Mathematical Problems with SATisfying Solutions

Curtis Bright
University of Windsor

One World Combinatorics on Words Seminar

June 3, 2025

SATisfiability

Formulae in Boolean logic consist of expressions formed with true/false variables connected with logical operators such as

$$\wedge \text{ (and)}, \vee \text{ (or)}, \neg \text{ (not)}, \rightarrow \text{ (implies)}$$

For example:

$$(x \vee y) \wedge (x \rightarrow \neg z)$$

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

SATisfiability

Formulae in Boolean logic consist of expressions formed with true/false variables connected with logical operators such as

$$\wedge \text{ (and)}, \vee \text{ (or)}, \neg \text{ (not)}, \rightarrow \text{ (implies)}$$

For example:

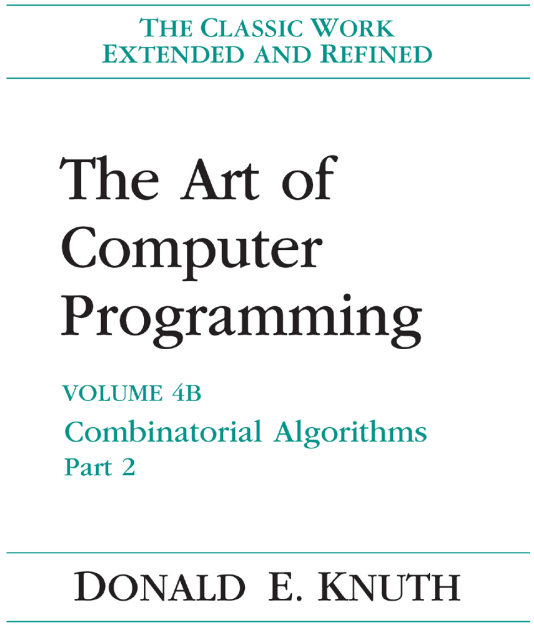
$$(x \vee y) \wedge (x \rightarrow \neg z)$$

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

The above example is satisfiable (take $x = y = \text{true}$, $z = \text{false}$).

SAT Solving

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

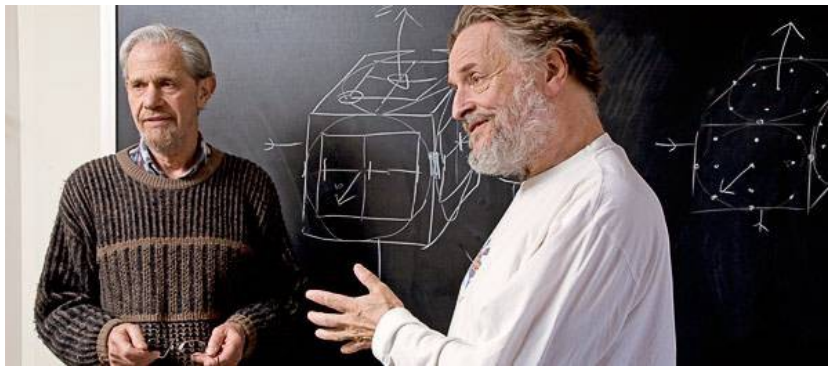


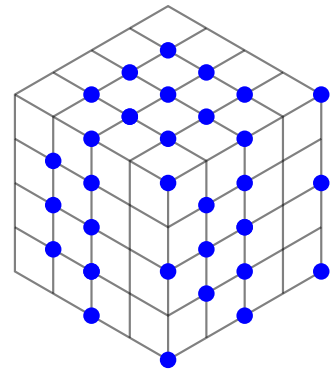
Despite having no provably fast algorithms, "SAT solvers" can be surprisingly effective and can be used solve a variety of problems seemingly unrelated to Boolean logic, like Sudoku or graph colouring.[1]

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

A SAT Success Story 1/5: Kochen–Specker Systems

Conway and Kochen proved the *Free Will Theorem*, which says if free will exists then quantum particles have free will. Their proof uses a set of 31 vectors called a Kochen–Specker (KS) system.





In 2021, I started mentoring Brian Li (now a PhD student at Georgia Tech). He used a SAT solver to show a KS system must have ≥ 23 vectors and the work has been extended to ≥ 24 vectors.[2,3]

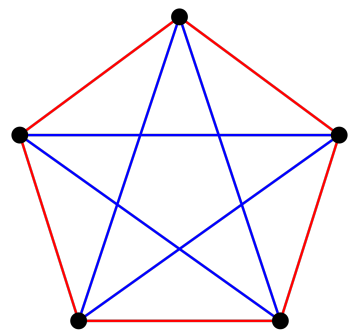
[2] Kirchweger, Peitl, Szeider. Co-Certifcate Learning with SAT Modulo Symmetries. *IJCAI 2023*.

[3] Li, Bright, Ganesh. A SAT Solver + Computer Algebra Attack on the Minimum Kochen–Specker Problem. *IJCAI 2024*.

A SAT Success Story 2/5: Ramsey Numbers

F. Ramsey proved that every red/blue edge colouring of a sufficiently large complete graph K_n must contain either a red clique of size s or a blue clique of size t (regardless of s and t). The smallest n for which this is true is called the *Ramsey number* $R(s, t)$.

The K_5 on the right has no blue triangles or red triangles, showing that $R(3, 3) > 5$. Every colouring of K_6 has a red or blue triangle, so $R(3, 3) = 6$.

