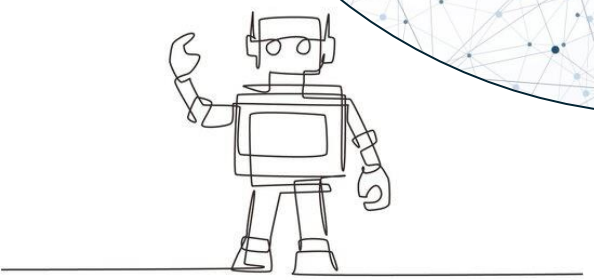
Computer-Assisted Intuition: SAT Solvers in Mathematical Discovery

Bernardo Subercaseaux

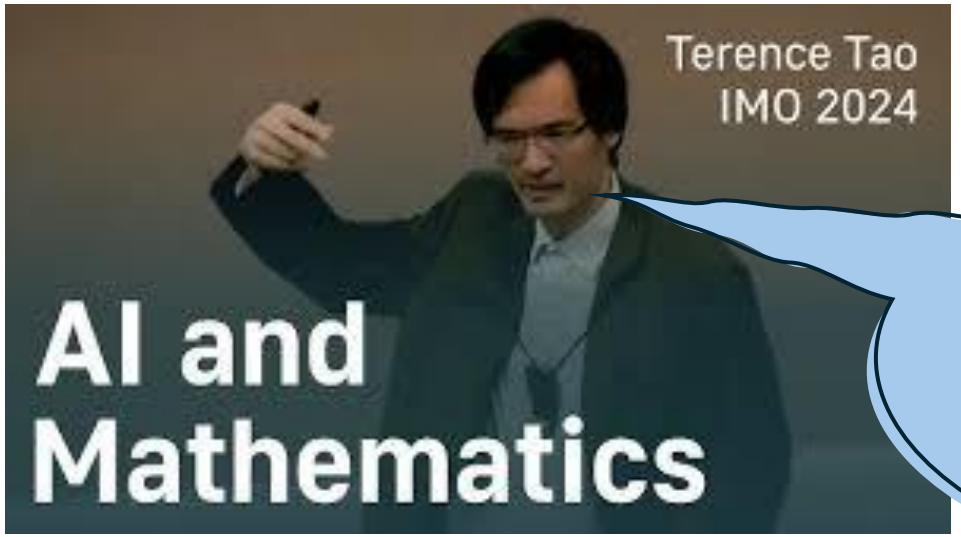
Carnegie Mellon University



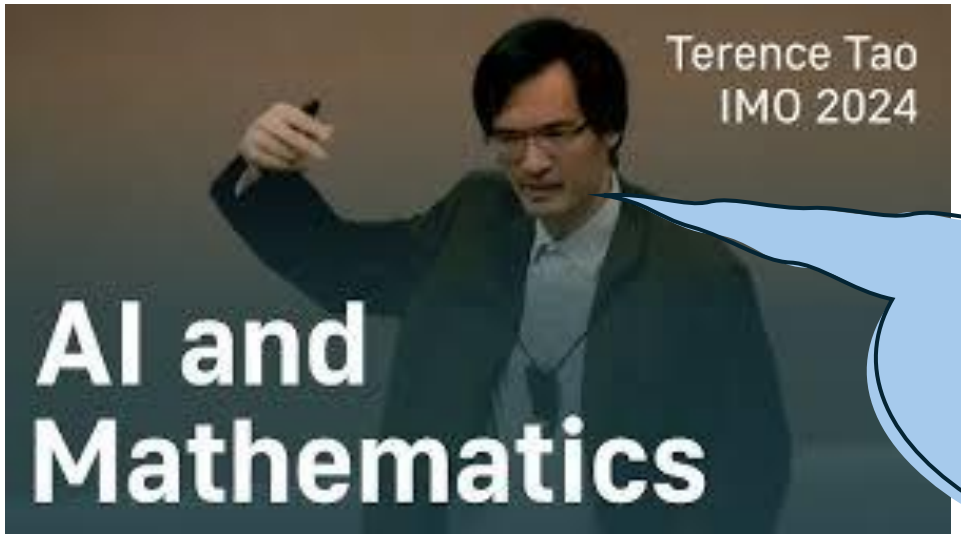
"What's most exciting to me is modeling **new modes of human-machine collaboration**.

I don't look to use these as a replacement for human mathematicians, but as **a force multiplier**."

- Jordan Ellenberg

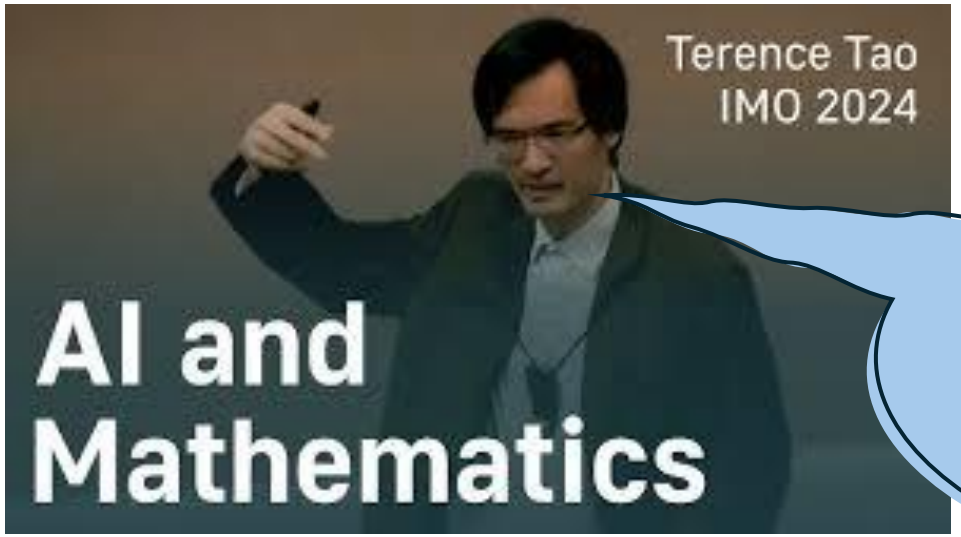


Computation has been assisting math since the beginning !



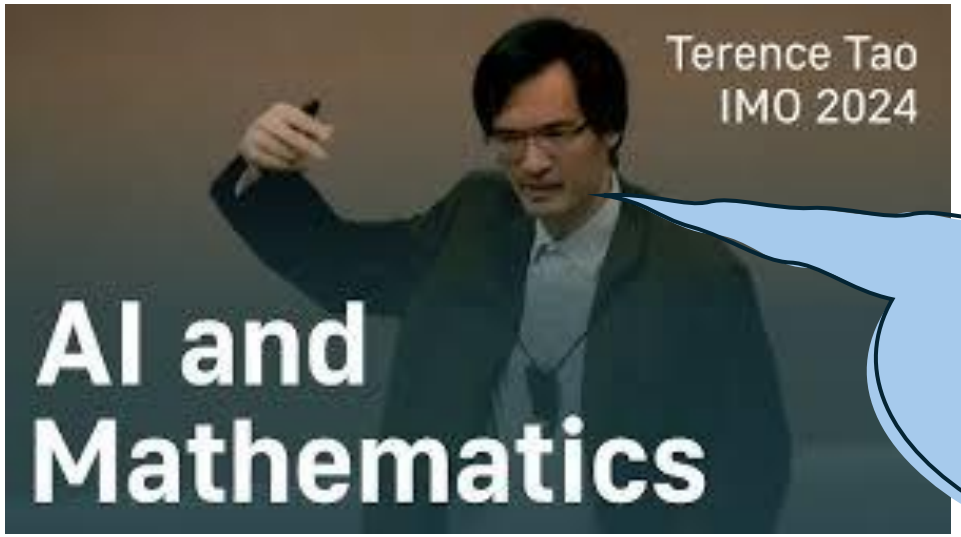
Computation has been assisting math since the beginning !

- Abacus



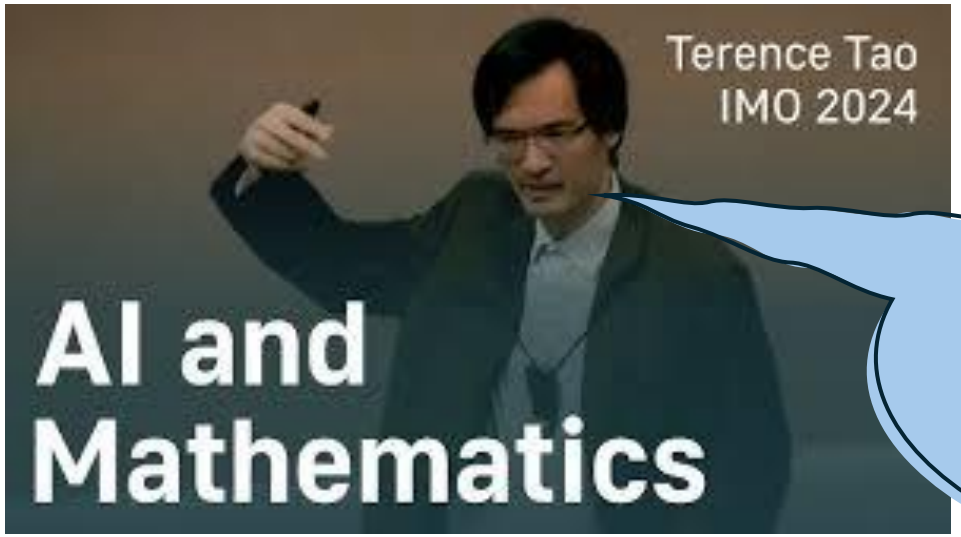
Computation has been assisting math since the beginning !

- Abacus
- Tables (Logarithm, Primes, OEIS, Elliptic curves, Groups)



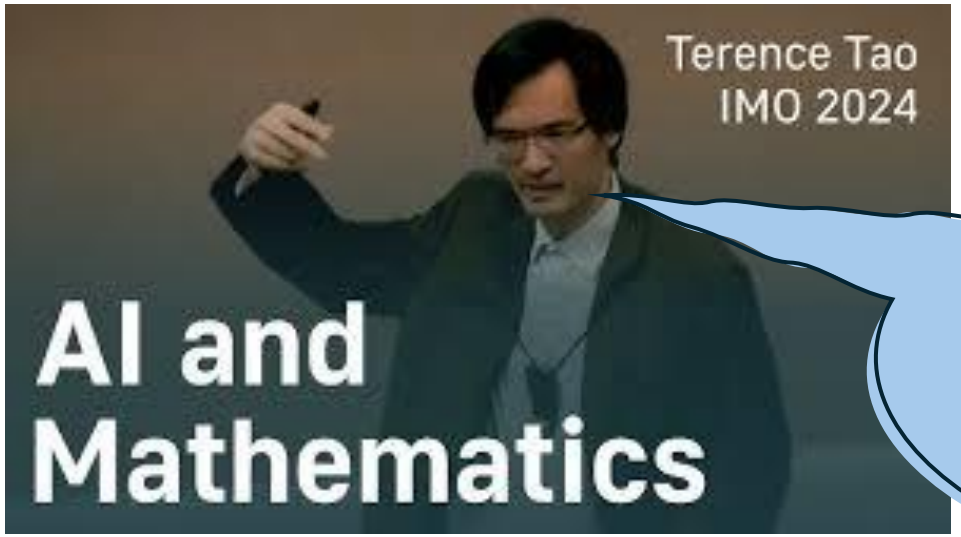
Computation has been assisting math since the beginning !

- Abacus
- Tables (Logarithm, Primes, OEIS, Elliptic curves, Groups)
- Computer Algebra Systems (Maple, SAGE, Mathematica)



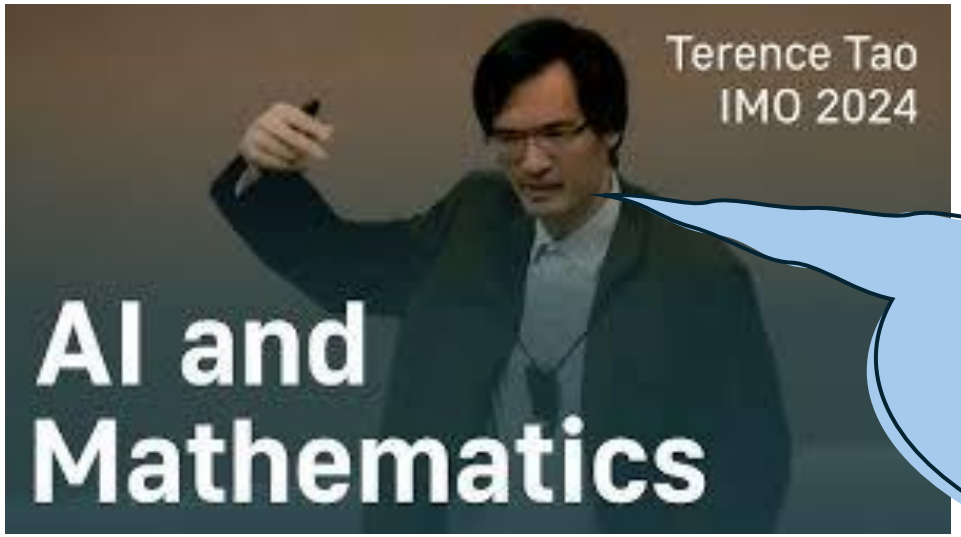
Computation has been assisting math since the beginning !

