SAT and Lattice Reduction for Integer Factorization

Yameen Ajani and Curtis Bright

University of Windsor

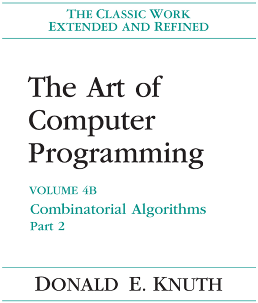
July 18, 2024
ISSAC 2024

curtisbright.com

SATisfiability

SAT: Given a Boolean logic expression, can it can be made true?

Donald Knuth's *The Art of Computer Programming Vol. 4B* (2022) is over 700 pages and half of it is devoted to SAT.

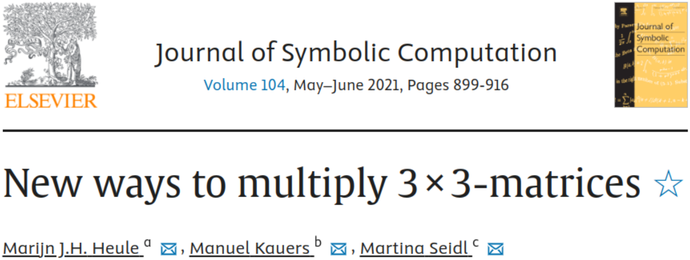


"SAT solvers" can be surprisingly effective and can solve many search problems seemingly unrelated to Boolean logic, like Sudoku.[1]

[1]Bright, Gerhard, Kotsireas, Ganesh. Effective Problem Solving Using SAT Solvers. *Maple Conference*, 2019.

A SAT Success Story

How fast can you multiply 3×3 matrices? Before 2021, four algorithms were known using 23 scalar multiplications. Then...


ELSEVIER

Journal of Symbolic Computation

Volume 104, May–June 2021, Pages 899-916

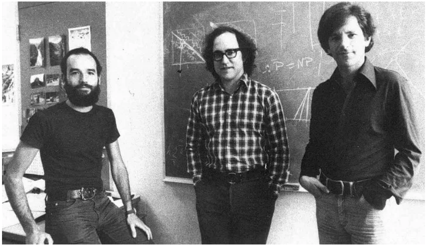
New ways to multiply 3×3-matrices ☆

Marijn J.H. Heule [a], Manuel Kauers [b], Martina Seidl [c]

Using a SAT solver, over 17,000 distinct 3×3 matrix multiplication algorithms were found using 23 scalar multiplications.

Rivest–Shamir–Adleman Cryptosystem

The popular cryptosystem RSA relies on the difficulty of factoring large integers into primes.



RSA encryption involves a semiprime $N = p \cdot q$ for two randomly chosen primes p and q of the same bitlength (known only to the recipient).

The best known attack on RSA involves factoring N. No efficient integer factoring algorithms are known (unless you allow quantum computation).

Reduction of Factoring to SAT

Multiplication circuits can be converted to SAT by operating directly on the bit-representation of the integers.

Say $[N_3 N_2 N_1 N_0]_2$ is the binary representation of N. Use variables p_1, p_0 and q_1, q_0 to denote the bits of the prime factors of N:

$$
\begin{array}{r}
q_1 \; q_0 \\
\times \; \underline{p_1 \; p_0} \\
a_1 \; a_0 \\
\underline{b_1 \; b_0 \quad\;} \\
\underline{c_1 \; c_0 \qquad\quad} \\
N_3 N_2 N_1 N_0
\end{array}
\qquad
\begin{aligned}
[a_1 a_0]_2 &= [q_1 q_0]_2 \times p_0 \\
[b_1 b_0]_2 &= [q_1 q_0]_2 \times p_1
\end{aligned}
\qquad
\begin{aligned}
N_0 &= a_0 \\
[c_0 N_1]_2 &= a_1 + b_0 \\
[c_1 N_2]_2 &= b_1 + c_0 \\
N_3 &= c_1
\end{aligned}
$$

These equations can be broken into logical expressions, e.g., $a_0 \leftrightarrow (q_0 \wedge p_0)$, $N_1 \leftrightarrow (a_1 \oplus b_0)$, and $c_0 \leftrightarrow (a_1 \wedge b_0)$, etc.

