Programmatic SAT for SHA-256 Collision Attack

M.Sc. Thesis Defense

Nahiyan Alamgir

Chair: Dr. Asish Mukhopadhyay
Supervisor: Dr. Curtis Bright
Internal Reader: Dr. Saeed Samet
External Reader: Dr. Huapeng Wu

August 14, 2024

University of Windsor

The work has been published in the SC^2 Workshop 2024.[1]

Thanks to Dr. Saeed Nejati for collaborating as a mentor.

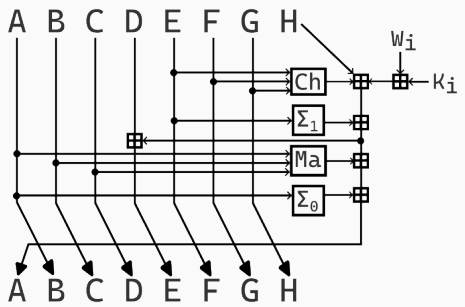
[1] Alamgir, N., Nejati, S., & Bright, C. (2024). SHA-256 Collision Attack with Programmatic SAT.

Motivation

Cryptographic hash functions are highly relied on for security and integrity, producing fixed-length hashes from variable-length inputs.

SHA-256 is a widely used hash function in the Bitcoin protocol for transaction signatures, file checksums, and other key applications.

SHA-256 has 64 steps, and each step scrambles the input using this circuit:



Collision Attacks

Let $h(x)$ be a hash function that takes an input x.

Can we find x and a different input x' that produces the same hash?

In other words, the collision problem is about solving

$$h(x) = h(x')$$

for x and x' where $x \neq x'$.

Objectives

Our objective is finding collisions of simplified SHA-256.

Collisions are found using a satisfiability (SAT) solver, which is a tool for finding solutions to logical problems.

We improved the performance of a SAT solver by augmenting it with computer algebraic techniques.

The idea is to dynamically guide the solver with information that the solver is not able to deduce on its own.

Cryptographic Hash Functions

One of the properties (in practice) is generating a distinct hash for each unique input.

An important property is the avalanche effect:

Input	Hash

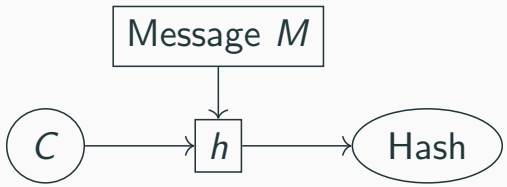
SC2	`b26804e0 7a71b4e6`
	`e31f0db3 2d554f8d`
	`16d11cbf e111e80e`
	`c92f9fd8 0348a514`

SC3	`1290a827 f49b6c4e`
	`08e3883b 7833978f`
	`45dd20df e2b290a3`
	`6fea6688 fc21decc`

The avalanche effect contributes to a hash function's collision resistance.

Semi-free-start (SFS) Collisions

The collision finding problem can be made easier by using an arbitrary variable C in place of the "initialization vector" (a constant in the SHA-256 specification).



C is an input variable.

A semi-free-start (SFS) collision consists of an input C and message pair M, M' for which

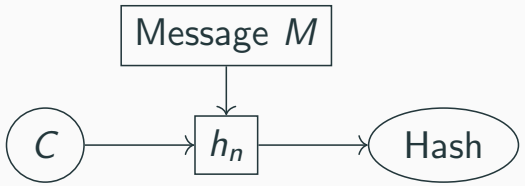
$$h(C, M) = h(C, M').$$

List of Simplifications

The number of steps in SHA-256 is reduced (from 64 steps) to $n \leq 38$ steps to make the problem easier.

Inputs with exactly 512 bits are used (instead of arbitrary size).

The initialization vector is now part of the input (instead of fixed).



h_n is SHA-256 reduced to n steps.

SAT Solving

Boolean Satisfiability (or SAT) is the problem of solving a Boolean formula.

A SAT solver is a program that searches for solutions to these formulas.

Example of a formula is $(\neg x_0 \vee x_1) \wedge (x_2 \vee x_3 \vee \neg x_4)$ where $x_0, \ldots, x_4 \in \{0, 1\}$.

