Local Search and Its Application in CDCL/CDCL(T) Solvers for SAT/SMT

Shaowei Cai

Institute of Software, Chinese Academy of Sciences

FMCAD 2023

October 23 to 27, 2023,

Ames, Iowa, USA

SAT, SMT

SAT: Propositional Satisfiability

$$(A \vee B) \wedge (\neg C \vee \neg B) \wedge (\neg C \vee A)$$

SMT: Satisfiability Modulo Theories

$$\neg(b+2 = c) \wedge (A[3] \neq A[c-b+1] \vee s = 10)$$

Engines in formal Tools:

EDA, program analysis,

software verification ...

To solve SAT and SMT:

- conflict-driven clause learning (CDCL)
- local search

...

Outline

- Local Search for SAT
  - Basis and Early Methods
  - Modern Local Search Solvers
- Local Search for SMT
  - Local Search for Bit Vectors //slides in this part provided by Aina Niemetz
  - Local Search for Arithmetic Theories
- Improving CDCL/CDCL(T) solvers by Local Search

Outline

- Local Search for SAT
  - Basis and Early Methods
  - Modern Local Search Solvers
- Local Search for SMT
  - Local Search for Bit Vectors //slides in this part provided by Aina Niemetz
  - Local Search for Arithmetic Theories
- Improving CDCL/CDCL(T) solvers by Local Search

SAT

Definition Boolean Satisfiability/Propositional Satisfiability/SAT

Given a propositional formula φ, test whether there is an assignment to the variables that makes φ true.

- Boolean variables: $x_1, x_2, \ldots$
- A literal is a Boolean variable x (positive literal) or its negation $\neg x$ (negative literal)
- A clause is a disjunction (∨) of literals

  $x_2 \vee x_3,$

  $\neg x_1 \vee \neg x_3 \vee x_4 \quad (\{\neg x_1, \neg x_3, x_4\})$

- A Conjunctive Normal Form (CNF) formula is a conjunction (∧) of clauses.

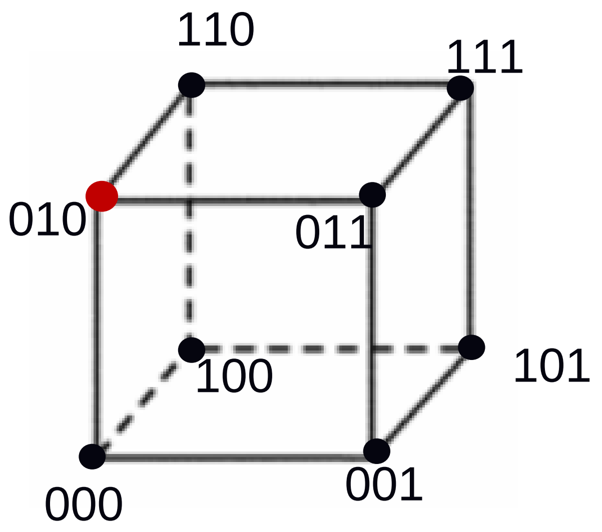
e.g., $\varphi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$

Local Search

F={$\neg x_1 \vee \neg x_2,\ x_1 \vee x_2, \neg x_2 \vee \neg x_3, x_2 \vee x_3,\ \neg x_1 \vee x_2 \vee \neg x_3$}

Assignment (x_1, x_2, x_3)	Cost	falsified Clauses
0 0 0	2	$(x_1 \vee x_2), (x_2 \vee x_3)$
0 0 1	1	$(x_1 \vee x_2)$
0 1 0	0	None √
0 1 1	1	$(\neg x_2 \vee \neg x_3)$
1 0 0	1	$(x_2 \vee x_3)$
1 0 1	1	$(\neg x_1 \vee x_2 \vee \neg x_3)$
1 1 0	1	$(\neg x_1 \vee \neg x_2)$
1 1 1	2	$(\neg x_1 \vee \neg x_2), (\neg x_2 \vee \neg x_3)$

a CNF with 3 variables



Search space:
all complete assignments

Organized by neighboring relation

- Local search walks in the search space, trying to visit a satisfying assignment
--- incomplete, cannot prove unsatisfiability.

Local Search

search space S (consists of all candidate solutions)

SAT: set of all complete truth assignments to variables

solution set S′ ⊆ S

SAT: models of given formula

neighbourhood relation N ⊆ S × S

SAT: Hamming distance 1

objective function f : S → R+

SAT: number of falsified clauses under given assignment

evaluation function g : S → R

$cost(\alpha)$=number of falsified clauses under given assignment

We can have other cost functions...

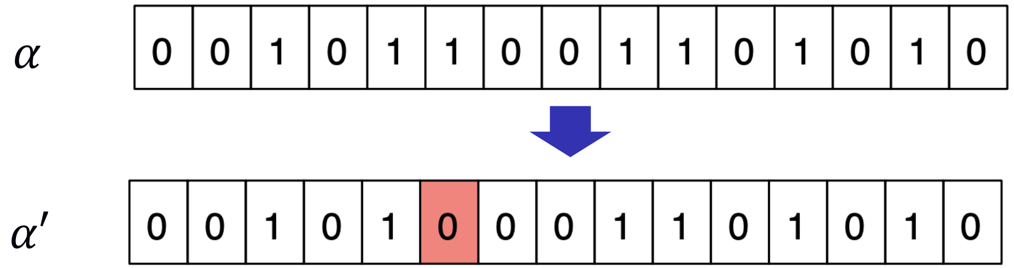
- Local search views SAT as a minimization problem.

Operator

Neighboring relation→ defined by operators

An operator defines how to modify the candidate solution in one step.

(e.g., Hamming distance 1 neighboring relation ←→ operator of flipping one variable)



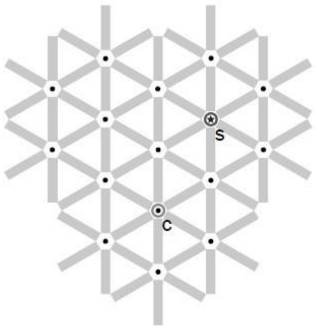
Flip operator for SAT

When an operator is instantiated with a variable (and a value), we obtain an operation.

(e.g., flip(x_1))

Scoring Function

We need an evaluation function to guide the search.



Instead of calculating cost function for candidate solutions, we calculate scoring function for operations.

(We have efficient method for calculating scores.)

A common function: $\boldsymbol{score(x) = cost(\alpha) - cost(\alpha')}$

Example.

F={$\neg x_1 \vee \neg x_2,\ x_1 \vee x_2, \neg x_2 \vee \neg x_3, x_2 \vee x_3,\ \neg x_1 \vee x_2 \vee \neg x_3$}

Assignment (x_1, x_2, x_3)	Cost	falsified Clauses
0 1 1	1	$(\neg x_2 \vee \neg x_3)$
1 1 1	2	$(\neg x_1 \vee \neg x_2), (\neg x_2 \vee \neg x_3)$
0 0 1	1	$(x_1 \vee x_2)$
0 1 0	0	None

$$score(x_1) = cost(011) - cost(111) = -1$$
$$score(x_2) = cost(011) - cost(001) = 0$$
$$score(x_3) = cost(011) - cost(010) = 1$$

Score Computation

Cache based computation

$N(x)$ ={variables share clauses with x}

- Initially calculate score(x) for each variable
- When flip a variable x, only score for those in $N(x)$ should be updated
  - Go through all clauses where x appears, need to update scores in 4 cases
    - 2-satisfied → 1 satisfied
    - 1-satisfied → falsified
    - falsified → 1-satisfied
    - 1-satisfied → 2-satisfied

Non-cache computation

- Simply compute score according to definition, by going through the x's clauses and compute the contribution (either +1 or -1) of each clause

More Scoring Functions

Mainly consider the objective function

- make(x): the number of currently falsified clauses that would become satisfied by flipping x.
- break(x): the number of currently satisfied clauses that would become falsified by flipping x.
- It is easy to see that score(x) = make(x)-break(x).
- $\text{score}(x)^B$
- $A^{-\text{break}(x)}$
- ...

May also consider the algorithm’s behavior

- age(x): the number of steps since the last time x was flipped.
- score(x)+age(x)/T
- …

Dynamic Scoring functions

- Change the parameters or the expression of the scoring function during the search

Local Search for SAT

Design of local search SAT algorithms

- operator
- initialization
- Scoring functions
- Search heuristics

Algorithm : Local Search Framework for SAT

```
begin
    α ← a complete assignment;
    while not reach terminal condition do
        if α satisfies F then return α;
        pick a variable x;            //PickVar
        α ← α with x flipped;
    return "Solution not found";
```

Scoring functions

Search strategies

GSAT

GSAT [SelmanLevesqueMitchell, AAAI'92]

```
S := a random complete assignment;
while (!termination condition)
        if (S is a solution) return S;
        x := a variable with the best score;
        S := S with x flipped;
return S;
```

Tested on hard random 3-SAT, and instances encoded from graph coloring and N-queens, showing promising results at that time.

Random Walk [Papadimitriou,FOCS'91]

Focus on complexity analysis

Start with any truth assignment. While there are unsatisfied clauses, pick one and flip a random literal in it.

WalkSAT[SelmanKautzCohen,AAAI'94]

<u>WalkSAT-PickVar</u>

C := a random falsified clause

If ∃ variable with 0-break

x := a 0-break variable, breaking ties randomly;

else

with probability p

x := a random variable in C;

otherwise

x := a variable with the smallest break, randomly;

- WalkSAT (with p=0.567) performs very well on random 3-SAT, tested on up to half million variables [KrocSabharwalSelman,SAT'10]

Local search algorithms for SAT mainly fall into two types:

- focused random walk (also called focused local search):

  always picks the flip variable from an unsatisfied clause. (conflict driven)

- two-mode local search

  switches between global mode (usually for intensification) and focused mode (usually for diversification).

Focused Random Walk

Novelty
[McAllesterSelmanKautz, AAAI'97]

Novelty-PickVar
Select a random unsatisfied clause;
if the best-score variable is not most recently flipped in the clause
    x:=the best-score variable;
else
    with probability p, x:=the second-best-score variable;
    with probability 1-p, x:=the best-score variable;

Novelty+
[Hoos,AAAI'99]

With a fixed probability wp, choose a random variable from the clause; //PAC
The remaining case, do as Novelty;

Novelty++
[LiHuang,SAT'05]

With a fixed probability dp, choose the oldest variable from the clause;
The remaining case, do as Novelty;

AdaptiveNovelty+
[Hoos,AAAI'02]

adapt wp during the search (initially wp:=0)

- if no improvement in a period of time, wp:=wp+(1-wp)$\cdot\theta$
- if improvement is observed, wp:=wp-wp $\cdot\theta/2$

Two mode Local Search

GWSAT [SelmanKautz, IJCAI'93]

```
S := a random complete assignment;
while (!termination condition)
        if (S is a solution) return S;
        with probability p
                x := a variable in  a random unsatisfied clause
        otherwise
                x := a variable with the best score
        S := S with x flipped;
return S;
```

Two mode Local Search

G2WSAT

[LiHuang,SAT'05]

<u>G2WSAT-PickVar</u>
if ∃ promising decreasing variables
    x:=the best-score promising variable;
else
    x:= the variable picked by Novelty++ heuristic;

promising decreasing: becomes decreasing (i.e., positive score) due to the flip of other variables

gNovelty+
[PhamThorntonGretton Sattar, AAI'07]

use AdaptNovelty+ in the focused mode
use clause weighting

Sparrow
[BalintFröhlich,SAT'10]

use a probability-based heuristic in focused mode
use clause weighting

Clause Weighting

Clause weighting serve as a form of diversification in local search.

- Associate each clause with a weight, and use weighted cost function:

$$wcost(F,\alpha) = \Sigma_{c \in UC(F,\alpha)} w(C)$$

then,

$$score(x) = wcost(F,\alpha) - wcost(F,\alpha')$$

- Date back to the Breakout method for SAT [Morris,AAAI'93]

increase the weight of each falsified clause by one when reaching local optima.

□ The basic idea of using weight penalties, or Lagrangian multipliers, to solve discrete optimization problems was developed in the operations research (OR) community much earlier. [Everett, OR'63]

Clause Weighting

Clause weighting schemes usually have a mechanism to decrease clause weights.

- Decrease weights by subtraction
  - Discrete Lagrangian method (DLM) [WuWah,AAAI'00] , PAWS [Thorton et al,JAR'05]:

    decreases clause weights by a constant amount after a fixed number of increases.
  - Probabilistic PAWS: with a probability, decrease the weights of clauses with large weights
- Pull to the mean value
  - $w(c) = \rho w(c) + (1-\rho)\overline{w}$ or $w(c) = \rho w(c) + (1-\rho)\overline{w_{sat}}$

  SDF[SchuurmansSouthey AIJ'01], ESG [SchuurmansSoutheyHolte IJCAI'01], SAPS[HutterTompkinsHoos,CP'02]
  - DDWF: transfer weights from neighbouring satisfied clauses to falsified ones. [Ishtaiwi et.al,CP'05]

□ Clause weighting has been the most significant line in recent progress of local search for MaxSAT, including SATLike [AAAI'20] NuWLS [AAAI'23] // need to distinguish hard and soft clauses

Outline

- Local Search for SAT
  - Basis and Early Methods
  - Modern Local Search Solvers
- Local Search for SMT
  - Local Search for Bit Vectors //slides in this part provided by Aina Niemetz
  - Local Search for Arithmetic Theories
- Improving CDCL/CDCL(T) solvers by Local Search

Efficient Local Search on Structured Formulas

After 2010, more attention on evaluating local search solvers on structured instances, including crafted and industrial instances, with promising results.

