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

Alberto Griggio, Roberto Sebastiani, and Silvia Tomasi

DISI, Università di Trento, Italy.

# Outline

# Outline

# Motivations and Goals

## Motivations

- "Lazy" approach to SMT: integration of DPLL and a $\mathcal{T}$-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 $\mathcal{T}$-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

# 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)

# 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)

# The WalkSAT Family Schema

**Require:** CNF formula $\varphi$, MAX_TRIES, MAX_FLIPS
1: **for** $i = 1$ to MAX_TRIES **do**
2: $\mu \leftarrow$ INITIALTRUTHASSIGNMENT($\varphi$)
3: **for** $j = 1$ to MAX_FLIPS **do**
4: **if** ($\mu \models \varphi$) **then**
5: **return** SAT
6: **else**
7: $c \leftarrow$ CHOOSEUNSATISFIEDCLAUSE($\varphi, \mu$)
8: $\mu \leftarrow$ NEXTTRUTHASSIGNMENT($\varphi, c, \mu$)
9: **end if**
10: **end for**
11: **end for**
12: **return** UNKNOWN

- differ for different techniques for INITIALTRUTHASSIGNMENT, CHOOSEUNSATISFIEDCLAUSE and NEXTTRUTHASSIGNMENT
- different degrees and forms of greediness and randomness

# The WalkSAT Family Schema

**Require:** CNF formula $\varphi$, MAX_TRIES, MAX_FLIPS
1: **for** $i = 1$ to MAX_TRIES **do**
2: $\mu \leftarrow$ INITIALTRUTHASSIGNMENT($\varphi$)
3: **for** $j = 1$ to MAX_FLIPS **do**
4: **if** ($\mu \models \varphi$) **then**
5: **return** SAT
6: **else**
7: $c \leftarrow$ CHOOSEUNSATISFIEDCLAUSE($\varphi, \mu$)
8: $\mu \leftarrow$ NEXTTRUTHASSIGNMENT($\varphi, c, \mu$)
9: **end if**
10: **end for**
11: **end for**
12: **return** UNKNOWN

- differ for different techniques for INITIALTRUTHASSIGNMENT, CHOOSEUNSATISFIEDCLAUSE and NEXTTRUTHASSIGNMENT
- different degrees and forms of greediness and randomness

# Outline

# SMT from a SAT perspective

SMT on $\varphi$: solve a *partially-invisible* SAT formula $\varphi^p \wedge \tau^p$ 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 *partially-invisible* SAT formula $\varphi^p \wedge \tau^p$ 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]

# Outline

# BASIC-WALKSMT Schema

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

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

Intuition: $\mathcal{T}$-solver plays the role of CHOOSEUNSATISFIEDCLAUSE on $\varphi^p \wedge \tau^p$ when no unsatisfied clause is found in $\varphi^p$.

# BASIC-WALKSMT Schema

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

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

Intuition: $\mathcal{T}$-solver plays the role of CHOOSEUNSATISFIEDCLAUSE on $\varphi^p \wedge \tau^p$ when no unsatisfied clause is found in $\varphi^p$.

# Outline

# 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

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

## static learning

- augment $\varphi$ with short "obvious" $\mathcal{T}$-lemmas generated without invoking the $\mathcal{T}$-solver (e.g., $\neg(x > y) \vee \neg(y > z) \vee (x > z)$)

$\Longrightarrow$ 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

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

## static learning

- augment $\varphi$ with short "obvious" $\mathcal{T}$-lemmas generated without invoking the $\mathcal{T}$-solver (e.g., $\neg(x > y) \vee \neg(y > z) \vee (x > z)$)

$\Longrightarrow$ 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

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

## static learning

- augment $\varphi$ with short "obvious" $\mathcal{T}$-lemmas generated without invoking the $\mathcal{T}$-solver (e.g., $\neg(x > y) \vee \neg(y > z) \vee (x > z)$)

$\Longrightarrow$ prevents investigating obviously-inconsistent assignments

# Learning and Unit simplification

## Learning

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

$\Longrightarrow$ 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.

$\Longrightarrow$ avoid useless flips on these variables

# Learning and Unit simplification

## Learning

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

$\Longrightarrow$ 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.