SAT vs. Algebraic Methods

> *It's somewhat mind-boggling to realize that numbers can be factored without using any number theory! No greatest common divisors, no applications of Fermat's theorems, etc., are anywhere in sight. [...] Of course we can't expect this method to compete with the sophisticated factorization algorithms...*
>
> Donald Knuth, TAOCP 4B

As might be expected, computer algebra methods *dramatically* outperform SAT. The *number field sieve* can factor an n-bit number heuristically in time $\exp(O^{\sim}(n^{1/3}))$ (super-polynomial, but sub-exponential in n).

Side-channel Attacks

Cryptographic implementations have an Achilles heel—they are implemented in the real world, *not* a platonic universe.

Side-channel attacks exploit the fact that cryptographic implementations may leak information about the private key in practice.

Motivating Example

Suppose you are using disk encryption with RSA. In order to read from the disk, your private key, including the prime factors of N, is kept in memory.

What if an attacker steals your screen-locked machine? Is there any way they can extract your private key?

Motivating Example

Suppose you are using disk encryption with RSA. In order to read from the disk, your private key, including the prime factors of N, is kept in memory.

What if an attacker steals your screen-locked machine? Is there any way they can extract your private key?

Experiments have shown that after an hour without power, 99.87% of bits in DRAM modules remain readable—assuming the DRAM was kept in liquid nitrogen.[2]

[2] Halderman et al. Lest We Remember: Cold-Boot Attacks on Encryption Keys. *Communications of the ACM*, 2009.

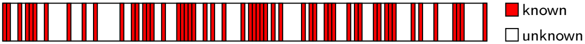
Motivating Example II

When power is removed, bits in DRAM modules decay to a predictable ground state (say 0).

Any bits that are 1 after the power is removed **must originally have been 1**, while 0 bits may have been 0 or 1.

The result is that the attacker learns bits of the private key **at bit positions they don't control** (in practice, at essentially random positions).

bits of p:



Exploiting Leaked Bits

Algebraic methods like the number field sieve cannot seem to exploit leaked bits.

With SAT, it is easy assign any leaked bits of the prime factors to their correct value. This speeds up the solver—but SAT solvers are slow for this problem, as they don't exploit algebraic properties.

Question we address: *Can we use algebraic methods to improve SAT solvers on random leaked-bit factorization problems?*

Coppersmith's Method

Don Coppersmith showed that if the lowest or highest 50% of the bits of a prime factor of N are leaked...

bits of p:	unknown	known	or	known	unknown

then N can be factored in polynomial time via lattice reduction.[3]

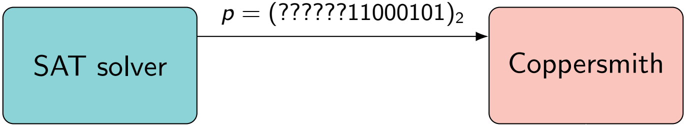
However, Coppersmith's method is not effective if the leaked bits are randomly distributed.

[3] Coppersmith. Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known. *EUROCRYPT*, 1996.

SAT + Computer Algebra System (CAS)

As SAT solvers search for solutions, they find "partial" solutions (where some variables will be unassigned).

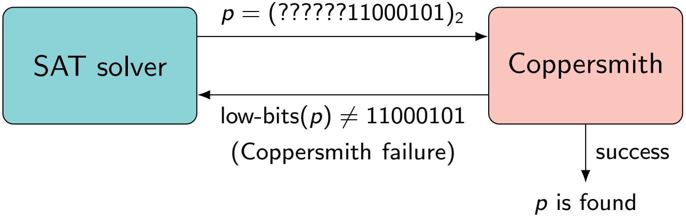
Say that a partial solution has assigned values to all of the bottom-half of the bits of p:



SAT + Computer Algebra System (CAS)

As SAT solvers search for solutions, they find "partial" solutions (where some variables will be unassigned).

