varphi^p$ is the “visible” part (Boolean abstraction of $\varphi$)
- $\tau^p$ is the “invisible” part (the B.a. of the set of $\mathcal{T}$-lemmas induced by the theory $\mathcal{T}$ on the atoms of $\varphi$)

In traditional “lazy” SMT solvers
- DPLL solver knows $\varphi^p$ but not $\tau^p$ ($\mathcal{T}$-solver “knows” $\tau^p$)
- whenever $\mu^p \models \varphi^p$, $\mathcal{T}$-solver checks whether $\mu^p$ falsifies $\tau^p$ and returns one falsified clause $c^p$ in $\tau^p$
- DPLL uses $c^p$ to drive the future search (optionally add it to $\varphi^p$)

[the superscript $^p$ denotes the Boolean abstraction of a $\mathcal{T}$-formula]
SMT from a SAT perspective

SMT on \( \varphi \): solve a \textit{partially-invisible} SAT formula \( \varphi^p \land \tau^p \) s.t.
- \( \varphi^p \) is the “visible” part (Boolean abstraction of \( \varphi \))
- \( \tau^p \) is the “invisible” part (the B.a. of the set of \( \mathcal{T} \)-lemmas induced by the theory \( \mathcal{T} \) on the atoms of \( \varphi \))

In traditional “lazy” SMT solvers
- DPLL solver knows \( \varphi^p \) but not \( \tau^p \) (\( \mathcal{T} \)-solver “knows” \( \tau^p \))
- whenever \( \mu^p \models \varphi^p \), \( \mathcal{T} \)-solver checks whether \( \mu^p \) falsifies \( \tau^p \) and returns one falsified clause \( c^p \) in \( \tau^p \)
- DPLL uses \( c^p \) to drive the future search (optionally add it to \( \varphi^p \))

[the superscript \( p \) denotes the Boolean abstraction of a \( \mathcal{T} \)-formula]
Motivations and goals

Background

Stochastic Local Search for SMT
- \textit{WALKSMT: basic schema}
- Enhancements

Preliminary Experimental Evaluation
- Experiments on SMT-LIB Instances
- Experiments on Random Instances

Conclusions and potential research directions
Basic-WalkSMT Schema

Require: SMT($T$) CNF formula $\varphi$, MAXTRIES, MAXFLIPS

1: \textbf{for} $i = 1$ to MAXTRIES \textbf{do}
2: \hspace{1em} $\mu^p \leftarrow \text{INITIALTRUTHASSIGNMENT}(\varphi^p)$
3: \hspace{1em} \textbf{for} $j = 1$ to MAXFLIPS \textbf{do}
4: \hspace{2em} \textbf{if} ($\mu^p \models \varphi^p$) \textbf{then}
5: \hspace{3em} $\langle \text{status}, c^p \rangle \leftarrow T$-solver($\varphi^p, \mu^p$)
6: \hspace{3em} \textbf{if} (status $==$ SAT) \textbf{then}
7: \hspace{4em} \textbf{return} SAT
8: \hspace{3em} \textbf{end if}
9: \hspace{3em} $\mu^p \leftarrow \text{NEXTTRUTHASSIGNMENT}(\varphi^p, c^p, \mu^p)$
10: \hspace{2em} \textbf{else}
11: \hspace{3em} $c^p \leftarrow \text{CHOOSEUNSATISFIEDCLAUSE}(\varphi^p, \mu^p)$
12: \hspace{3em} $\mu^p \leftarrow \text{NEXTTRUTHASSIGNMENT}(\varphi^p, c^p, \mu^p)$
13: \hspace{2em} \textbf{end if}
14: \hspace{1em} \textbf{end for}
15: \hspace{1em} \textbf{end for}
16: \hspace{1em} \textbf{return} UNKNOWN

Intuition: $T$-solver plays the role of CHOOSEUNSATISFIEDCLAUSE on $\varphi^p \land \tau^p$ when no unsatisfied clause is found in $\varphi^p$. 
BASIC-WALK\textsc{SMT} Schema