$\Longrightarrow$ avoid useless flips on these variables

# Learning and Unit simplification

## Learning

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

$\Longrightarrow$ 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.

$\Longrightarrow$ avoid useless flips on these variables

# Learning and Unit simplification

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

**for** $i = 1$ to MAX_TRIES **do**
  $\mu^p \leftarrow$ INITIALTRUTHASSIGNMENT $(\varphi^p)$
  **for** $j = 1$ to MAX_FLIPS **do**
    **if** $(\mu^p \models \varphi^p)$ **then**
      $\langle status, c^p \rangle \leftarrow \mathcal{T}$-solver $(\varphi^p, \mu^p)$
      **if** ($status$ == SAT) **then**
        **return** SAT
      **end if**
      $c^p \leftarrow$ UNIT-SIMPLIFICATION$(\varphi^p, c^p)$
      $\varphi^p \leftarrow \varphi^p \wedge c^p$
      $\mu^p \leftarrow$ NEXTTRUTHASSIGNMENT $(\varphi^p, c^p, \mu^p)$
    **else**
      $c^p \leftarrow$ CHOOSEUNSATISFIEDCLAUSE $(\varphi^p, \mu^p)$
      $\mu^p \leftarrow$ NEXTTRUTHASSIGNMENT $(\varphi^p, c^p, \mu^p)$
    **end if**
  **end for**
**end for**
**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 $\mu^p$

$\Longrightarrow$

- 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 $\mu^p$

$\Longrightarrow$

- 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 $\mu^p$

$\Longrightarrow$

- 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

# 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

# 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

# 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

## 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)

# Outline

# A prototype SLS-based SMT solver for $\mathcal{LA}(\mathbb{Q})$: WalkSMT

## Implementation

- implemented in C++
- built on top of UBCSAT SLS platform [Tompkins & Hoos'04]
  - implements many existing SLS procedures
  - after various attempts and preliminary testing, we selected the 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 $\mathcal{LA}(\mathbb{Q})$: WalkSMT

## Implementation

- implemented in C++
- built on top of UBCSAT SLS platform [Tompkins & Hoos'04]
  - implements many existing SLS procedures
  - after various attempts and preliminary testing, we selected the 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

# Comparison: 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 $\mathcal{T}$-propagation disabled

# Comparison: 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 $\mathcal{T}$-propagation disabled

# Outline

# SMT-LIB Instances

"industrial" formulas on $\mathcal{LA}(\mathbb{Q})$ (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

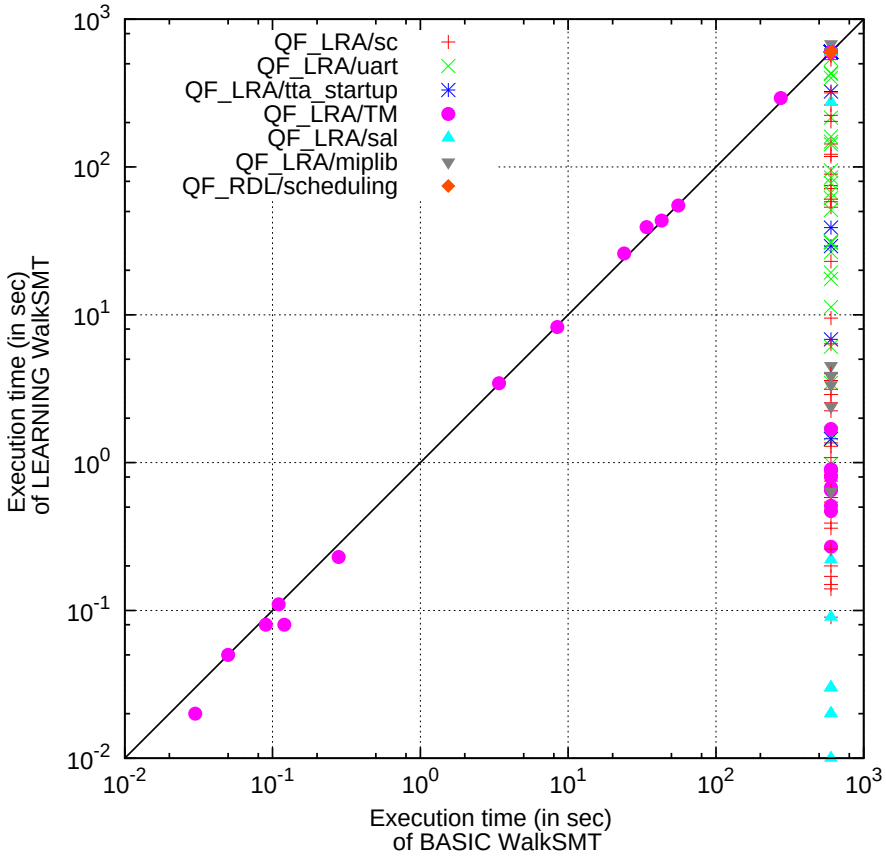
"industrial" formulas on $\mathcal{LA}(\mathbb{Q})$ (encoding real-word problems)

