Chaff: Engineering an Efficient SAT Solver

Matthew W.Moskewicz,
Concor F. Madigan, Ying Zhao, Lintao Zhang, Sharad Malik
Princeton University

Slides: Tamir Heyman
Some are from Malik's presentation

Last modified by Will Klieber on Sep 7, 2011

Boolean Algebra Notation

- "$+$" denotes logical OR ("$\vee$").
- "$\cdot$" denotes logical AND ("$\wedge$").
- Overbar or postfix "$'$" denotes negation.
- Example:
  "$(A \vee (\neg B \wedge C))$" corresponds to
  "$(A + (B' \cdot C))$".

Chaff Philosophy

- Make the core operations fast
  - profiling driven, most time-consuming parts:
    - Boolean Constraint Propagation (BCP) and Decision
- Emphasis on coding efficiency
- Emphasis on optimizing data cache behavior
- Search space pruning:
  - conflict resolution and learning

Chaff's Main Procedures

- Efficient BCP
  - Two watched literals
  - Fast backtracking
- Efficient decision heuristic
  - Localizes search space
- Restarts
  - Increases robustness

Implication

- What "causes" an implication?
- When can it occur?
- All literals in a clause but one are assigned False.

Implication example

- The clause (v1 + v2 + v3) implies values only in the following cases.
- In case (F + F + v3)
  - implies v3=T
- In case (F + v2 + F)
  - implies v2=T
- In case (v1 + F + F)
  - implies v1=T

Implication for N-literal clause

- Implication occurs after N-1 assignments to False to its literals.
- We can ignore the first N-2 assignments to this clause.
- The first N-2 assignments won't have any effect on the BCP.

Watched Literals

- Each clause has two watched literals.
- Ignore any assignments to the other literals in the clause.
- BCP maintains the following invariant:
  - By the end of BCP, one of the watched literals is true or both are unassigned.
  - (Can watch a false literal only if other watch is true.)
- Guaranteed to find all implications found by normal unit prop.

BCP with watched Literals

- Identifying conflict clauses
- Identifying unit clauses
- Identifying associated implications
- Maintaining “BCP Invariant”

Example (1/13)

Input formula has the following clauses:

v2 + v3 + v1 + v4

v1 + v2 + v3′

v1 + v2′

v1′+ v4

(v1′) means (¬v1)

Example (2/13)

Initially, we identify any two literals in each clause as the watched ones

Watched literals

v2 + v3 + v1 + v4

v1 + v2 + v3′

v1 + v2′

v1′+ v4

(v1′) means (¬v1)

Example (3/13)

Stack:(**v1=F**)

v2 + v3 + v1 + v4
v1 + v2 + v3'
v1 + v2'
v1' + v4

Assume we decide to set v1 the value F

Example (4/13)

Stack:(**v1=F**)

v2 + v3 + v1 + v4

v1 + v2 + v3′

v1 + v2′

→ v1′+ v4

- Ignore clauses with a watched literal whose value is T.
  - (Such clauses are already satisified.)

Example (5/13)

Stack:(**v1=F**)

→ v2 + v3 + v1 + v4

v1 + v2 + v3'

v1 + v2'

v1'+ v4

- Ignore clauses where neither watched literal value changes

Example (6/13)

Stack:(**v1=F**) → v2 + v3 + v1 + v4

→ v1 + v2 + v3’

→ v1 + v2’

v1’+ v4

- Examine clauses with a watched literal whose value is F

Example (7/13)

v2 + v3 + v1 + v4
v1 + v2 + v3’
v1 + v2’
v1’+ v4

Example (7/13)

<u>v2</u> + <u>v3</u> + v1 + v4	→	<u>v2</u> + <u>v3</u> + v1 + v4
<u>v1</u> + <u>v2</u> + v3’		<u>v1</u> + <u>v2</u> + <u>v3’</u>
<u>v1</u> + <u>v2’</u>		<u>v1</u> + <u>v2’</u>
<u>v1’</u>+ <u>v4</u>		<u>v1’</u>+ <u>v4</u>
Stack:(**v1=F**)		Stack:(**v1=F**)

- In the second clause, replace the watched literal v1 with v3’

Example (8/13)

v2 + v3 + v1 + v4
v1 + v2 + v3'
v1 + v2'
v1' + v4

Stack:(**v1=F**)

→

v2 + v3 + v1 + v4
v1 + v2 + v3'
v1 + v2'
v1' + v4

Stack:(v1=F)
Pending: (v2=F)

- The third clause is a unit and implies v2=F
- We record the new implication, and add it to a queue of assignments to process.