Truth table for $\vee$ (logical or), $\wedge$ (logical and), and $\neg$ (logical not) operations:

x	y	$\neg y$	$x \vee y$	$x \wedge y$
T	T	F	T	T
F	T	F	T	F
T	F	T	T	F
F	F	T	F	F

CNF Formula

CNF (Conjunctive Normal Form) consists of clauses that all have to be true.

A clause is disjunction (logical or) of variables (or negated variables).

Example of a clause is $u \vee v \vee \neg w$.

CNF is a conjunction (logical and) of clauses.

Previous Best Step-Reduced SHA-256 Collisions

Publication Year	Author	Collision	SFS Collision
2006	Mendel et al.	18	–
2008	Sanadhya et al.	24	–
2011	Mendel et al.	27	32
2013	Mendel et al.	28	38
2024	Li et al.	31	39

Mendel et al.[2] held the record for 11 years using a custom-written sophisticated search tool and differential cryptanalysis. In May 2024, this was improved by Li et al.[3] (using a different SMT/SAT technique).

[2] F. Mendel, T. Nad, M. Schläffer, Improving local collisions: new attacks on reduced SHA-256, in: Advances in Cryptology-EUROCRYPT 2013

[3] Li, Y., Liu, F., Wang, G. (2024). New Records in Collision Attacks on SHA-2. In: Joye, M., Leander, G. (eds) Advances in Cryptology - EUROCRYPT 2024

Example of an SFS Collision

A SAT-only approach could only find SFS collisions up to 28 steps.
Adding one step makes the problem drastically more hard for pure SAT.

Our SAT + computer algebra solver found SFS collisions up to **38 steps**:

h_0	afea2566 1e0a73e2 da747de7 34381a7f 06f4c0d9 8897dd98 c592ba6a d2aa5e80
M	5b5058d2 901f87fb 254bcfa2 5f8d7dc1 fb1053be 0622e1f8 da8801c2 a951cfbb 5db42ffd 683b4391 f87eabbd e928b976 3675cc55 6ebe78be e3031536 c2de906f
M'	5b5058d2 901f87fb 254bcfa2 5f8d7dc1 fb1053be 0622e1f8 da8801c2 9737d17b 5db43001 683b4391 f8812bbd e928b976 3675cc55 6ebe78be e3031536 c2de906b
h_1	d0e019f7 408269d3 24296a7b 30df8e7f 95d2bff8 34e2bca6 6c50a294 ddb4254a

Note that M and M' differ only in a few spots.

This matches the best known SHA-256 SFS collision prior to 2024.

Differential Cryptanalysis

Differential cryptanalysis is a technique that we used for tracking the differences in the variable pairs.

This technique allows us to analyze the flow of differences from the input pair to the hash output pair with a fine-grained precision.

The flow of differences in a function $f(x) = y$ denoted by

$$\Delta x \xrightarrow{f} \Delta y$$

is called a differential, where the Δ denotes an XOR difference.

Differential Example

Let '-' represent a difference of 0 and '$\mathtt{x}$' represent a difference of 1.

An example of a differential is: $[\mathtt{-}, \mathtt{x}, \mathtt{x}] \xrightarrow{\text{XOR}} [\mathtt{-}]$.

If the XOR function takes inputs a, b, c and returns d, the differences can be expressed as

$$\Delta a = 0, \qquad \Delta b = 1, \qquad \Delta c = 1, \qquad \Delta d = 0.$$

Starting Points

The differentials for all the operations are linked together. The entire chain across the hash function is called a "differential path".

The differential path is constrained (from the start) to define the collision problem and also refine the search space, forming a "starting point".

For example, the starting point constrains the differentials of the hash output and certain internal state variables to be zero.

Differential Conditions

Let x be a Boolean variable in one hash function instance, and x' be its counterpart in a second instance. The possibilities of (x, x') are concisely described by "differential conditions".