Require: SMT(\(T\)) CNF formula \(\varphi\), MAX\_TRIES, MAX\_FLIPS

1: \textbf{for} \(i = 1\) to MAX\_TRIES \textbf{do}
2: \(\mu^p \leftarrow \text{INITIALTRUTHASSIGNMENT}(\varphi^p)\)
3: \textbf{for} \(j = 1\) to MAX\_FLIPS \textbf{do}
4: \textbf{if} \(\mu^p \models \varphi^p\) \textbf{then}
5: \(\langle \text{status}, c^p \rangle \leftarrow T\)-solver(\(\varphi^p, \mu^p\))
6: \textbf{if} \(\text{status == SAT}\) \textbf{then}
7: \hspace{1em} \text{return SAT}
8: \textbf{end if}
9: \(\mu^p \leftarrow \text{NEXTTRUTHASSIGNMENT}(\varphi^p, c^p, \mu^p)\)
10: \textbf{else}
11: \(c^p \leftarrow \text{CHOOSEUNSATISFIEDCLAUSE}(\varphi^p, \mu^p)\)
12: \(\mu^p \leftarrow \text{NEXTTRUTHASSIGNMENT}(\varphi^p, c^p, \mu^p)\)
13: \textbf{end if}
14: \textbf{end for}
15: \textbf{end for}
16: \textbf{return} UNKNOWN

Intuition: \(T\)-solver plays the role of \textsc{ChooseUnsatisfiedClause} on \(\varphi^p \wedge \tau^p\) when no unsatisfied clause is found in \(\varphi^p\).
Outline

1. Motivations and goals
2. Background
3. Stochastic Local Search for SMT
   - WALKSMT: basic schema
   - Enhancements
4. Preliminary Experimental Evaluation
   - Experiments on SMT-LIB Instances
   - Experiments on Random Instances
5. Conclusions and potential research directions
Preprocessing

Simplify the input formula $\varphi$ through:

**unit propagation**

- unit-propagate each literal occurring as unit clause in $\varphi$
- add to $\varphi$ the conjunction of all *non-propositional* unit literals eliminated

$\implies$ eliminates purely-propositional variables (and possibly others)

**static learning**

- augment $\varphi$ with short “obvious” $T$-lemmas generated without invoking the $T$-solver (e.g., $\neg (x > y) \lor \neg (y > z) \lor (x > z)$)

$\implies$ prevents investigating obviously-inconsistent assignments
Preprocessing

Simplify the input formula $\varphi$ through:

**unit propagation**

- unit-propagate each literal occurring as unit clause in $\varphi$
- add to $\varphi$ the conjunction of all non-propositional unit literals eliminated

$\implies$ eliminates purely-propositional variables (and possibly others)

**static learning**

- augment $\varphi$ with short “obvious” $T$-lemmas generated without invoking the $T$-solver (e.g., $\neg(x > y) \lor \neg(y > z) \lor (x > z)$)

$\implies$ prevents investigating obviously-inconsistent assignments
Preprocessing

Simplify the input formula $\phi$ through:

**unit propagation**
- unit-propagate each literal occurring as unit clause in $\phi$
- add to $\phi$ the conjunction of all **non-propositional** unit literals eliminated

$\implies$ eliminates purely-propositional variables (and possibly others)

**static learning**
- augment $\phi$ with short “obvious” $T$-lemmas generated without invoking the $T$-solver (e.g., $\neg(x > y) \lor \neg(y > z) \lor (x > z)$)

$\implies$ prevents investigating obviously-inconsistent assignments
Learning and Unit simplification

Learning

- Learn the $\mathcal{T}$-lemmas generated by the $\mathcal{T}$-solver

$\Rightarrow$ avoid finding the same $\mathcal{T}$-conflict multiple times

Unit simplification

- Before returning a $\mathcal{T}$-lemma, remove from it (set them to TRUE) all the literals occurring as unit clauses in the (preprocessed) input problem.