- Abacus
- Tables (Logarithm, Primes, OEIS, Elliptic curves, Groups)
- Computer Algebra Systems (Maple, SAGE, Mathematica)
- SAT solvers



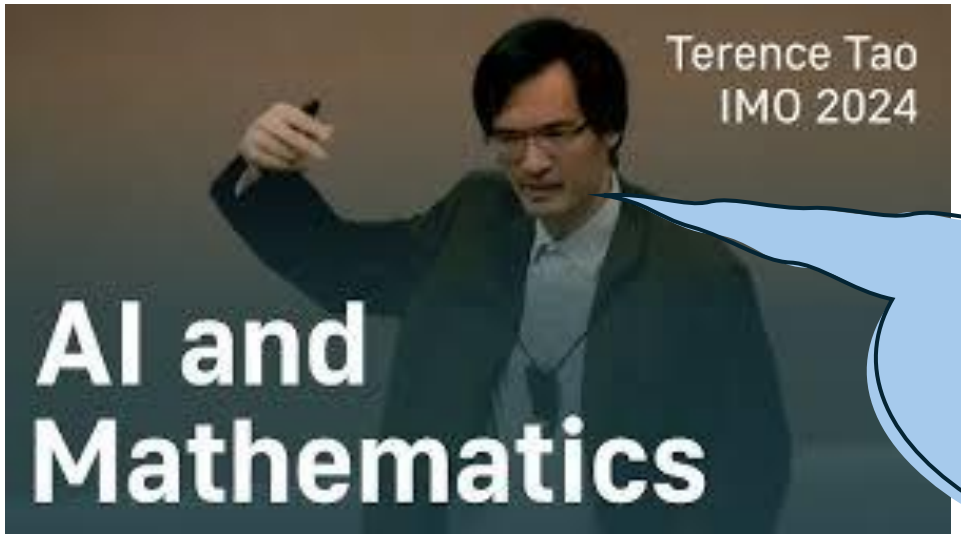
Computation has been assisting math since the beginning !

- Abacus
- Tables (Logarithm, Primes, OEIS, Elliptic curves, Groups)
- Computer Algebra Systems (Maple, SAGE, Mathematica)
- SAT solvers
- Formal theorem provers (Lean)



Computation has been assisting math since the beginning !

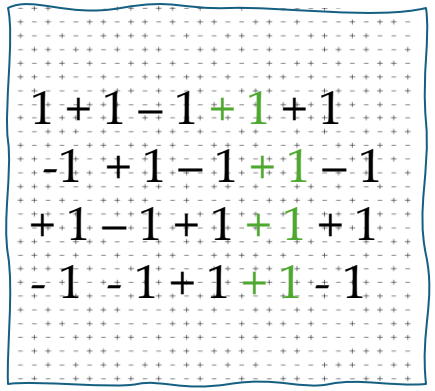
- Abacus
- Tables (Logarithm, Primes, OEIS, Elliptic curves, Groups)
- Computer Algebra Systems (Maple, SAGE, Mathematica)
- SAT solvers
- Formal theorem provers (Lean)
- LLMs?



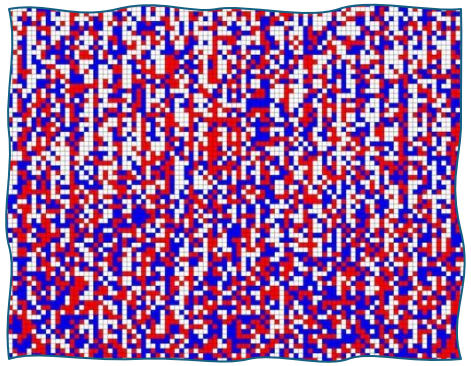
Computation has been assisting math since the beginning !

- Abacus
- Tables (Logarithm, Primes, OEIS, Elliptic curves, Groups)
- Computer Algebra Systems (Maple, SAGE, Mathematica)
- **SAT solvers**
- Formal theorem provers (Lean)
- LLMs?

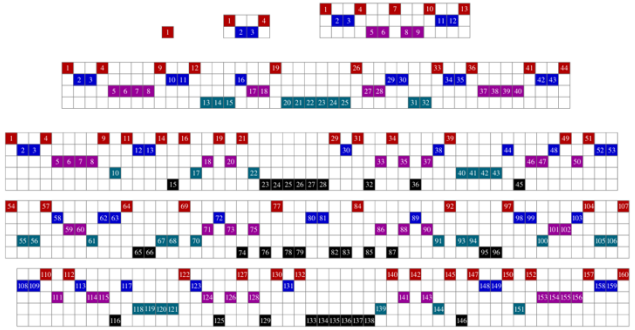
You've already heard about SAT for Math (?)



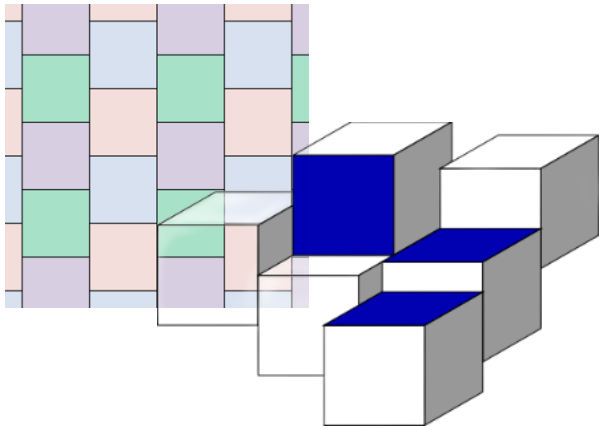
(2014) Boolean Erdős Discrepancy



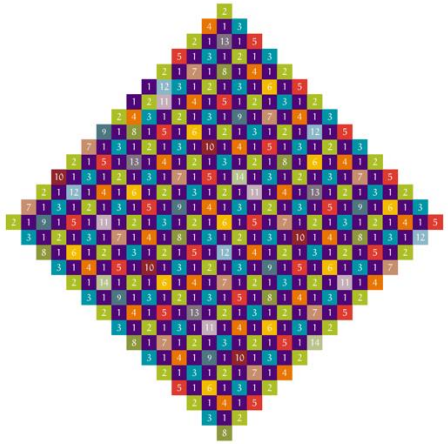
(2016) Boolean Pythagorean Triples



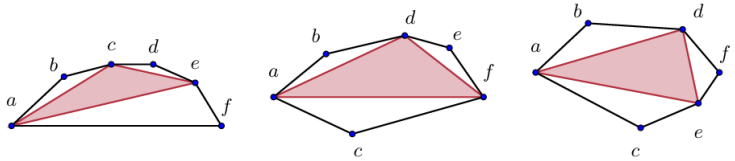
(2018) Schur Number 5



(2019) Keller's Conjecture



(2023) Packing Chromatic Number of Z^2

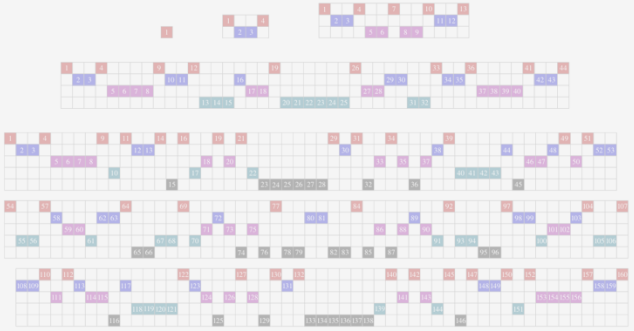


(2024) Empty Hexagon every 30 Points

mpactful in math!

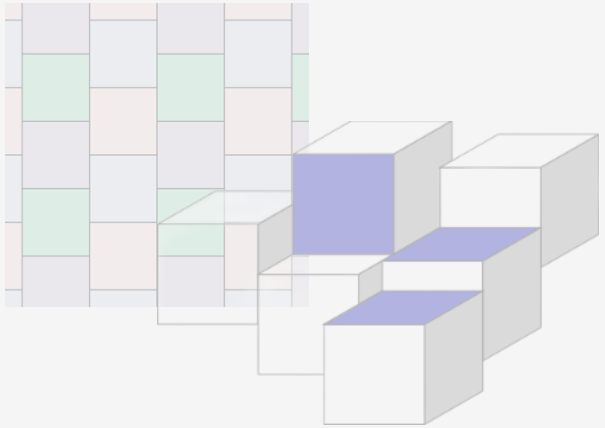
These all follow a common pattern:

Using SAT solvers to put an end to a combinatorial problem where naïve brute force (i.e., backtracking) would take forever

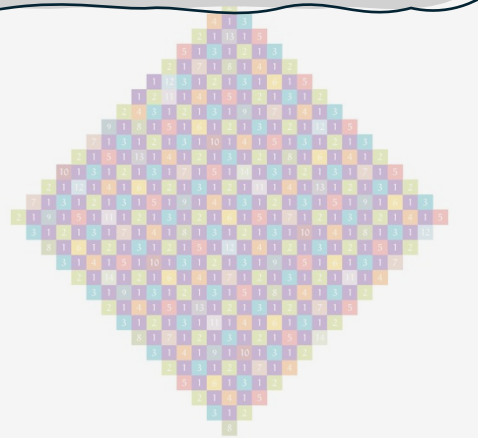


Triples

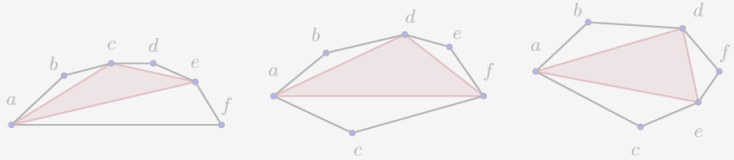
(2018) Schur Number 5



(2019) Keller's Conjecture



(2023) Packing Chromatic Number of Z^2

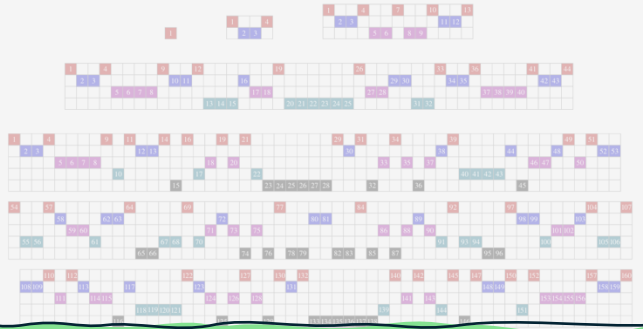


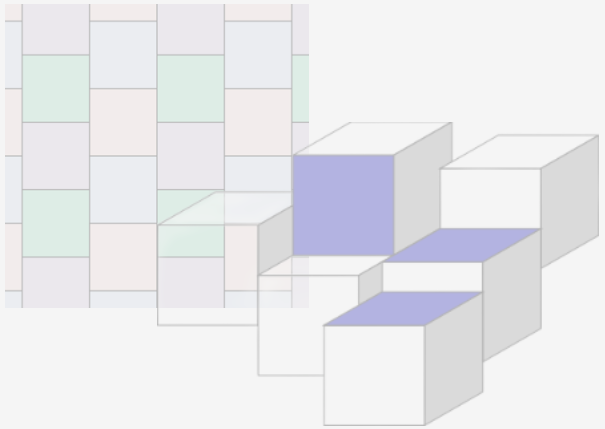
(2024) Empty Hexagon every 30 Points

mpactful in math!

These all follow a common pattern:

Using SAT solvers to put an end to a combinatorial problem where naïve brute force (i.e., backtracking) would take forever





(2019) Keller's Conjecture

(2023) Packing C

What this talk is about:

SAT solvers can be used at different stages of mathematical research; to build examples, get ideas, and elicit conjectures.

Vision:

P1. (Counter)examples help drive **mathematical progress**

Vision:

P1. (Counter)examples help drive **mathematical progress**

P2. Finding such (counter)examples can be (NP)-**hard**

Vision:

P1. (Counter)examples help drive **mathematical progress**

P2. Finding such (counter)examples can be (NP)-**hard**

P3. **SAT solvers** solve instances of NP-hard problems in practice

Vision:

P1. (Counter)examples help drive **mathematical progress**

P2. Finding such (counter)examples can be (NP)-**hard**

P3. **SAT solvers** solve instances of NP-hard problems in practice

C. **SAT solvers can help drive mathematical progress**

Leonhard Euler (1761)

Specimen de usu observationum in mathesi pura

(Example of the use of observation in pure mathematics)

(Part 1)

"It will seem not a little ***paradoxical to ascribe a great importance to observations even in that part of the mathematical sciences which is usually called Pure Mathematics****, since the current opinion is that observations are restricted to physical objects that make impression on the senses. As we must refer the numbers to the pure intellect alone, we can hardly understand how observations and quasi-experiments can be of use in investigating the nature of the numbers.* ***Yet, in fact, as I shall show here with very good reasons, the properties of the numbers known today have been mostly discovered by observation, and discovered long before their truth has been confirmed by rigid demonstrations****."*

(Part 2)

"There are even many properties of the numbers with which we are well acquainted, but which we are not yet able to prove; only observations have led us to their knowledge. Hence ***we see that in the theory of numbers, which is still very imperfect, we can place our highest hopes in observations; they will lead us continually to new properties which we shall endeavor to prove afterwards****. The kind of knowledge which is supported only by observations and is not yet proved must be carefully distinguished from the truth; it is gained by induction, as we usually say. Yet we have seen cases in which mere induction led to error. Therefore, we should take great care not to accept as true such properties of the numbers which we have discovered by observation and which are supported by induction alone. Indeed, we should use such a discovery as an opportunity to investigate more exactly the properties discovered and to prove or disprove them; in both cases we may learn something useful."*

In the next 45 minutes

I will try to convince you that SAT solvers are:

In the next 45 minutes

I will try to convince you that SAT solvers are:

- Easy to use

In the next 45 minutes

I will try to convince you that SAT solvers are:

- Easy to use
- A powerful tool for assisting discrete math research

In the next 45 minutes

I will try to convince you that SAT solvers are:

- Easy to use
- A powerful tool for assisting discrete math research
- Free of hallucinations ;)

Quick Q&A

Quick Q&A

Q: What is a SAT solver?
A: A highly optimized program for deciding if a Boolean formula is satisfiable

Quick Q&A

Q: What is a SAT solver?
A: A highly optimized program for deciding if a Boolean formula is satisfiable

Q: A Boolean formula?

A: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2)$

Quick Q&A

Q: What is a SAT solver?
A: A highly optimized program for deciding if a Boolean formula is satisfiable

Q: A Boolean formula?

A: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2)$

Q: Satisfiable?

A: Yes! $(x_1, x_2, x_3) := (0, 1, 1)$

Quick Q&A

Q: What is a SAT solver?
A: A highly optimized program for deciding if a Boolean formula is satisfiable

Q: A Boolean formula?

A: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2)$

Q: Satisfiable?

A: Yes! $(x_1, x_2, x_3) := (0, 1, 1)$

Q: And how do I write this on my computer ?

A:

test.cnf

```
p cnf 3 2
1 -2 3 0
-1 2 0
```

Quick Q&A

Q: What is a SAT solver?
A: A highly optimized program for deciding if a Boolean formula is satisfiable

Q: A Boolean formula?