Example (9/13)

v2 + v3 + v1 + v4 → v2 + v3 + v1 + v4
v1 + v2 + v3’ → v1 + v2 + v3’
v1 + v2’   v1 + v2’
v1’+ v4   v1’+ v4

Stack:(v1=F, **v2=F**)

Stack:(v1=F, v2=F)
Pending: (v3=F)

- Next, we process v2.
- We only examine the first 2 clauses

Example (10/13)

v2 + v3 + v1 + v4 → v2 + v3 + v1 + v4

v1 + v2 + v3’ → v1 + v2 + v3’

v1 + v2’   v1 + v2’

v1’+ v4   v1’+ v4

Stack:(v1=F, **v2=F**)   Stack:(v1=F, v2=F)
Pending: (v3=F)

- In the first clause, we replace v2 with v4
- The second clause is a unit and implies v3=F
- We record the new implication, and add it to the queue

Example (11/13)

v2 + v3 + v1 + v4 → v2 + v3 + v1 + v4
v1 + v2 + v3′ → v1 + v2 + v3′
v1 + v2′ → v1 + v2′
v1′+ v4 → v1′+ v4

Stack:(v1=F, v2=F, **v3=F**) → Stack:(v1=F, v2=F, v3=F)
Pending: ()

- Next, we process v3′. We only examine the first clause.

Example (12/13)

v2 + v3 + v1 + v4
v1 + v2 + v3'
v1 + v2'
v1'+ v4

Stack:(v1=F, v2=F, **v3=F**)

→

v2 + v3 + v1 + v4
v1 + v2 + v3'
v1 + v2'
v1'+ v4

Stack:(v1=F, v2=F, v3=F)
Pending: (v4=T)

- The first clause is a unit and implies v4=T.
- We record the new implication, and add it to the queue.

Example (13/13)

v2 + <u>v3</u> + v1 + <u>v4</u>
v1 + <u>v2</u> + <u>v3'</u>
<u>v1</u> + <u>v2'</u>
<u>v1'</u>+ <u>v4</u>

Stack:(v1=F, v2=F, v3=F, v4=T)

- There are no pending assignments, and no conflict
- Therefore, BCP terminates and so does the SAT solver

Identify conflicts

v2 + v3 + v1
v1 + v2 + v3'
v1 + v2'
v1'+ v4

Stack:(v1=F, v2=F, **v3=F**)

- What if the first clause does not have v4?
- When processing v3', we examine the first clause.
- This time, there is no alternative literal to watch.
- BCP returns a conflict

Backtrack

$\underline{v2} + \underline{v3} + v1$

$v1 + \underline{v2} + \underline{v3'}$

$\underline{v1} + \underline{v2'}$

$\underline{v1'} + \underline{v4}$

Stack:()

- We do not need to move any watched literal

BCP Summary

- During forward progress (decisions, implications)
  - Examine clauses where watched literal is set to F
  - Ignore clauses with assignments of literals to T
  - Ignore clauses with assignments to non-watched literals

Backtrack Summary

- Unwind Assignment Stack
- No action is applied to the watched literals
- Overall
  - Minimize clause access

Chaff Decision Heuristic VSIDS

- Variable State Independent Decaying Sum
  - Rank variables based on literal count in the initial clause database.
  - Only increment counts as new clauses are added.
  - Periodically, divide all counts by a constant.

VSIDS Example (1/2)

Initial data base

x1 + x4
x1 + x3′ + x8′
x1 + x8 + x12
x2 + x11
x7′ + x3′ + x9
x7′ + x8 + x9′
x7 + x8 + x10′

Scores:
4: x8
3: x1,x7
2: x3
1: x2,x4,x9,x10,x11,x12

New clause added

x1 + x4
x1 + x3′ + x8′
x1 + x8 + x12
x2 + x11
x7′ + x3′ + x9
x7′ + x8 + x9′
x7 + x8 + x10′
x7 + x10 + x12′

Scores:
4: x8,x7
3: x1
2: x3,x10,x12
1: x2,x4,x9,x11

watch what happens to x8, x7 and x1

VSIDS Example (2/2)

Counters divided by 2

x1 + x4
x1 + x3′ + x8′
x1 + x8 + x12
x2 + x11
x7′ + x3′ + x9
x7′ + x8 + x9′
x7 + x8 + x10′
x7 + x10 + x12′

Scores:
2: x8,x7
1: x3,x10,x12,x1
0: x2,x4,x9,x11

New clause added

x1 + x4
x1 + x3′ + x8′
x1 + x8 + x12
x2 + x11
x7′ + x3′ + x9
x7′ + x8 + x9′
x7 + x8 + x10′
x7 + x10 + x12′
x12′ + x10

Scores:
2: x8,x7,x12,x10
1: x3,x1
0: x2,x4,x9,x11

watch what happens to x8, x10

VSIDS - Summary

- Quasi-static:
  - Static because it is independent of variable values
  - Not static because it gradually changes as new clauses are added
    - Decay causes bias toward *recent* conflicts.
- Use heap to find an unassigned variable with the highest ranking

Interplay of BCP and the Decision Heuristic

- This is only an intuitive description …
  - Reality depends heavily on specific instances
- Take some variable ranking
  - Assume several decisions are made
  - Say v2=T, v7=F, v9=T, v1=T (and any implications thereof)

Interplay of BCP and the Decision Heuristic (cont')

- Then a conflict is encountered and forces v2=F
- The next decisions may still be v7=F, v9=T, v1=T
  - VSIDS variable ranks change **slowly**…
- But the BCP engine has recently processed these assignments …
  - so these variables are unlikely to **still be watched**.