$\Rightarrow$ avoid useless flips on these variables
Learning and Unit simplification

**Learning**

- *Learn* the \( \mathcal{T} \)-lemmas generated by the \( \mathcal{T} \)-solver

\[ \Rightarrow \] avoid finding the same \( \mathcal{T} \)-conflict multiple times

**Unit simplification**

- Before returning a \( \mathcal{T} \)-lemma, remove from it (set them to TRUE) all the literals occurring as unit clauses in the (preprocessed) input problem.

\[ \Rightarrow \] avoid useless flips on these variables
Learning and Unit simplification

### Learning
- *Learn* the $\mathcal{T}$-lemmas generated by the $\mathcal{T}$-solver
- $\implies$ avoid finding the same $\mathcal{T}$-conflict multiple times

### Unit simplification
- Before returning a $\mathcal{T}$-lemma, remove from it (set them to `TRUE`) all the literals occurring as unit clauses in the (preprocessed) input problem.
- $\implies$ avoid useless flips on these variables
Learning and Unit simplification

**Require:** SMT($T$) CNF formula $\varphi$, MAX_TRIES, MAX_FLIPS

1: for $i = 1$ to MAX_TRIES do
2:  $\mu^p \leftarrow$ INITIALTRUTHASSIGNMENT ($\varphi^p$)
3:  for $j = 1$ to MAX_FLIPS do
4:    if ($\mu^p \models \varphi^p$) then
5:      \langle status, c^p \rangle \leftarrow T$-solver ($\varphi^p, \mu^p$)
6:      if (status == SAT) then
7:        return SAT
8:    end if
9:    $c^p \leftarrow$ UNIT-SIMPLIFICATION($\varphi^p, c^p$)
10:   $\varphi^p \leftarrow \varphi^p \land c^p$
11:   $\mu^p \leftarrow$ NEXTTRUTHASSIGNMENT ($\varphi^p, c^p, \mu^p$)
12:  else
13:    $c^p \leftarrow$ CHOOSEUNSATISFIEDCLAUSE ($\varphi^p, \mu^p$)
14:    $\mu^p \leftarrow$ NEXTTRUTHASSIGNMENT ($\varphi^p, c^p, \mu^p$)
15:  end if
16: end for
17: end for
18: return UNKNOWN
Filtering the assignments given to $\mathcal{T}$-solver

**pure-literal filtering**

If non-Boolean $\mathcal{T}$-atoms occur only positively [negatively] in the original formula $\varphi$, drop every negative [positive] occurrence of them from $\mu^p$.

$\Rightarrow$

- reduce the work of $\mathcal{T}$-solver by removing $\mathcal{T}$-atoms from the assignment $\mu^p$ to be checked,
- improve the chances of finding a $\mathcal{T}$-consistent assignment.
Filtering the assignments given to $\mathcal{T}$-solver

**pure-literal filtering**

If non-Boolean $\mathcal{T}$-atoms occur only positively [negatively] in the original formula $\varphi$, drop every negative [positive] occurrence of them from $\mu^p$ to

- reduce the work of $\mathcal{T}$-solver by removing $\mathcal{T}$-atoms from the assignment $\mu^p$ to be checked,
- improve the chances of finding a $\mathcal{T}$-consistent assignment
Filtering the assignments given to \( T \)-solver

**pure-literal filtering**

If non-Boolean \( T \)-atoms occur only positively [negatively] in the original formula \( \varphi \), drop every negative [positive] occurrence of them from \( \mu^p \)

\[ \Rightarrow \]

- reduce the work of \( T \)-solver by removing \( T \)-atoms from the assignment \( \mu^p \) to be checked,
- improve the chances of finding a \( T \)-consistent assignment
Learning multiple $\mathcal{T}$-lemmas

Idea: Since $\mu^p$ are total assignments, they may be $\mathcal{T}$-inconsistent for several different reasons.

$\implies$ Learn more than one $\mathcal{T}$-lemma.