Say that a partial solution has assigned values to all of the bottom-half of the bits of p:



If Coppersmith's method succeeds, N is factored. If not, tell the solver that at least one of the low bits of p must change.

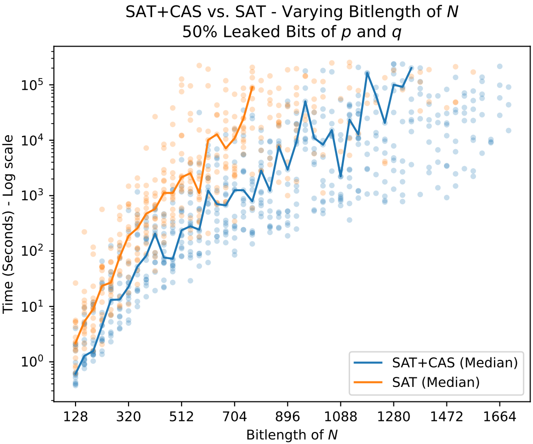
Experimental Setup

For varying bitlengths and percentages of leaked bits, we compared the SAT solver MapleSAT with a version of MapleSAT calling Coppersmith's method on 15 randomly generated instances.

Coppersmith's method (implemented with fplll)[4] is used when at least 60% of the low bits of p are known, as this allows using a lattice of fixed dimension 5 (regardless of the size of N).

[4] fplll, a lattice reduction library, `https://github.com/fplll/fplll`

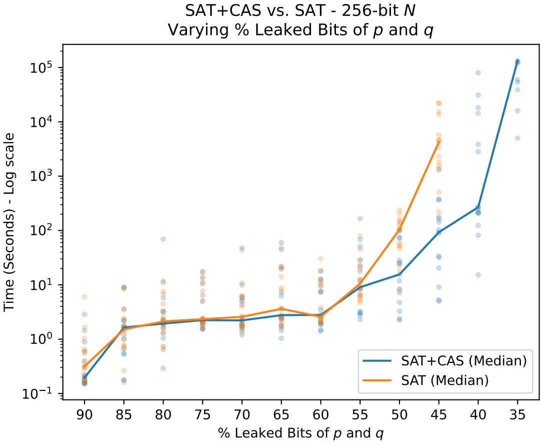
Results



Each instance was run for 3 days. For comparison, the number field sieve on 512-bit N uses around 2770 CPU hours.[5]

[5] Valenta et al. Factoring as a Service. *Financial Cryptography and Data Security*, 2016.

Results II



Each instance was run for 3 days and used at most 0.5 GiB of RAM. For comparison, an algebraic "branch and prune" technique with 40% leaked bits used around 2000 seconds and 90 GiB.[6]

---

[6] Heninger and Shacham. Reconstructing RSA Private Keys from Random Key Bits. *CRYPTO* 2009.

Final Thoughts

I've been working on combining SAT with computer algebra for almost 10 years. I regularly see SAT+CAS solvers providing exponential speedups over pure SAT or pure CAS approaches.

The approach works for problems requiring *both* search and advanced mathematics...

review articles

The science of less-than-brute force.

BY CURTIS BRIGHT, ILIAS KOTSIREAS, AND VIJAY GANESH

When Satisfiability Solving Meets Symbolic Computation

MATHEMATICIANS HAVE LONG been fascinated by objects that exhibit exceptionally nice combinatorial properties. However, it is often difficult to determine whether objects satisfying a given combinatorial property exist. Sometimes, the only feasible method of definitively answering the question of existence is simply to perform a systematic search. A famous example of this is the proof of the *four-color theorem*—the notion that four colors suffice to color the regions of a planar map with adjacent regions colored differently.[2] The theorem has been known to be true since 1977, but every known proof relies on computer calculations in an essential way. Mathematical arguments are used to reduce the search for counterexamples to a finite number of cases, and the cases are then

Communications of the ACM, 2022