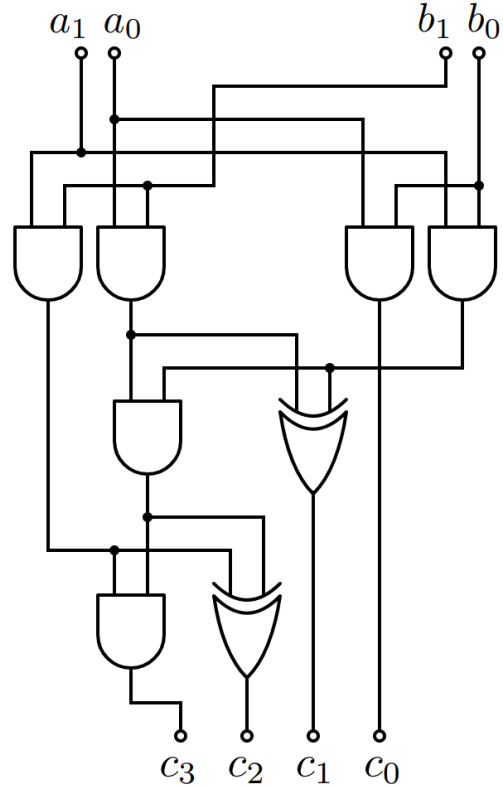


With Conor Duggan and Brian Li (Waterloo Master's students co-supervised with Vijay Ganesh), we used a SAT solver and about 211 days of compute time to generate certificates proving $R(3, 8) = 28$ and $R(3, 9) = 36$.[4]

[4] Li, Duggan, Bright, Ganesh. Verified Certificates via SAT and Computer Algebra Systems for the Ramsey $R(3, 8)$ and $R(3, 9)$ Problems. *IJCAI 2025*.

A SAT Success Story 3/5: Integer Factorization

The integer factorization problem is to write an integer as a product of primes. This problem can be reduced to a SAT by converting a binary multiplication circuit into Boolean logic, then enforcing the output bits to be the binary representation of the number to factor.

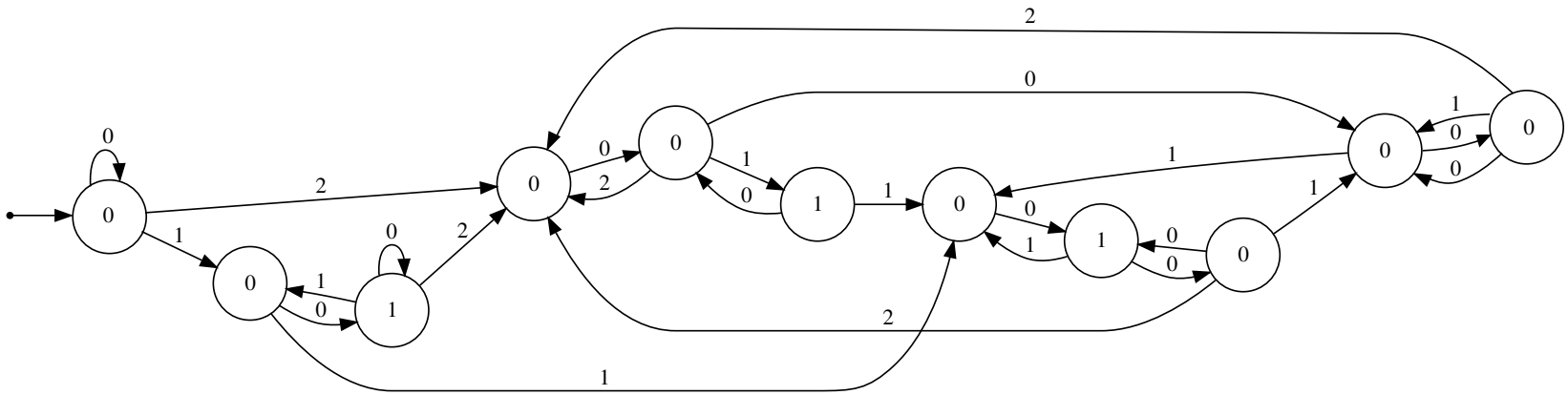


Unfortunately, SAT solvers perform poorly on factorization instances. However, if leaked bits of the prime factors are known SAT solvers can outperform algebraic methods, especially when augmented with algebraic techniques like lattice reduction.[5]

[5] Ajani, Bright. SAT and Lattice Reduction for Integer Factorization. *ISSAC 2024*.

A SAT Success Story 4/5: Minimizing Finite Automata

The following DFAO seemingly computes the nth binary digit of $(\sqrt{3}-1)/2$ when given input 2^n represented in the "Ostrowski $\frac{\sqrt{3}-1}{2}$-representation".[6]



The software Walnut finds a 12-state DFAO that provably works for all n. A SAT solver quickly finds the above 11-state automaton and shows it is minimal, assuming it works for all n (the SAT solver only ensures that the automaton works for n up to a fixed bound).

[6] Barnoff, Bright, Shallit. Using Finite Automata to Compute the Base-b Representation of the Golden Ratio and Other Quadratic Irrationals. *CIAA 2024*.

A SAT Success Story 5/5: Rado Numbers

For a linear equation $\mathcal{E}$, the 3-colour *Rado number* $R_3(\mathcal{E})$ is the smallest integer n, if it exists, such that every 3-colouring of the integers $[1 \,..\, n]$ contains a monochromatic solution to $\mathcal{E}$.

For $\mathcal{E}: x + 3y = 3z$, we have $R_3(\mathcal{E}) = 27$, since

001001002001001002001001 00

provides a 3-colouring of $[1 \,..\, 26]$, but there is no way to colour $[1 \,..\, 27]$ without introducing a monochromatic solution to $\mathcal{E}$.

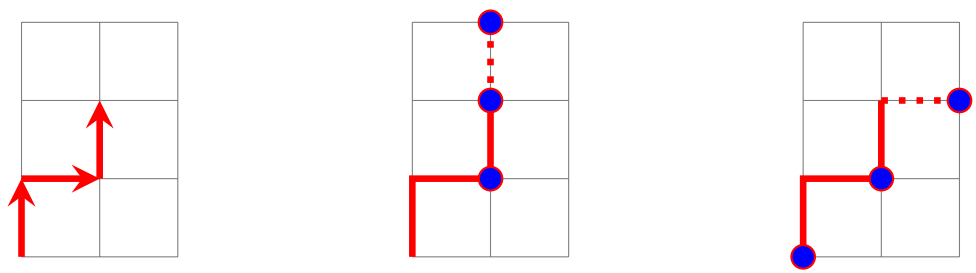
For $\mathcal{E}: 15x + 15y = 8z$, a SAT solver finds $R_3(\mathcal{E}) = 97875$ in about 2.5 days.[7]

[7] Ahmed, Zaman, Bright. Symbolic Sets for Proving Bounds on Rado Numbers. 2025.