Multiple learning

- invoke the $\mathcal{T}$-solver on $\mu$ to find a new conflict set $\eta$

- if a conflict set $\eta$ is found,
  - unit-simplify and learn the $\mathcal{T}$-lemma $\neg \eta$
  - compute a sub-assignment $\mu'$ by removing from $\mu$ (part or all) the literals occurring in $\neg \eta$

- invoke the $\mathcal{T}$-solver on $\mu'$ to find a new conflict set $\eta'$, etc.

Note: CHOOSE_UNSATISFIED-Clause picks randomly one of the $\neg \eta$s
Learning multiple $\mathcal{T}$-lemmas

Idea: Since $\mu^p$ are total assignments, they may be $\mathcal{T}$-inconsistent for several different reasons.

$\implies$ Learn more than one $\mathcal{T}$-lemma.

Multiple learning

- invoke the $\mathcal{T}$-solver on $\mu$ to find a new conflict set $\eta$

- if a conflict set $\eta$ is found,
  - unit-simplify and learn the $\mathcal{T}$-lemma $\neg\eta$
  - compute a sub-assignment $\mu'$ by removing from $\mu$ (part or all) the literals occurring in $\neg\eta$

- invoke the $\mathcal{T}$-solver on $\mu'$ to find a new conflict set $\eta'$, etc.

Note: CHOOSEUNSATISFIEDCLAUSE picks randomly one of the $\neg\eta$s
Learning multiple $\mathcal{T}$-lemmas

Idea: Since $\mu^P$ are total assignments, they may be $\mathcal{T}$-inconsistent for several different reasons.

$\Longrightarrow$ Learn more than one $\mathcal{T}$-lemma.

### Multiple learning

- invoke the $\mathcal{T}$-solver on $\mu$ to find a new conflict set $\eta$
- if a conflict set $\eta$ is found,
  - unit-simplify and learn the $\mathcal{T}$-lemma $\neg \eta$
  - compute a sub-assignment $\mu'$ by removing from $\mu$ (part or all) the literals occurring in $\neg \eta$
- invoke the $\mathcal{T}$-solver on $\mu'$ to find a new conflict set $\eta'$, etc.

Note: **CHOOSEUNSATISFIEDCLAUSE** picks randomly one of the $\neg \eta$s
Efficient $\mathcal{T}$-solvers for LS

DPLL-based SMT solvers
- truth assignments are updated in a stack-based manner
- $\mathcal{T}$-solvers typically incremental and backtrackable

SLS-based SMT solvers
- truth assignments are updated by flipping an arbitrary literal
- backtrackable $\mathcal{T}$-solvers are of little use
- it is desirable to be able to remove and add arbitrary literals from a $\mathcal{T}$-solver without the need of resetting its internal state (some $\mathcal{T}$-solvers for $\mathcal{DL}$ and $\mathcal{LA}(\mathbb{Q})$ have this capability)
Efficient $\mathcal{T}$-solvers for LS

<table>
<thead>
<tr>
<th>DPLL-based SMT solvers</th>
</tr>
</thead>
<tbody>
<tr>
<td>- truth assignments are updated in a <strong>stack-based</strong> manner</td>
</tr>
<tr>
<td>- $\mathcal{T}$-solvers typically <strong>incremental</strong> and <strong>backtrackable</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SLS-based SMT solvers</th>
</tr>
</thead>
<tbody>
<tr>
<td>- truth assignments are updated by flipping an <strong>arbitrary</strong> literal</td>
</tr>
<tr>
<td>- <strong>backtrackable</strong> $\mathcal{T}$-solvers are of little use</td>
</tr>
<tr>
<td>- it is desirable to be able to remove and add <strong>arbitrary</strong> literals from a $\mathcal{T}$-solver without the need of resetting its internal state</td>
</tr>
<tr>
<td>(some $\mathcal{T}$-solvers for $\mathcal{DL}$ and $\mathcal{LA}(\mathbb{Q})$ have this capability)</td>
</tr>
</tbody>
</table>
Outline