Condition	Possibilities	# of Possibilities
`?`	$(0,0),(0,1),(1,0),(1,1)$	4
`-`	$(0,0),(1,1)$	2
`x`	$(0,1),(1,0)$	2
`0`	$(0,0)$	1
`n`	$(0,1)$	1
`u`	$(1,0)$	1
`1`	$(1,1)$	1

Let ∇X denote a differential vector over two words (X, X'). As an example, $\nabla X =$ `[x-1]` defines a differential with 4 possibilities:

$$(001,101),(011,111),(101,001),(111,011).$$

Programmatic SAT

Programmatic SAT involves writing custom code to dynamically guide the SAT solver.

It is usually used when adding high-level properties inside the Boolean formula increases its size significantly.

For example, in 7-bit addition, when 4 bits are known, we can derive extra information (high carry) that the SAT solver wouldn't derive right away.

We used the IPASIR-UP[4] programmatic SAT interface.

[4] K. Fazekas, A. Niemetz, M. Preiner, M. Kirchweger, S. Szeider, A. Biere, IPASIR-UP: User propagators for CDCL, in: 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023), Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023

Programmatic SAT: Bitsliced Propagation

SHA-256 performs operations on 32-bit words.

Each output bit position of an operation can be propagated independently by taking "slices".

Example of propagation in modular addition (denoted by ⊞):

	`??-`	Carries
	`[x-x-]`	Addend X
⊞	`[x---]`	Addend Y
	`[???-]`	Sum

Programmatic SAT: Bitsliced Propagation

An output condition is propagated by enumerating all possibilities conforming to the dependent input conditions.

After enumerating all possibilities in the slice, we get:

```
      ??-
     [x-x-]
⊞    [x---]
-----------
     [??x-]
```

The underlined sum differential is derived to be an '`x`'.

Programmatic SAT: Wordwise Propagation

Each step of SHA-256 involves several modular addition operations.

Bitsliced propagation can be repeated to propagate conditions in modular addition, but each propagation step is local to a bitslice.

The locality means all of the the relations between the variables aren't captured.

Programmatic SAT: Wordwise Propagation

Propagation can be performed on an entire operation in a single step instead of small slices, but this would involve enumerating a vast set of possibilities.

Our solution is a global "wordwise" propagation technique that is cheaper than propagating over an entire modular addition operation, yet often derives more information than bitsliced propagation.

The idea is to take advantage of modular differences of the words in the two hash instances.

Programmatic SAT: Wordwise Propagation

Wordwise propagation can be performed when the modular difference (δ) of one or more word pairs is known.

Suppose $\delta A \boxplus \delta B = \delta C$ for which

$$\nabla A = \texttt{[----x]},$$
$$\nabla B = \texttt{[??-?x]},$$
$$\delta C = 0.$$

Providing it to the wordwise propagator breaks it down to the problem

$$\begin{array}{rr} & [\ v_5,\ v_4,\ v_3,\ v_0,\ 0\ ] \\ & [\ v_6,\ 0,\ c_0,\ v_1,\ 0\ ] \\ \boxplus & [\ 0,\ 0,\ 0,\ v_2,\ 0\ ] \\ \hline & 1\ \ 1\ \ 1\ \ 0\ \ 0 \end{array}$$

where $v_0, \ldots, v_6$ and $c_0 \in \{0, 1\}$.

Programmatic SAT: Wordwise Propagation

We begin by performing unit constraint propagation

$$\begin{array}{rc} & [\ v_5,\ v_4,\ v_3,\ v_0,\ 0\ ] \\ & [\ v_6,\ 0,\ c_0,\ v_1,\ 0\ ] \\ \boxplus & [\ 0,\ 0,\ 0,\ v_2,\ 0\ ] \\ \hline & 1\ \ 1\ \ 1\ \ 0\ \ 0 \end{array}$$

which derives $v_4 = 1$.

Each '?' can have a difference of 0, 1, or 2. If a low variable is 1, the high variable must be 0.

∇B_3 translates into low and high variables v_4 and v_5 respectively. Since $v_4 = 1$, v_5 must be 0.

Programmatic SAT: Wordwise Propagation

After the first round of unit propagation, we have

$$\begin{array}{rc} & [\ 0,\ 1,\ v_3,\ v_0,\ 0\ ] \\ & [\ v_6,\ 0,\ c_0,\ v_1,\ 0\ ] \\ \boxplus & [\ 0,\ 0,\ 0,\ v_2,\ 0\ ] \\ \hline & 1\ \ 1\ \ 1\ \ 0\ \ 0 \end{array}$$

while the next round deduces $v_6 = 1$.

