CrystalBall: **How to create your custom SAT solver**

Arijit Shaw[1] Mate Soos[1] Raghav Kulkarni[2] Kuldeep S. Meel[1]

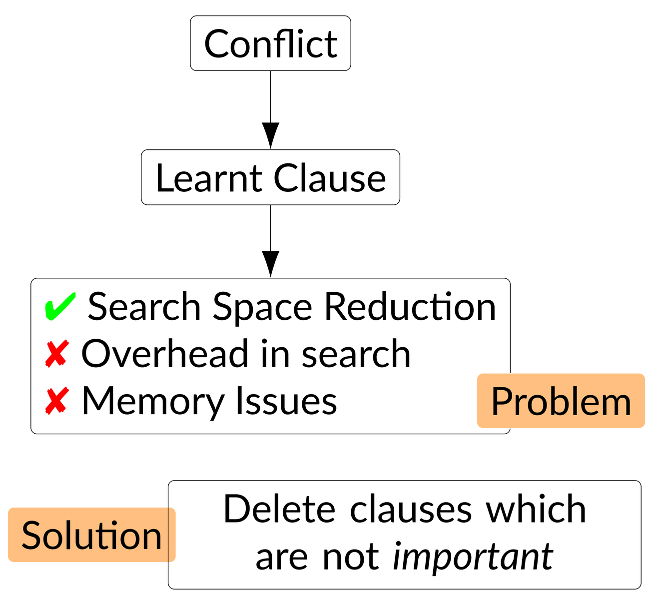
[1]School of Computing, National University of Singapore [2]Chennai Mathematical Institute

CDCL(F)

$A \leftarrow \{\}$
while $hasUnassignedVars(F, A)$ do
  $A \leftarrow A \cup$ PickBranchingLiteral(F, A)
  while UnitPropagation$(F, A) = conflict$ do
    $\langle b, c \rangle \leftarrow$ AnalyzeConflict()
    if $b < 0$ then
      return $unsat$
    else
      $Backtrack(F, A, b)$
  if $ClauseDeletionRequired(F)$ then
    ReduceLearntClauseDB(F)
return sat

Figure 1. Framework of a CDCL Algorithm. The green parts are those where we need heuristics.

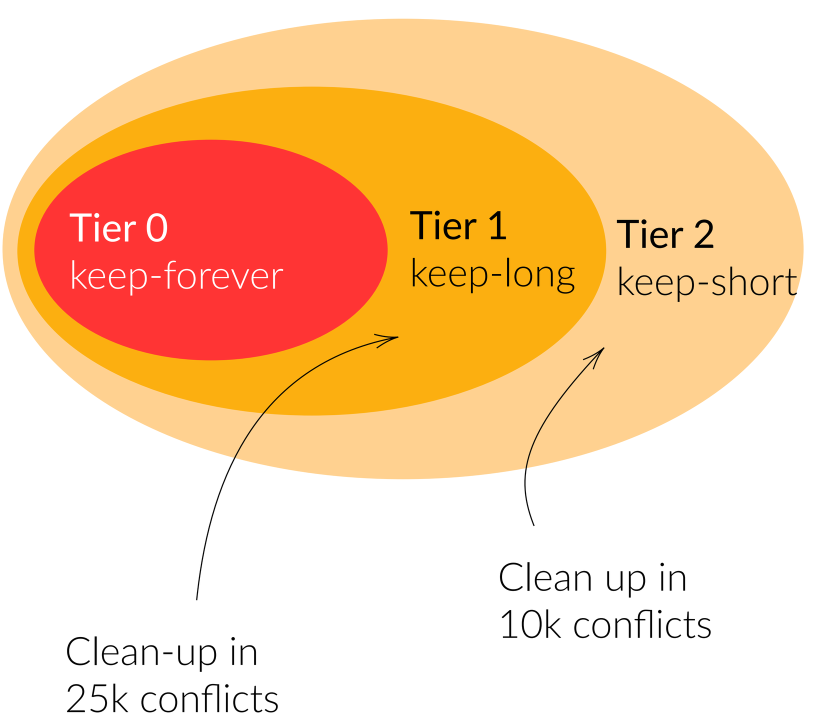
Clause deletion



Modern Solvers	CrystalBall(v1)
Delete clauses based on activity / LBD.	Learn the heuristic as a classifier by looking at execution data.