1. Motivations and goals
2. Background
3. Stochastic Local Search for SMT
   - WALKSMT: basic schema
   - Enhancements
4. Preliminary Experimental Evaluation
   - Experiments on SMT-LIB Instances
   - Experiments on Random Instances
5. Conclusions and potential research directions
A prototype SLS-based SMT solver for $\mathcal{LA}(\mathbb{Q})$: \textsc{WalkSMT}

Implementation

- implemented in C++
- built on top of \textsc{UbcSat} SLS platform [Tompkins & Hoos’04]
  - implements many existing SLS procedures
  - after various attempts and preliminary testing, we selected the \textsc{AdaptiveNovelty+} procedure
- uses MathSAT preprocessor and $\mathcal{LA}(\mathbb{Q})$-solver
- implements the optimizations previously described

Execution

- on a 2.66GHz 4GB RAM Xeon machine on linux
- 600s timeout
- multiple runs with different seeds for each formula
A prototype SLS-based SMT solver for $\mathcal{L}A(\mathbb{Q})$: \textsc{WalkSMT}

**Implementation**

- implemented in C++
- built on top of \textsc{Ubcsat} SLS platform [Tompkins & Hoos’04]
  - implements many existing SLS procedures
  - after various attempts and preliminary testing, we selected the \textsc{AdaptiveNovelty+} procedure
- uses MathSAT preprocessor and $\mathcal{L}A(\mathbb{Q})$-solver
- implements the optimizations previously described

**Execution**

- on a 2.66GHz 4GB RAM Xeon machine on linux
- 600s timeout
- multiple runs with different seeds for each formula
Comparison: \textsc{WalkSMT} vs. MathSAT

**SLS-based SMT solver**

- **Basic-WalkSMT** has no optimizations
- **Learning-WalkSMT** combines **Basic-WalkSMT** with preprocessing, unit simplification and learning
- **Best-WalkSMT** extends **Learning-WalkSMT** with multiple learning, pure-literal filtering optimizations

**DPLL-based SMT solver**

- MathSAT with all the optimizations enabled,
- MathSAT with early pruning and $\tau$-propagation disabled
Comparison: \textsc{WalkSMT} vs. MathSAT

\begin{itemize}
\item \textsc{Basic-WalkSMT} has no optimizations
\item \textsc{Learning-WalkSMT} combines \textsc{Basic-WalkSMT} with preprocessing, unit simplification and learning
\item \textsc{Best-WalkSMT} extends \textsc{Learning-WalkSMT} with multiple learning, pure-literal filtering optimizations
\end{itemize}

\begin{itemize}
\item MathSAT with all the optimizations enabled,
\item MathSAT with early pruning and \(\tau\)-propagation disabled
\end{itemize}
Outline

1. Motivations and goals

2. Background

3. Stochastic Local Search for SMT
   - WALKSMT: basic schema
   - Enhancements

4. Preliminary Experimental Evaluation
   - Experiments on SMT-LIB Instances
   - Experiments on Random Instances

5. Conclusions and potential research directions
SMT-LIB Instances

“industrial” formulas on $\mathcal{L}(\mathbb{Q})$ (encoding real-word problems)
- 7 categories of formulas
- 20 formulas per category

Plots
- scatter-plots with logscale
- for WALKSMT, 3 runs with different seeds for each formula
SMT-LIB Instances

“industrial” formulas on $\mathcal{L}A(\mathbb{Q})$ (encoding real-word problems)

- 7 categories of formulas
- 20 formulas per category

Plots

- scatter-plots with logscale
- for WALKSMT, 3 runs with different seeds for each formula
Configurations of WALKSMT on SMT-LIB Instances

Execution time (in sec) of LEARNING WalkSMT

Execution time (in sec) of BASIC WalkSMT

QF_LRA/sc
QF_LRA/uart
QF_LRA/tta_startup
QF_LRA/TM
QF_LRA/sal
QF_LRA/miplib
QF_RDL/scheduling

Execution time (in sec) of BEST WalkSMT

Execution time (in sec) of LEARNING WalkSMT

QF_LRA/sc
QF_LRA/uart
QF_LRA/tta_startup
QF_LRA/TM
QF_LRA/sal
QF_LRA/miplib
QF_RDL/scheduling
WALKSMT vs MathSAT on SMT-LIB Instances