Interplay of BCP and the Decision Heuristic (cont’)

- In a more general sense
- The more “active” a variable is, the more likely it is to *not* be watched.
- Because BCP is likely to replace it

Interplay of Learning and the Decision Heuristic

- Again, this is an intuitive description …
- Learned clauses capture relationships between variables
- Decision heuristic influences which variables appear in learned clauses
  - Decisions →implications →conflicts →learned clause

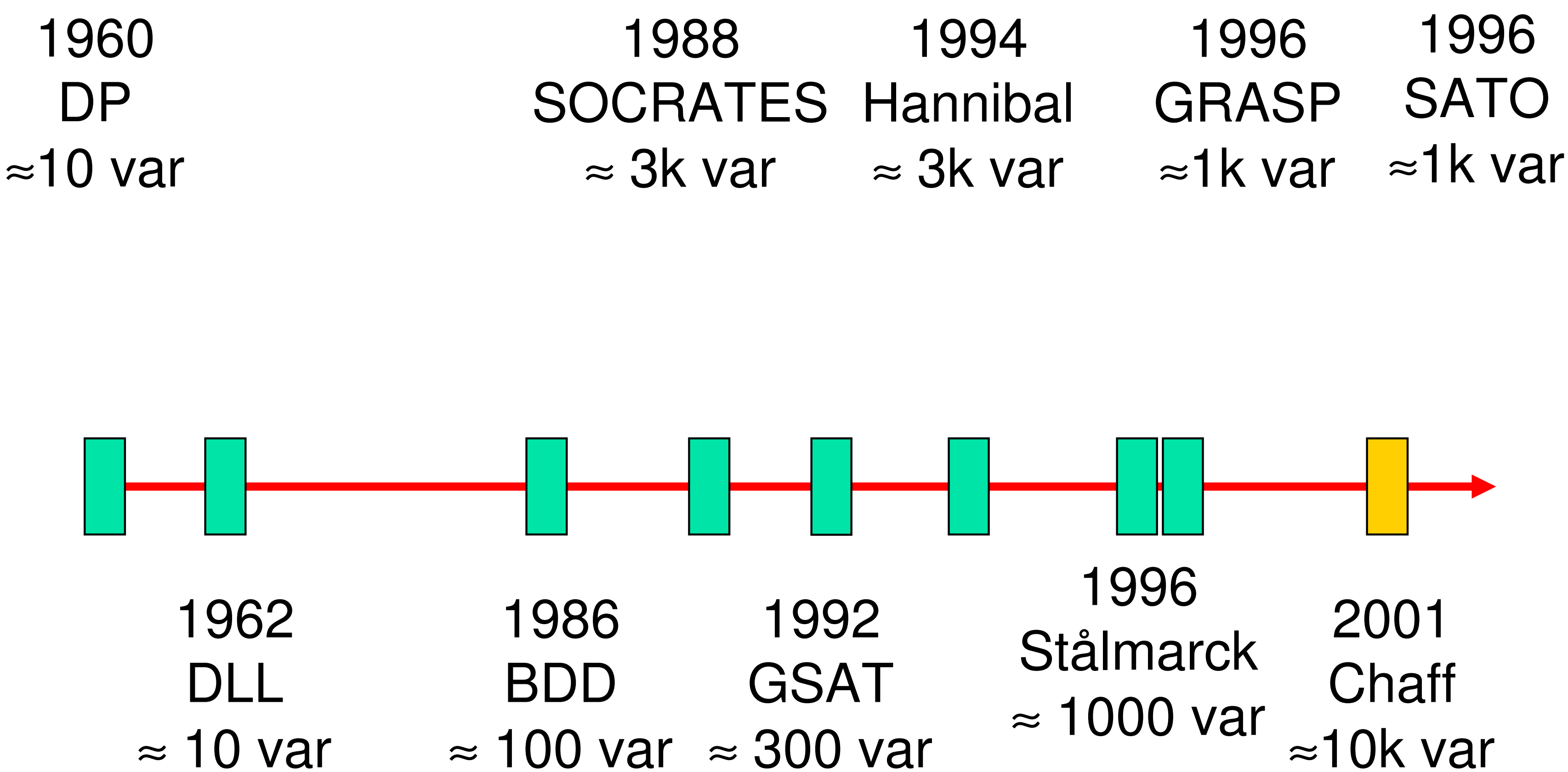
Interplay of Learning and the Decision Heuristic (cont')

- Important for decisions to keep the search strongly localized
  - Especially when there are 100k variables!
- In VSIDS, learned clauses bias decision strategy
  - Focusing in a smaller set of variables

Restart

- Abandon the current search tree and reconstruct a new one
- Helps reduce runtime variance between instances- adds to robustness of the solver
- The clauses learned prior to the restart are *still there* after the restart and can help pruning the search space

Timeline


1960
DP
≈10 var
1988
SOCRATES
≈ 3k var
1994
Hannibal
≈ 3k var
1996
GRASP
≈1k var
1996
SATO
≈1k var
1962
DLL
≈ 10 var
1986
BDD
≈ 100 var
1992
GSAT
≈ 300 var
1996
Stålmarck
≈ 1000 var
2001
Chaff
≈10k var