This happens with new LS solvers: Sattime, probSAT, CCASat/CCAnr...

- Sattime ranked 4th in the crafted track, the top 3 were portfolios.
- complementary with CDCL solvers on crafted benchmarks. (top 3 solvers are CDCL+LS in SC'14)
- CCAnr (or its variant) shows good performance on instances from test generation, spectrum allocation, and math problems [Brown et al, AAAI'16;Fröhlich et al,AAAI'15;Cai et al, CP'21]
- local search solver for SAT instances from matrix multiplication [HeuleKauersSeidl,SAT'19]

☐ No random track in SAT Competition after 2017.

Two Modern Local Search SAT Solvers

CCAnr (two-mode local search)

developed: Cai, 2013

[AIJ’13,SAT’15]

- configuration checking
- second level score
- clause weighting

probSAT (focused local search)

developed: Balint, 2012

[SAT’12, SAT’14]

- probability distribution using break
- second-/multi-level break

Second Level Scoring Functions

Example. Given an assignment {x_1 =1, x_2 =1, x_3 =0, x_4 =1, x_5 =1}

c1=$x_1 \vee x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5$, c2=$x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4 \vee \neg x_5$

Both clauses are satisfied.
But c1 is a 4-satised clause, while c2 is 1-satised.

- Satisfaction degree

1-satised clauses are the most endangered satisfied clauses.

→encourage 1-satisfied clauses change to 2-satisfied.

Second Level Scoring Functions [CaiSu, AIJ'13, AAAI'13]

- $make_2$(x) is the number of 1-satifised clauses → 2-satisfied by flipping x.
- $break_2$(x) is the number of 2-satifised clauses → 1-satisfied by flipping x.
- $score_2(x) = make_2(x) - break_2(x)$
- First used in CCASat (with name 'subscore'), formally defined in WalkSATlm, also used in CScoreSAT, probSAT, Sattime2014r...

Second Level Scoring Functions

Proposition For a random 3-SAT formula F(n,m), under any satisfying assignment α to F, the number of 1-satised clauses is more than m/2. [CaiSu,AIJ'13]

This proposition says, second level functions are not suitable for 3-SAT formulas, as at least half clauses are 1-satisfied under any solution.

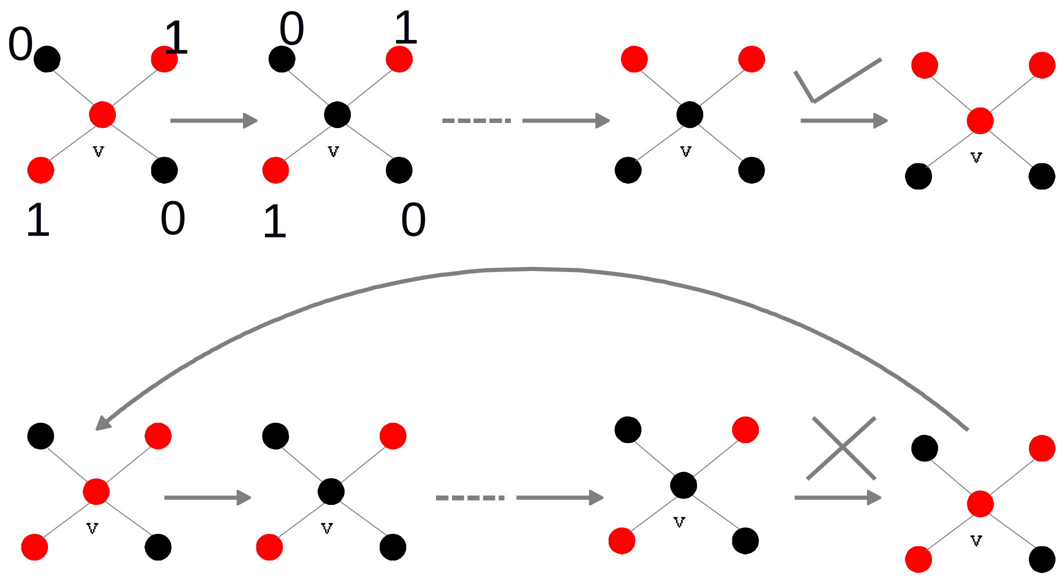
Generally, this indicates it is likely that second level functions are not helpful for formulas with short clauses.

In fact, all local search solvers using second/multi-level functions only use them for 5- and 7-SAT, while the experiment studies show that it is not good for 3-SAT.

Configuration Checking (CC)

Definition the configuration of a variable x is a vector C_x consisting of truth value of all variables in $N(x)$ under current assignment α (i.e., $C_x = \alpha|_{N(x)}$). [AAAI'12,AIJ'13]

$N(x)$ ={variables share clauses with x}



- CC aims to address cycling problem, i.e., revisiting candidate solutions
- We can have different definitions of CC

A simple CC for SAT: if the configuration of x has not changed since x's last flip, then it is forbidden to flip.

Configuration Checking

Observation when a variable is flipped, the configuration of all its neighboring variables has changed.

Efficient implementation of CC:

- Auxiliary data structure --- CC array
  - CC[x] = 1 means the configuration of x has changed since x's last flip;
  - CC[x] = 0 on the contrary.
- Maintain the CC array
  - for each variable x, CC[x] is initialized as 1.
  - when flipping x, CC[x] is reset to 0, and for each $y \in \mathrm{N}(x)$, CC[y] is set to 1.

When to use (or not use) CC?

The effectiveness of the typical CC is related to the neighborhood of variables.

Proposition. For a uniform random k-SAT formula F, its the number of variables n and the clause-variable ratio r, if $\ln(n-1) < \frac{k(k-1)r}{n-1}$, then each variable is expected to have a complete neighborhood, and thus the CC strategy degrades to the trivial case that forbids only one variable.

	3-SAT ($\boldsymbol{r = 4.2}$)	4-SAT ($\boldsymbol{r = 9.0}$)	5-SAT ($\boldsymbol{r = 20}$)	6-SAT ($\boldsymbol{r = 40}$)	7-SAT ($\boldsymbol{r = 85}$)
n^*	11.652	32.348	90.093	223.095	564.595

- Generally, CC is effective for formulas with short clauses.

$f(n) = ln(n-1) - \frac{k(k-1)r}{n-1}$ is a monotonic increasing with n $(n > 1)$.
$f(n) < 0$ *(thus cc fails)* iff $n \leq \lfloor n^* \rfloor$, *where n* is a number s.t.* $f(n^*) = 0$.
This table list the n^* value near phase transition for k-SAT.

CCASat and CCAnr

CCA-PickVar

If ∃ $\boldsymbol{CCD}$ variables //configuration checking
x:=the best-score CCD variable;
else if ∃ $\boldsymbol{SD}$ variables // aspiration
x:=the best-score SD variable;
else
Select a random unsatisfied clause;
x:=pick a variable from the clause

CCD:={x|score(x)>0 and CC[x]=1}.

SD:={x|score(x) > $\overline{w}$}

x:= oldest variable from the clause

x:= best variable from the clause

CCASat (won random track of SAT Challenge 2012)

- Variants ranked top 3 in random SAT track in following SCs.
- using second level score for k-SAT

CCAnr

- good at structured instances
- not using second level score

probSAT and YalSAT

probSAT (won random SAT track of SC'13)

- Choose a random unsatisfied clause C;
- Pick a variable from C according to probability $\frac{f(x)}{\sum_{z \in C} f(z)}$

$$f(x) = cb^{-break(x)} \quad \rightarrow \quad f(x) = \prod_{l} cb_l^{-break_l(x)}$$

$$f(x) = (1 + break(x))^{-cb} \quad \rightarrow \quad f(x) = \prod_{l} (1 + break_l)^{-cb_l}$$

Original probSAT
[Balint Schöning,SAT'12]

2[nd] level and multi-level break
[BalintSchöningFröhlichBiere,SAT'14]

- 3-SAT: only use $break(x)$
- Two scenarios for 5-SAT and 7-SAT
  - use $break(x)$ *and* $break_2(x)$
  - use all $break_l(x)$ for $l \in \{1,2,\ldots,k\}$

YalSAT (won random SAT track of SC'17)

[Biere,SC-Proc'14]

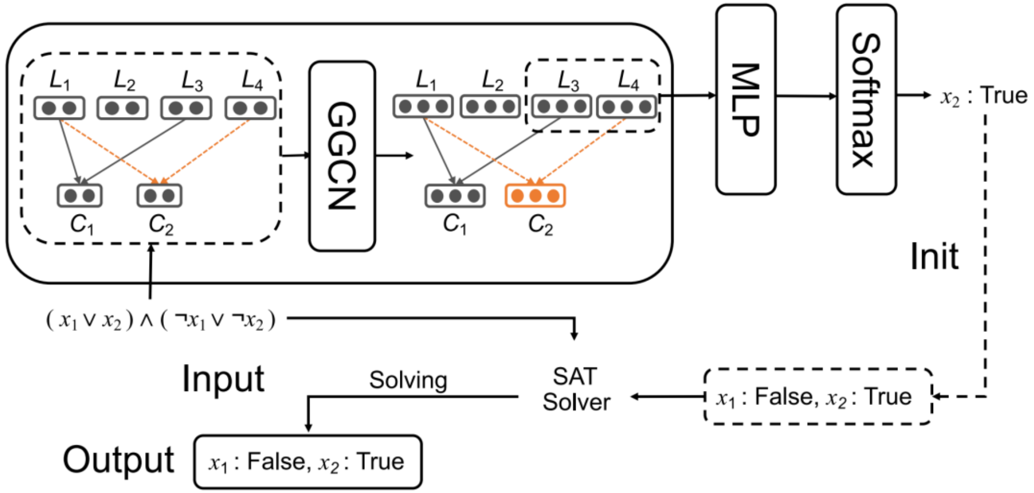
- implements several variants of probSAT
- these variants are scheduled by Luby restarts.

- Besides the traditional random k-SAT instances, random SAT track of SC'17 also includes random instances of a model called sgen.

Improving Local Search via Machine Learning

NLocalSAT[Zhang et,al.,IJCAI'20]:

using Gated Graph Convolutional Network to predict solution, used as initial assignment



Solver	Predefined(165)	Uniform(90)	Total(255)
CCAnr	107.3 ± 1.2	18.0 ± 0.8	125.3 ± 1.2
CCAnr with *NLocalSAT*	165.0 ± 0.0	12.7 ± 0.9	$\mathbf{177.7 \pm 0.9}$
Sparrow	126.7 ± 0.5	23.7 ± 1.7	150.3 ± 1.2
Sparrow with *NLocalSAT*	165.0 ± 0.0	31.0 ± 0.8	$\mathbf{196.0 \pm 0.8}$
CPSparrow	128.0 ± 0.8	27.0 ± 1.6	155.0 ± 1.4
CPSparrow with *NLocalSAT*	165.0 ± 0.0	32.0 ± 0.8	$\mathbf{197.0 \pm 0.8}$
YalSAT	75.0 ± 0.0	49.5 ± 1.5	124.5 ± 1.5
YalSAT with *NLocalSAT*	165.0 ± 0.0	37.3 ± 0.9	$\mathbf{202.3 \pm 0.9}$
probSAT	75.7 ± 0.5	51.0 ± 0.0	126.7 ± 0.5
probSAT with *NLocalSAT*	165.0 ± 0.0	40.7 ± 1.2	$\mathbf{205.7 \pm 1.2}$
Sparrow2Riss	165	23	188
gluHack	165	0	165
MapleLCMDistBT	165	0	165