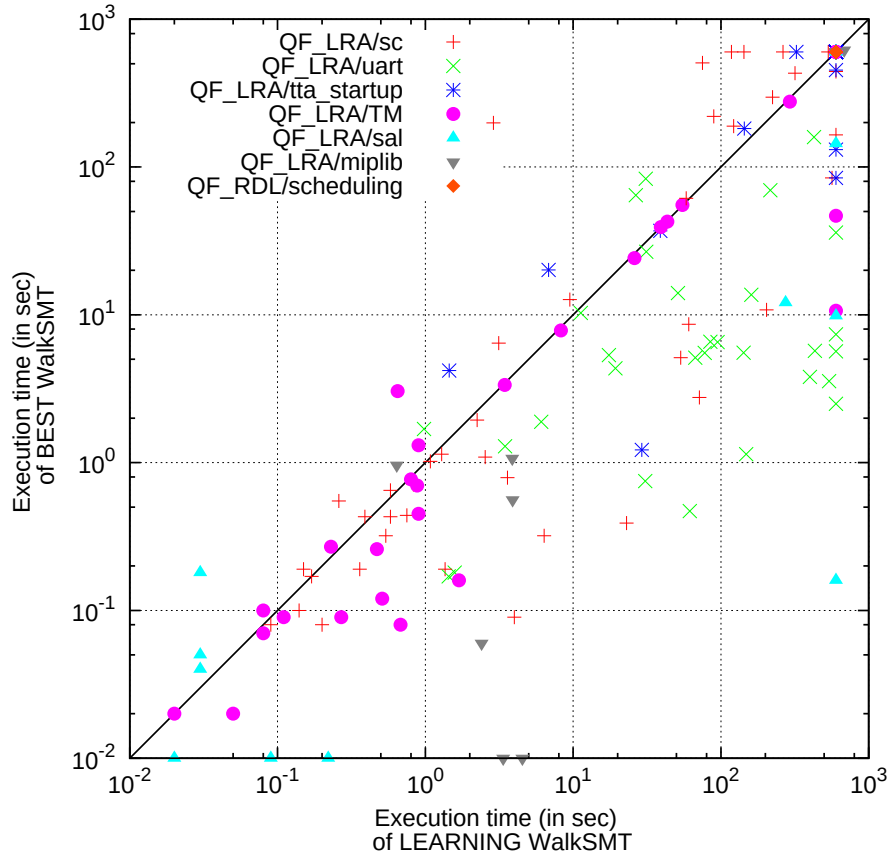
- 7 categories of formulas
- 20 formulas per category