A: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2)$

Q: Satisfiable?

A: Yes! $(x_1, x_2, x_3) := (0, 1, 0)$

Q: And how do I write this on my computer ?

A:

test.cnf

```
p cnf 3 2
1 -2 3 0
-1 2 0
```

Quick Q&A

Q: What is a SAT solver?
A: A highly optimized program for deciding if a Boolean formula is satisfiable

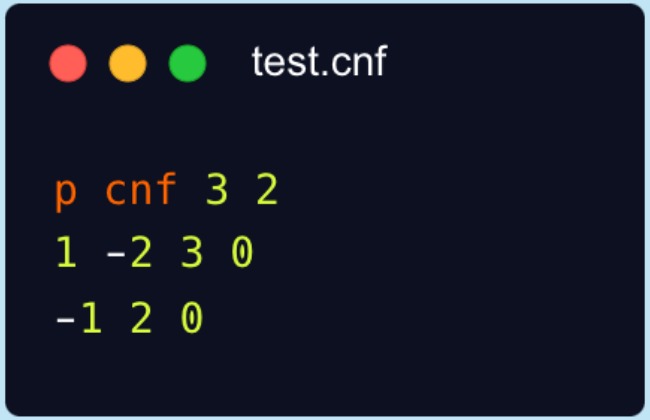
Q: A Boolean formula?

A: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2)$

Instead of writing this manually, you write high-level code that generates this!

Q: And how do I write this on my computer ?

A:



Quick Q&A

Q: And then what?

Quick Q&A

Q: And then what?

A:

or

Quick Q&A

Q: And then what?

A:

or

You can also get a reproducible proof!

Quick Q&A

Q: And then what?

A:

terminal

```
> kissat test.cnf
[...]
s SATIISFIABLE
v -1 2 3 0
```

or

terminal

```
> kissat test2.cnf
[...]
s UNSATISFIABLE
```

or

You can also get a reproducible proof!

Quick Q&A

Q: Wait but isn't SAT an NP-complete problem?!

Quick Q&A

Q: Wait but isn't SAT an NP-complete problem?!

A: Exactly!

Quick Q&A

Q: Wait but isn't SAT an NP-complete problem?!

A: Exactly!

It's hard !

Writing your own efficient algorithm is a large engineering challenge

Quick Q&A

Q: Wait but isn't SAT an NP-complete problem?!

A: Exactly!

It's hard !

Writing your own efficient algorithm is a large engineering challenge

It's universal !

Can represent a lot of different math problems

E.g., Graphs, matroids, groups, geometry

A toy example: **Sudoku**

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

Unsolved Sudoku

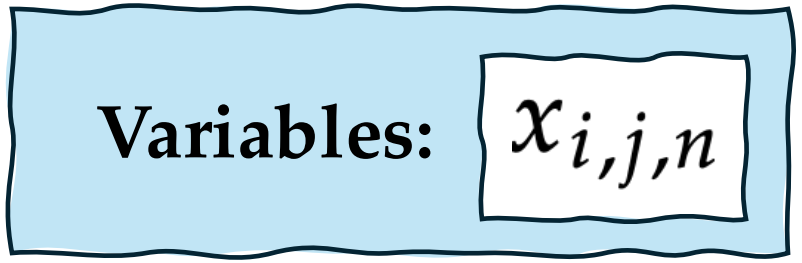
4	2	*6*	5	*7*	1	*3*	9	*8*
8	*5*	*7*	2	*9*	3	*1*	*4*	6
1	3	*9*	*4*	6	*8*	*2*	7	*5*
9	*7*	1	*3*	*8*	*5*	6	*2*	*4*
5	4	*3*	*7*	*2*	*6*	*8*	1	9
6	*8*	2	*1*	*4*	*9*	7	*5*	*3*
7	9	*4*	*6*	3	*2*	*5*	8	*1*
2	*6*	*5*	8	*1*	4	*9*	*3*	7
3	1	*8*	9	*5*	7	*4*	6	*2*

Solved Sudoku

Each cell must contain a number from {1, …, 9}

Each number from {1, …, 9} needs to appear exactly once per:
- Row
- Column
- 3x3 *major subgrid*

A toy example: **Sudoku**



"cell *(i, j)* gets number *n*"

A toy example: **Sudoku**

Variables: $x_{i,j,n}$

"cell *(i, j)* gets number *n*"

sudoku_sat.py

```python
def encode(clues):
    """
    Encode a sudoku puzzle as a SAT problem.
    clues : 9x9 int matrix, non-zero values represent clues
    """
    enc = modeler.Modeler() # my library

    # create variables
    for i in range(9):
        for j in range(9):
            for n in range(1, 10):
                enc.add_var(f"x_{i, j, n}")
```

Sudoku Constraints

Each cell must contain a number from {1, …, 9}

$$\forall i \in [9], \forall j \in [9], \quad \left(\bigvee_{n \in [9]} x_{i,j,n} \right)$$

Respect clues!

$$(x_{1,4,5}) \wedge (x_{2,1,8}) \wedge \cdots$$

Sudoku Constraints

Each cell must contain a number from {1, …, 9}

$$\forall i \in [9], \forall j \in [9], \left(\bigvee_{n \in [9]} x_{i,j,n} \right)$$

Respect clues!

$$(x_{1,4,5}) \wedge (x_{2,1,8}) \wedge \cdots$$

sudoku_sat.py

```python
# at least one number per cell
for i in range(9):
    for j in range(9):
        enc.add_clause([f"x_{i, j, n}" for n in range(1, 10)])

# respect clues
for i in range(9):
    for j in range(9):
        if clues[i][j] != 0:
            enc.add_clause([f"x_{i, j, clues[i][j]}"])
```

Sudoku Constraints

How to encode "**Exactly one** of the variables $x_1, x_2, \ldots, x_n$ is true"?

Sudoku Constraints

How to encode "**Exactly one** of the variables $x_1, x_2, \ldots, x_n$ is true"?

At least one $\left(\bigvee_{i=1}^{n} x_i \right)$

Sudoku Constraints

How to encode "**Exactly one** of the variables $x_1, x_2, \ldots, x_n$ is true"?

At least one $\left(\bigvee_{i=1}^{n} x_i \right)$

At most one (naïve) $\bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} (\neg x_i \vee \neg x_j)$

SAT encodings can be interesting ;)

If all we have are the variables $x_1, x_2, \ldots, x_n$
then AMO($x_1, x_2, \ldots, x_n$) requires $O(n^2)$ clauses

Introduce "auxiliary" variable y that *summarizes* information

SAT encodings can be interesting ;)

If all we have are the variables $x_1, x_2, \ldots, x_n$
then AMO($x_1, x_2, \ldots, x_n$) requires $O(n^2)$ clauses

Introduce "auxiliary" variable $\boldsymbol{y}$ that *summarizes* information

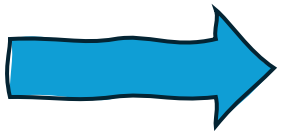
Suppose $\boldsymbol{y}$ summarizes whether a variable in $\{x_1, x_2, x_3\}$ is true

SAT encodings can be interesting ;)

If all we have are the variables $x_1, x_2, \ldots, x_n$
then AMO($x_1, x_2, \ldots, x_n$) requires $O(n^2)$ clauses

Introduce "auxiliary" variable y that *summarizes* information

Suppose y summarizes whether a variable in $\{x_1, x_2, x_3\}$ is true



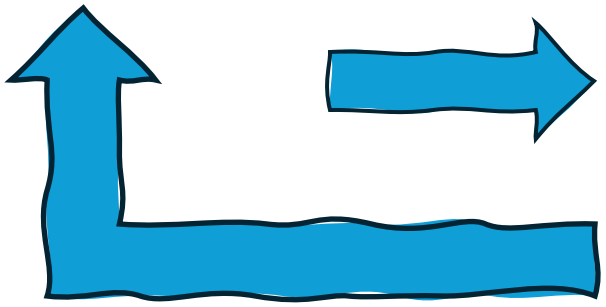
$$\text{AMO}(y, x_4, \ldots, x_n)$$

SAT encodings can be interesting ;)

If all we have are the variables $x_1, x_2, \ldots, x_n$
then AMO($x_1, x_2, \ldots, x_n$) requires $O(n^2)$ clauses

Introduce "auxiliary" variable y that *summarizes* information

Suppose y summarizes whether a variable in $\{x_1, x_2, x_3\}$ is true



$$\text{AMO}(y, x_4, \ldots, x_n)$$

$$\text{AMO}(x_1, x_2, x_3, \neg y)$$

SAT encodings can be interesting ;)

$$\mathrm{AMO}(x_1, \ldots, x_n) \cong$$
$$\mathrm{AMO}(x_1, x_2, x_3, \neg y) \wedge \mathrm{AMO}(y, x_4, \ldots, x_n)$$

SAT encodings can be interesting ;)

$$\mathrm{AMO}(x_1, \ldots, x_n) \cong$$
$$\mathrm{AMO}(x_1, x_2, x_3, \neg y) \wedge \mathrm{AMO}(y, x_4, \ldots, x_n)$$

Number of clauses $f(n)$ satisfies $f(n) = f(4) + f(n\text{-}2)$,
therefore, we have $f(n) = O(n)$

SAT encodings can be interesting ;)

$$\mathrm{AMO}(x_1, \ldots, x_n) \cong$$
$$\mathrm{AMO}(x_1, x_2, x_3, \neg y) \wedge \mathrm{AMO}(y, x_4, \ldots, x_n)$$

Number of clauses $f(n)$ satisfies $f(n) = f(4) + f(n\text{-}2)$,
therefore, we have $f(n) = O(n)$

This is just the tip of the iceberg!
Good encodings are cool math on their own!

Cardinality constraints included in libraries

sudoku_sat.py

```python
# exactly-one constraints
for n in range(1, 10):
    # rows
    for i in range(9):
        enc.exactly_one([f"x_{i, j, n}" for j in range(9)])

    # cols
    for j in range(9):
        enc.exactly_one([f"x_{i, j, n}" for i in range(9)])

    # sub_grids
    sub_grids = [[[] for sj in range(3)] for si in range(3)]
    for i in range(9):
        for j in range(9):
            sub_grids[i//3][j//3].append((i, j))
    for si in range(3):
        for sj in range(3):
            enc.exactly_one([f"x_{cell, n}" for cell in sub_grids[si][sj]])
```

Okay, now on to some math examples!

Okay, now on to some math examples!

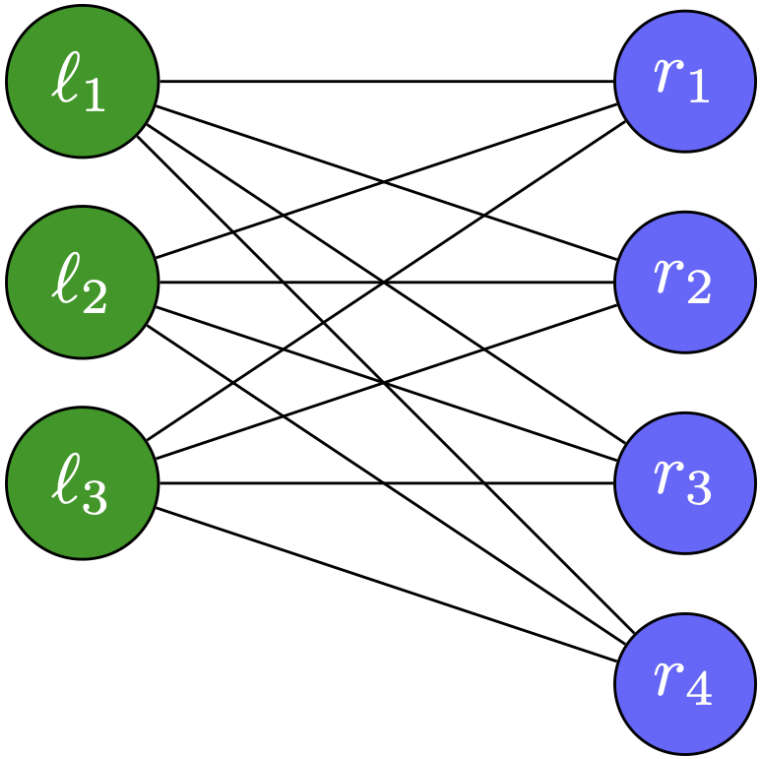
Main idea: to gain insight on a problem, we will search for finite objects.
The (non-)existence of such objects will be encoded in a Boolean formula

Mantel's problem

What's the maximum number of edges on
a triangle-free graph on n vertices?

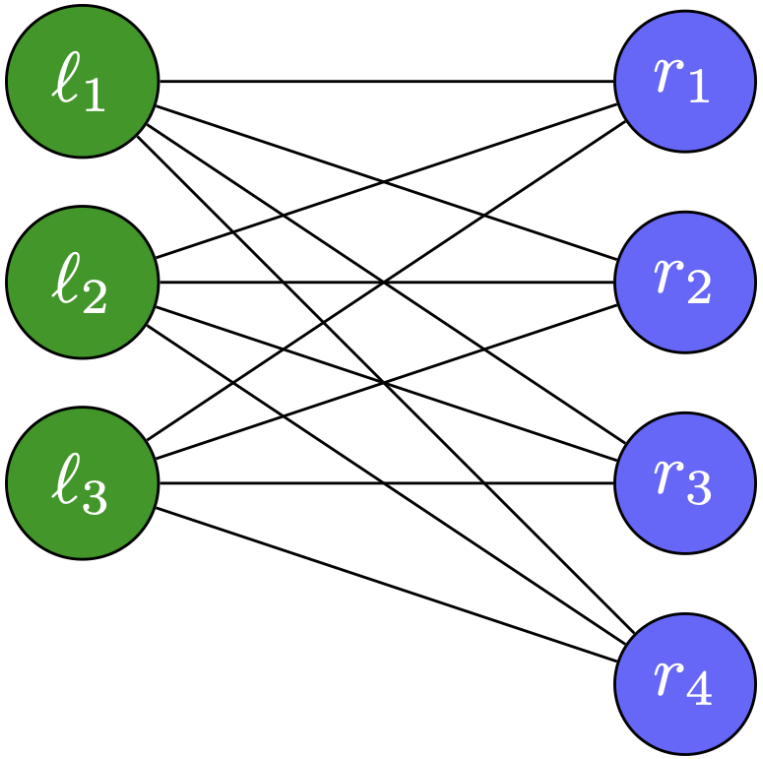
Mantel's problem

What's the maximum number of edges on a triangle-free graph on n vertices?