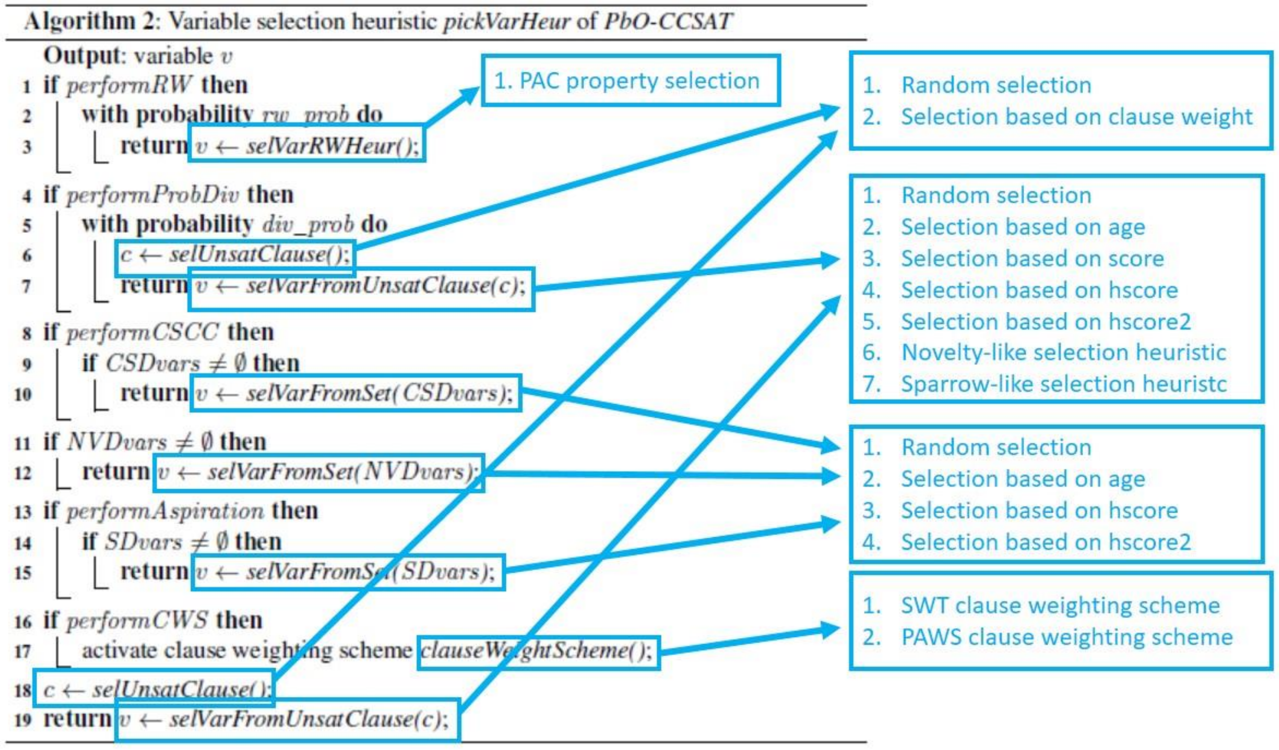
improve local search solvers, tested on uniform random instances and those generated by Balyo's model in SC'18

Improving Local Search via Machine Learning

PbO-CCSAT [LuoHoosCai,PPSN'20]

CC-based local search framework

larger design space → automatic configuration by SMAC [HutterHoosBrown,LION'11]



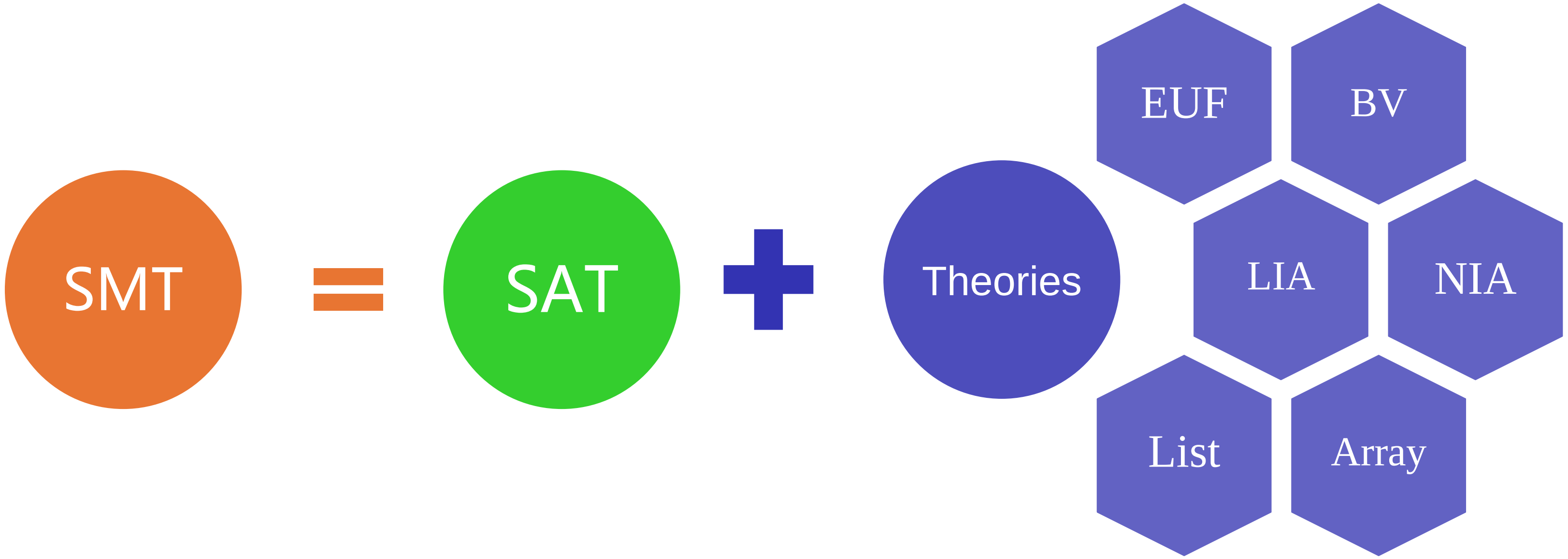
- Improve LS on application benchmarks
- Better than CDCL solvers in some problems

Outline

- Local Search for SAT
  - Basis and Early Methods
  - Modern Local Search Solvers
- Local Search for SMT
  - Local Search for Bit Vectors //slides in this part provided by Aina Niemetz
  - Local Search for Arithmetic Theories
- Improving CDCL/CDCL(T) solvers by Local Search

SMT

Satisfiability Modulo Theories



$$(x_1 - x_2 \leq 13 \vee f(x_2) \neq f(x_3)) \wedge (B_1 \rightarrow x_4 > x_5) \wedge \neg B_2$$

SMT

Example:
$\phi = (x_1 - x_2 \leq 13 \vee x_2 \neq x_3) \wedge (B_1 \rightarrow x_4 > x_5) \wedge \neg B_2$

Propositional Skeleton $\mathrm{PS}_{\Phi}{=}(b_1 \vee \neg b_2) \wedge (B_1 \rightarrow b_3) \wedge \neg B_2$

$b_1 : x_1 - x_2 \leq 13$

$b_2 : x_2 = x_3$

$b_3 : x_4 > x_5$

SMT

- Fixed Sized Bit vectors (BV)

  (x << 001) ≥s 000 ∧ x <u 100 ∧ (x · 010) mod 011 = x + 001

- Linear integer/real arithmetic (LIA/LRA)

  $(x_1 - x_2 \leq 13 \vee x_2 \neq x_3) \wedge (B_1 \rightarrow x_4 > x_5) \wedge \neg B_2$

- Nonlinear integer/real arithmetic (NIA/NRA)

  $(B_1 \vee x_1 x_2 \leq 2) \wedge (\mathrm{B}_2 \vee 3x_2^3 x_4 + 4x_4 + 5x_5 = 12 \vee x_2 - x_3 \leq 3\ )$

Local Search at Boolean Skeleton

WalkSMT [Griggio,Phan,Sebastiani,Tomasi FroCos 2011]

Combine WalkSAT and MathSAT, for LRA

- WalkSAT is used to solve the Boolean skeleton of the SMT formula,
- the conjunction of the literals in the solution μ is sent to the theory solver to check.
- Learn lemmas: if μ is inconsistent, sample some of the literals to check consistency, if they are inconsistent, we learn a lemma

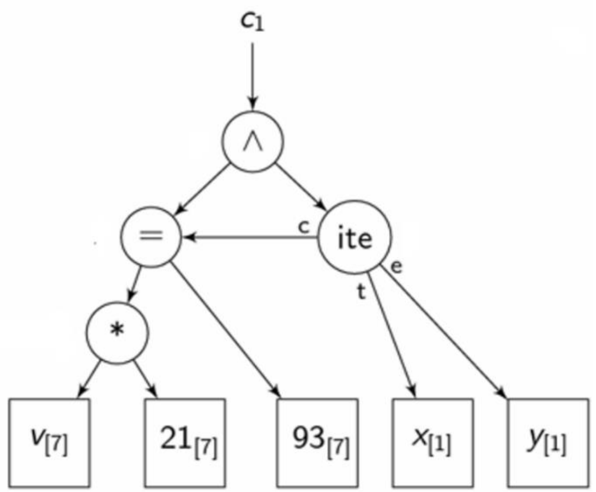
Experiment results:

- SMTLIB: “globally MATHSAT4 performs much better than WALKSMT, often by orders of magnitude.”
- Random instances: “a very small difference ”

Local Search for Bit Vector

BV-SLS [FröhlichBiereWintersteigerHamadi,AAAI'15] in Z3, Boolector

- Represent formula as a directed acyclic graph (DAG) with (possibly) multiple roots
- Use single bit operators



Example.

$$\phi \equiv \underset{\downarrow \atop 0010101}{21_{[7]}} * v_{[7]} = \underset{\downarrow \atop 1011101}{93_{[7]}}$$

Candidate: $v_{[7]} := 0000000$ (initial)

- Single Bit Flips:
  - $v_{[7]} := 0000001$
  - $v_{[7]} := 0000010$
  - $v_{[7]} := 0000100$
  - $v_{[7]} := 0001000$
  - $v_{[7]} := 0010000$
  - $v_{[7]} := 0100000$
  - $v_{[7]} := 1000000$
- Increment
  - $v_{[7]} := 0000001$
- Decrement
  - $v_{[7]} := 1111111$
- Bit-Wise Negation
  - $v_{[7]} := 1111111$

Local Search for Bit Vector

- A function score to evaluate each possible assignment obtained by each operation.
- Recursively defined, compute via bottom-up way, i.e., starting from the inputs

Boolean literal

$$s(x,\alpha) = \alpha(x)$$
$$s(\neg x,\alpha) = \neg\alpha(x)$$

and-expression

$$s(a \wedge b,\alpha) = \frac{1}{2}(s(a,\alpha) + s(b,\alpha))$$
$$s(\neg(a \wedge b),\alpha) = max(s(\neg a,\alpha) + s(\neg b,\alpha))$$

(as $\neg(a \wedge b) \equiv \neg a \vee \neg b$)

equality expression

$$s(a = b, \alpha) = \begin{cases} 1.0 & \text{if } \alpha(a) = \alpha(b) \\ c_1 \cdot \left(1 - \frac{h(\alpha(a),\, \alpha(b))}{n}\right) & \text{otherwise} \end{cases}$$

$$s(a \neq b, \alpha) = \begin{cases} 1.0 & \text{if } \alpha(a) \neq \alpha(b) \\ 0.0 & \text{otherwise.} \end{cases}$$

inequality expression

$$s(a < b, \alpha) = \begin{cases} 1.0 & \text{if } \alpha(a) < \alpha(b) \\ c_1 \cdot \left(1 - \frac{m_<(\alpha(a), \alpha(b))}{n}\right) & \text{otherwise} \end{cases}$$

$$s(a \geq b, \alpha) = \begin{cases} 1.0 & \text{if } \alpha(a) \geq \alpha(b) \\ c_1 \cdot \left(1 - \frac{m_\geq(\alpha(a), \alpha(b))}{n}\right) & \text{otherwise.} \end{cases}$$

Local Search for Bit Vector

$$
\begin{array}{l}
\textbf{for } i = 1 \textbf{ to } \infty \\
\quad \alpha = \mathit{initialize}(F) \\
\quad \textbf{for } j = 1 \textbf{ to } \mathit{maxSteps}(i) \\
\quad\quad V = \mathit{selectCandidates}(F, \alpha) \\
\quad\quad \mathit{move} = \mathit{findBestMove}(f, \alpha, V) \\
\quad\quad \textbf{if } \mathit{move} \neq \mathit{none} \textbf{ then } \alpha = \mathit{update}(\alpha, \mathit{move}) \\
\quad\quad \textbf{else } \alpha = \mathit{randomize}(\alpha, V)
\end{array}
$$

single bit operations

- compute the score of each possible assignment obtained by each bit operation,
- then choose the best one.