## Plots

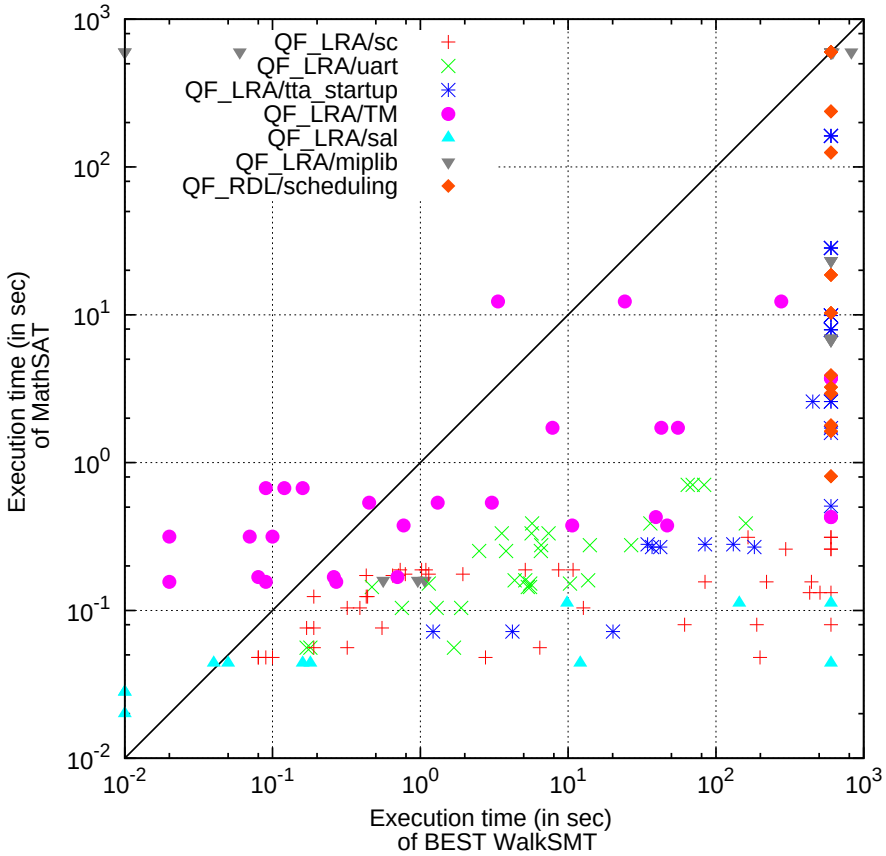
- scatter-plots with logscale
- for WALKSMT, 3 runs with different seeds for each formula

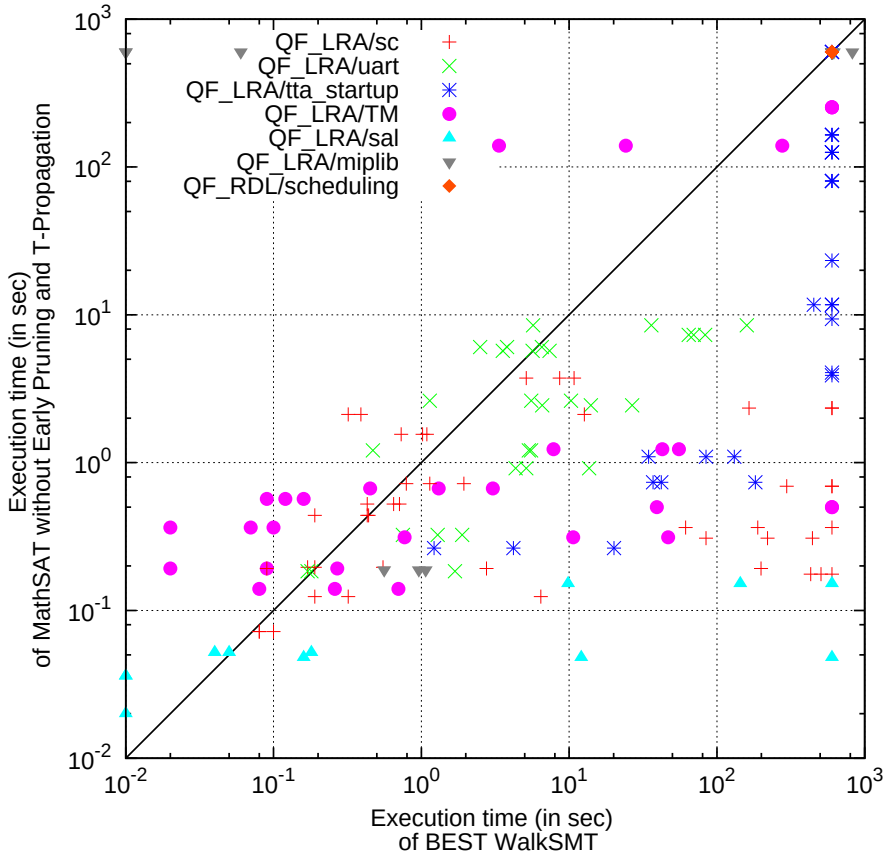
# Configurations of WalkSMT on SMT-LIB Instances