Mantel's problem

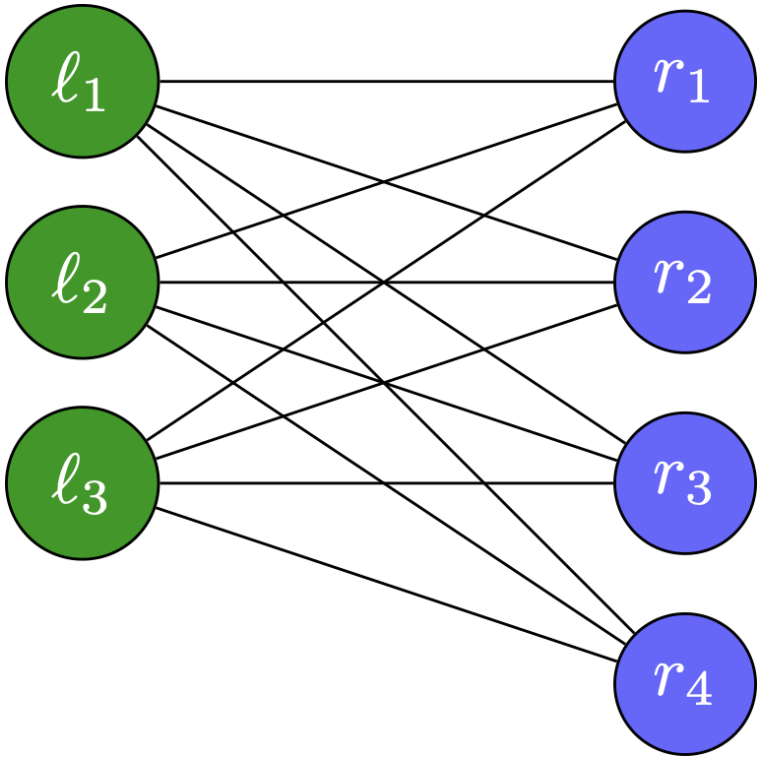
What's the maximum number of edges on a triangle-free graph on n vertices?



Could we have more edges?

Mantel's problem

What's the maximum number of edges on a triangle-free graph on n vertices?



Could we have more edges?

Is there a non-isomorphic graph that also achieves this?

Mantel's problem

What's the maximum number of edges on a triangle-free graph on n vertices?

mantel_sat.py

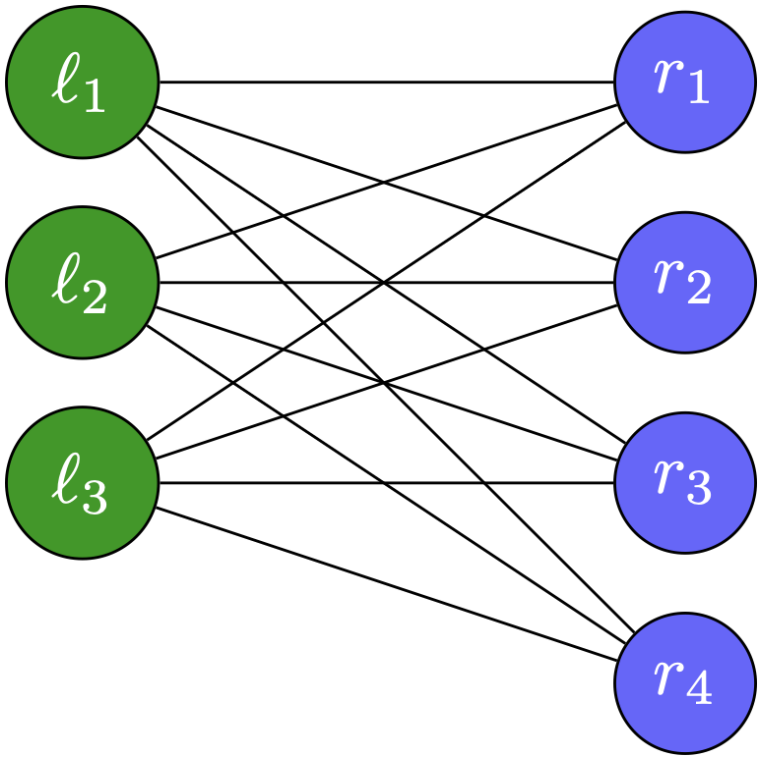
```
def encode_mantel(n, m):
    """ Encodes whether a graph with n vertices and m edges can have no triangles
    """
    enc = modeler.Modeler()
    for u in range(n):
        for v in range(u+1, n):
          enc.add_var(f"e_{u, v}") # edge variables

    # no triangles
    for u in range(n):
        for v in range(u+1, n):
            for w in range(v+1, n):
                enc.add_clause([f"-e_{u, v}", f"-e_{v, w}", f"-e_{u, w}"])

    edge_vars = [f"e_{u, v}" for u in range(n) for v in range(u+1, n)]
    enc.at_least_k(edge_vars, m)
```

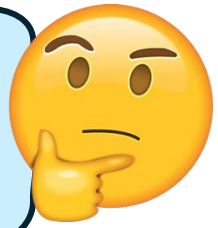
Mantel's problem

What's the maximum number of edges on a triangle-free graph on n vertices?



Could we have more edges?

Is there a non-isomorphic graph that also achieves this?



Mantel's problem & Symmetry Breaking

If sum of degrees is $2m$, some vertex has degree at least $2m/n$

WLOG, that's vertex 0, and that it's connected to 1, 2, …, ceil($2m/n$)

Mantel's problem & Symmetry Breaking

If sum of degrees is $2m$, some vertex has degree at least $2m/n$

WLOG, that's vertex 0, and that it's connected to 1, 2, …, ceil($2m/n$)

mantel_sat.py

```python
def encode_mantel(n, m):
    [...]
    # symmetry breaking
    for i in range(1, 2*math.ceil(m/n)+1):
        enc.add_clause([f"e_{0, i}"])
```

Mantel's problem & Symmetry Breaking

If sum of degrees is $2m$, some vertex has degree at least $2m/n$

WLOG, that's vertex 0, and that it's connected to 1, 2, …, ceil($2m/n$)

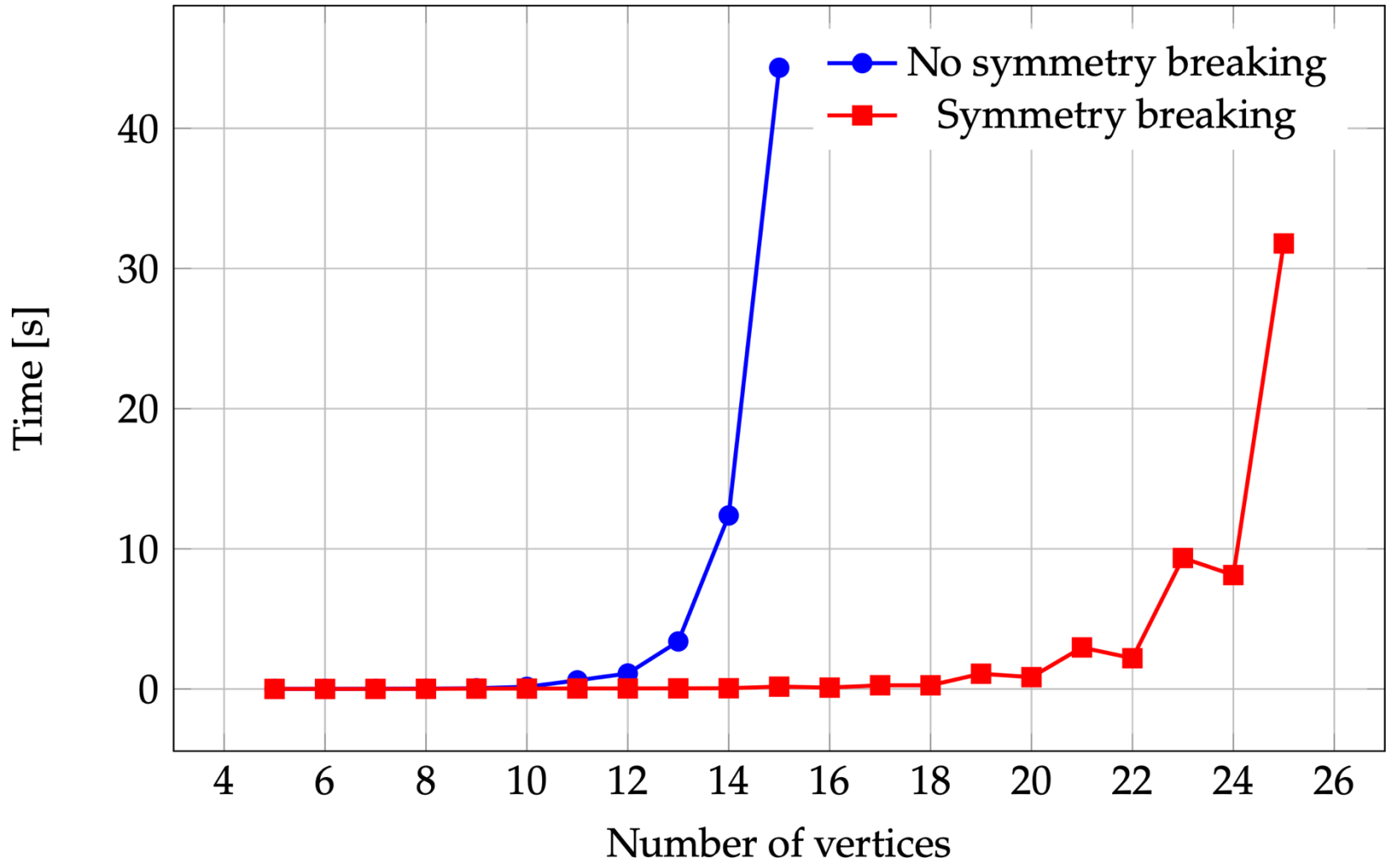
mantel_sat.py

```python
def encode_mantel(n, m):
    [...]
    # symmetry breaking
    for i in range(1, 2*math.ceil(m/n)+1):
        enc.add_clause([f"e_{0, i}"])
```

This is not enough to break all symmetries of the problem, but if we also force the edges of 0 to be sorted...

Then solution is unique!

Mantel's problem & Symmetry Breaking



Lagrange's theorem

For every subgroup **H** of a finite group **G**, |**H**| divides |**G**|

We encode, given (*n, k*), the existence of a group of size *n* with a subgroup of size *k*.

E.g., (8, 1), (8, 2), (8, 4), and *(8, 8)* are SAT, the other (*8, k*) cases are UNSAT.

Lagrange's theorem

For every subgroup **H** of a finite group **G**, |**H**| divides |**G**|

lagrange_sat.py

```python
# "multiplication" table
for i in elements:
    for j in elements:
        for k in elements:
            enc.add_var(f"m_{i, j, k}", description=f"{i} * {j} = {k}")

# i * j has a unique answer
for i in elements:
    for j in elements:
        enc.exactly_one([f"m_{i, j, k}" for k in elements])

# Neutral element is assumed to be 0 wlog. 0 * i = i * 0 = i
for i in elements:
    enc.add_clause([f"m_{0, i, i}"])
    enc.add_clause([f"m_{i, 0, i}"])
```

Lagrange's theorem

For every subgroup **H** of a finite group **G**, |**H**| divides |**G**|

lagrange_sat.py

```python
# Inverse elements:
for i in elements:
    for j in elements:
        enc.add_var(f"inv_{i, j}", description=f"{j} is the inverse of {i}")
for i in elements:
    enc.add_clause([f"inv_{i, j}" for j in elements])
    for j in elements:
        # inv_{i, j} -> m_{i, j, 0} & m_{j, i, 0}
        enc.add_clause([f"-inv_{i, j}", f"m_{i, j, 0}"])
        enc.add_clause([f"-inv_{i, j}", f"m_{j, i, 0}"])

# associativity: (i * j) * k = i * (j * k)
for i in elements:
    for j in elements:
        for k in elements:
            for l in elements:
                for m in elements:
                    for s in elements:
                        enc.add_clause([f"-m_{i, j, l}", f"-m_{j, k, m}", f"-m_{l, k, s}", f"m_{i, m, s}"])
```

Lagrange's theorem

For every subgroup **H** of a finite group **G**, |**H**| divides |**G**|

lagrange_sat.py

```
# desired subgroup:
for i in elements:
    enc.add_var(f"e_{i}", description=f"element {i} is in the subgroup")

# neutral element is in the subgroup
enc.add_clause([f"e_{0}"])

# inverses are in the subgroup
for i in elements:
    for j in elements:
        # e_i & inv_{i, j} -> e_j
        enc.add_clause([f"-e_{i}", f"-inv_{i, j}", f"e_{j}"])

# closure
for i in elements:
    for j in elements:
        for k in elements:
            enc.add_clause([f"-e_{i}", f"-e_{j}",  f"-m_{i, j, k}", f"e_{k}"])

# subgroup size
enc.exactly_k([enc.v(f"e_{i}") for i in elements], S)
```

Lagrange's theorem

For every subgroup **H** of a finite group **G**, |**H**| divides |**G**|

What about the "converse"?

Is there a group **G** on n elements, and some divisor d of n such that **G** has **no** subgroup of size d?

Lagrange's theorem

For every subgroup **H** of a finite group **G**, |**H**| divides |**G**|

What about the "converse"?

Is there a group **G** on n elements, and some divisor d of n such that **G** has **no** subgroup of size d?

