Watched Literals for Constraint Propagation in Minion

Ian Gent, Chris Jefferson, Ian Miguel

Minion History

- Introduced at ECAI ’06.
- Designed to be a fast black-box solver.
- Question: Can a CSP solver be as fast as a SAT solver at SAT?

Minion Is Not...

- Bug-free.
- General purpose.
    - Limited Dynamic Heuristics.
- Capable of hybridising.
- Good with very large domain variables.
- Have many global constraints.

SAT

- SAT is a tiny subset of CSP.
  - Boolean variables
  - Conjunctions of literals.
- Example: $x \vee \neg y \vee z$
- All SAT problems are CSP problems.
- So why use a SAT solver?

Results - QG7.13

	Nodes/sec	*Slower than SAT*
ILOG 6.3	*25*	*197*
Minion	*397*	*12*
WL-Minion	*1,728*	*1.8*
MiniSAT	*4,932*	*1*

Why is SAT fast?

- Highly optimised black-box solvers.
  - Minion our attempt at an optimised black-box, but still much slower than SAT solvers.
- What else do SAT solvers have?

Why is SAT fast?

- Complete SAT (and CSP) solvers have 3 major components:

Why is SAT fast?

- Complete SAT (and CSP) solvers have 3 major components:
  - Variable / value heuristic.

Why is SAT fast?

- Complete SAT (and CSP) solvers have 3 major components:
    - Variable / value heuristic.
    - Learning.

Why is SAT fast?

- Complete SAT (and CSP) solvers have 3 major components:
  - Variable / value heuristic.
  - Learning.
  - Propagation.

Propagation in CP

- Constraints attach a trigger to each variable they want to be informed about.
- Different types of trigger:
  - Domain Value (Literal) Removed.
  - Bounds Changed.
  - Variable Assigned.

Propagation Example

$a \vee b \vee c \vee d \vee e$

- 1 simple rule to get all propagation:
  - If all but one variable assigned false: assign other variable true.
- This implies: If variables false, fail.

Propagation in CSPs

- Propagation in a traditional CSP solver:
  - Algorithm run whenever a variable assigned.
  - Add 1 to a counter if variable assigned false.
  - When counter high enough, find unassigned variable and assign.

Propagation

- Can we reduce / change those requirements?
  - Need to trigger on all assignments?
  - Need to count assigned variables?

‘Watched Literals’

- Different from normal triggers:
  - Cheap to move to different literals.
  - Not restored on backtrack.

Watched Literals for SAT

- Idea: If two variables are either unassigned or assigned true, no need to do anything.
- So just find two variables which satisfy this condition.
- If can't find two, may have to propagate / fail.

Propagation Example

0/1	*0/1*	*0/1*	*0/1*
a	*b*	*c*	*d*

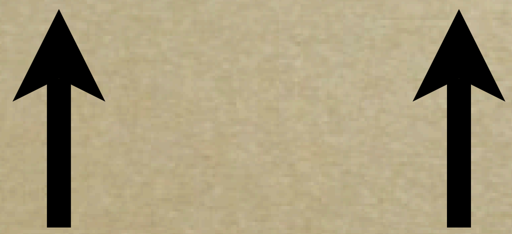
Triggers: ↑ ↑

- $a \vee b \vee c \vee d$

Propagation Example

0	*0/1*	*0/1*	*0/1*
a	*b*	*c*	*d*

Triggers:



- *a* assigned false.
- Update pointer.

Propagation Example

0	*0/1*	*0/1*	*0/1*
a	*b*	*c*	*d*

Triggers:

- *a* assigned false.
- Update pointer.

Propagation Example

0/1	*0/1*	*0/1*	*0/1*
a	*b*	*c*	*d*

Triggers:

- Backtrack. *a* unassigned.
- **Pointers do not move back**

Propagation Example

0/1	***1***	*0/1*	*0/1*
a	***b***	*c*	*d*

Triggers:

- If *b* is assigned true, pointer doesn't move.

Propagation Example

0	*0/1*	*0/1*	***0***
a	*b*	*c*	***d***

Triggers: ↑ ↑

- If other variables assigned, nothing happens!

Propagation Example

0	***0***	*0/1*	*0*
a	***b***	*c*	*d*

Triggers: ↑ (b) ↑ (c)

- If we cannot find something new to watch...