Avoiding Collinear Points in North-East Lattice Paths

In 1971, Tom C. Brown of Simon Fraser University proposed and solved the following problem on North-East (NE) lattice paths.[8]

Show a lattice path with steps in $\{(0,1),(1,0)\}$ must pass through k collinear points for every fixed $k \geq 1$.



The path on the left must extend to a path with 3 collinear points.

[8] Brown. Collinear Points on a Monotonic Polygon. *The American Mathematical Monthly*, 1972.

Gerver–Ramsey Theorem

In 1979, Joseph L. Gerver and L. Thomas Ramsey proved an effective version of Brown's result.[9]

Theorem

A North-East lattice path of length at least

$$2^{2^{13}(k-1)^4+\log_2(k-1)}$$

must pass through k collinear points.

[9] Gerver, Ramsey. On Certain Sequences of Lattice Points. *Pacific Journal of Mathematics*, 1979.

Looseness of the Gerver–Ramsey Bound

The effective bound by provided Gerver–Ramsey is superexponential in k and extremely loose.

For example, their bound implies that any NE-path with at least $2^{2^{13}} \approx 10^{2466}$ steps must contain at least $k = 2$ collinear points.

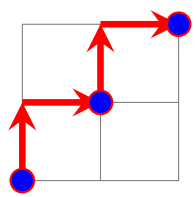
However, any NE-path with at least **one** step must have at least two collinear points!

Avoiding 3 Collinear Points

If $k = 3$ then you cannot go in the same direction twice in a row since that would produce 3 collinear points.

Thus, the longest path avoiding 3 collinear points alternates *north* and *east* steps.

After 4 steps of this, three collinear points on the line $y = x$ are produced:



How Far Can You Avoid k Collinear Points?

A231255 a(n) is the smallest integer t such that every length-t walk from the origin (0,0) taking steps of either (0,1) or (1,0) is guaranteed to have n points that are collinear.

0, 1, 4, 9, 29, 97 (list; graph; refs; listen; history; text; internal format)

OFFSET 1,3

COMMENTS By length-t we mean the number of steps, one less than the number of points.
It is known that a(7) >= 261.

LINKS Table of n, a(n) for n=1..6.
J. L. Gerver and L. T. Ramsey, On certain sequences of lattice points, Pacific J. Math. 83 (1979), 357-363.

EXAMPLE a(3) = 4 because two consecutive identical steps from (0,0) generate 3 collinear points, so the first three steps must be (0,1), (1,0), (0,1) or its complement. Then no matter what is chosen for the next step, three collinear points are generated.

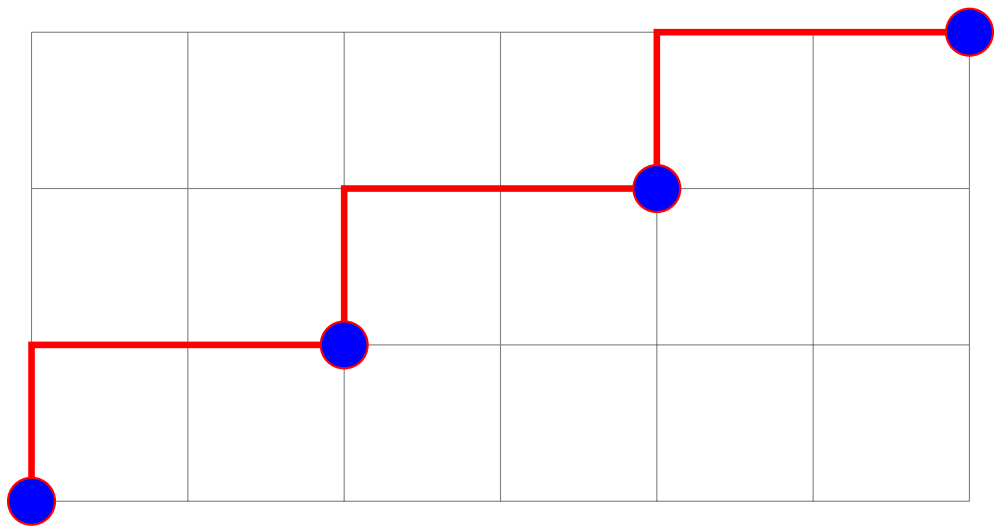
CROSSREFS Sequence in context: A210969 A059345 A127768 * A241393 A186650 A091658
Adjacent sequences: A231252 A231253 A231254 * A231256 A231257 A231258