# WalkSMT vs MathSAT on SMT-LIB Instances



# 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

# 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 $\sum_{i=0}^{4} c_i x_i \leq c_0$ 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

# Randomly-generated Instances

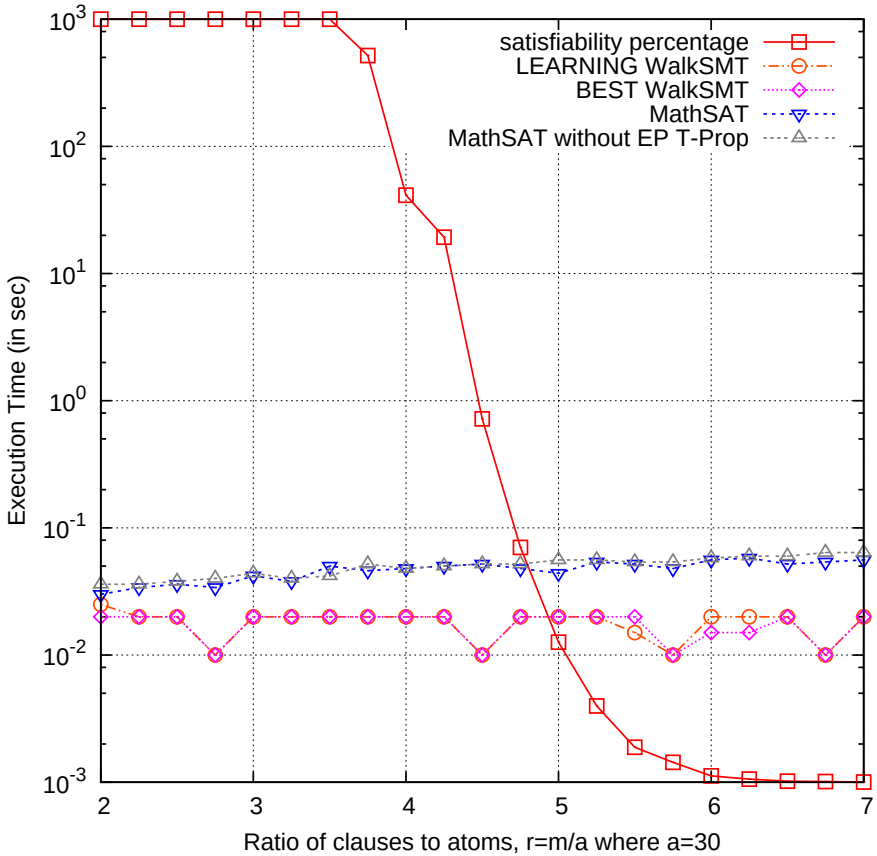
3-CNF formula generated in terms of $\langle m, n, a \rangle$