If no improving operation was found, then perform a random step

Clause weighting:
updated whenever no improving move could be found

	QF_BV	SAGE2
CCAnr	**5409**	64
CCASat	4461	8
probSAT	3816	10
Sparrow	3806	12
VW2	2954	4
PAWS	3331	**143**
YalSAT	3756	142
Z3 (Default)	7173	5821
BV-SLS	**6172**	**3719**

Path Propagation

Example.

$$\phi \equiv 274177_{[65]} * v_{[65]} = 18446744073709551617_{[65]}$$

Candidate: $v_{[65]} := 000000...000000$ (initial)

Assume: no preprocessing (rewriting, simplification)

$\longrightarrow$ 355837 moves, 21 restarts

$\longrightarrow$ unable to determine (single) solution $v_{[65]} = 67280421310721_{[65]}$

- within a time limit of 1200 seconds
- on a 3.4GHz Intel Core i7-2600 machine

$\longrightarrow$ solved within **one** single propagation move

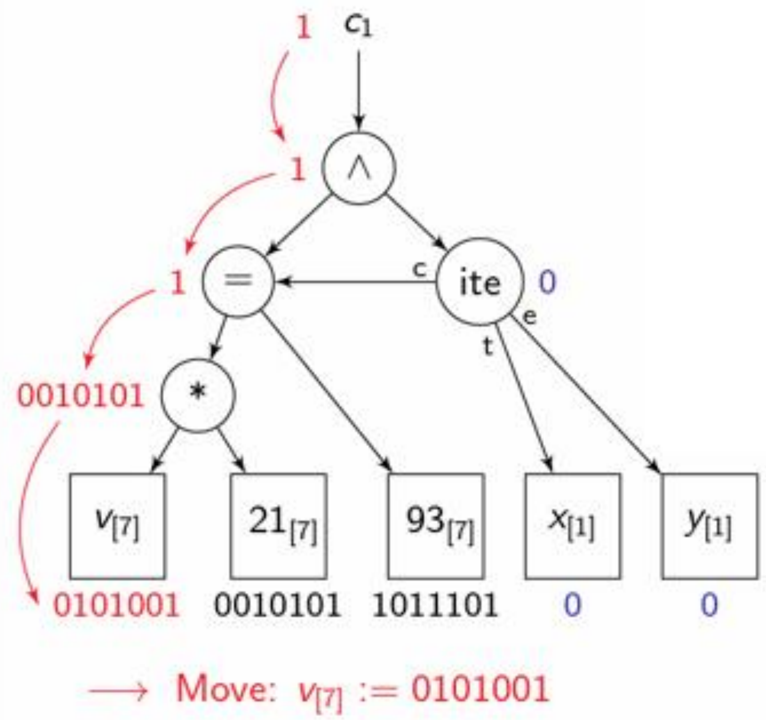
extends BV-SLS [AAAI'15] by path propagation

[NiemetzPreinerFröhlichBiere, DIFTS'15]

Path Propagation

Path propagation (aka. backtracing)

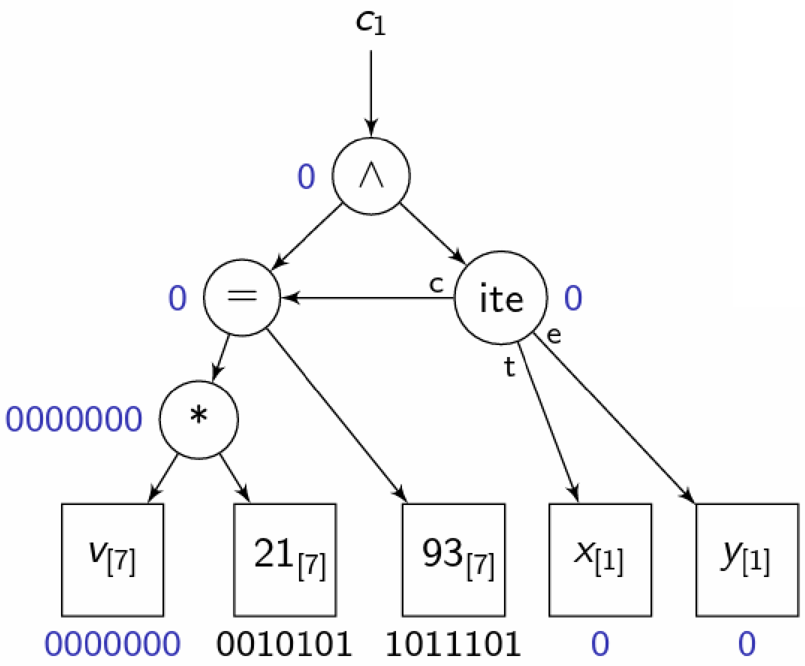
- Force root r to assume its target value to be 1.
- propagate this information along a path towards the primary inputs, update assignment
- propagate this information along another path towards the primary inputs, update assignment
- ...



Path Propagation

Example.

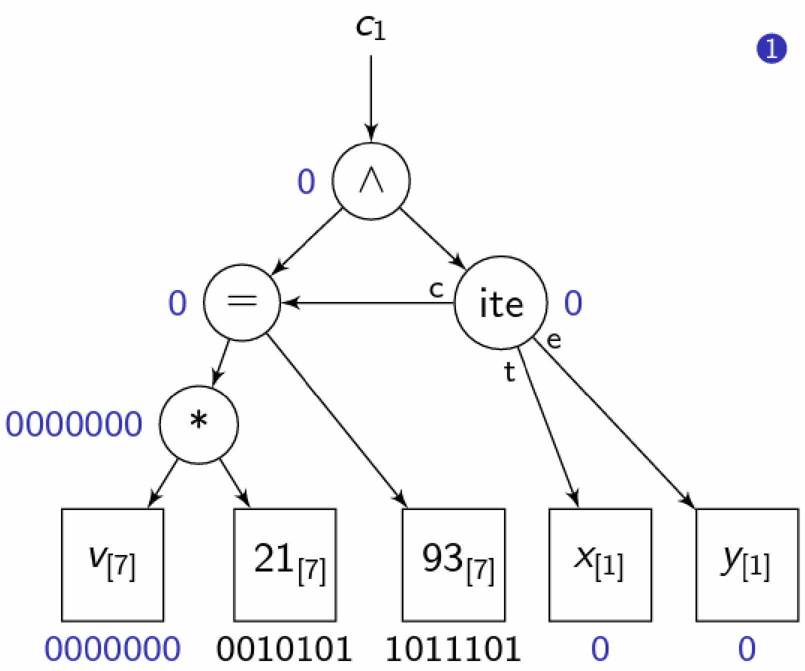
$$\phi \equiv c_1 \wedge c_2 \wedge c_3$$



Path Propagation

Example.

$$\phi \equiv c_1 \wedge c_2 \wedge c_3$$

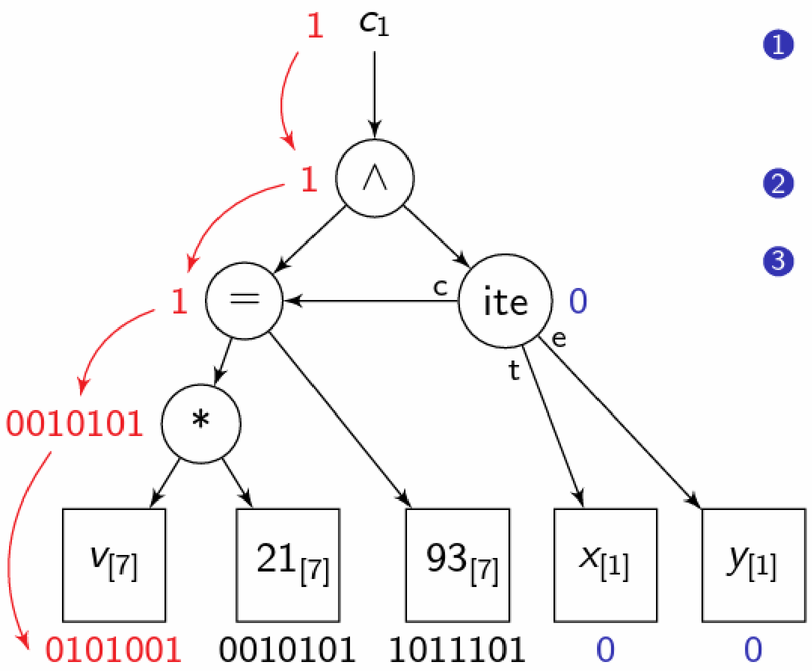


1. initial assignment
   $\{v_{[7]} := 0000000,\ x_{[1]} := 0,\ y_{[1]} := 0\}$

Path Propagation

Example.

$$\phi \equiv c_1 \wedge c_2 \wedge c_3$$



$\longrightarrow$ Move: $v_{[7]} := 0101001$

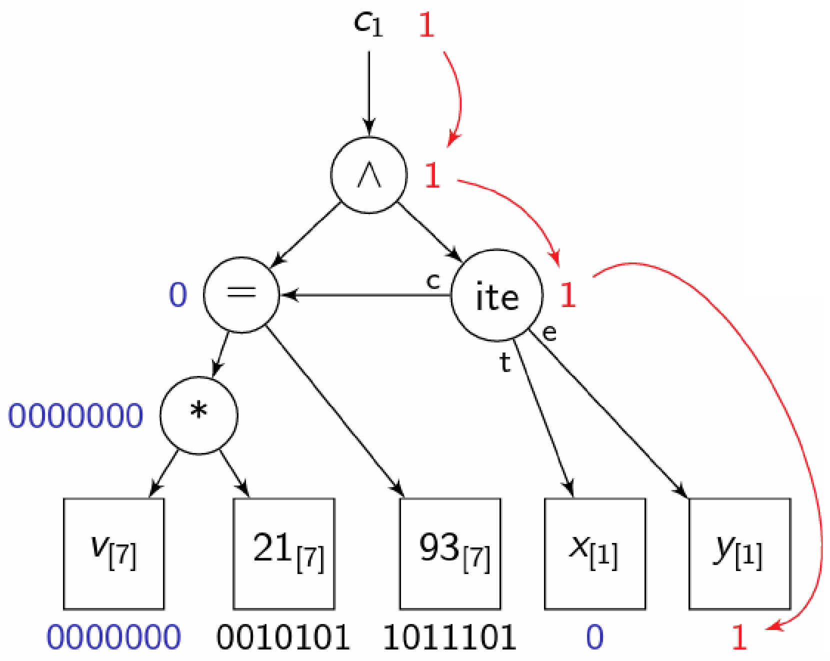
1. initial assignment
   $\{v_{[7]} := 0000000,\ x_{[1]} := 0,\ y_{[1]} := 0\}$
2. force $c_1 = 1$
3. choose path and propagate down
   1. Boolean $\wedge$

      $\longrightarrow$ justification-based selection
      - $0 \to 1$: choose single controlling input
      - else choose randomly

Path Propagation

Example.

$$\phi \equiv c_1 \wedge c_2 \wedge c_3$$



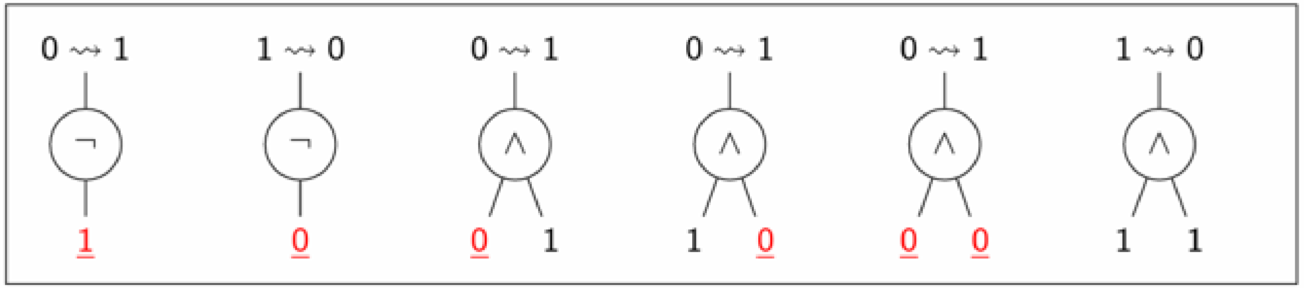
$\longrightarrow$ Move: $y_{[1]} := 1$

To change the value of 'ite' node from 0 to 1, we need to change the value of variable y.

Path Propagation: How to Choose A Path

Definition An input to a node is controlling, if the node can not assume a given target value as long as the value of the input does not change.

Example **Bit-Level** - controlling inputs



How to extend a path?

For each node, prefer to pick to choose a controlling input, otherwise pick a random input

Path Propagation

Down Propagation of Assignments

- via inverse computation
- Restricted set of bit-vector operations
  - **Unary** operations `bvnot` `extract`
  - **Binary** operations `=` `bvult` `bvshl` `bvshr` `bvadd` `bvand` `bvmul` `bvudiv` `bvurem` `concat`
- for some operations no well-defined inverse operation exists

$\longrightarrow$ produce non-unique values

$\longrightarrow$ via randomization of bits or bit-vectors

e.g. $c_{[4]} := a_{[4]} \wedge b_{[4]}$

assume as fixed: $b := 1011$

down propagated: $c := 0001$

selected path, inverse $a := 0X01$

don't care

Path propagation (aka. backtracing)

- Force root r to assume its target value to be 1.
- Iteratively propagate this information along a path towards the primary inputs.

Path Propagation

Down Propagation of Assignments (cntd.)