KEYWORD nonn,more

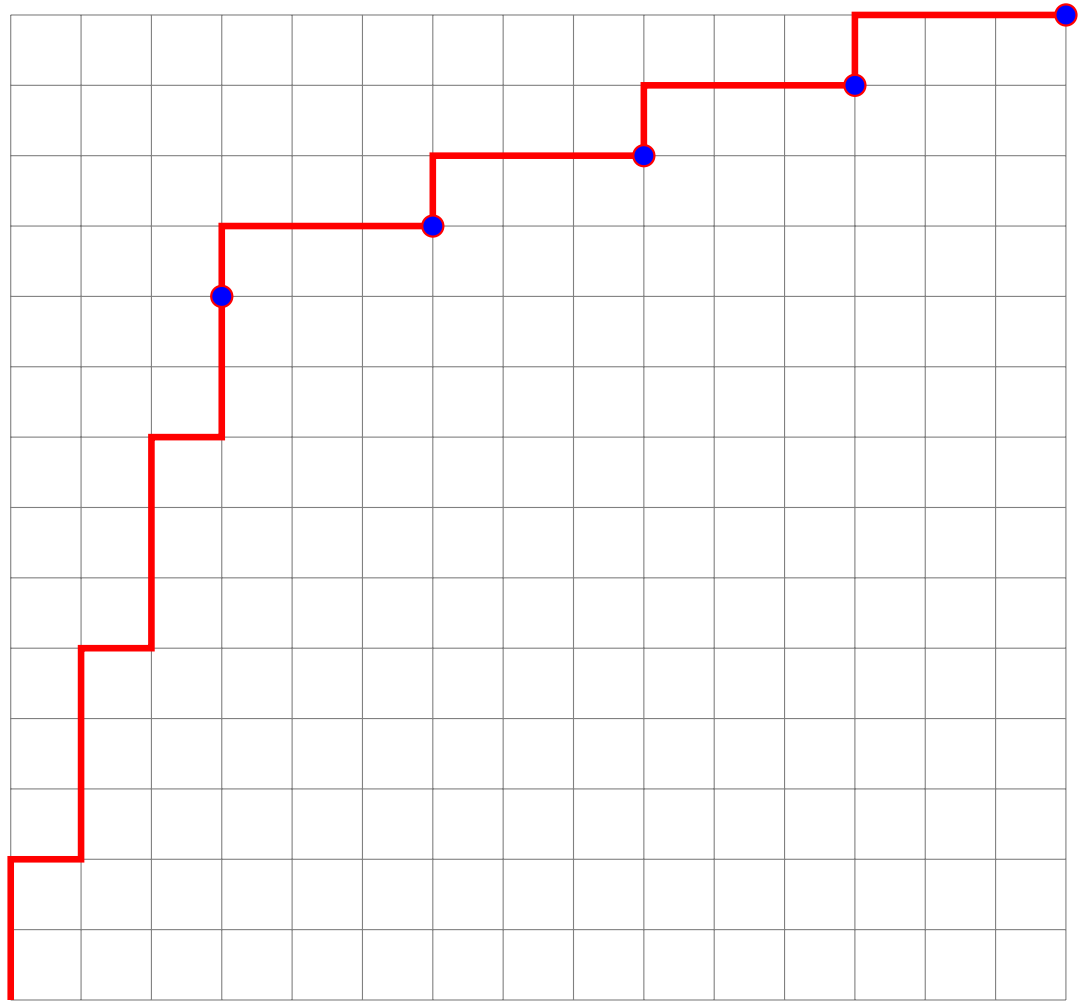
AUTHOR Jeffrey Shallit, Nov 06 2013

STATUS approved

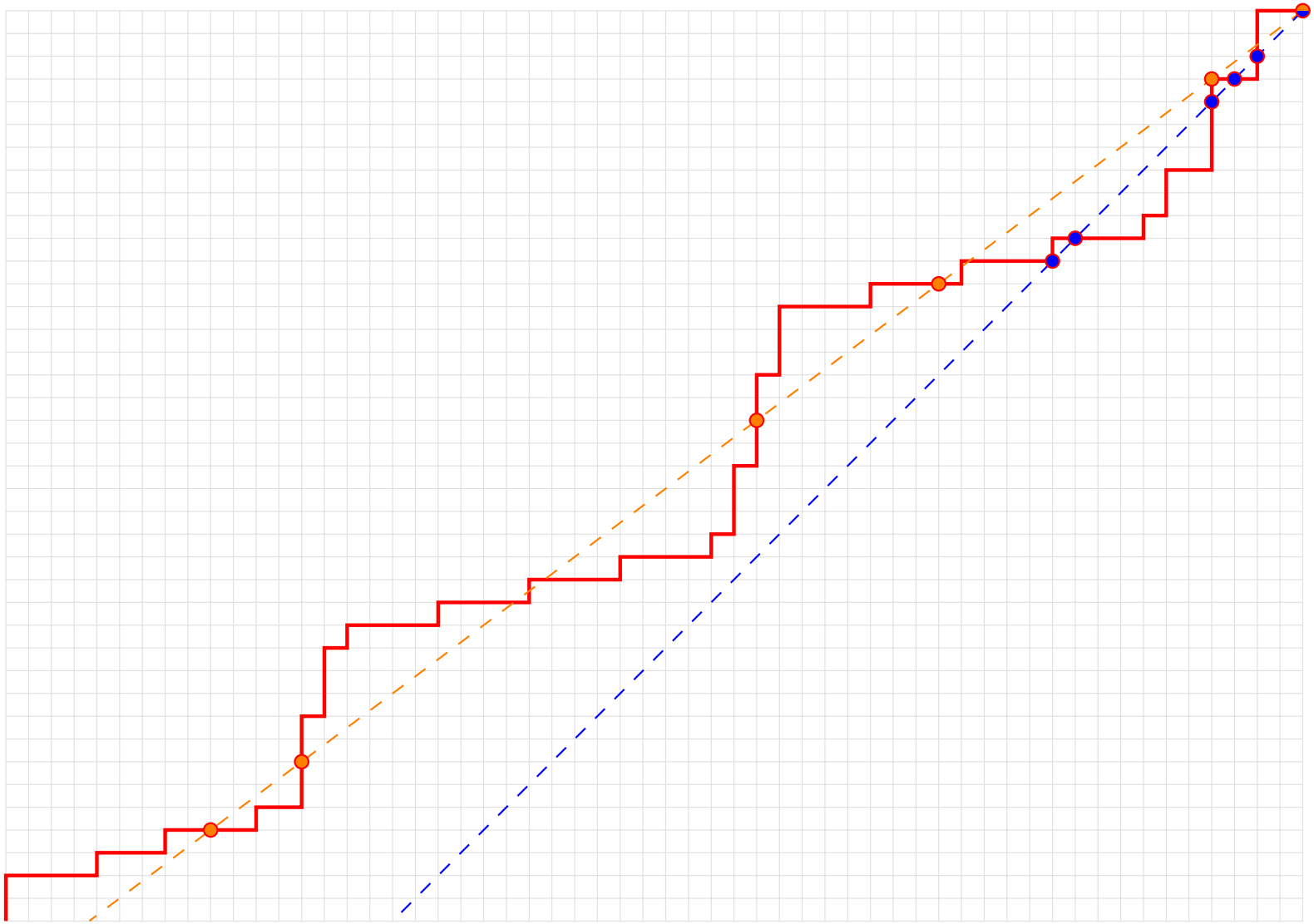
The longest path avoiding $k = 4$ collinear points (until the final step) has 9 steps. Here is one example:



The longest path avoiding $k = 5$ collinear points (until the final step) has 29 steps. Here is one example:



The longest path avoiding $k = 6$ collinear points (until the final step) has 97 steps. Here is one example:



Summary

Let A_k denote the smallest integer n such that every NE-path with n steps is guaranteed to have k collinear points.

The following were computed by Jeff Shallit using APL:

k	2	3	4	5	6	7
A_k	1	4	9	29	97	≥ 261

Can we find A_7 or improve its lower bound using automated reasoning tools like SAT solvers?

SAT Encoding

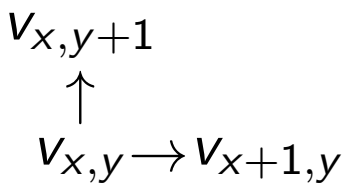
Fix k (the number of collinear points to avoid) and n (the number of steps in the path). We will construct a SAT instance that is satisfiable exactly when there exists a length-n NE-path avoiding k collinear points.

Let $v_{x,y}$ be a Boolean variable corresponding to the point (x, y). The variable will be *true* when the path we are constructing includes (x, y) and *false* otherwise.

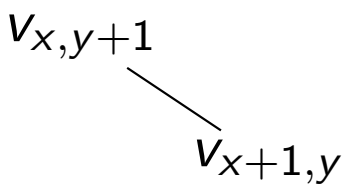
$$
\begin{array}{ccccc}
v_{4,0} & & & & \\
v_{3,0} & v_{3,1} & & & \\
v_{2,0} & v_{2,1} & v_{2,2} & & \\
v_{1,0} & v_{1,1} & v_{1,2} & v_{1,3} & \\
v_{0,0} & v_{0,1} & v_{0,2} & v_{0,3} & v_{0,4}
\end{array}
$$

One Small Step

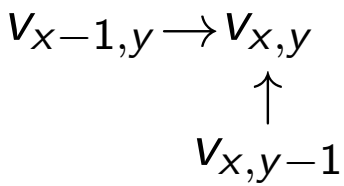
We want to encode that if (x, y) is a point in the middle of the path then you must move up or right (but not both).



$$v_{x,y} \to (v_{x+1,y} \vee v_{x,y+1})$$



$$\neg(v_{x+1,y} \wedge v_{x,y+1})$$



$$v_{x,y} \to (v_{x-1,y} \vee v_{x,y-1})$$

Step Constraints

The step constraints can be expressed as the following *clauses* (disjunction of variables or negated variables).

$$\neg v_{x,y} \vee v_{x+1,y} \vee v_{x,y+1}$$
$$\neg v_{x+1,y} \vee \neg v_{x,y+1}$$
$$\neg v_{x,y} \vee v_{x-1,y} \vee v_{x,y-1}$$

We use these for all points $(x, y) \in \mathbb{N}^2$ in the acceptable range (e.g., $x + y < n$). The variable $v_{0,0}$ is set true to start the path.

Preventing k Vertical Points

Preventing k vertical points is easy; if (x, y) is on the path then $(x, y + k - 1)$ cannot be on the path.

Thus, k vertical points are prevented by

$$v_{x,y} \rightarrow \neg v_{x,y+k-1}$$

over all valid points (x, y) in the instance.

Preventing k Collinear Points

Consider the $v_{x,y}$ as $\{0,1\}$-variables. We would like to encode

$$\sum_{x=0}^{n-1} v_{x,mx+b} < k \qquad \text{where } mx+b \in \mathbb{N}$$

and m and b are constants defining a line.

Problem: Modern SAT solvers typically require their input to be given as *clauses*, but a constraint like $\sum_{i=1}^{n} x_i < k$ is not a clause.

Encoding Cardinality Constraints

The cardinality constraint $\sum_{i=1}^{n} x_i < k$ can be efficiently encoded into clauses by introducing new variables.[10]

Let $s_{k,n}$ denote that at least k of $L := [x_1, \ldots, x_n]$ are true. Note $s_{k,n}$ can be defined in terms of $L' := [x_1, \ldots, x_{n-1}]$:

- $s_{0,j}$ is true for $j \geq 0$;
- $s_{i,0}$ is false for $i > 0$;
- if at least k of L' are true then $s_{k,n}$ is true;
- if x_n is true and at least $k-1$ of L' are true, then $s_{k,n}$ is true.

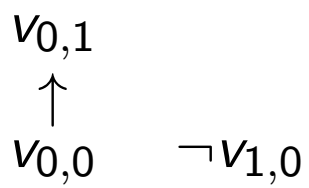
These can be encoded via $\bigwedge_{j=0}^{n} s_{0,j}$, $\bigwedge_{i=1}^{k} \neg s_{i,0}$, $s_{k,n-1} \rightarrow s_{k,n}$ and $(x_n \wedge s_{k-1,n-1}) \rightarrow s_{k,n}$. Finally, $\sum_{i=1}^{n} x_i < k$ is enforced by $\neg s_{k,n}$.

[10] Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. *CP 2005*.