The propagated variables give us

$$\nabla A = [\texttt{----x}],$$
$$\nabla B = [\texttt{---?x}].$$

Programmatic SAT: Wordwise Propagation

$$
\begin{array}{rc}
 & [\,0,\,1,\,v_3,\,v_0,\,0\,] \\
 & [\,1,\,0,\,c_0,\,v_1,\,0\,] \\
\boxplus & [\,0,\,0,\,0,\,v_2,\,0\,] \\
\hline
 & 1\;\;1\;\;1\;\;0\;\;0
\end{array}
$$

The remaining variables may be derived by enumerating all the solutions for each column.

The constraints imposed by the carry variables are applied during the enumeration.

For example, if c_0 was 0, the only solution of (v_0, v_1, v_2) would've been $(0, 0, 0)$.

In this problem, brute forcing doesn't deduce any variable.

Programmatic SAT: Wordwise Propagation Heuristic

The effectiveness of wordwise propagation is dramatically increased through a heuristic, by assuming '?' differentials of auxiliary variables are '–' differentials.

This makes it more likely for modular differences to be known, and increases the frequency of variable deduction.

This significantly improved the solver's performance and found SFS collisions for a larger number of steps.

Implementation

The IPASIR-UP interface in CaDiCaL is used for programmatic SAT.

Bitsliced propagation is done alongside BCP with reason clauses provided when required.

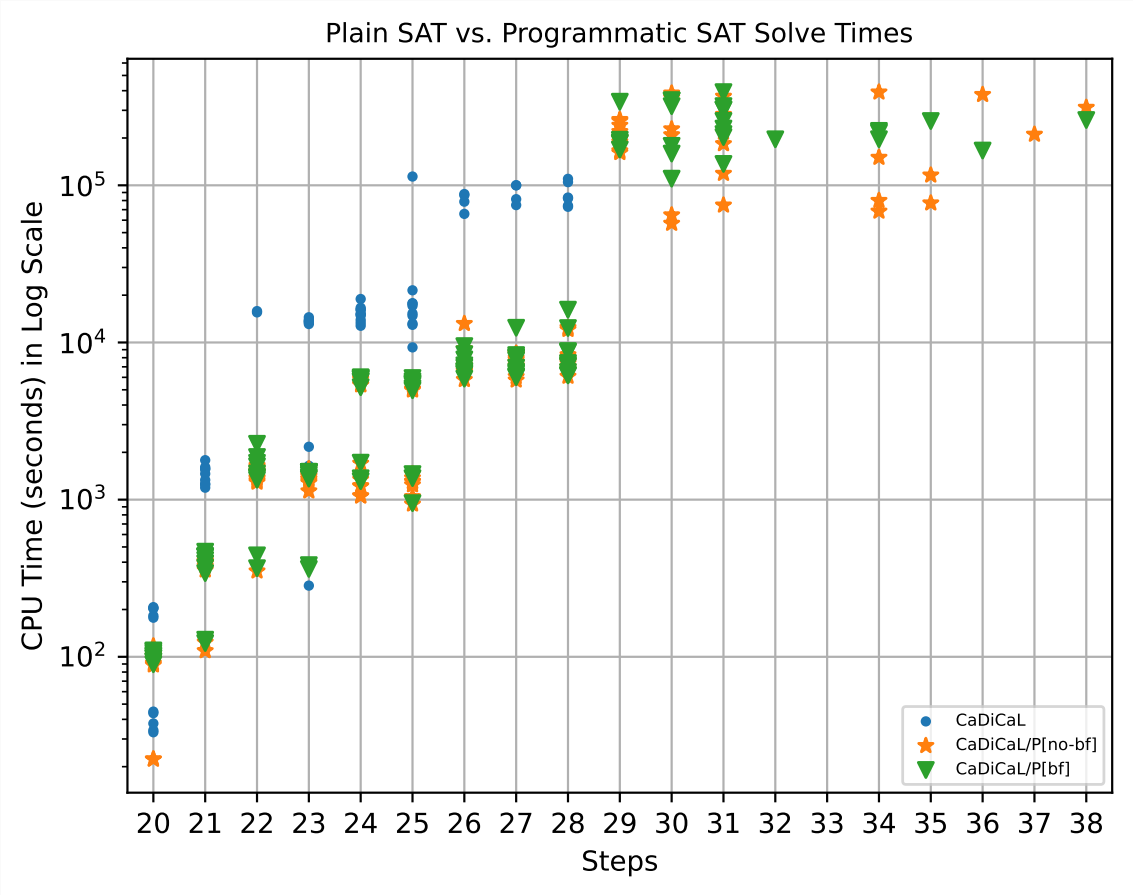
Wordwise propagation involves significantly longer reasoning clauses, so it is performed by branching on unassigned variables necessary to induce the propagated differential.