- if no inverse found
  - $c_{[n]} := a_{[n]} \text{ op } b_{[n]}$

    $\longrightarrow$ disregard b

    $\longrightarrow$ choose inverse value for a that matches assignment of c

    e.g. $c_{[4]} := a_{[4]} \wedge b_{[4]}$

    disregard $b := 1110$

    down propagated: $c := 0001$

    selected path, choose $a := 0001$

  - $c_{[n]} := a_{[n]} \text{ op } bvconst_{[n]}$

    $\longrightarrow$ assignments of b and c are conflicting

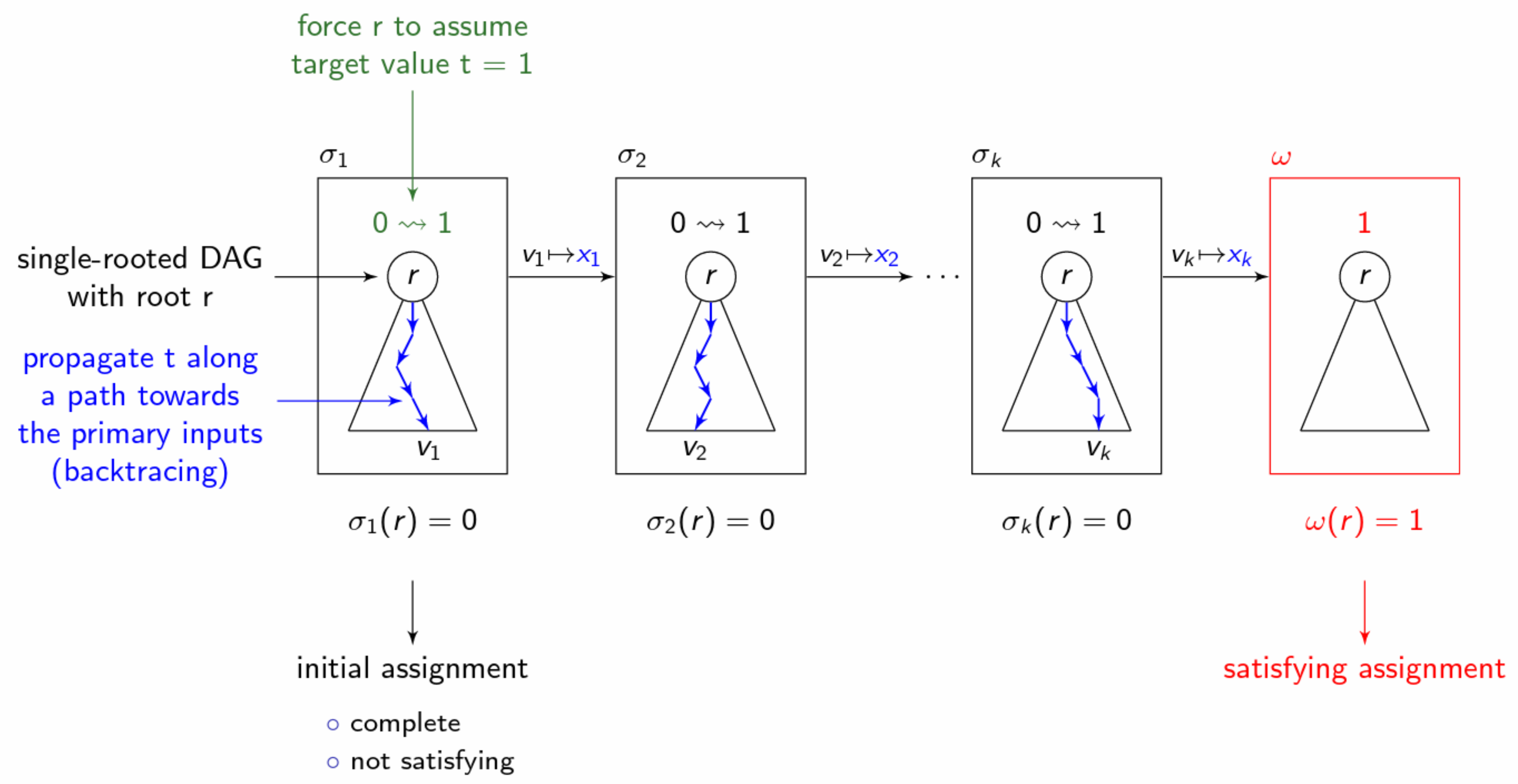
    $\longrightarrow$ no value for a found

    $\longrightarrow$ recover with **regular SLS** move

Path propagation (aka. backtracing)

- Force root r to assume its target value to be 1.
- Iteratively propagate this information along a path towards the primary inputs.

Path Propagation



- prioritizes selecting controlling inputs, else choose randomly

Path Propagation

- Two scenarios
  - Propagation (Bprop) vs. LS moves (frw) with a ratio
  - Propagation moves only

Implemented in Boolector: Bit blasting + focused random walk + path propagation

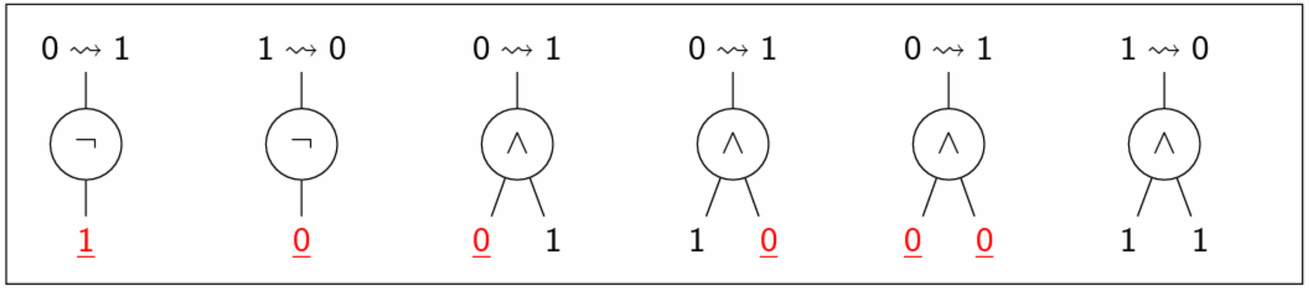
	Solved [#]	Time [s]		
Bb	14806	2623801		
Bb+Bprop+frw (1s)	14844	2538616	+38	97.1%
Bb+Bprop+frw (2s)	14852	2535600	+46	96.9%
Bb+Bprop+frw (3s)	14858	2534900	+52	96.9%
Bb+Bprop+frw (4s)	14861	2538266	+55	97.1%
Bb+Bprop+frw (5s)	14862	2544488	+56	97.3%
Bb+Bprop+frw (6s)	14862	2551784	+56	97.6%
Bb+Bprop+frw (7s)	14862	2558002	+56	97.9%
Bb+Bprop+frw (8s)	14862	2565357	+56	98.1%
Bb+Bprop+frw (9s)	14862	2572600	+56	98.0%

Time limit: 1200 seconds, **Memory limit**: 7GB

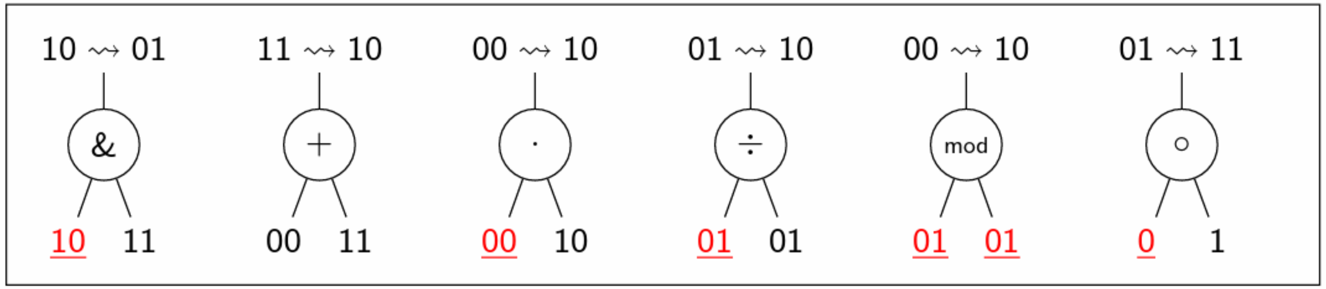
Word Level Propagation

Definition An input to a node is controlling (essential), if the node can not assume a given target value as long as the value of the input does not change.

Example **Bit-Level** - controlling inputs



Example **Word-Level** - essential inputs



Lift propagation
from bit level to word level
[NiemetzPreinerBiere, CAV’16]

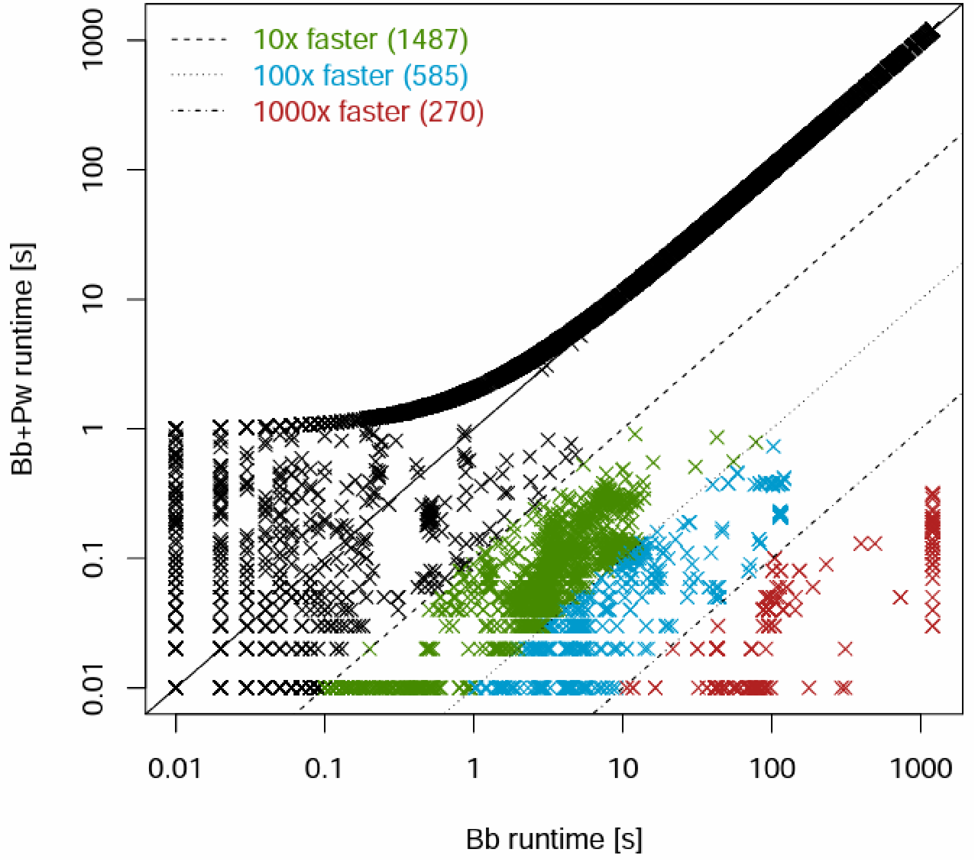
Word Level Propagation

Boolector Configurations:

- **Bit-blasting engine: Bb**
  winner of QF_BV main track of SMT-COMP'15
- **Propagation-based: Pw**
- **Sequential portfolio: Bb+Pw**
  Bb with Pw as a preproc. step

Results:

time limit	Pw 1 sec	Bb 1200 sec	Bb+Pw 1200 sec	
# solved	7632	14806	14866	+60
total time	9106	2611840	2513348	

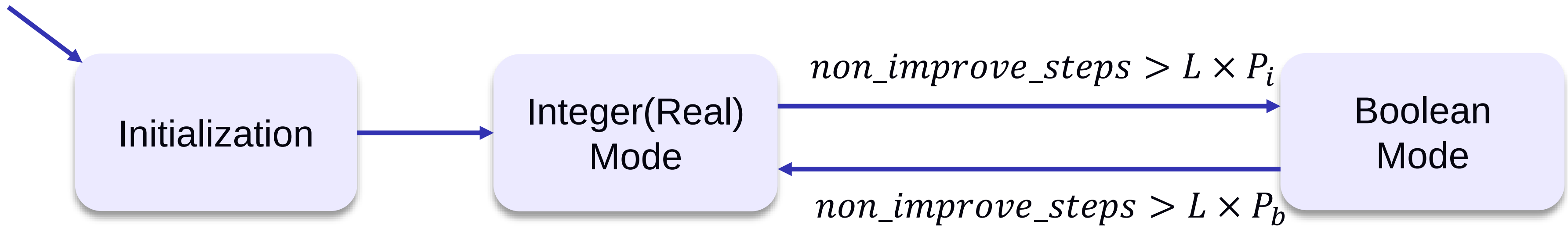


Outline

- Local Search for SAT
  - Basis and Early Methods
  - Modern Local Search Solvers
- Local Search for SMT
  - Local Search for Bit Vectors //slides in this part provided by Aina Niemetz
  - Local Search for Arithmetic Theories
- Improving CDCL/CDCL(T) solvers by Local Search

A Local Search Algorithm for Arithmetic Theories

LS-LIA → LocalSMT (LIA and NIA) [Cai,Li,Zhang, CAV'22,TOCL'23]



- Top level of LocalSMT

P_b , P_i : the proportion of Boolean and integer literals to all literals in falsified clauses

Consecutively performing X (Boolean or Integer) operations can help algorithm focus on the subformula with only X variables

Critical Move

The critical move operator, $cm(x, \ell)$, assigns an integer variable x to the threshold value making literal ℓ true, where ℓ is a falsified literal containing x.

LIA: *let* $\Delta = \sum_i a_i \alpha(x_i) - k$

- *for the case* $\ell: \sum_i a_i x_i \leq k$, $cm(x_i, \ell)$ *makes* $\alpha(x_i) = \left\lceil \left| \frac{\Delta}{a_i} \right| \right\rceil$ for each x_i
- *for the case* $\ell: \sum_i a_i x_i = k$, $cm(x_i, \ell)$ *increases* $\alpha(x_i)$ *by* $-\frac{\Delta}{a_i}$, if $a_i | \Delta$

...