Clause deletion is shown in the CDCL algorithm (above) as ReduceLearntClauseDB(F).

Clause Maintainance in Modern Solvers



With CrystalBall(v1) we create classifier for this Tier 1 and Tier 2. These classifiers are named *keep-long* and *keep-short* respectively.

Open Ends

1. **Knowing exact objective for clause learning.** Current labelling is based on usage in future.
2. **More features.** = More accuracy = Better solver. CrystalBall always crave for new features.
3. **Normalized features.** SVM or random forest often work better while features are normalized.
4. **Learning other policies** like restart and branching.
5. **Other models** like neural nets or reinforcement learning.

The Goal

The annual SAT competition witnesses :

1. The top solvers are based on Conflict-Driven Clause Learning (CDCL), the structures are mostly same.
2. Yet there is an impressive improvement in performance from last year.
3. The major difference is made by some newly invented heuristics.

In this context, we ask, given white-box access to the execution of SAT solver, can we synthesize algorithmic heuristic for the solver?

The project CrystalBall aims to seek an answer to this.

CrystalBall(v1) aimed at learning heuristic for **clause deletion**.

CrystalBall : Data Pipeline

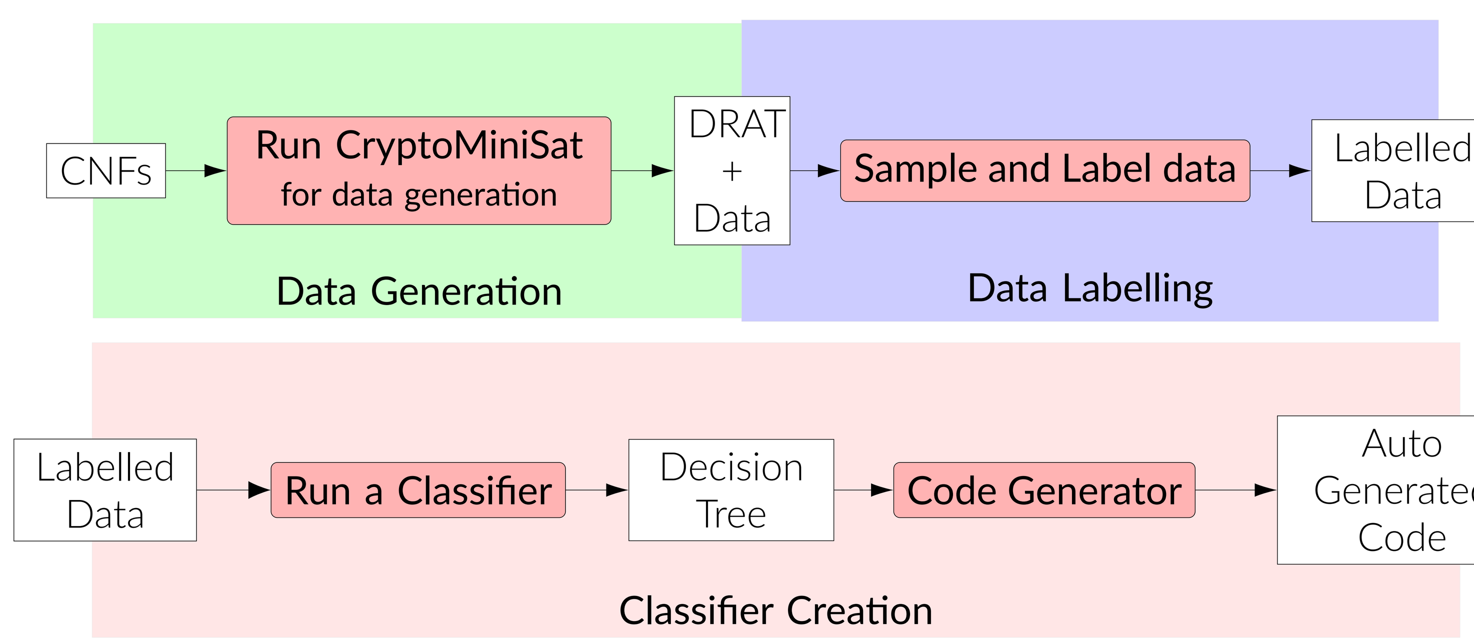


Figure 2. Steps in CrystalBall

Phase 1 : Data Collection

- On UNSAT instances,run a customized version of CryptoMiniSat, that do not employ clause deletion at all.
- This logs a lot (212 in v1) of *features* about learnt clauses while the clause gets generated or used.

Phase 2 : Data Labeling

- Hack into DRAT proofs, and this gives us idea about usage of the learnt clause.
- Based on this usage we *label* a clause as *important* (should be *kept*) or not (should be *thrown away*).

Phase 3 : Classifier Creation

- On this labelled data, use scikit-learn to create a classifier. Choose from decision trees, random forests, or, SVMs.
- Now we parse this decision tree and spit out C++ code that is plugged back to CryptoMiniSat. We call this version PredCryptoMiniSat.

Feature Engineering

Feature Type	Example
Global	# variables
Contextual	# literals in clause, LBD score.
Restart	trail depth, branch depth
Performance	last time used in a conflict