Propagation Example

0	*0*	***1***	*0*
a	*b*	***c***	*d*

Triggers:

- Assign other watch!

Watched Literals vs. CP

- CP:
  - Trigger on all variables.
  - O(1) cost on trigger.
- Watched:
  - Trigger on 2 variables
  - O(n) cost on trigger.
- Exactly the same propagation.

Advantages of WL

- **ZERO** cost if a literal not watched.
- **ZERO** cost on backtrack.

Practical Advantages of WL

- If watches move to assigned variables - no work.
- Usually takes few checks to find a new literal to watch.
- During search, watches move to “safe” literals and not triggered often.

Advantages of WL

- With watched literals, the “less important” a constraint is, the cheaper it is during search.
  - Observed many times in SAT.
  - Why SAT solvers can add a huge number of learned clauses with little cost.

Implementation Difficulties

- Changes deep in solver.
  - Important to make moving cheap.
- A constraint can watch same literal multiple times.
- Watches can be left on deleted literals.
- Important to make moving watched literals very cheap.

Implemented Constraints

- Table (extensional) constraint
- Element
- Array $\neq$
- Max, Min
- Non-GAC AllDiff
- Occurrence (not GCC)

Implementing Element with WL

- $M[Index] = Result$
- Array of variables M.
- Variables $Index$ and $Result$.
- There are 3 conditions which must be satisfied for this constraint to be GAC.

Element: Condition 1

- *Index* can be assigned i if $M[i] = R$ is possible.

Element: Condition 1

- *Index* can be assigned i if $M[i] = R$ is possible.
  - Look for v where:
    v in domain of $M[i]$
    v in domain of R.

Element: Condition 1

- *Index* can be assigned i if $M[i] = R$ is possible.
  - Look for v where:
    v in domain of $M[i]$
    v in domain of R.
  - If found, watch v in $M[i]$ and R.

Element: Condition 1

- *Index* can be assigned i if $M[i] = R$ is possible.
  - Look for v where:
    v in domain of $M[i]$
    v in domain of R.
  - If found, watch v in $M[i]$ and R.
  - If not found, remove i from *Index*.

Element: Condition 2

- *Result* can be assigned r if $M[X] = r$ is possible.

Element: Condition 2

- *Result* can be assigned r if $M[X] = r$ is possible.
  - Look for x where:
    x in domain of X
    r in domain of $M[x]$.

Element: Condition 2

- *Result* can be assigned r if $M[X] = r$ is possible.
  - Look for x where:
    x in domain of X
    r in domain of $M[x]$.
  - If found, watch x in X and r in $M[x]$

Element: Condition 2

- *Result* can be assigned r if $M[X] = r$ is possible.
  - Look for x where:
    x in domain of X
    r in domain of $M[x]$.
  - If found, watch x in X and r in $M[x]$
  - If not found, remove r from *Result*.

Element: Condition 3

- Once X is assigned, M[X] and R must have the same domain.
- Can be implemented in an old-fashioned way.

Element Constraint

- Algorithm very simple (I think).
- Follows naturally from maths.
- | dom(Result) | = r, | dom(Index) | = i
- Watches = $2r + 2i + 2i$
- Literals = $r + i + \boldsymbol{ri}$

Watched Literals

- All watched literals found so far follow a similar basis.
  - Find ‘proof’ assignments should not be removed, watch it.
  - When no proof can be found, remove values.

Conclusions

- Watched Literals are good when a "proof" the constraint is true is small.
- SAT : 1 variable.
- Element : 3 Variables (X,Y, M[X]).
- Array ≠ : 2 variables (1 index).
- Improvements on Minion's table too.

Conclusions

- Watched literals can massively improve the performance of constraints solvers.
- They can be used to implement many types of constraints.
- May provide an easier way of designing and implementing some propagators?
- Close the gap between SAT and CP.

Results - QG7.13

	Nodes/sec	*Slower than SAT*
ILOG 6.3	*25*	*197*
Minion	*397*	*12*
WL-Minion	*1,728*	*1.8*
MiniSAT	*4,932*	*1*

Results - QG7.13

	Time	*Nodes Searched*
ILOG 6.3	*>1h*	*312,108*
Minion	*786*	
WL-Minion	*180*	
MiniSAT	*0.27*	*1,307*

Thank you

Any Questions?