Results

We utilized the SAT solver CaDiCaL 1.8.

For each step count and solver, we ran 10 instances each with a different seed for 500,000 seconds (roughly 5.8 days).

Lack of a data point indicates no collisions were found within the time limit.



Conclusion

We demonstrate that employing computer algebraic techniques can significantly improve the performance of SAT solvers on hash collision instances, allowing us to find 38-step SFS collisions.

Just a few months ago, a 39-step attack by Li et al. was published. Our computer algebraic techniques weren't found to be useful with their encoding for reasons that are currently unclear.

Exploring or adjusting techniques to enhance the solver's performance with this new encoding is a potential future work.

Thanks to the committee and the audience!

Starting Point for 38-step SFS Collisions

i	∇A_i	∇E_i	∇W_i
−4	----------	----------	
−3	----------	----------	
−2	----------	----------	
−1	----------	----------	
0	----------	----------	----------
1	----------	----------	----------
2	----------	----------	----------
3	----------	----------	----------
4	----------	----------	----------
5	----------	----------	----------
6	----------	----------	----------
7	??????????	??????????	??????????
8	??????????	??????????	??????????
9	??????????	??????????	----------
10	----------	??????????	??????????
11	----------	??????????	----------
12	----------	??????????	----------
13	----------	??????????	----------
14	----------	??????????	----------
15	----x-----x----------	??????????	----------x--
16	----------x--	??????????	----------
17	----------	??????????	----------
18	----------	----------	----------
19	----------	??????????	----------
20	----------	??????????	----------
21	----------	----------	----------
22	----------	----------	----------
23	----------	----------	----x--------xx----------
24	----------	----------	----------x--
25	----------	----------	----------
26	----------	----------	----------
27	----------	----------	----------
28	----------	----------	----------
29	----------	----------	----------
30	----------	----------	----------
31	----------	----------	----------
32	----------	----------	----------
33	----------	----------	----------
34	----------	----------	----------
35	----------	----------	----------
36	----------	----------	----------
37	----------	----------	----------

Programmatic SAT: Inconsistency Blocking

There are cases when a differential path is inconsistent.

Specifically, relations in one section of the differential path contradict those in another section.

The relations we study are in the form $x \oplus y = z$ where $x, y, z \in \{0, 1\}$.

Programmatic SAT: Inconsistency Blocking

The variable relations can be derived from the differentials of bitwise functions and modular addition.

A trivial contradiction can be seen in the two-bit conditions $a = b$ and $a \neq b$.

For non-trivial cases, we have cycles of contradictions like $a = b = c \neq a$.

Programmatic SAT: Inconsistency Blocking

We use a graph data structure for tracking the variable relations

Each variable in the graph is a vertex and every relation is an edge.

We analyze the cycles in the graph and watch for inconsistencies.

The shortest inconsistent cycle is used for blocking the (invalid) differential path.

