P-Optimal CSP Solvers Applied to Software Security

Ahmed Abdel Mohsen and Karl Lieberherr
College of Computer and Information Science
Northeastern University, Boston, MA 02115
mohsen@ccs.neu.edu lieber@ccs.neu.edu

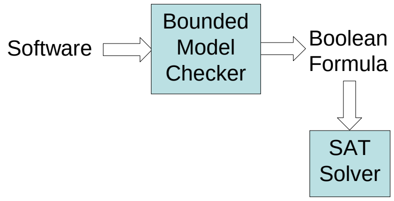
Northeastern UNIVERSITY
Higher Learning. Richer Experience.

The Institute for Information Assurance

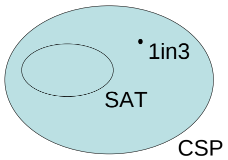
Application of SAT solvers to security:

Reps and Bryant et al. (2005) [1] use a bounded model checker called UCLID to check for exploits at the API level. UCLID translates the problem into one of checking the validity of a Boolean formula, which is checked using a SAT solver (CSP solver).

CSP solvers have many applications besides software security. We hope that our CSP Solver technology will also be useful in Computational Biology, such as pathway modeling based on planning and in Bayesian Inference.



[1] "Automatic Discovery of API-Level Exploits" vinod Ganapathy, Sanjit Seshia, Somesh Jha, Thomas Reps and Randal Bryant. International Conference on Software Engineering (ICSE 2005).



CSP:

A CSP solver accepts a formula consisting of a fixed set of Boolean relations. An example of such a Boolean relation is "1in3 (x y z)" which is satisfied if exactly one of its parameters is true. The following example shows a sample input to a CSP solver:

1in3 (X1 X2 X3), 1in3 (X1 X3 X4), 1in3 (X1 X2 X4)

Our Approach:

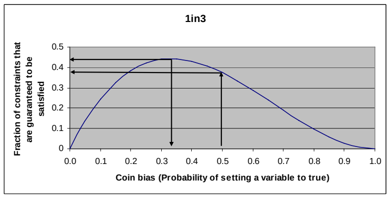
1. **Optimally biased coin:** Universal P-optimal algorithm for fixed set of relations. [2]&[3] prove that if there is a better polynomial algorithm than using the optimally biased coin then P=NP. Explore value orderings and variable orderings.
2. **Derandomization:** explore with and without repeated optimization.
3. **Clause learning:** explore back jump clauses versus semi-superresolvents.
4. **Bitwise relation reduction:** experiment with bit-wise relation manipulation for CSP to speed up Unit Propagation, a very important operation in CSP solvers.

[2] "Algorithmic extremal problems in combinatorial optimization" Karl J. Lieberherr. Journal of Algorithms (1982).
[3] "Complexity of Partial Satisfaction" Karl J. Lieberherr and Ernst Specker. Journal of the ACM (1981).

Optimally Biased Coin:

Suppose that: for every variable in the formula, we flip a coin. If head, we set the variable to true otherwise we set it to false. If our coin is fair (i.e., it produces heads and tails with equal probability) then we can expect that half of the variables will be set to true and the other half will be set to false. However, this does not always lead to a the best possible result.

We can improve over this by biasing our coin (based on the formula) such that we can guarantee the maximum possible fraction of satisfied clauses. This fraction is called a P-optimal threshold because if anyone can guarantee a trillionth more, then P=NP. For example, for the 1in3 example described above, if we biased our coin so that it produces heads with probability 1/3 we guarantee that 4/9 of the clauses are guaranteed to be satisfied. The set of 1in3 problems where the fraction 4/9+trillionth can be satisfied is NP-complete. If we used a fair coin, only 3/8 of the clauses are guaranteed to be satisfied.



Derandomization:

Even if we are using a fair coin. We might flip our coin and get heads all the time. This means that we might be unable to satisfy the P-optimal threshold. Derandomization is about "guaranteeing" the P-optimal threshold deterministically.

Clause Learning:

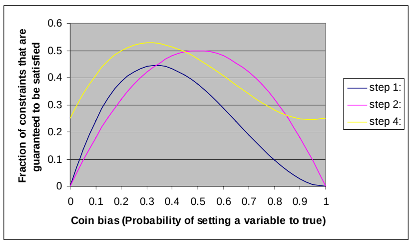
Our goal is to satisfy all of the clauses not just the maximum fraction that can be set in polynomial time. Therefore, once we find out that a partial assignment lets one or more clauses unsatisfied, we add a new clause to the formula so that we never make the same mistake again. The following example demonstrates the learning process:

Step 1:
{ }
1in3 (X1 X2 X3)
1in3 (X1 X3 X4)
1in3 (X1 X2 X4)
Decide X1 = false

Step 2:
{ !X1 }
1in2 (X2 X3)
1in2 (X3 X4)
1in2 (X2 X4)
Decide X2 = true

Step 3:
{ !X1, X2 }
not (X3)
1in3 (X3 X4)
not (X4)
Conflict reached via unit propagation:
Learn or (X1, !X2)

Step 4:
{ }
1in3 (X1 X2 X3)
1in3 (X1 X3 X4)
1in3 (X1 X2 X4)
or (X1 !X2)



Bitwise Relation Reduction:

Fast Unit Propagation for CSP: DPLL SAT solvers spend up to 90%[4] of their time doing unit propagation. Unit propagation for CSP is not so efficient. We have developed a representation scheme for relations, that allows us to do Unit Propagation for CSP in an extremely fast manner using bitwise operations.

[4] "Fast Incremental Unit Propagation by Unifying Watched-literals and Local Repair" Shen Qu. Master thesis at MIT (2006).