Symmetry Removal

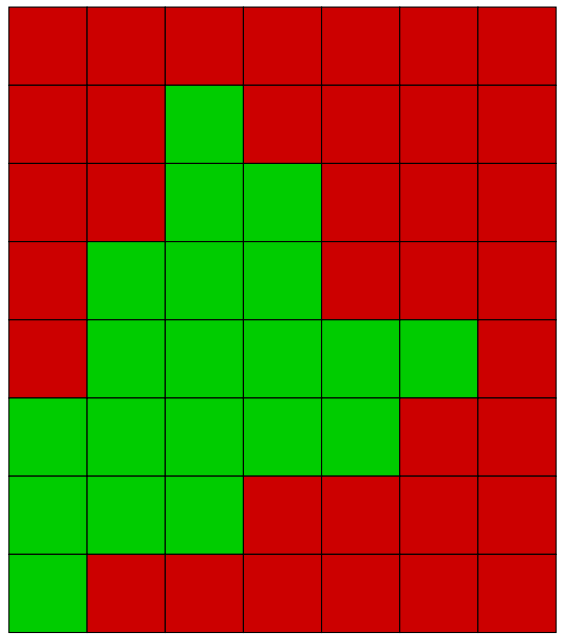
Without loss of generality we can suppose the first step is upwards.

This can simply be encoded by setting $v_{0,1}$ true (which implies $v_{1,0}$ is false).



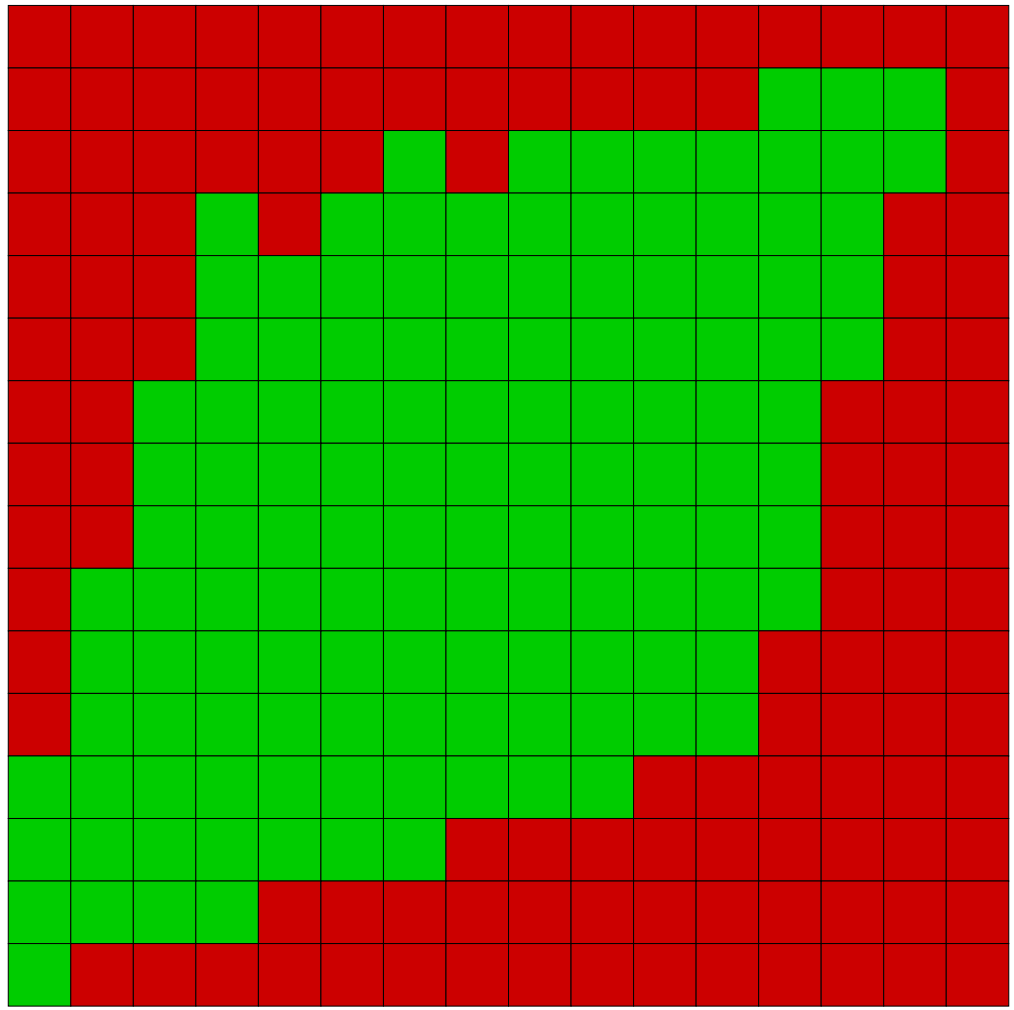
Reachability of Points for $k = 4$

For fixed k, a SAT instance was generated using this encoding by setting $v_{x,y}$ true for every possible n-step finishing point (x, y) for $n = 1, 2, 3, \ldots$ in sequence.