Execution time (in sec) of MathSAT
Execution time (in sec) of BEST WalkSMT
QF\_LRA/sc
QF\_LRA/uart
QF\_LRA/tta\_startup
QF\_LRA/TM
QF\_LRA/sal
QF\_LRA/miplib
QF\_RDL/scheduling

Griggio, Sebastiani, Tomasi ()
SLS for SMT
Results on SMT-LIB Instances

- Learning the discovered $\mathcal{T}$-lemmas is crucial
- optimizations are very significant
- huge gap between WALKSMT and MathSAT:
  - early pruning and $\mathcal{T}$-propagation cannot be applied to WALKSMT since it works on complete truth assignment
  - DPLL-based SAT solvers outperform SLS-based ones on industrial, structured instances since Boolean Constraint Propagation is fully exploited
Outline

1. Motivations and goals
2. Background
3. Stochastic Local Search for SMT
   - WALKSMT: basic schema
   - Enhancements
4. Preliminary Experimental Evaluation
   - Experiments on SMT-LIB Instances
   - Experiments on Random Instances
5. Conclusions and potential research directions
Randomly-generated Instances

3-CNF formula generated in terms of $\langle m, n, a \rangle$

- $m$ clauses
- $n$ $\mathcal{T}$-variables
- $a$ $\mathcal{T}$-atoms $\sum_{i=0}^{4} c_ix_i \leq c_0$ where
  - variable $x_i$ chosen with probability $1/n$
  - constant terms $c$ and $c_i$ randomly taken in $[-100, 100]$

Plots

- plots represent the execution time vs ratio $r = m/a$
- each point corresponds to the median time on 50 different formulas
- for WALKSMT, time is the median value of 10 runs with different seed
Randomly-generated Instances

3-CNF formula generated in terms of $\langle m, n, a \rangle$

- $m$ clauses
- $n T$-variables
- $a T$-atoms $\sum_{i=0}^{4} c_i x_i \leq c_0$ where
  - variable $x_i$ chosen with probability $1/n$
  - constant terms $c$ and $c_i$ randomly taken in $[-100, 100]$

Plots

- plots represent the execution time vs ratio $r = m/a$
- each point corresponds to the median time on 50 different formulas
- for WALKSMT, time is the median value of 10 runs with different seed
Random Instances with 20 $T$-variables

Griggio, Sebastiani, Tomasi ()

SLS for SMT
Random Instances with 20 $\mathcal{T}$-variables
Random Instances with 20 $T$-variables

![Graphs showing execution time and satisfiability percentage for different solvers with varying ratios of clauses to atoms.](image-url)
Results on Random Instances

- multiple learning and filtering not very effective
- small difference between performance of \textsc{WalkSMT} and MathSAT
Outline

1. Motivations and goals
2. Background
3. Stochastic Local Search for SMT
   - WALKSMT: basic schema
   - Enhancements
4. Preliminary Experimental Evaluation
   - Experiments on SMT-LIB Instances
   - Experiments on Random Instances
5. Conclusions and potential research directions
Conclusions and potential research directions

Conclusions

- SLS for SMT: interesting research problem
- presented \textsc{WalkSMT}, a prototype SLS-based SMT procedure
- so far performances far from DPLL-based SMT solvers on SMT-LIB instances, comparable on random ones

Potential research directions

- investigate theory-driven techniques
- focus on specific problems (e.g., scheduling?)
- explore optimization problems
- ...
Conclusions and potential research directions

Conclusions

- SLS for SMT: interesting research problem
- presented WALKSMT, a prototype SLS-based SMT procedure
- so far performances far from DPLL-based SMT solvers on SMT-LIB instances, comparable on random ones

Potential research directions

- investigate theory-driven techniques
- focus on specific problems (e.g., scheduling?)
- explore optimization problems
- ...

Griggio, Sebastiani, Tomasi
Thank you for your attention!