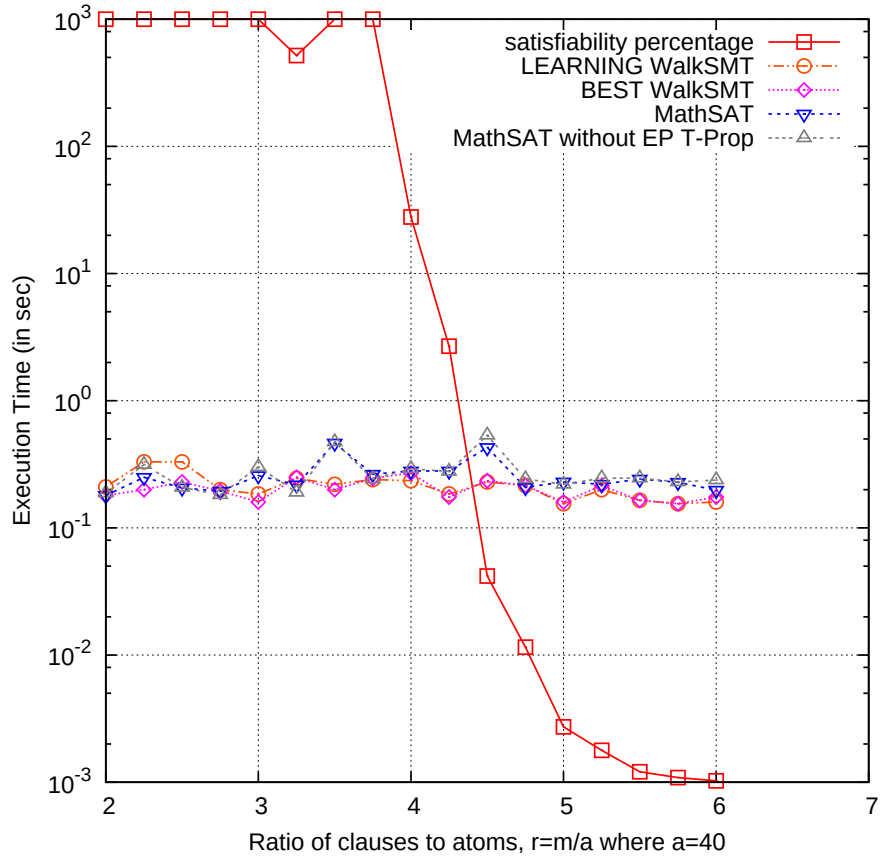
- $m$ clauses
- $n$ $\mathcal{T}$-variables
- $a$ $\mathcal{T}$-atoms $\sum_{i=0}^{4} c_i x_i \leq c_0$ where
  - variable $x_i$ chosen with probability $1/n$
  - constant terms $c$ and $c_i$ randomly taken in $[-100, 100]$

## Plots

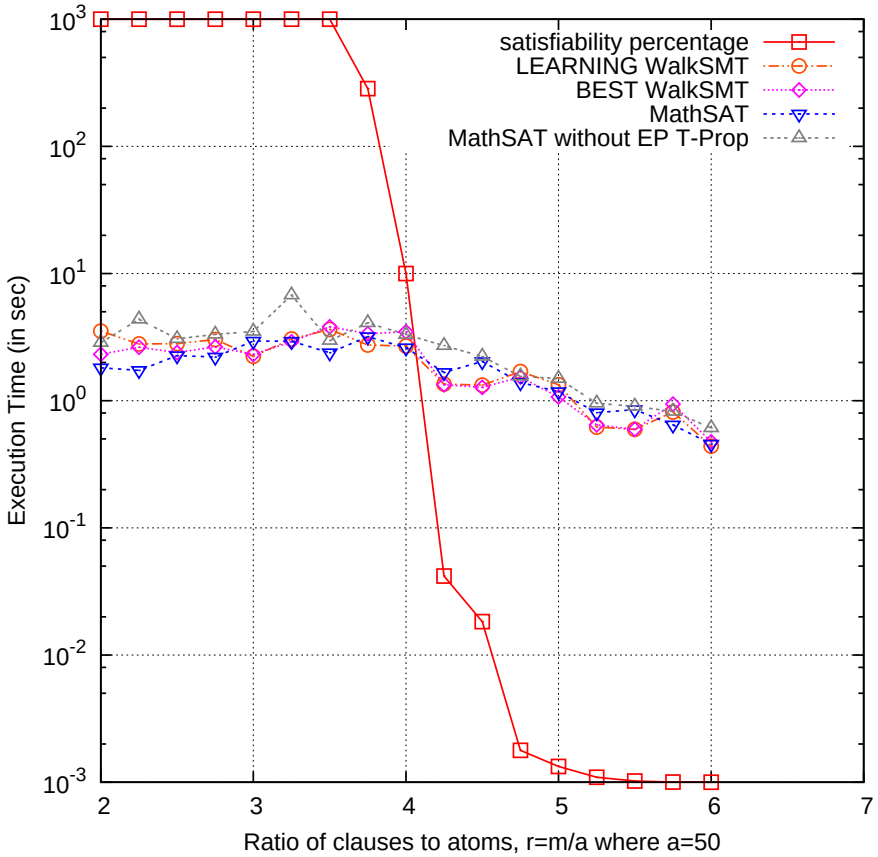
- plots represent the execution time vs ratio $r = m/a$
- each point corresponds to the median time on 50 different formulas
- for WALKSMT, time is the median value of 10 runs with different seed