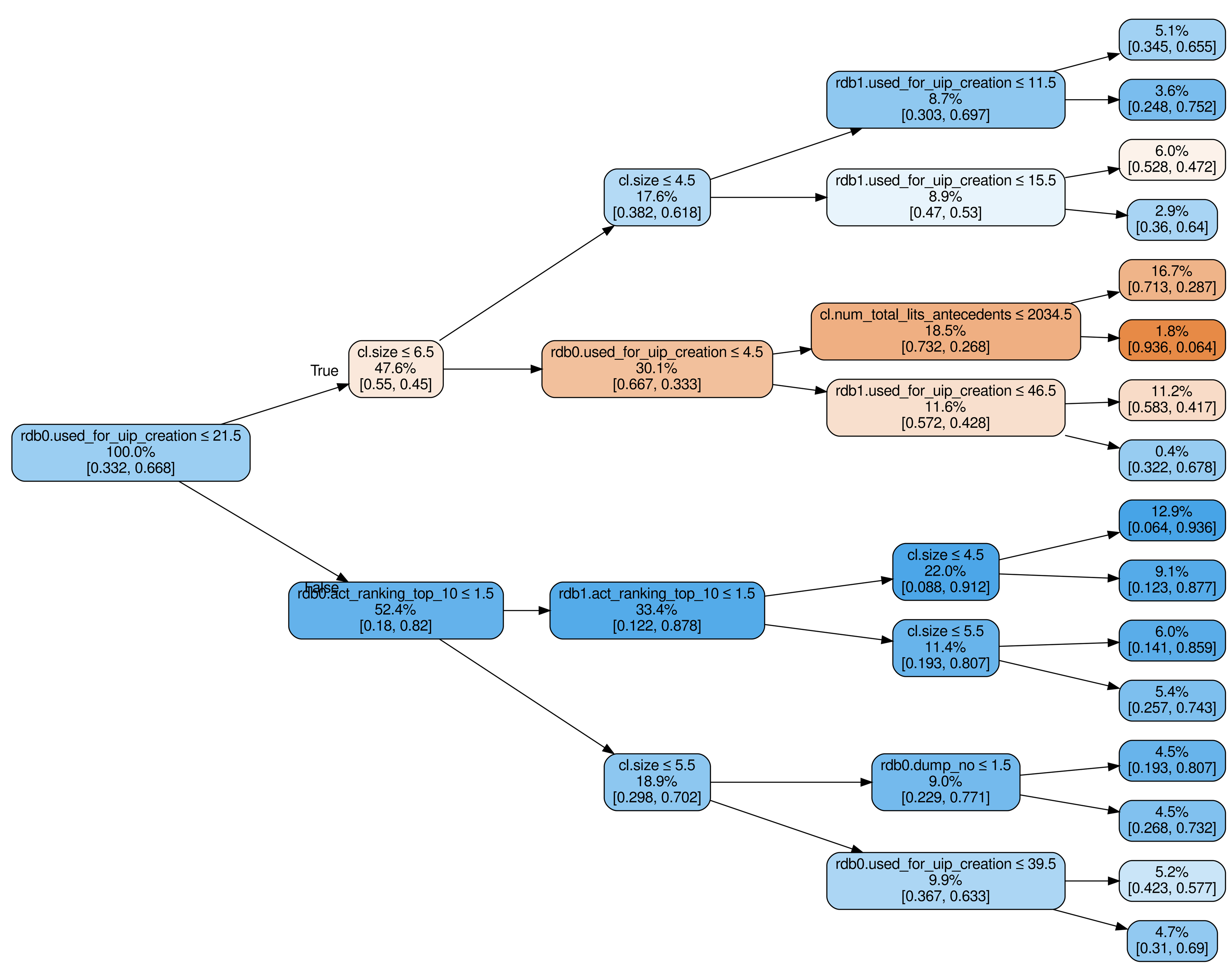


Figure 3. Decision tree from SHA1 preimage attack benchmarks.

Some examples please . . .

A DRAT Proof

```
p cnf          1 -2  3 0 0
-2  3 0        2  1  3 0 0
 1  3 0        3 -1  2 0 0
-1  2 0        4 -1 -2 0 0
-1 -2 0        5  1 -2 0 0
 1 -2 0        6  2 -3 0 0
 2 -3 0        7 -2  0 4 5 0
 3  6 0        8  3  0 1 2 3 0
               9  0  6 7 8 0
```

Labelled Data

glue	size	used_last_10k	activity rank	label
10	15	3	top half	keep
7	10	1	bottom half	throw
3	7	0	bottom half	throw

Figure 4. Excerpt from a table generated by CrystalBall

Performance

We create two instance of PredCryptoMiniSat :

1. PCMS-satcomp : trained with SAT competition benchmarks.
2. PCMS-sha1 : trained with CNFs that are example of preimage attack on SHA1 algorithm.

	Benchmark	
solver	SAT comp	SHA1
CryptoMiniSat	2176	1129
PCMS-satcomp	2440	1263
PCMS-sha1	2805	1165

Some Machine Learning Statistics

During training with SHA-1 benchmark, the normalized confusion matrix looked like the following:

		Predicted	
		Throw	Keep
Actual	Throw	0.82	0.18
	Keep	0.11	0.89

The heuristics learnt for SHA1 benchmarks shown to the left.

Heuristics we've learnt

According to CrystalBall, while training with SAT competition, the top features that should decide are the following :

keep-short	keep-long
rdb0.used for uip create	rdb0.sum uip1 used
rdb0.last touched diff	rdb1.sum uip1 used
rdb0.activity rel	rdb0.used for uip create
rdb0.sum uip1 used	rdb0.act ranking
rdb1.sum uip1 used	rdb0.act ranking top 10
rdb1.activity rel	rdb0.last touched diff
(a)	(b)

What can we expect from CrystalBall ?

1. **A "Configurable" SAT solver.** For specific industrial / academic purpose.
2. **A portfolio solver**, Like SATZilla, this will look at the problem instance and decide which heuristic it should use.
3. **Aid in designing heuristics** with an in-depth data-driven understanding.

More Resources here

Paper [1] Mate Soos, Raghav Kulkarni, and Kuldeep S Meel. **Crystalball: Gazing in the black box of sat solving.** In International Conference on Theory and Applica ons of Satisfiability Testing, pages 371–387. Springer, 2019.

Blog [2] CrystalBall: SAT solving, Data Gathering, and Machine Learning. `https://www.msoos.org/2019/06/crystalball-sat-solving-data-gathering-and-machine-learning/`