Yes! A group on 12 elements without subgroups of size 6

grange's theorem

f a finite group **G**, |**H**| divides |**G**|

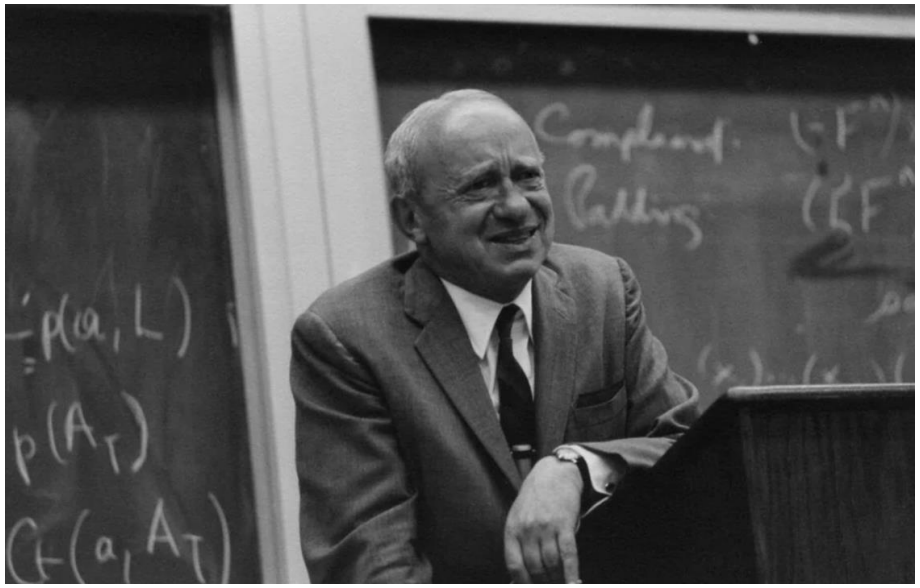
Isn't this an ∃∀ - statement that is beyond SAT?

What about the "converse"?

Is there a group **G** on n elements, and some divisor d of n such that **G** has **no** subgroup of size d?

Yes! A group on 12 elements without subgroups of size 6

"However, the machine would permit us to test the hypothesis for any special value of *n*. We could carry out such tests for a sequence of consecutive values *n*=2,3,.. up to, say, *n=100*. If the result of at least one test were negative, the hypothesis would prove to be false;



Alfred Tarski (1948)

Otherwise, our confidence in the hypothesis would increase, and we should feel encouraged to attempt establishing the hypothesis (by means of a normal mathematical proof), instead of trying to construct a counterexample."

Discrete Geometry

How many points, without 3 on a line, guarantee a **convex quadrilateral**?

Discrete Geometry

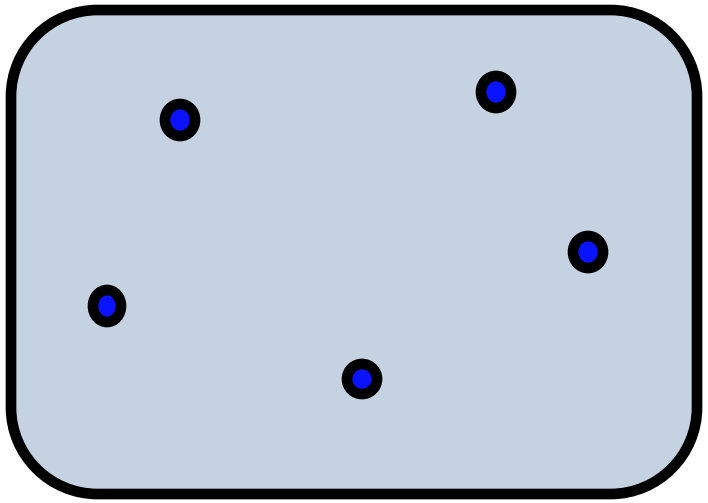
How many points, without 3 on a line, guarantee a **convex quadrilateral**?

Answer: 5, Klein, *"Happy Ending Theorem"*

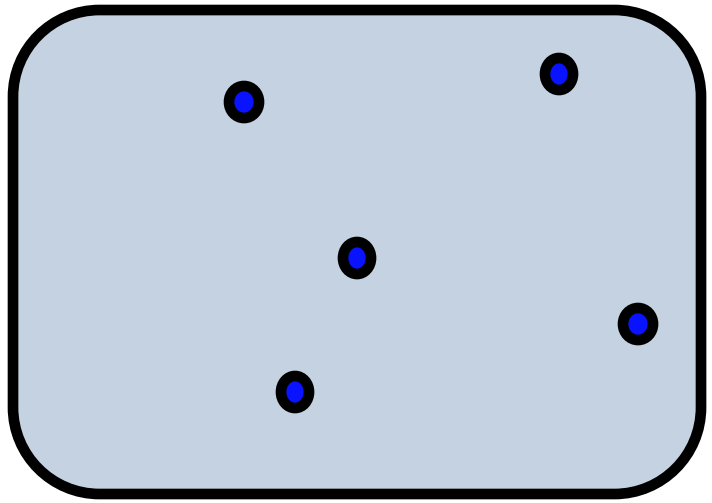
Discrete Geometry

How many points, without 3 on a line, guarantee a **convex quadrilateral**?

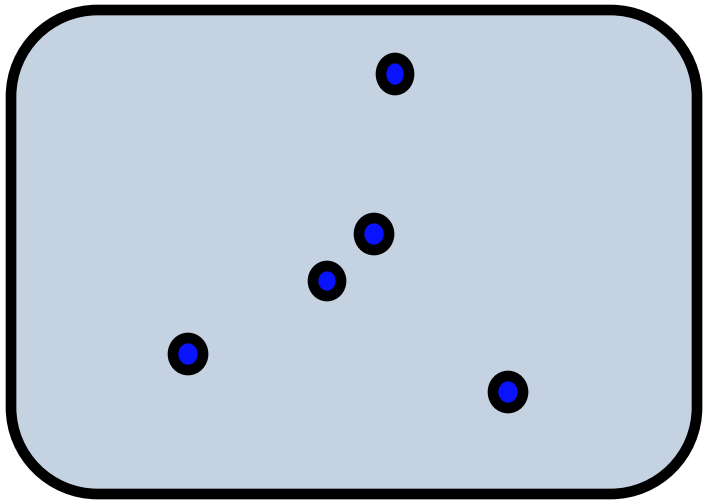
Answer: 5, Klein, *"Happy Ending Theorem"*



Case A: 5 points in convex hull



Case B: 4 points in convex hull

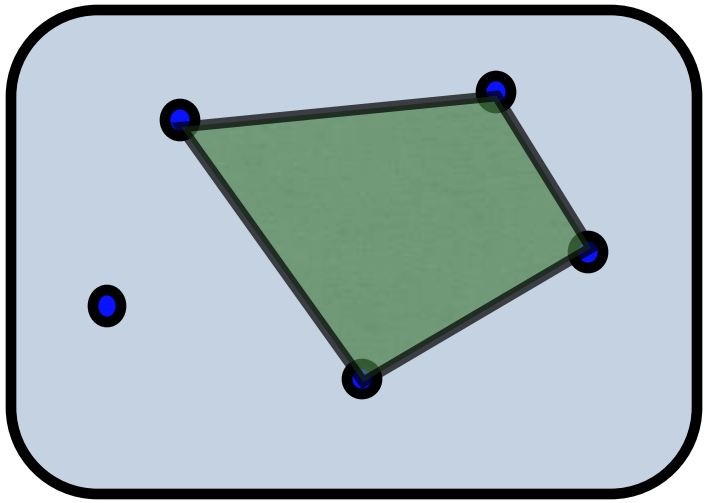


Case C: 3points in convex hull

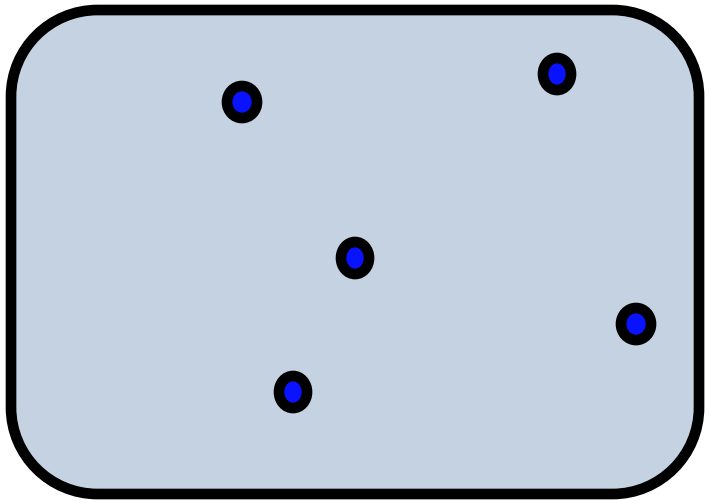
Discrete Geometry

How many points, without 3 on a line, guarantee a **convex quadrilateral**?

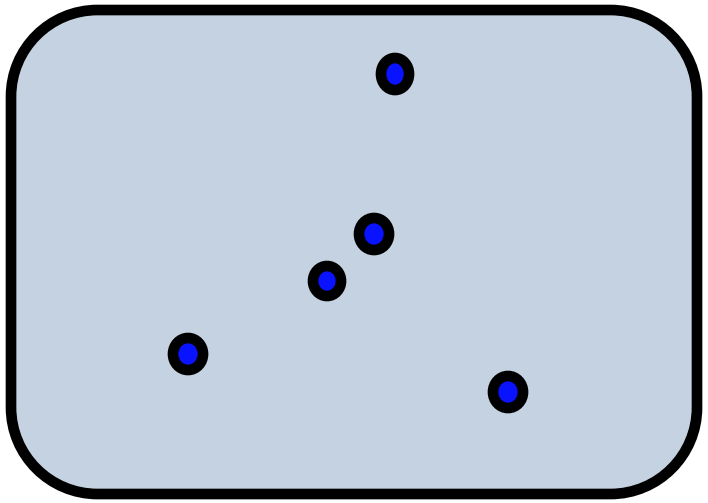
Answer: 5, Klein, *"Happy Ending Theorem"*



Case A: 5 points in convex hull



Case B: 4 points in convex hull

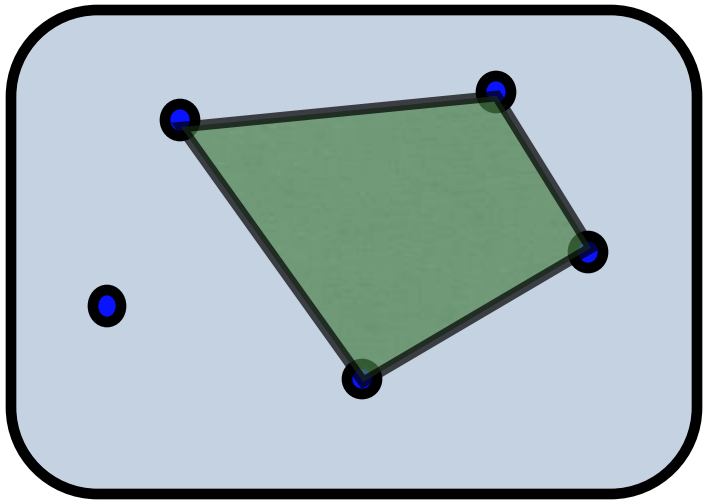


Case C: 3points in convex hull

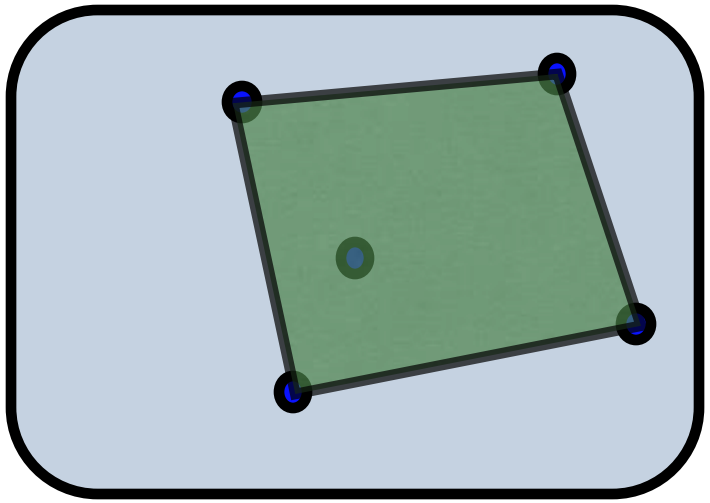
Discrete Geometry

How many points, without 3 on a line, guarantee a **convex quadrilateral**?

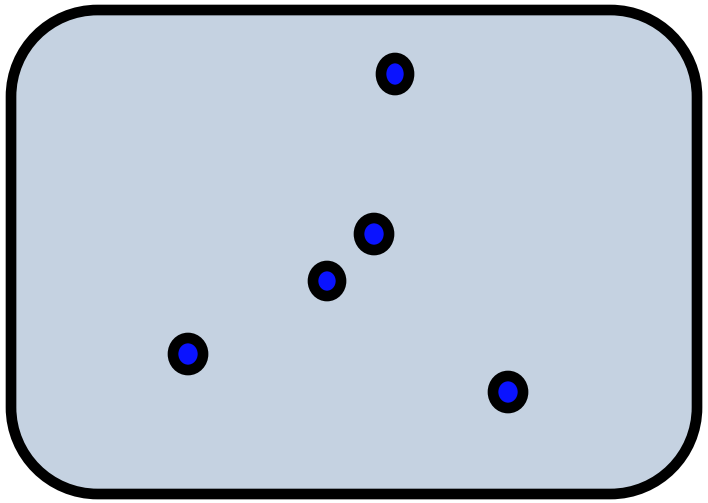
Answer: 5, Klein, *"Happy Ending Theorem"*



Case A: 5 points in convex hull



Case B: 4 points in convex hull

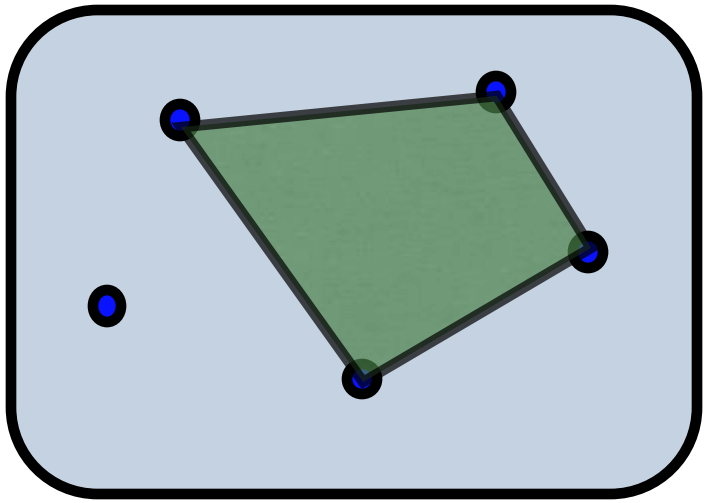


Case C: 3points in convex hull

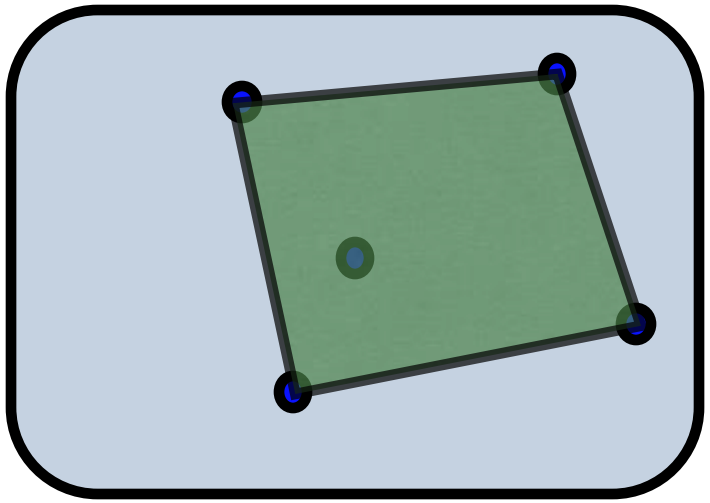
Discrete Geometry

How many points, without 3 on a line, guarantee a **convex quadrilateral**?