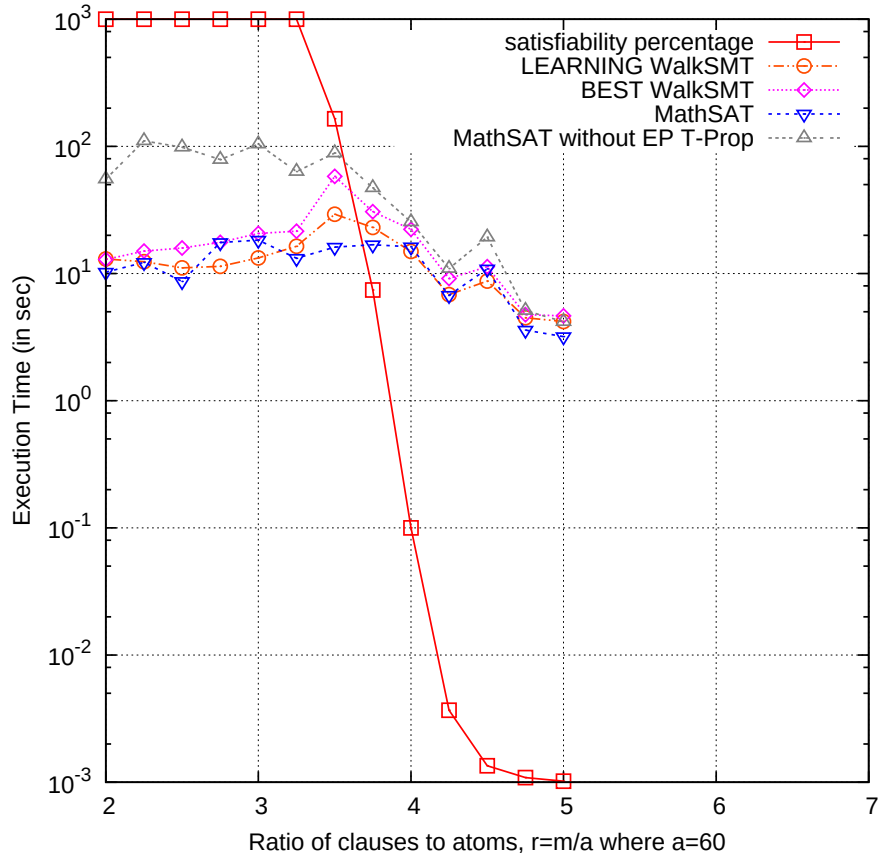
# Random Instances with 20 $\mathcal{T}$-variables