Example

given two literals $l_1: 2b - a \leq -3$ and $l_2: 5c - d + 3a = 5$ and the assignment $\{a = b = c = d = 0\}$

- $cm(a, l_1)$ refers to assigning a to 3, $cm(c, l_2)$ assign c to 1.
- Note that there exists no $cm(a, l_2)$ since $3 \nmid 5$

Critical Move

The critical move operator, $cm(x, \ell)$, assigns an integer variable x to the threshold value making literal ℓ true, where ℓ is a falsified literal containing x.

NIA: Suppose x has n different roots for $\sum_i a_i m_i(x) = k$, listed as $r_1 < r_2 < \cdots < r_n$

for the case $\ell : \sum_i a_i m_i \leq k$,

$$cm_{NIA}(x, \ell) = \bigcup_{j \in S^-} \{op(x, I_{min}[r_j, r_{j+1}]), op(x, I_{max}[r_j, r_{j+1}])\}$$

for the case $\ell : \sum_i a_i m_i = k$,

$$cm_{NIA}(x, \ell) = \{op(x, r_j) | r_j \; is \; an \; integer \; root\}$$

...

- For a variable, there may be more than one critical moves w.r.t. a literal

Critical Move

The critical move operator, $cm(x, \ell)$, assigns an integer variable x to the threshold value making literal ℓ true, where ℓ is a falsified literal containing x.

Substitute all variables but x with their values → Solve feasible intervals → Determine the largest and smallest integer in each feasible interval

Example. literal $l: -2bc^2 + 3\ ab\ + c \leq -3$

current assignment $\{a = 1, b = 1, c = 1, d = 1\}$.

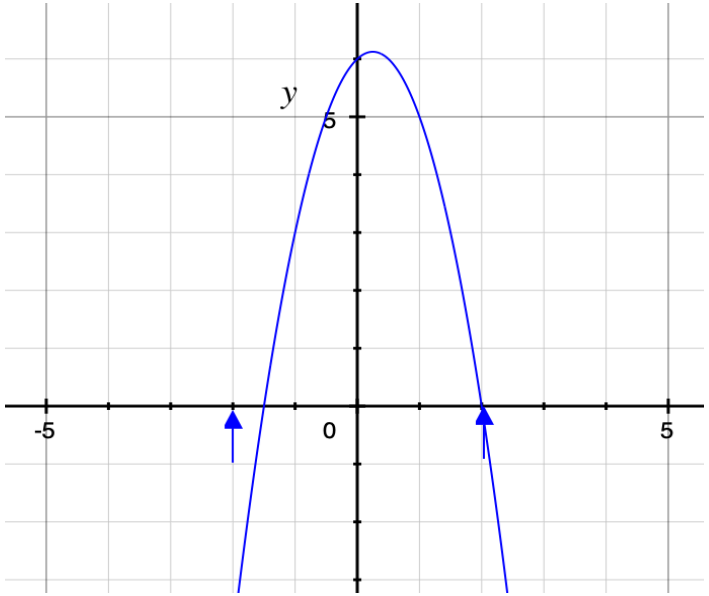
solve

$$-2c^2 + c + 6 \leq 0$$

feasible intervals: $(-\infty, -1.5] \cup [2, \infty)$

largest and smallest integer in these intervals: -2, 2.

→ $cm_{NIA}(c, l)$ contains two operations: assigning c to -2 and 2 respectively.



Two-level heuristic

To find a decreasing cm operation: whenever one exists, we need to scan all cm operations on false literals.

Time consuming!

The set of cm operations D

$S \subseteq D$, $S = \{cm(x, \ell) | \ell \text{ appears in at least one falsified clause}\}$

search for a decreasing cm operation from S

if fail

search for decreasing cm operation from D\S

Two-level heuristic

1. Efficiency of picking operation
2. Conflict driven

LocalSMT Algorithm

- LocalSMT switches between Boolean mode and integer mode
  - Each mode is based on the "two-mode local search" (global step and focused random walk)

<u>Picking Operation in Integer Mode of LocalSMT</u>

If ∃ decreasing cm operation in falsified clauses
    op:=the best-score cm operation;
else if ∃ decreasing cm operation in satisfied clauses
    op:=the best-score cm operation;
else
    update clause weights according to PAWS;
    c:=select a random falsified clause;
    op:=pick a cm operation from c with best dscore;

Two level heuristic

Score Based on Distance to Satisfaction

Distance to truth (dtt):

Given an assignment α and a literal ℓ, the distance to truth of ℓ is

- Inequality literal $\sum_i a_i x_i \leq k$: its $dtt(\ell, \alpha) = max\{\sum_i a_i \alpha(x_i) - k, 0\}$.
- Boolean or equality $\sum_i a_i x_i = k$: $dtt(\ell, \alpha) = 0$ if ℓ is true under α and 1 otherwise.

Distance to satisfaction (dts):

Given an assignment α and a clause C,

$$dts(C, \alpha) = \min_{l \in C}\{dtt(\ell, \alpha)\}$$

Example.

$C = \ell_1 \vee l_2 \vee l_3 = (a + b \geq 1) \vee (b \geq 2) \vee (c \leq -3)$

$\alpha = \{a = b = c = 0\}$

Then, $dtt(\ell_1) = 1$, $dtt(\ell_2) = 2$, $dtt(\ell_3) = 3$,

$and \quad dts(C) = 1$

Distance score (dscore)

For an operation op, $dscore(op) = \sum_{c \in F}(dts(c, \alpha) - dts(c, \alpha'))$

where α, α' denotes the assignment before and after performing op

LocalSMT on Integer Arithmetic Benchmarks

	#inst	MathSAT5	CVC5	Yices2	Z3	LocalSMT	**Z3+LS**
LIA_no_bool	6,670	6,442	6,242	5,994	6,385	**6,478**	6,536
LIA_with_bool	1,842	1,619	766	**1,662**	1,617	912	1,625
Total	8,512	**8,061**	7,008	7,656	8,002	7,390	8,161
IDL_no_bool	841	363	539	654	653	**687**	687
IDL_with_bool	770	514	586	658	**665**	319	661
Total	1,611	877	1,125	1,312	**1,318**	1,006	1,348
NIA_without_bool	16,439	10,497	7,535	9,157	11,806	**12,132**	12,946
NIA_with_bool	1,980	1,906	1,908	1,942	**1,959**	1,669	1,952
Total	18,419	12,403	9,443	11,099	13,765	**13,801**	14,898

Instances without and with Boolean variables are denoted by "no_bool" and "with_bool" respectively.

Tested on SMTLIB benchmarks of LIA, IDL and NIA, cutoff=1200s

Local Search for Linear/Multi-linear Real Arithmetic

- LocalSMT(RA), supports linear and multi-linear real arithmetic
  - e.g. $xy + 5yz - 2xyz \leq 100$ (multi-linear)

issue: infinite possible values for a variable

$\downarrow$

solution: interval-based operation

1. interval division
2. Consider a few options in a selected interval

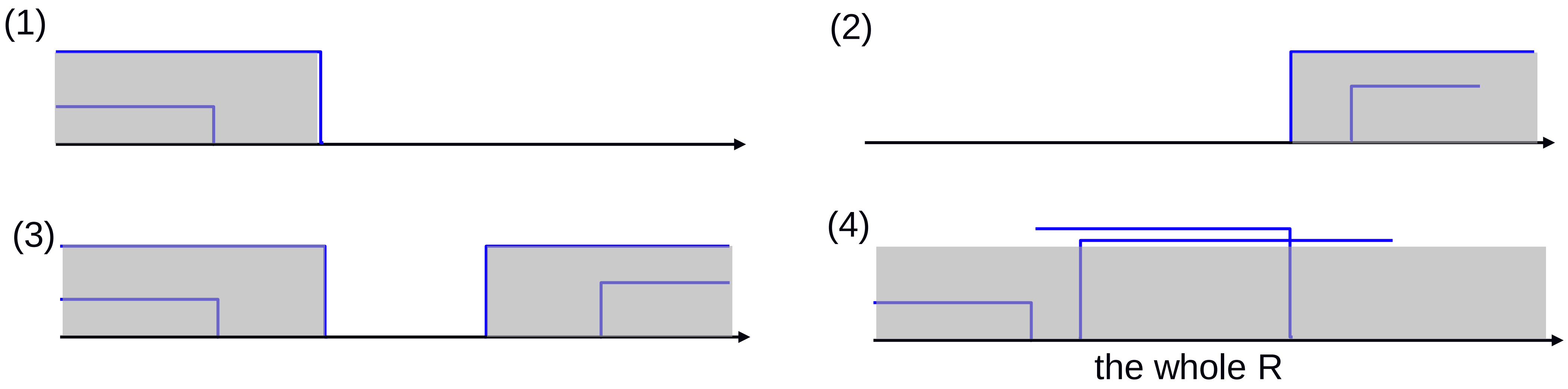
- [Li,Cai,FMCAD'23]

Satisfying Interval

For a literal of linear/multi-linear constraint, when all variables but one (say x) is substituted with their values, we can solve the constraint and get the satisfying interval of x

→either $x \leq ub$ or $x \geq lb$ (for strict inequation, $x < ub$ or $x > lb$)

For a clause with more than one literal, the satisfying interval of x is the union of its satisfying intervals w.r.t. all literals it appears.



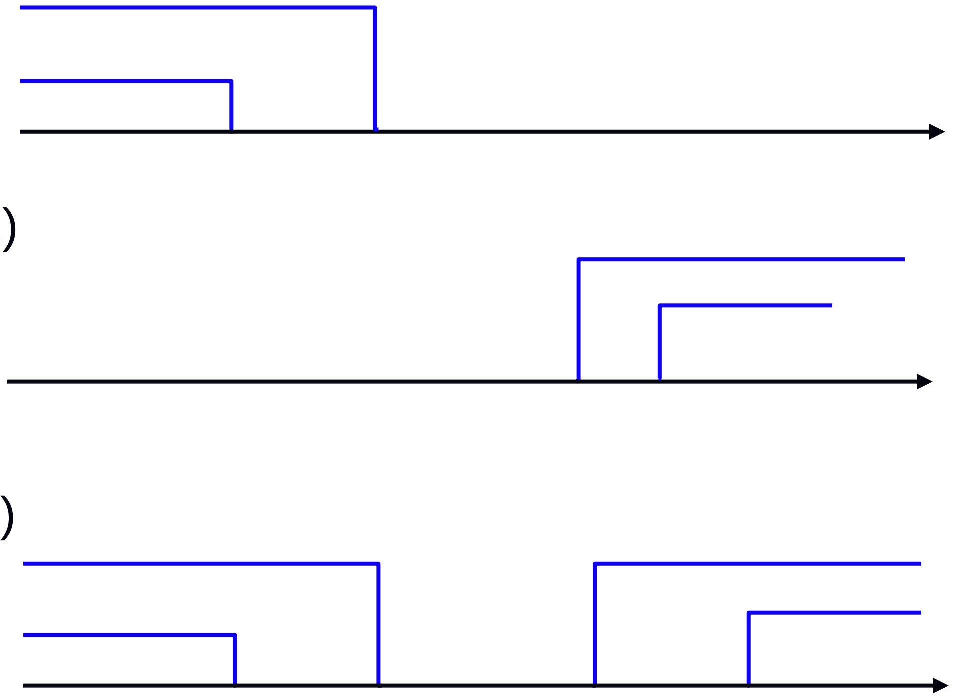
Satisfying Interval

- Consider all falsified clauses, for a variable x, put all satisfying intervals together:

(1)

(2)

(3)



This is the general case

- There is no case with crossing intervals. Suppose they are derived from two clauses C_1 and C_2, then at least one of them is satisfied.

Example. C_1: $x \geq 1$, C_2: $x \leq 2$, then not matter what value x is assigned, at least one of them is satisfied.

Equi-make Intervals

- Consider all falsified clauses, for a variable x, we obtain an interval division:

(3)

This is the general case

For each of the resulting intervals:

Assigning x to any value in the interval have the same $make$ value (making the same number of falsified clauses become true).

→such an interval is called euqi-make interval.

Example:

$F = C_1 \wedge C_2$

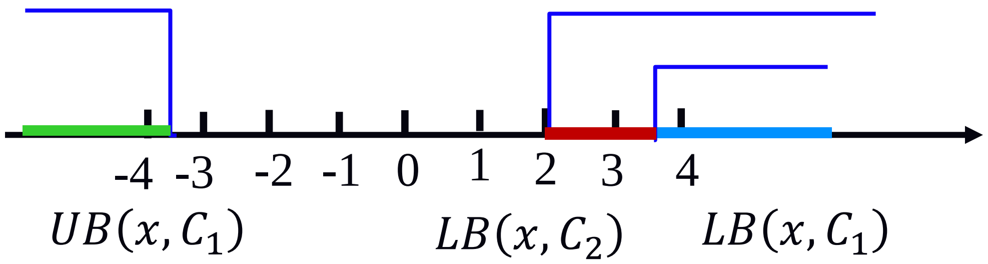
$= (a - b > 4 \vee 2a - b \geq 7 \vee 2a - c \leq -5)$

$\wedge\ (a - c \geq 2)$,

assignment $\{a = b = c = 0\}$, $C_1 and\ C_2$ falsified

for variable a:

- interval $[3.5, \infty)$ can satisfy 2 clauses;
- both interval $(-\infty, -2.5]$ and $[2, 3.5)$ can satisfy 1 clause



Choosing an Operation from Equi-make Interval

- After choosing an equi-make interval, we need to choose a value v.