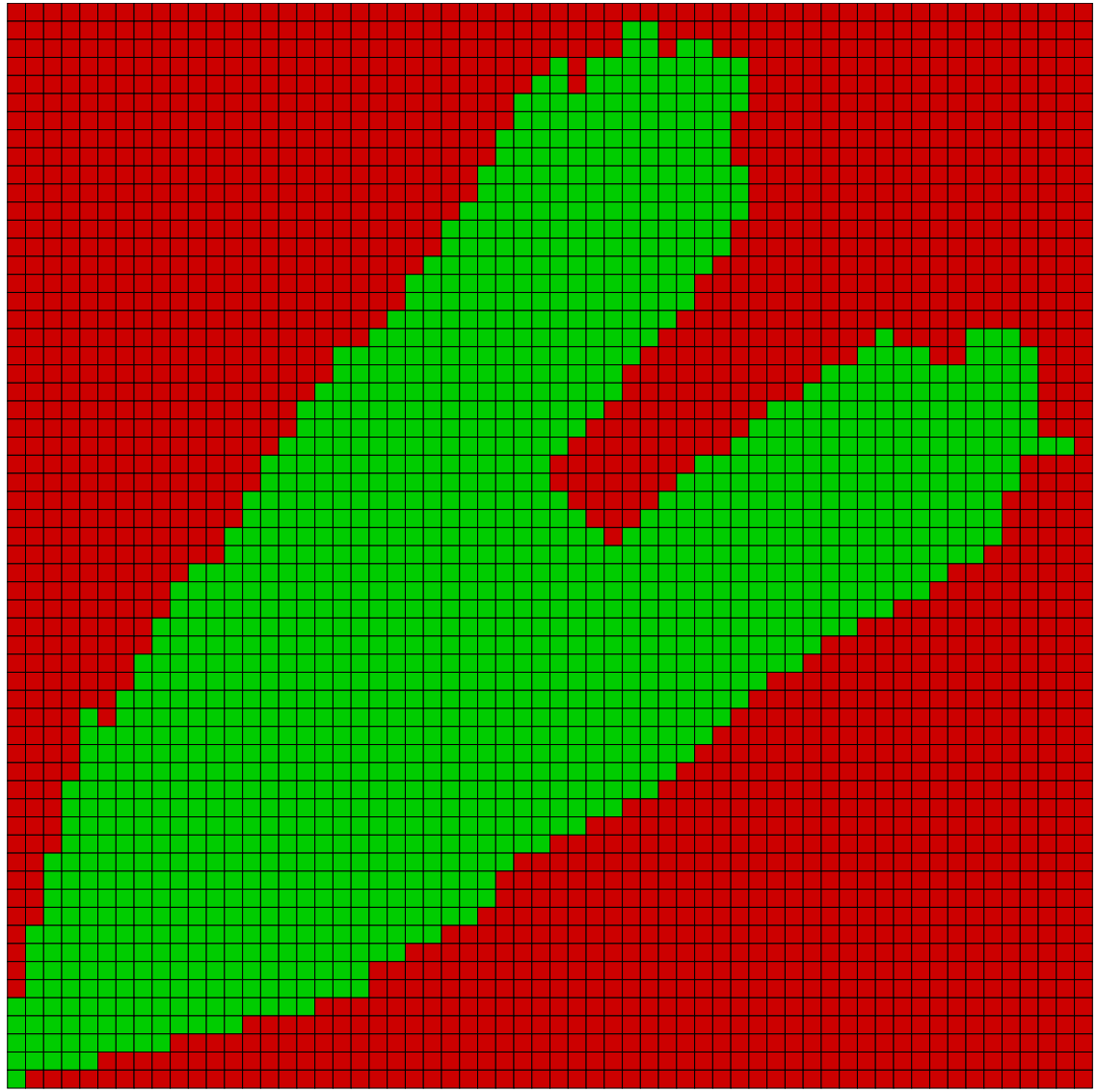
The SAT solver is able to determine the reachability of every point here instantly.

Reachability of Points for $k = 5$



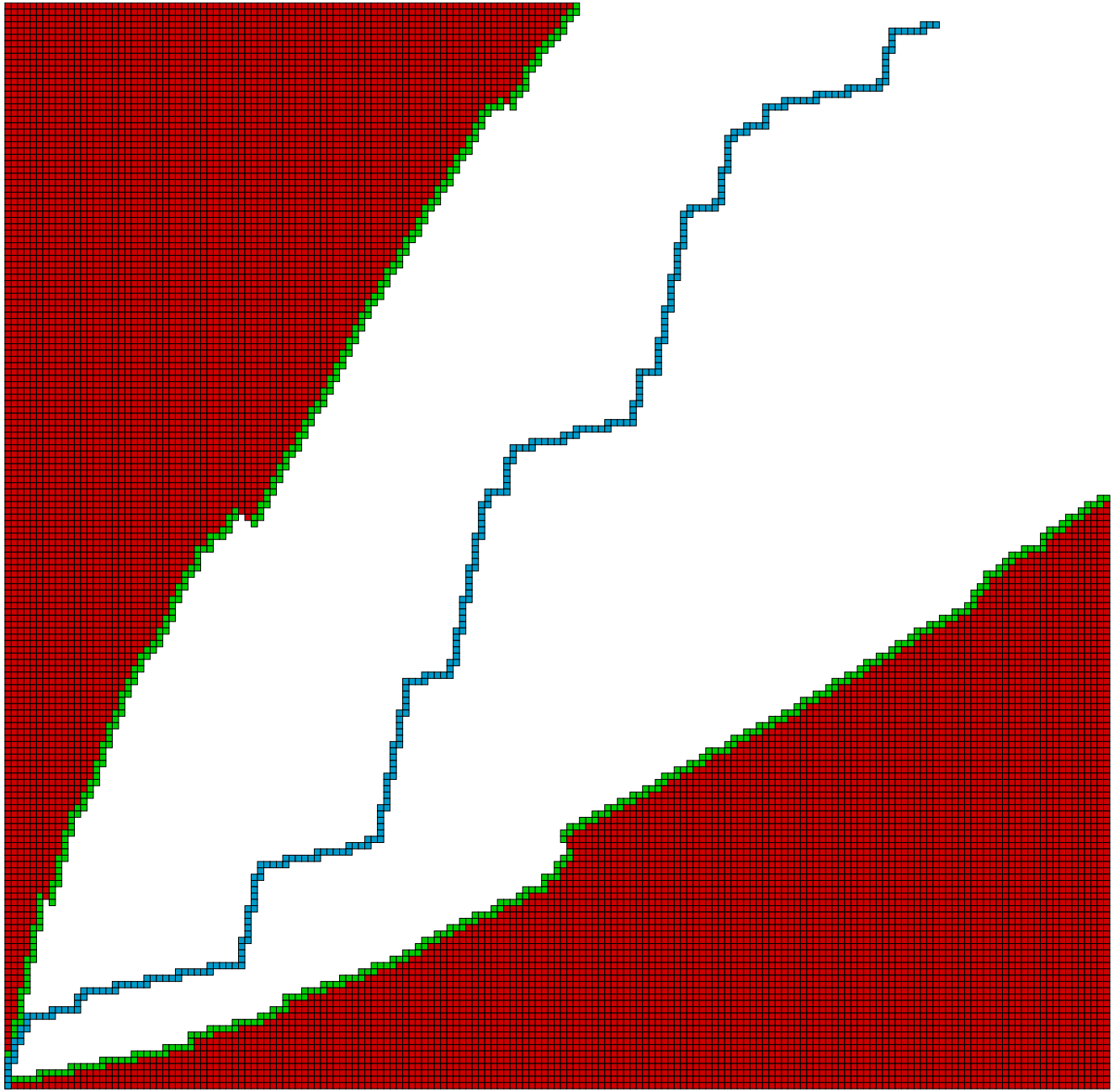
The SAT solver is able to determine the reachability of every point here in under a second in total.