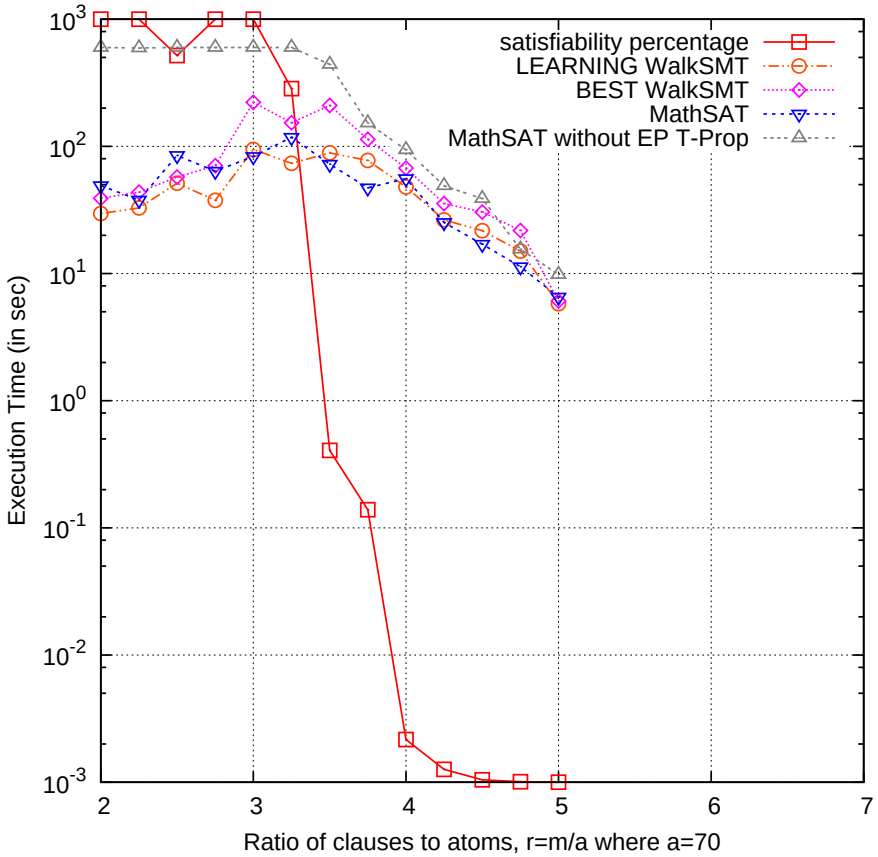



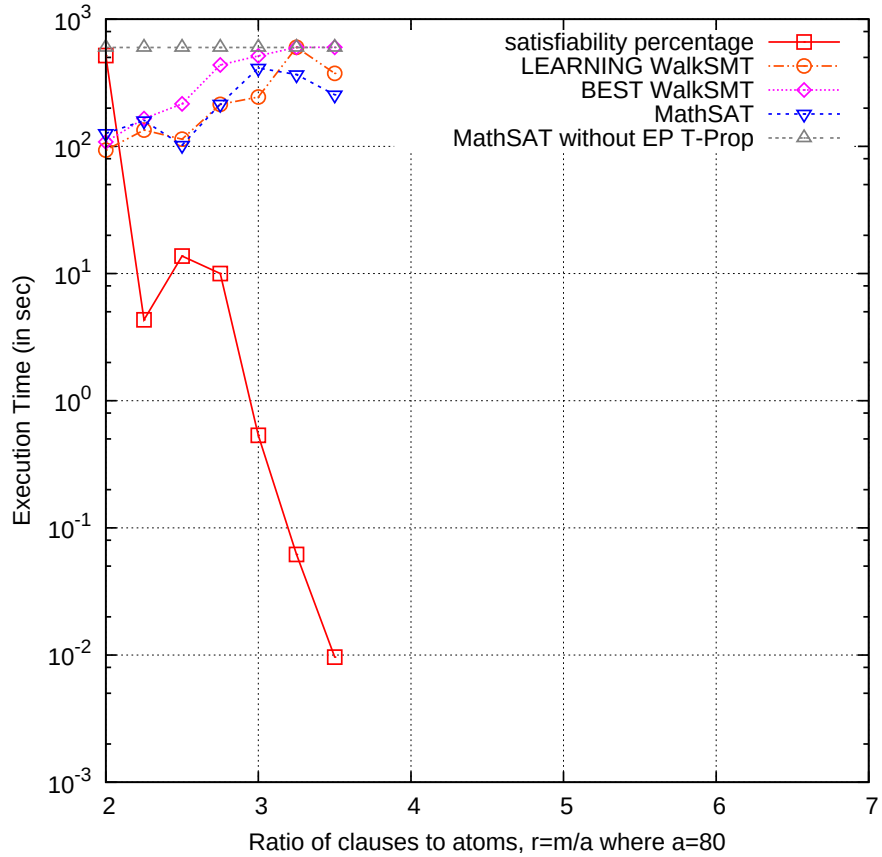
# Random Instances with 20 $\mathcal{T}$-variables



# Random Instances with 20 $\mathcal{T}$-variables



# Results on Random Instances

- multiple learning and filtering not very effective
- small difference between performance of WALKSMT and MathSAT

# Outline

# 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
- ...

# 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
- ...

Thank you for your attention!