  Four options

  1) Threshold: l, U
  2) Median: $(l+U)/2$
  3) Largest/Smallest integer in interval: $Z_1 > l$, $Z_2 < U$
  4) For $\left(\frac{b}{a}, \frac{d}{c}\right)$, another option is $\frac{b+d}{a+c}$

  → obtain an operation op(x, v)

LocalSMT(LRA):

- based on the framework of LocalSMT
- global step: collect K such operations, pick the best-score one.

LocalSMT for LRA/MLRA

TABLE I: Results on instances from SMTLIB-LRA

	#inst	cvc5	Yices	Z3	OpenSMT	LocalSMT(RA)
2017-Heizmann	8	4	3	4	4	**7**
2019-ezsmt	84	61	61	53	**62**	35
check	1	1	1	1	1	1
DTP-Scheduling	91	91	91	91	91	91
LassoRanker	271	232	**265**	256	262	240
latendresse	16	9	**12**	1	10	0
meti-tarski	338	338	338	338	338	338
miplib	22	14	**15**	**15**	**15**	4
sal	11	11	11	11	11	11
sc	108	108	108	108	108	108
TM	24	**24**	**24**	**24**	**24**	11
tropical-matrix	10	1	**6**	4	**6**	0
tta	24	24	24	24	24	24
uart	36	**36**	**36**	**36**	**36**	30
Total	1044	954	**995**	966	992	900

TABLE III: Results on instances from SMTLIB-MRA

	#inst	cvc5	Yices	Z3	SMT-RAT	LocalSMT(RA)
20170501-Heizmann	51	1	0	4	0	**17**
20180501-Economics	28	28	28	28	28	28
2019-ezsmt	32	31	**32**	**32**	21	28
20220314-Uncu	12	12	12	12	12	12
LassoRanker	347	**312**	124	199	0	297
meti-tarski	423	423	423	423	423	423
UltimateAutomizer	48	34	39	46	18	**48**
zankl	38	24	25	28	30	**38**
Total	979	865	683	772	532	**891**

Local Search for Nonlinear Real Arithmetic

Extension of the above algorithm to nonlinear real arithmetic need to deal with additional challenges:

1. Efficiency: while there are well-known algorithms for root isolation in higher-degree polynomials, they are time consuming and should be used sparingly.
   - Computation is especially slow when algebraic numbers are involved.

     Example. for constraint $x^2 + y^2 = 3$, if x is assigned to 1, then $y = \pm\sqrt{2}$.

2. Unlike linear equations, not all higher-degree polynomials have feasible solution for each variable.

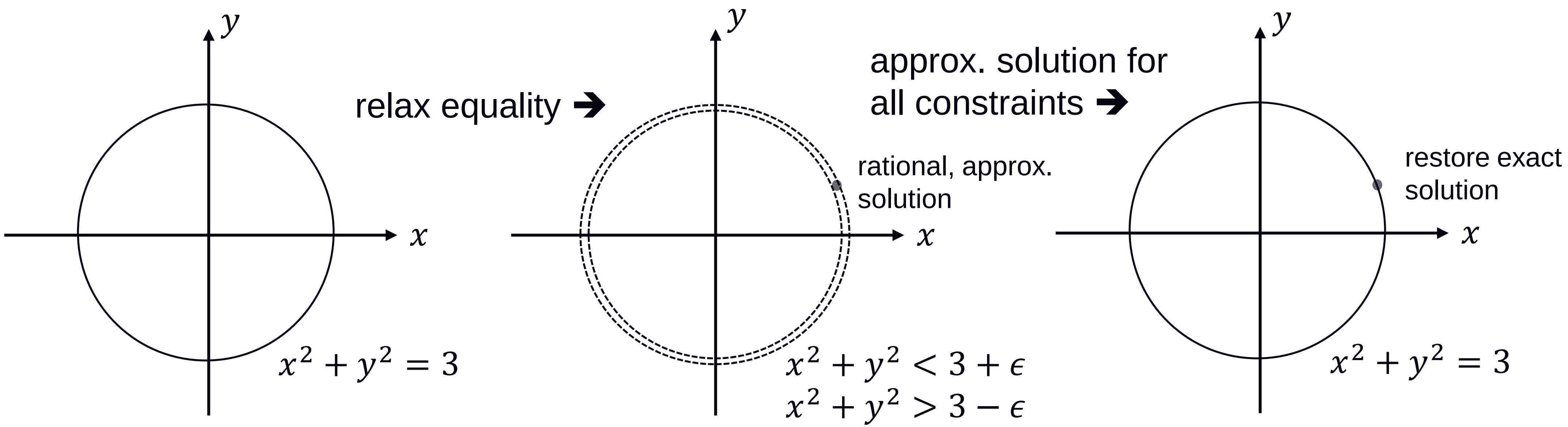
Additional improvements address the above issues, yielding a local search method that is competitive with state-of-the-art complete algorithms.

Relaxation and Restoration of Equalities

A challenge: equality constraints (e.g. $x^2 + y^2 = 3$) may force assignment of variables to irrational (algebraic) numbers, making computation very slow.

- We *relax* the equality constraints that force irrational assignments during most of local search.
- After *approximate* solutions are found, these equalities are restored, and solved to obtain an exact solution.

[WangZhanLiCai, VMCAI'24]



Local Search for Nonlinear Real Arithmetic

Category	#inst	Z3	cvc5	Yices	Ours	Unique
20161105-Sturm-MBO	120	0	0	0	**84**	84
20161105-Sturm-MGC	2	**2**	0	0	0	0
20170501-Heizmann	60	3	1	0	**6**	5
20180501-Economics-Mulligan	93	**93**	89	91	87	0
2019-ezsmt	61	**54**	51	52	18	0
20200911-Pine	237	**235**	201	**235**	224	0
20211101-Geogebra	112	**109**	91	99	100	0
20220314-Uncu	74	73	66	**74**	73	0
LassoRanker	351	155	**304**	122	284	15
UltimateAtomizer	48	**41**	34	39	26	2
hycomp	492	**311**	216	227	272	12
kissing	42	**33**	17	10	**33**	1
meti-tarski	4391	**4391**	4345	4369	4356	0
zankl	133	70	61	58	**99**	26
Total	6216	5570	5476	5376	**5662**	145

local search for NRA, competitive with complete algorithms such as MCSAT on the satisfiable instances QF_NRA in SMT-LIB.

Outline

- Local Search for SAT
  - Basis and Early Methods
  - Modern Local Search Solvers
- Local Search for SMT
  - Local Search for Bit Vectors //slides in this part provided by Aina Niemetz
  - Local Search for Arithmetic Theories
- Improving CDCL/CDCL(T) solvers by Local Search

Challenge of Combining CDCL and Local Search

Ten Challenges in Propositional Reasoning and Search

Bart Selman, Henry Kautz, and David McAllester
AT&T Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
{selman, kautz, dmac}@research.att.com
http://www. research, att.com/~selman/challenge

Challenge 7: Demonstrate the successful combination of stochastic search and systematic search techniques, by the creation of a new algorithm that outperforms the best previous examples of both approaches.

[Bart Selman, Henry Kautz and David McAllester, AAAI 1997]

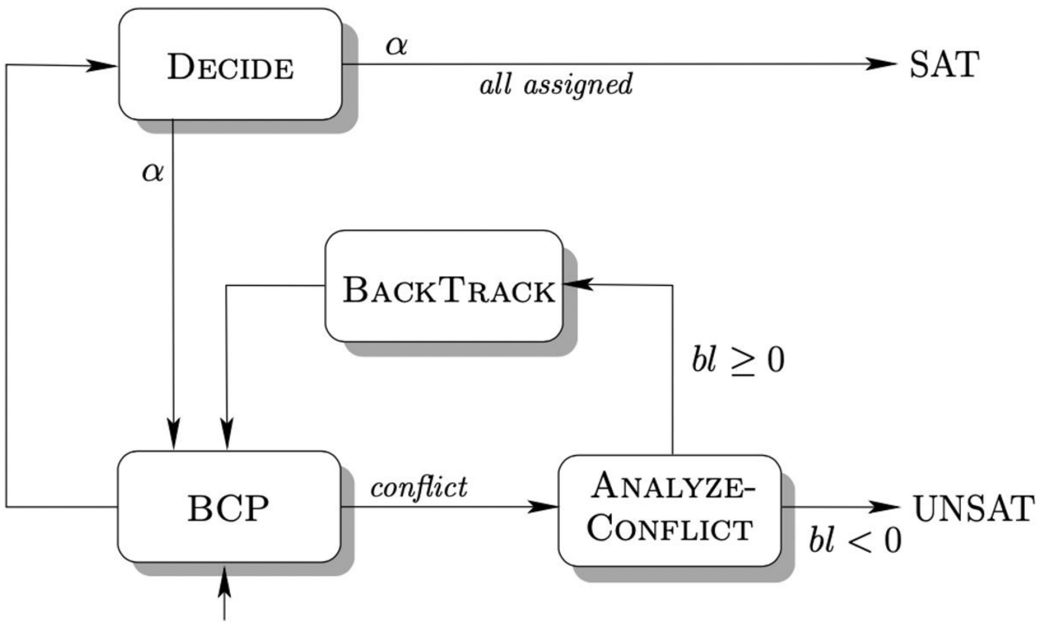
Challenge of Combining CDCL and Local Search

- Local search as main body
  - hybridGM （SAT 2009）, SATHYS （LPAR 2010）
  - GapSAT: use CDCL as preprocessor before local search (SAT 2020)
  - Use resolution in local search (AAAI 1996, AAAI 2005)
- DPLL/CDCL as main body
  - HINOTOS: local search finds subformulas for CDCL to solve (SAT 2008)
  - WalkSatz: calls WalkSAT at each node of a DPLL solver Satz (CP 2002)
  - CaDiCaL and Kissat: a local search solver is called when the solver resets the saved phases and is used only once immediately after the local search process (2019)
- Sequential portfolio
  - Sparrow2Riss, CCAnr+glucose, SGSeq

CDCL Solver Overview

CDCL solver

- Analyze-Conflict : non-chronological backtracking + clause learning + vivification
- Decide : Branching strategy and phasing strategy

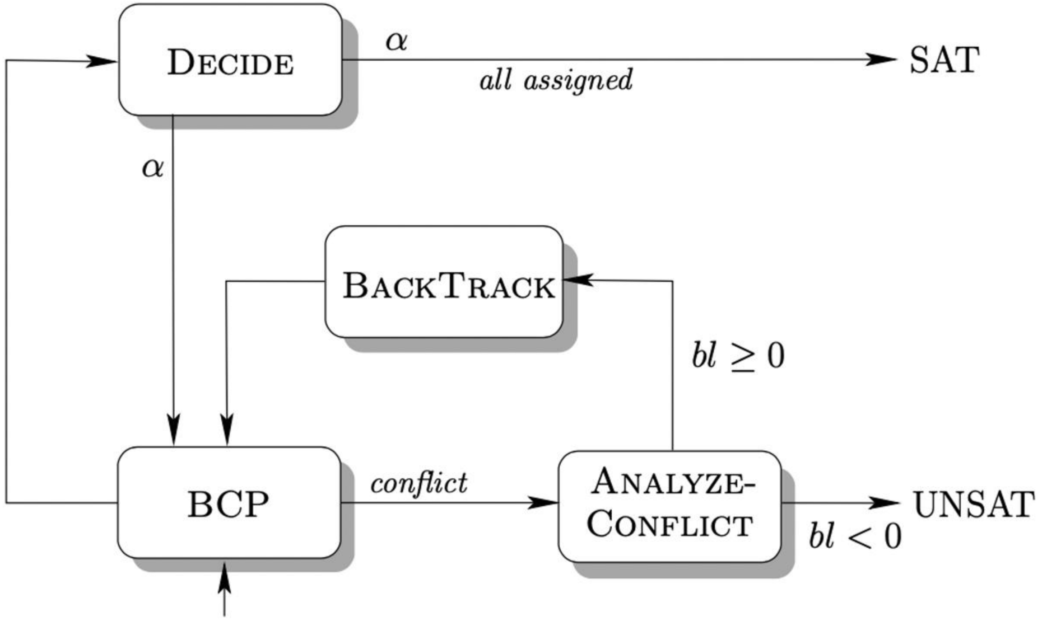


- Clause learning
- Clause management
- Lazy data structures
- Restarting
- Branching
- Phasing
- Mode Switching
- …

CDCL Solver Overview

CDCL solver

- Analyze-Conflict : non-chronological backtracking + clause learning + vivification
- Decide : Branching strategy and phasing strategy → can be improved by local search



- Clause learning
- Clause management
- Lazy data structures
- Restarting
- Branching
- Phasing
- Mode Switching
- …