Programmatic SAT: Inconsistency Blocking

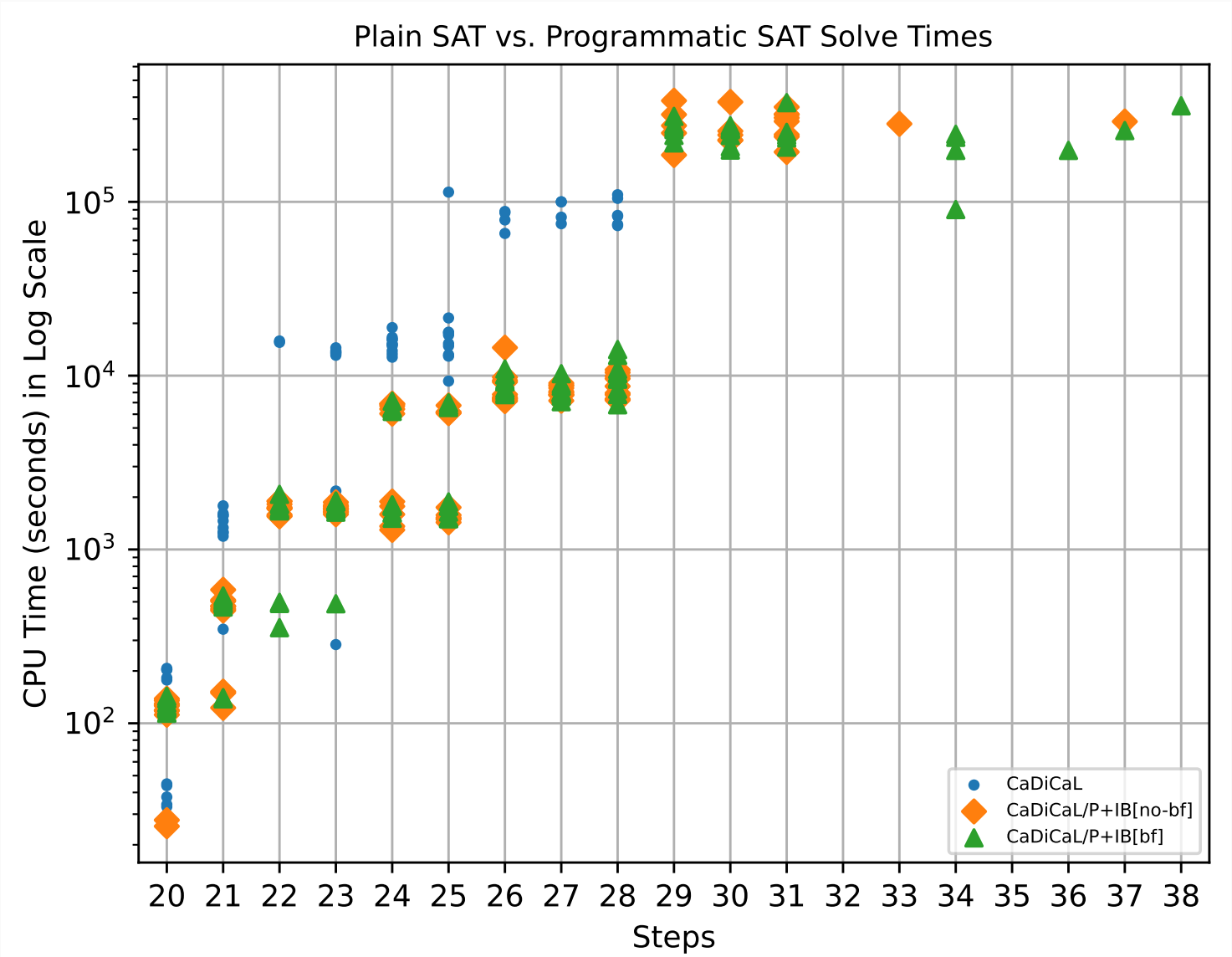
Breadth-first search (BFS) is used for finding all the cycles involving a newly added edge.

The cycles containing an odd number of inequal variables are determined to be inconsistent.

To block an inconsistent cycle, the variable assignments that led to that cycle are used to construct a conflict clause.

The conflict clause is injected to the solver, causing it to step out of the invalid differential path.

Programmatic SAT: Inconsistency Blocking



Li et al. 2024's Encoding

In Li et al.'s work, they use an SMT solver for finding characteristics for 39-step SFS collisions.

Their SMT encoding defines the relations between modular difference and word differentials.

A smaller set of differential conditions, $\{\mathtt{n}, \mathtt{u}, \mathtt{-}\}$, are used.

Programmatic SAT with Li et al. 2024's Encoding