Reachability of Points for $k = 6$



The SAT solver is able to determine the reachability of every point here in about 1.5 total hours.

Results for $k = 7$



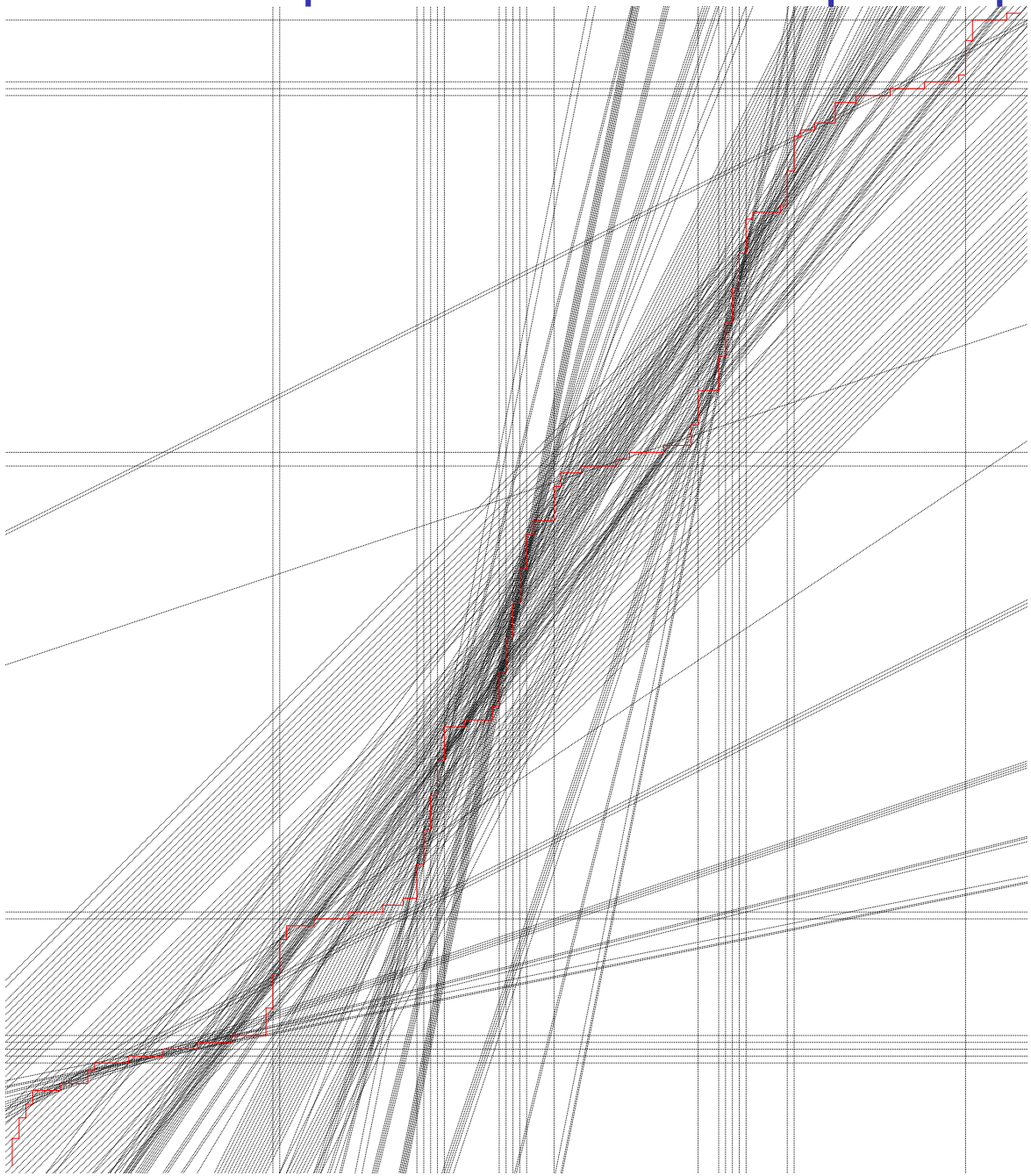
Result for $k = 7$

The lower and upper reachability bounds from the previous figure were computed by the SAT solver using about 195 days of compute time.

The instance asserting the existence of a 310-point NE-path avoiding 7 collinear points was partitioned into 131 subinstances by iteratively fixing a "splitting variable" to true and then false.

Each of the 131 instances were run for 2 days. The solver found a single 310-point path (one could be extended by 6 steps) and the other 130 instances timed out.

Lines with 6 collinear points on the 316-point path



Conclusion

SAT solvers are useful in searching for many interesting things in combinatorics—such as paths avoiding collinear points.

A SAT solver improved the lower bound on A_7 from 261 to 317.

k	2	3	4	5	6	7
A_k	1	4	9	29	97	$\geq$ **317**

A Master's student I am supervising (Aaron Barnoff) is currently working on improving this bound.