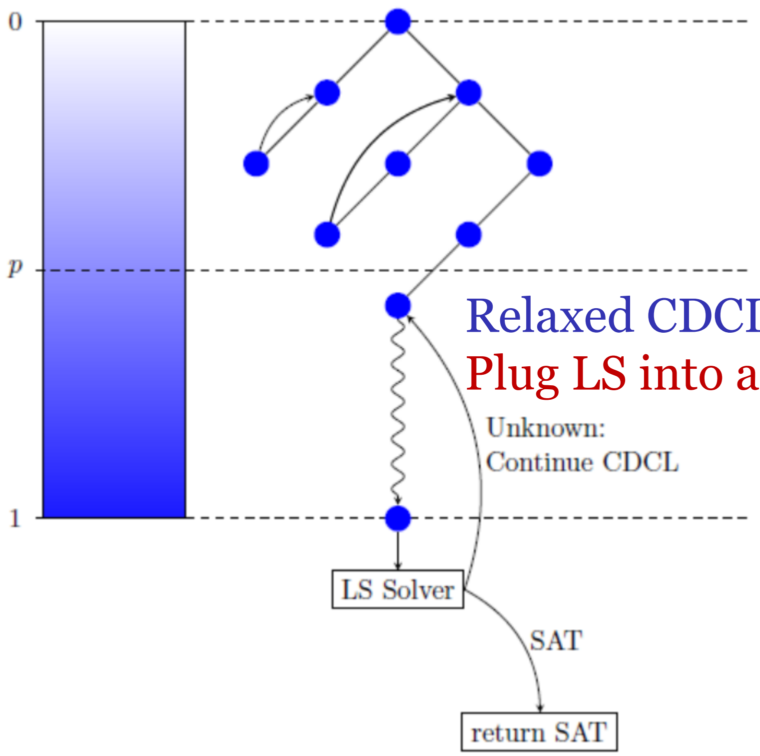
Deep Cooperation of CDCL and Local Search

CDCL focuses on a local space in a certain period

→Better to integrate reasoning techniques

Local search walks in the whole search space

→Better at sampling



Relaxed CDCL:
Plug LS into a CDCL solver

[Cai,Zhang, SAT '21] (best paper).

A short history of this work and similar works independently by Biere is described in [Cai,Zhang,Fleury,Biere, JAIR '22]

- ☐ How to create a full initial assignment?

Relax CDCL and complete the partial assignment by alternating decisions and propagations while ignoring all conflicts

- BCP when possible
- Pick a random unassigned variable, assign it with phase saving heuristic

Improve Branching Heuristics via Local Search

CDCL is powerful owing largely to the utilization of conflict information

CDCL solvers prefer the variable which may cause conflicts faster (e.g. VSIDS)

Can local search information be used to enhance branching heuristics?

Branching with conflict frequency in local search:

- calculate the conflict frequency: frequency of occurring in falsified clauses
- multiply $ls\_confl\_freq(\mathrm{x})$ with 100 , resulting $ls\_confl\_num(\mathrm{x})$

$$\downarrow$$

- improve VSIDS: for each variable x, its activity is increased by $ls\_confl\_num(x)$
- improve LRB: for each variable x, the number of learnt clause during its period I is increased by $ls\_confl\_num(x)$.

Local Search Rephasing

Phase selection is an important component of a CDCL solver.

Most modern CDCL solvers utilize the phase saving heuristic [PipatsrisawatDarwiche, SAT'07]

Local search rephasing

- After each restart of CDCL, reset the saved phases of all variables with assignments by local search.

Phase Name	α_longest_LS	α_latest_LS	α_best_LS	no change
Probability	20%	65%	5%	10%

$\alpha\_longest\_LS$: the assignment of the local search procedure in which the initial solution is extended from the longest branch during past CDCL search.
α_best_LS: the assignment with smallest cost among all local search procedures.
α_latest_LS: the assignment of the latest local search procedure.
(the assignment of a local search procedure is the best found assignment)

Deep Cooperation of CDCL and Local Search

solver	#SAT	#UNSAT	#Solved	PAR2	#SAT	#UNSAT	#Solved	PAR2
	SC2017(351)				SC2018(400)			
glucose_4.2.1	83	101	184	5220.0	95	95	190	5745.9
glucose+rx	88	95	183	20↑	113	95	208	51↑
glucose+rx+rp	112	94	206	[illegible]	141	87	228	[illegible]
glucose+rx+rp+cf	110	94	204	4668.5	150	91	241	4438.2
Maple-DL-v2.1	101	113	214	4531.0	133	102	235	4533.9
Maple-DL+rx	101	112	213	9↑	149	101	250	24↑
Maple-DL+rx+rp	111	103	214	[illegible]	158	93	251	[illegible]
Maple-DL+rx+rp+cf	116	107	223	4139.4	162	97	259	3927.6
Kissat_sat	115	114	229	3↓	167	98	265	17↑
Kissat_sat+cf	113	113	226	[illegible]	178	104	282	[illegible]
CCAnr	13	N/A	13	9629.9	56	N/A	56	8622.0
	SC2019(400)				SC2020(400)			
glucose_4.2.1	118	86	204	5437.6	68	91	159	6494.6
glucose+rx	120	84	204	21↑	93	88	181	62↑
glucose+rx+rp	134	85	219	[illegible]	130	85	215	[illegible]
glucose+rx+rp+cf	140	85	225	4923.6	134	87	221	4977.9
Maple-DL-v2.1	143	97	240	4601.8	86	104	190	5835.7
Maple-DL+rx	146	93	239	9↑	121	105	226	67↑
Maple-DL+rx+rp	155	94	249	[illegible]	142	99	241	[illegible]
Maple-DL+rx+rp+cf	154	95	249	4377.4	151	106	257	4171.1
Kissat_sat	159	88	247	5↑	146	114	260	10↑
Kissat_sat+cf	162	90	252	[illegible]	157	113	270	[illegible]
CCAnr	13	N/A	13	9678.3	45	N/A	45	8978.7

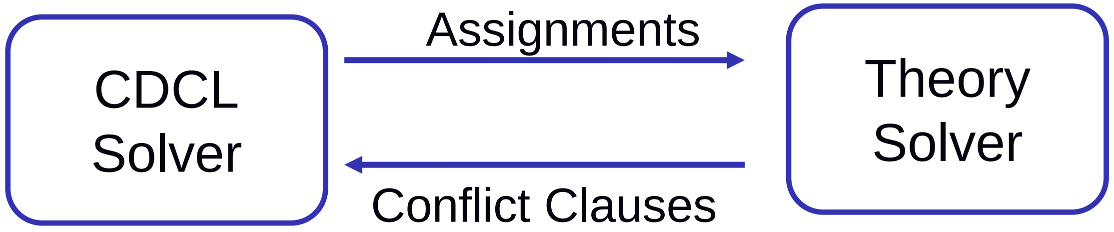
Most winners of main track in recent competitions use this method or similar idea.

#SAT_bonus: solved by hybrid solver, but both original CDCL and LS fail.

Solver	Analysis for SAT				Analysis for UNSAT	
	#byLS	#SAT_bonus	#LS_call	LS_time(%)	#LS_call	LS_time(%)
	SC2017(351)					
glucose+rx	20	11	24.28	21.66	16.36	5.52
glucose+rx+rp	10	33	17.77	18.46	14.33	4.86
glucose+rx+rp+cf	17	29	22.7	22.19	15.3	5.81
Maple+rx	16	9	13.86	7.52	11.18	2.03
Maple+rx+rp	11	15	9.63	10.43	6.54	2.36
Maple+rx+rp+cf	6	16	12.59	7.49	8.59	2.12
	SC2018(400)					
glucose+rx	50	4	11.27	20.66	29.62	4.94
glucose+rx+rp	47	31	9.46	18.4	21.66	5.64
glucose+rx+rp+cf	53	36	11.43	20.28	20.62	6.64
Maple+rx	52	7	4.8	13.02	11.69	2.81
Maple+rx+rp	56	13	4.84	15.21	8.7	3.04
Maple+rx+rp+cf	51	18	6.52	12.53	15.62	2.94
	SC2019(400)					
glucose+rx	14	8	26.46	10.79	17.42	6.39
glucose+rx+rp	10	26	22.68	8.67	14.59	5.14
glucose+rx+rp+cf	11	26	20.39	11.82	15.51	5.95
Maple+rx	14	7	12.66	2.67	12.94	1.98
Maple+rx+rp	9	14	8.6	3.17	16.59	2.53
Maple+rx+rp+cf	12	15	11.21	3.05	17.23	2.22
	SC2020(400)					
glucose+rx	30	9	14.94	11.75	14.67	10.27
glucose+rx+rp	23	37	13.17	10.79	9.4	9.71
glucose+rx+rp+cf	23	37	12.78	11.67	10.52	10.28
Maple+rx	19	13	14.21	6.69	10.24	5.25
Maple+rx+rp	30	29	8.53	6.62	11.7	6.18
Maple+rx+rp+cf	23	36	10.95	6.05	14.17	5.42

Lift the Hybrid Method to SMT

CDCL(T): CDCL deals with the skeleton, while theory solver solve the conjunction of theory literals and learn lemmas.



CDCL(T) guides local search:

When CDCL(T) finds a satisfying assignment to Boolean skeleton → extract a subformula F of the true literals → run local search at F

Example

$(p_1 \vee \neg p_2) \wedge (\neg(3x_1x_2 \leq 2) \vee (-x_2 - 3x_4 \leq 0))$

Boolean skeleton: $(p_1 \vee \neg p_2) \wedge (\neg p_{\sigma_1} \vee p_{\sigma_2})$

satisfying assignment to skeleton

$\{p_1 \rightarrow T, p_{\sigma_1} \rightarrow F, p_2 \rightarrow F\}$

→ $(p_1 \vee \neg p_2) \wedge \neg(3x_1x_2 \leq 2)$

Lift the Hybrid Method to SMT

Local search enhances phasing heuristic:

word-level assignments by local search → assignments to Boolean encoders → used in phasing heuristic of CDCL

Local search enhances ordering (branching) heuristic:

calculate the conflict frequency of each Boolean encoder (i.e., atomic formula),

add to VSIDS scoring function.

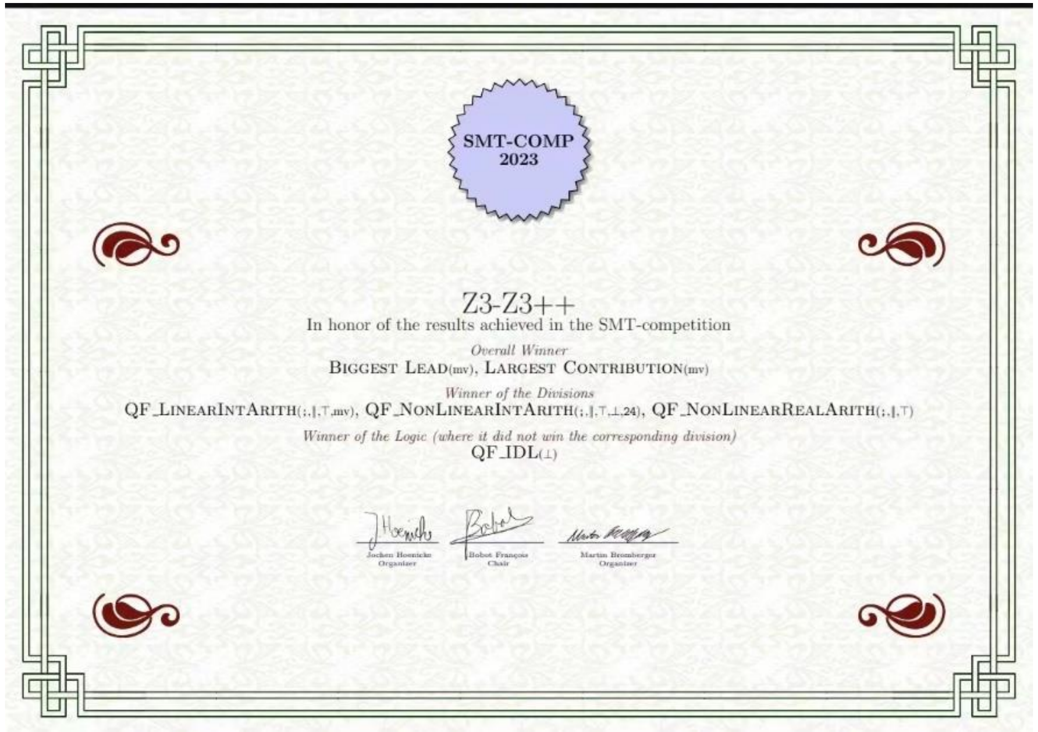
Integrate Local Search in Z3

Z3++

- integrating local search solvers for arithmetic theories into Z3.
- Cooperation between CDCL(T) and local search

Z++ in SMT-Comp 2022 and 2023

- Biggest Lead Model Validation
- Largest Contribution Model Validation
- Winning “single query” and “model validation” tracks of LIA, NIA, NRA Divisions



Local Search and Its Application in CDCL/CDCL(T) Solvers for SAT/SMT

Shaowei Cai

Institute of Software, Chinese Academy of Sciences

caisw@ios.ac.cn