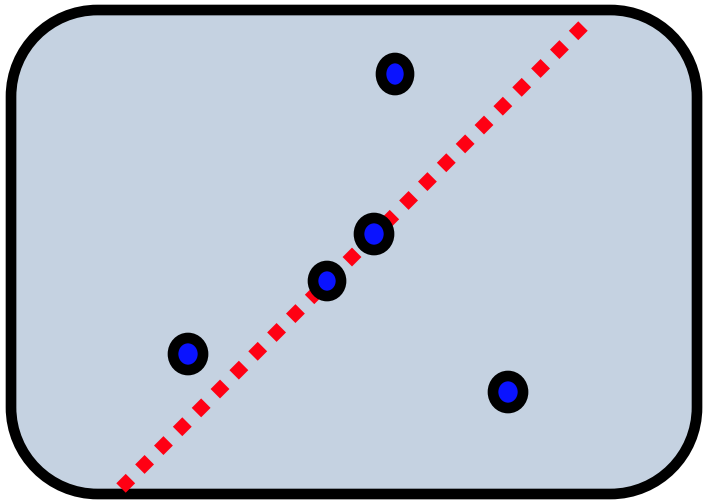
Answer: 5, Klein, *"Happy Ending Theorem"*



Case A: 5 points in convex hull



Case B: 4 points in convex hull

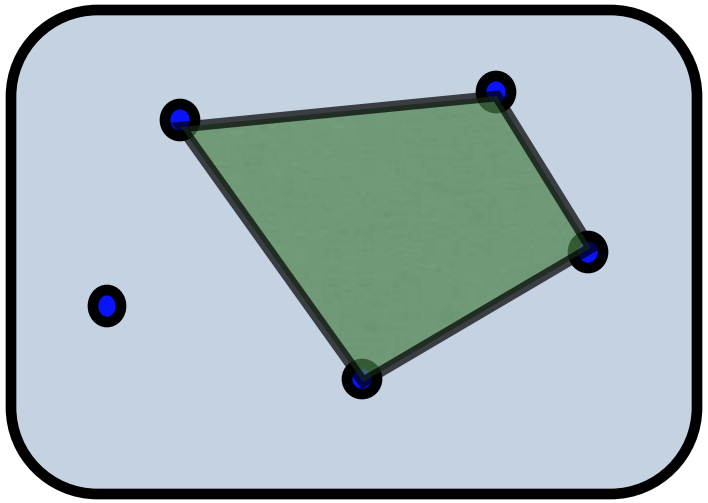


Case C: 3points in convex hull

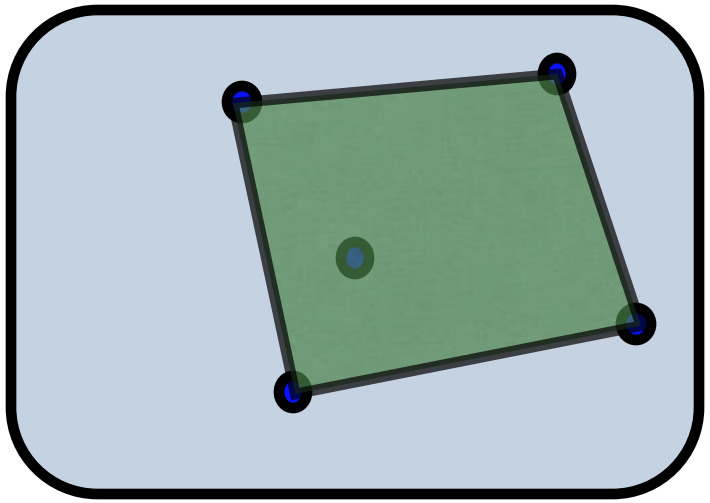
Discrete Geometry

How many points, without 3 on a line, guarantee a **convex quadrilateral**?

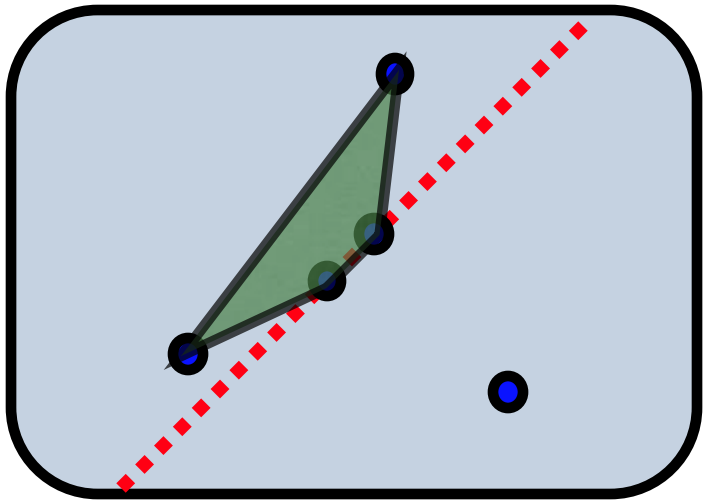
Answer: 5, Klein, *"Happy Ending Theorem"*



Case A: 5 points in convex hull



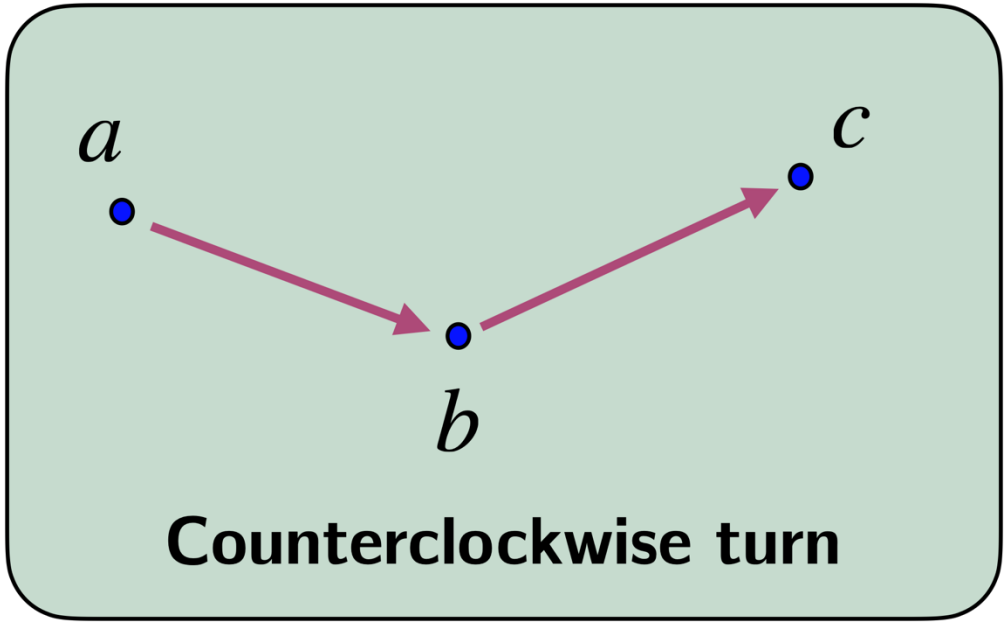
Case B: 4 points in convex hull



Case C: 3points in convex hull

Discrete Geometry

We can define variables $s_{(a,\,b,\,c)}$ that represent whether a,b,c is a CC-turn

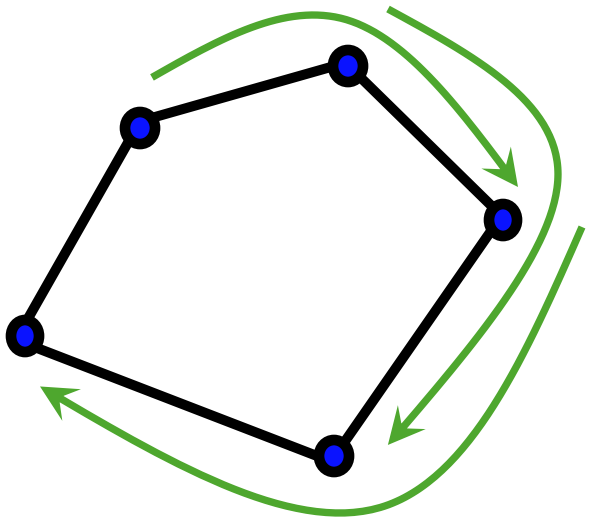


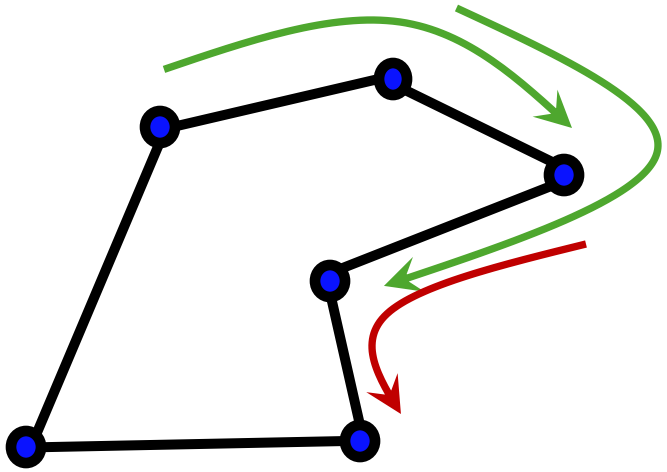
$$\det \begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ 1 & 1 & 1 \end{bmatrix} > 0$$

Discrete Geometry

A polygon is convex if and only if the turns in “cyclic order”

are either all clockwise or all counterclockwise



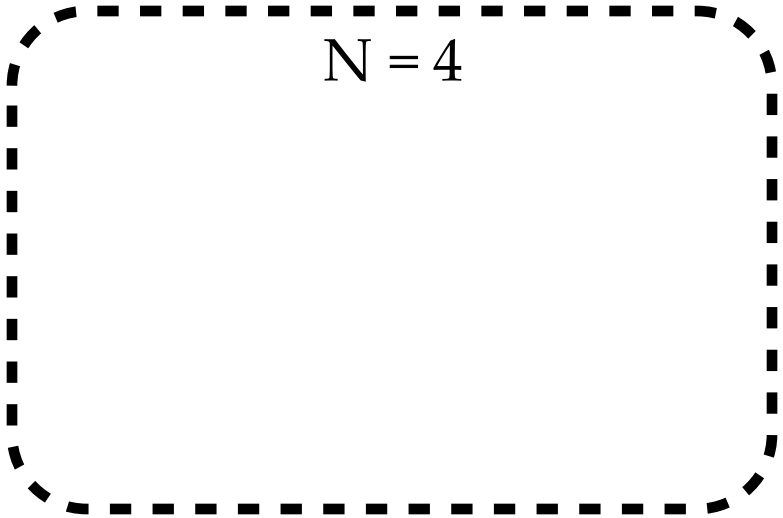


Problem: Minimizing Convex Pentagons

Given an integer N, what is, $\mu_5(N)$ the minimum number of convex pentagons we can get by placing N points in the Euclidean plane without 3 on a line?

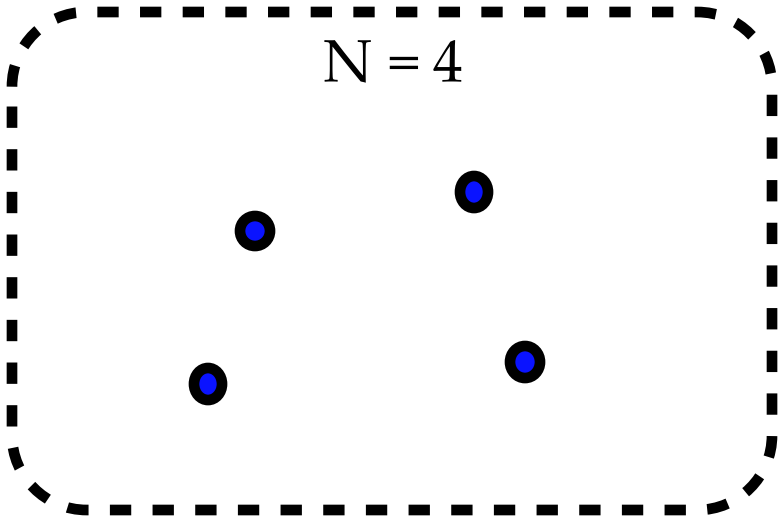
Problem: Minimizing Convex Pentagons

Given an integer N, what is, $\mu_5(N)$ the minimum number of convex pentagons we can get by placing N points in the Euclidean plane without 3 on a line?



Problem: Minimizing Convex Pentagons

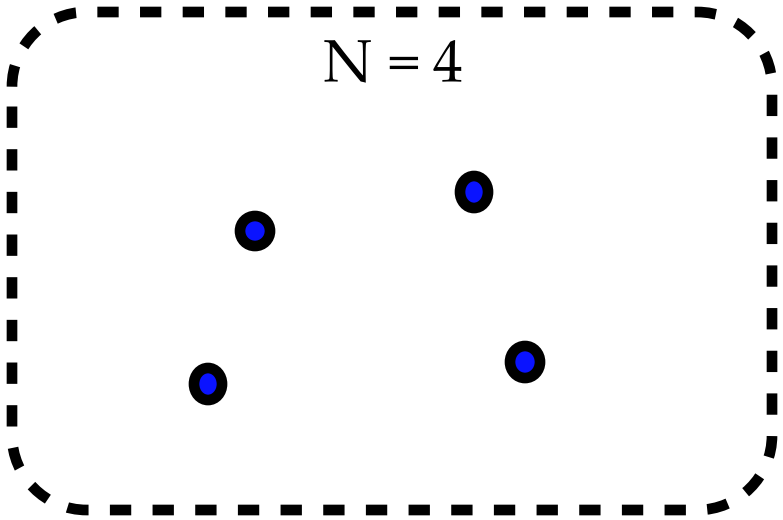
Given an integer N, what is, $\mu_5(N)$ the minimum number of convex pentagons we can get by placing N points in the Euclidean plane without 3 on a line?



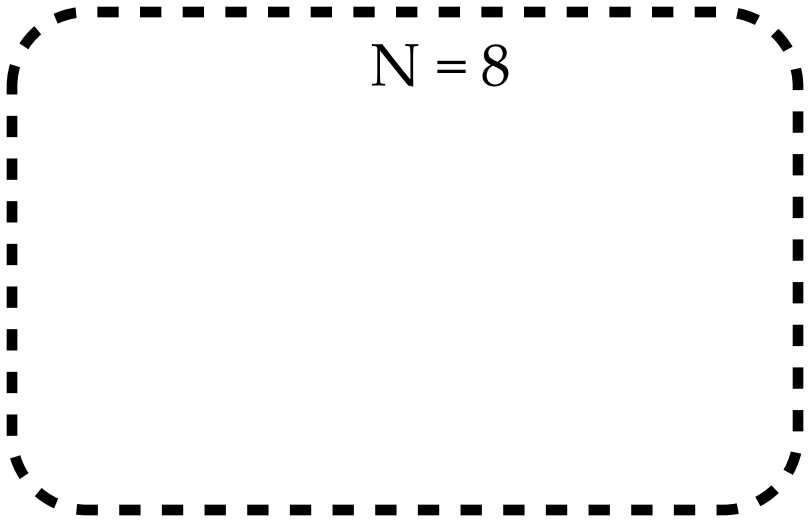
Answer = 0

Problem: Minimizing Convex Pentagons

Given an integer N, what is, $\mu_5(N)$ the minimum number of convex pentagons we can get by placing N points in the Euclidean plane without 3 on a line?

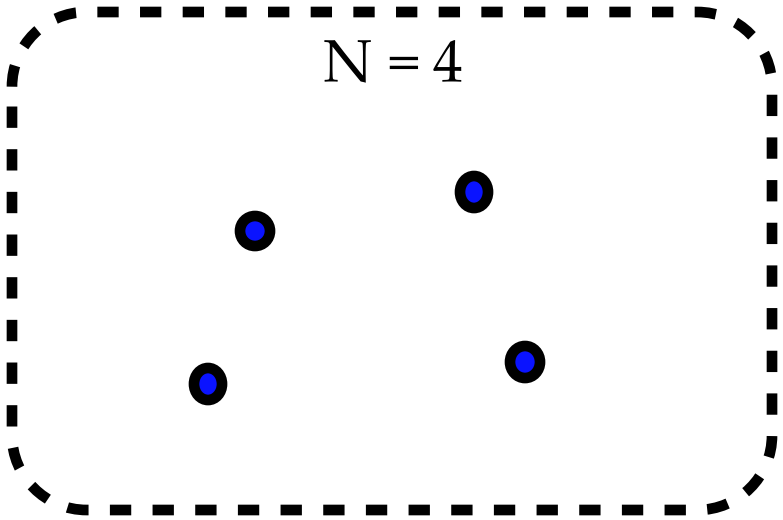


Answer = 0

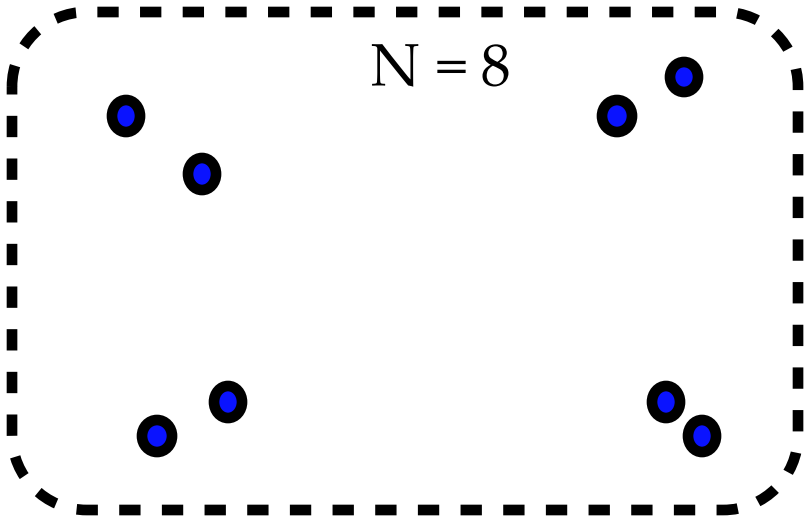


Problem: Minimizing Convex Pentagons

Given an integer N, what is, $\mu_5(N)$ the minimum number of convex pentagons we can get by placing N points in the Euclidean plane without 3 on a line?



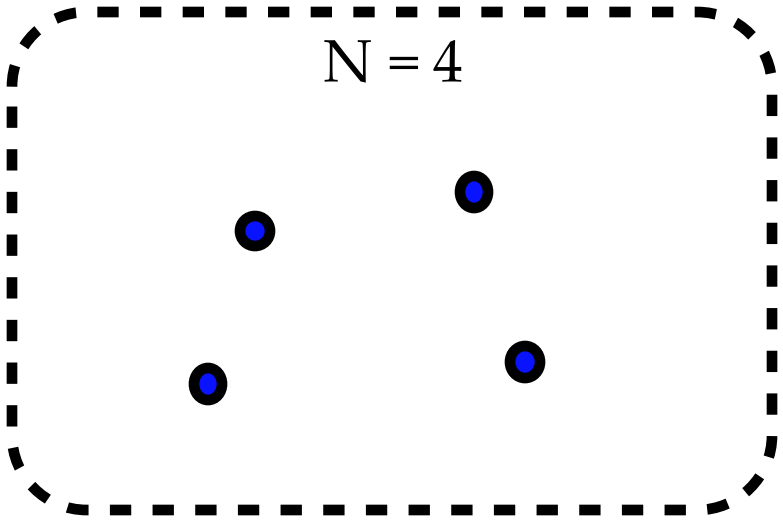
Answer = 0



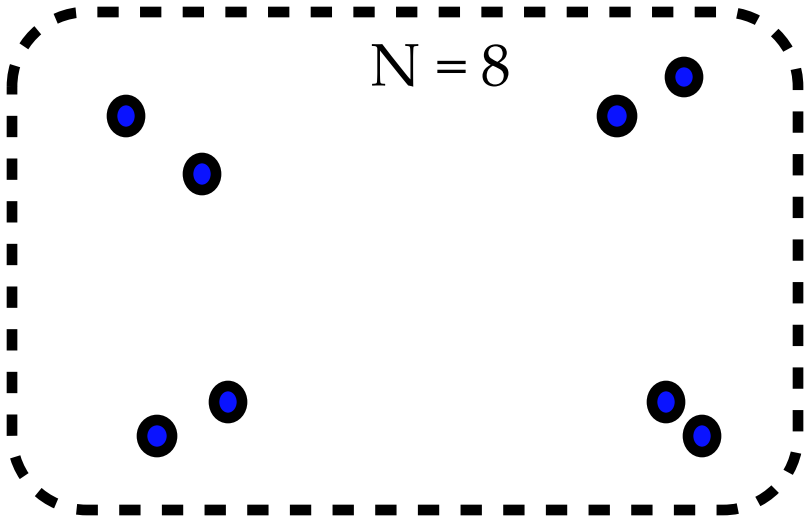
Answer = 0

Problem: Minimizing Convex Pentagons

Given an integer N, what is, $\mu_5(N)$ the minimum number of convex pentagons we can get by placing N points in the Euclidean plane without 3 on a line?



Answer = 0



Answer = 0

N	$\mu_5(N)$
4	0
8	0
9	1
15	77

Minimum number of convex pentagons among N points?

N	Best	Time
9	1	0.00 s
10	2	0.00 s
11	7	0.00 s
12	12	0.00 s
13	27	0.01 s
14	42	0.01 s
15	77	0.01 s
16	112	0.02 s

...

N	Best	Time
23	1254	12 s
24	1584	472 s
25	2079	64 s
26	2574	5269 s
27	3289	1556 s
28	4004	1792 s
29	5005	467 s
30	6007	18244 s

Minimum number of convex pentagons among N points?

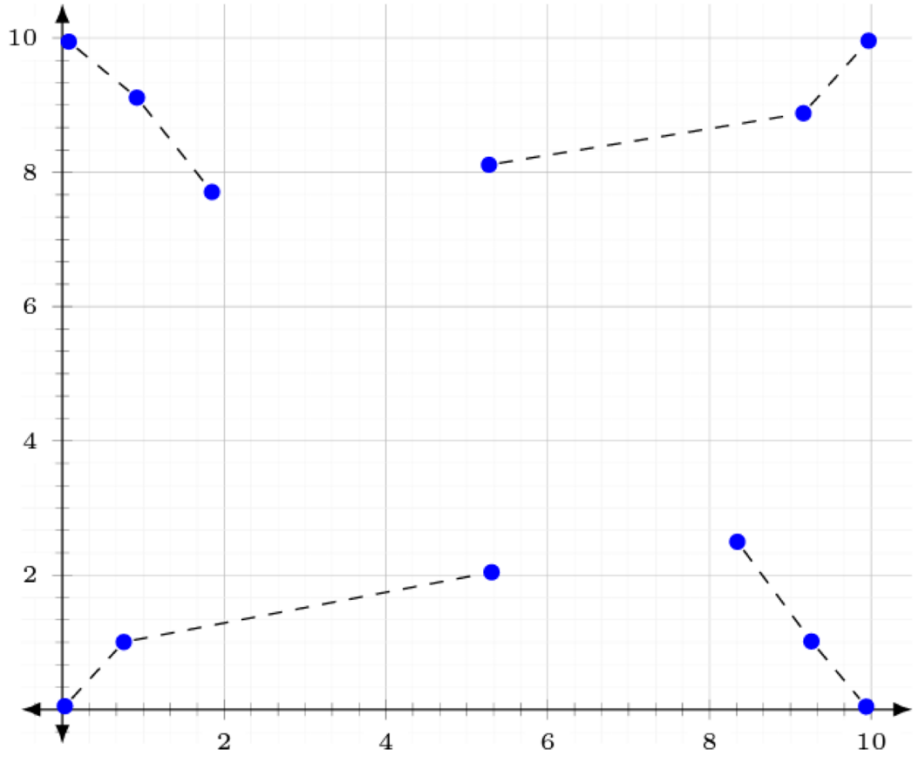
N	Best	Time
9	1	0.00 s
10	2	
11	7	
12	12	
13	27	
14	42	0.01 s
15	77	0.01 s
16	112	0.02 s

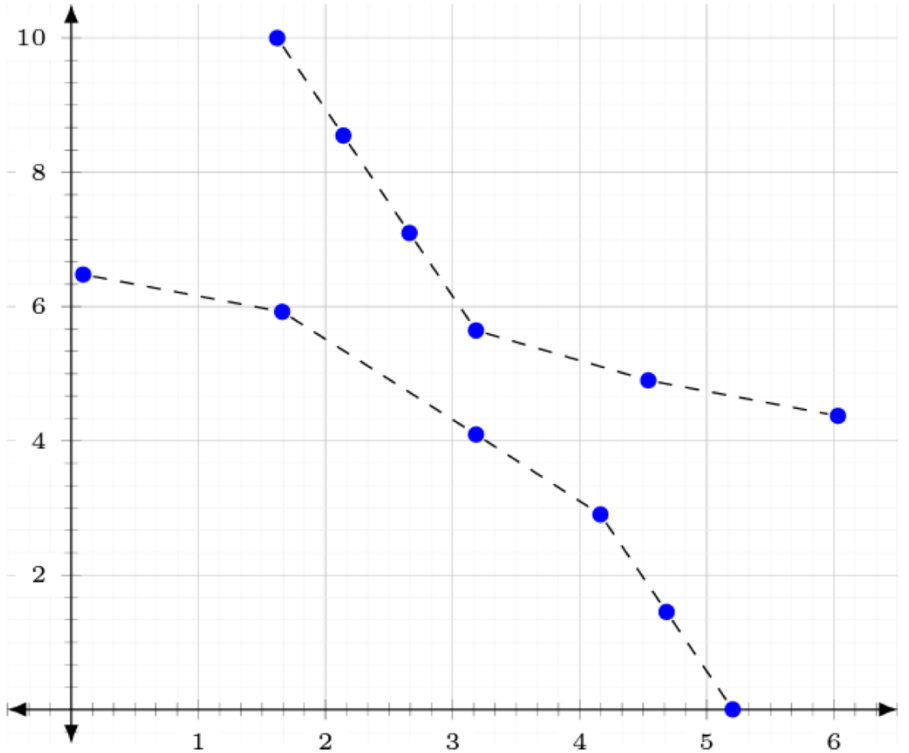
$$\mu_5(N) = \binom{\lfloor N/2 \rfloor}{5} + \binom{\lceil N/2 \rceil}{5}$$

N	Best	Time
23	1254	12 s
	4	472 s
	9	64 s
	4	5269 s
	9	1556 s
28	4004	1792 s
29	5005	467 s
30	6007	18244 s

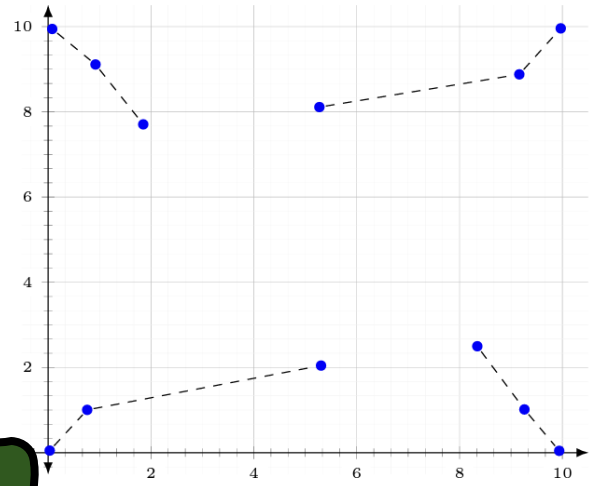
Realizability ($\exists\mathbb{R}$-complete)

How to minimize # of convex pentagons among N points in general position?

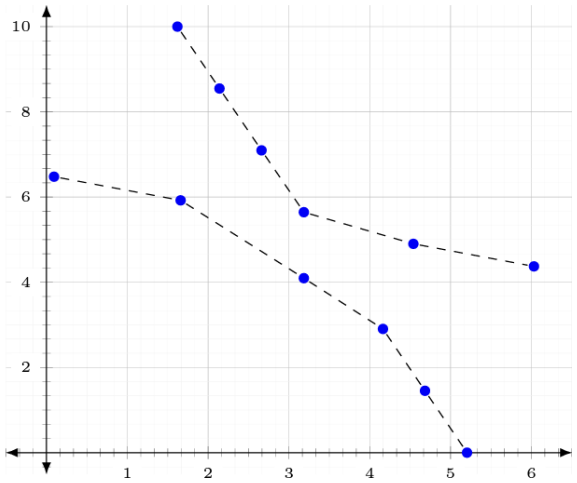




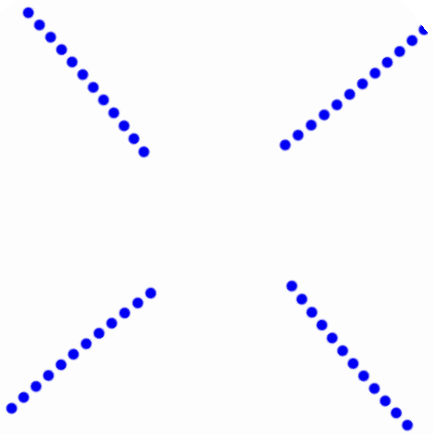
Generalize them as constructions!



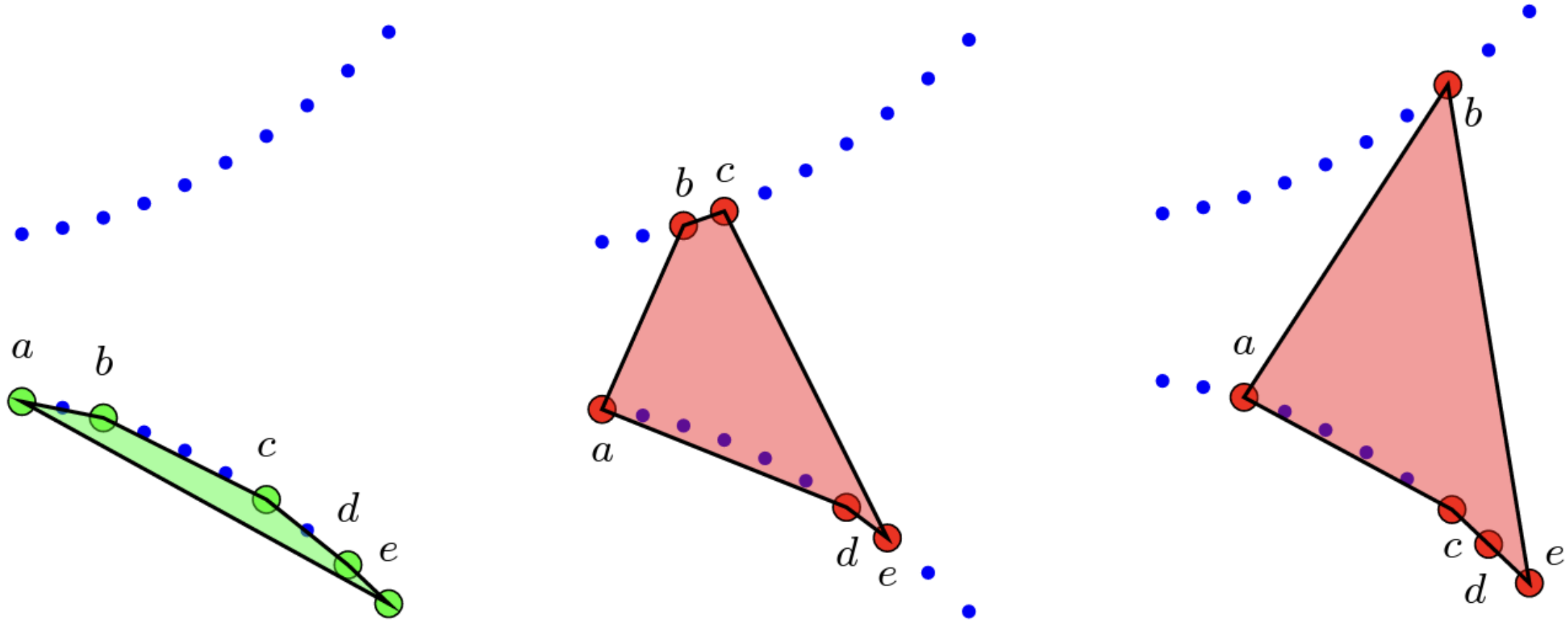
Pinwheel

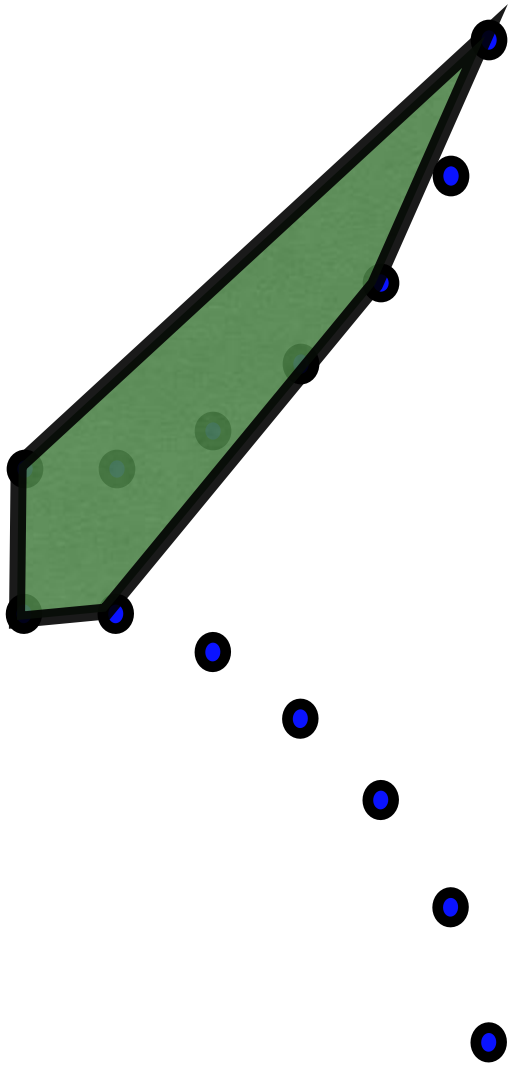


Parabolic

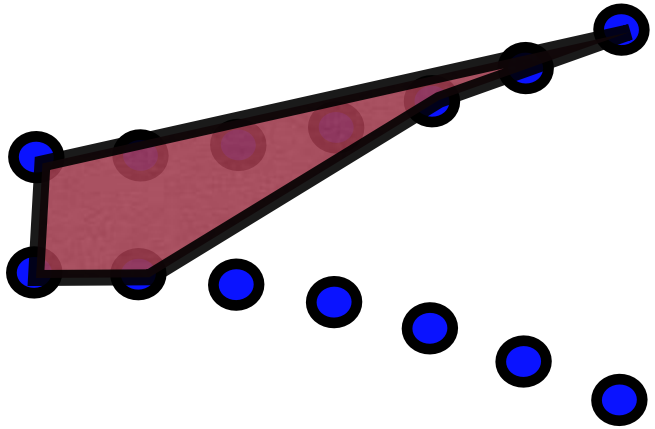


Parabolic Construction





We need to be careful though!



$$p_i^{\top} = \left(i, 2 + \frac{i^2}{n^2}\right), \forall i \in \left[\left\lfloor \frac{n}{2} \right\rfloor\right] \quad \text{and} \quad p_i^{\perp} = \left(i, -2 - \frac{i^2}{n^2}\right), \forall i \in \left[\left\lceil \frac{n}{2} \right\rceil\right]$$

Distance Digits problem

How can I find a non-exhaustive algorithm for the following problem? In the N-base number system, there are exactly $N^{(N+1)}$ numbers with $N+1$ digits. I would like to select N^2 numbers such that the Hamming distance between any two of them is the same. For example, for $N=3$, here are the $3^2=9$ numbers:

$$0000, 0111, 0222, 1012, 1120, 1201, 2021, 2102, 2210$$

all with a Hamming distance of 3 between any two pairs. I am looking for a feasible algorithm that finds N^2 numbers with the same Hamming distance, if they exist. The value of N is sufficient if it is maximum 10. In this case, the distance can be a predefined value, it is not necessary for the program to vary it. As background information: I started with a geometry problem in N-dimensional Euclidean space and ended up with this combinatorial question.

algorithms python

Can we have **N^2** many words over alphabet **{0, …., N-1}**, with length **N+1**, that are all at same Hamming distance from each other?

Distance Digits problem

Can we have $\mathbf{N^2}$ many words over alphabet **{0, …., N-1},** with length **N+1,** that are all at same Hamming distance from each other?

Encoded into SAT and found solutions for **N = 4** and **N = 5** right away.
Not able to find a solution for **N = 6**…

Distance Digits problem

Can we have $\mathbf{N^2}$ many words over alphabet **{0, …., N-1},** with length **N+1,** that are all at same Hamming distance from each other?

Encoded into SAT and found solutions for **N = 4** and **N = 5** right away.
Not able to find a solution for **N = 6**…

I started looking at solutions for **N = 5** and see if I could enforce more patterns: e.g., that one string is just **00000**.

Distance Digits problem

Can we have N^2 many words over alphabet **{0, …., N-1}**, with length **N+1**, that are all at same Hamming distance from each other?

I started looking at solutions for **N = 5** and see if I could enforce more patterns: e.g., that one string is just **000000**.

000000	112340	224130	331420	443210
011111	123401	230241	342031	404321
022222	134012	241302	303142	410432
033333	140123	202413	314203	421043
044444	101234	213024	320314	432104

Distance Digits problem

000000	112340	224130	331420	443210
011111	123401	230241	342031	404321
022222	134012	241302	303142	410432
033333	140123	202413	314203	421043
044444	101234	213024	320314	432104

Let $S(a, b)$ be string number b from block a (both indexed from 0).
Then $S(a, b)[0] = a$, and for $i > 0$, $S(a, b)[i] = (a \cdot i + b) \bmod 5$

Distance Digits problem

000000	112340	224130	331420	443210
011111	123401	230241	342031	404321
022222	134012	241302	303142	410432
033333	140123	202413	314203	421043
044444	101234	213024	320314	432104

Let $S(a, b)$ be string number b from block a (both indexed from 0).
Then $S(a, b)[0] = a$, and for $i > 0$, $S(a, b)[i] = (a \cdot i + b) \bmod 5$

Claim: $S(a_1, b_1)$ has exactly one position equal to any other $S(a_2, b_2)$.

Distance Digits problem

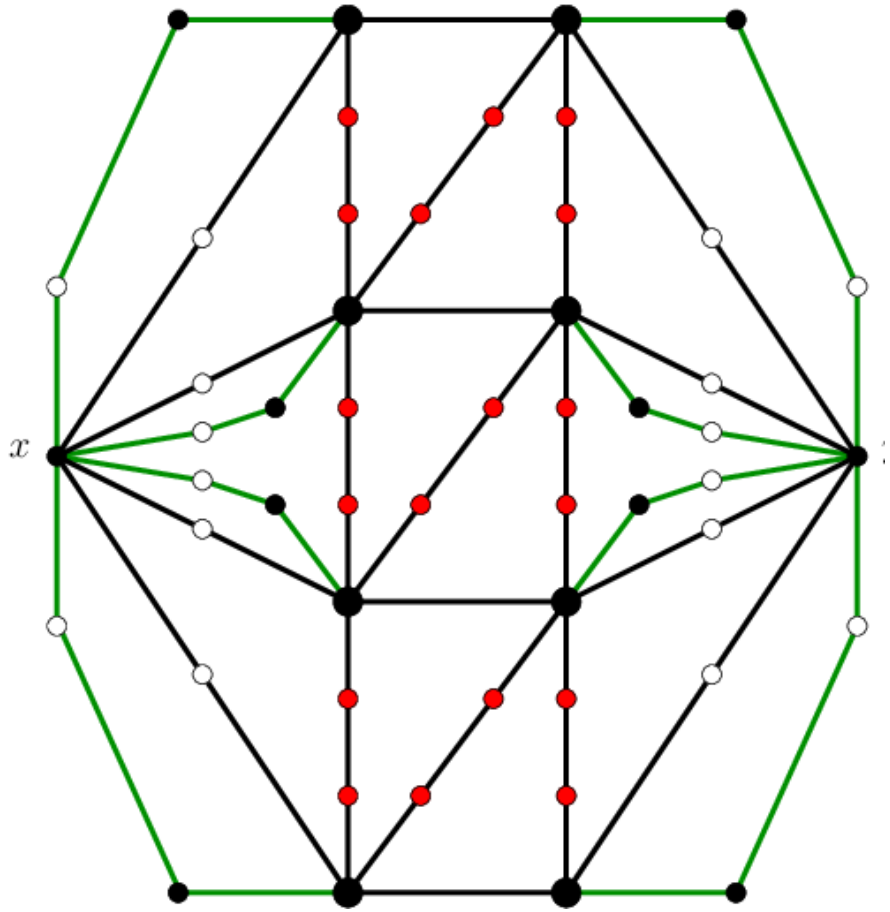
000000	112340	224130	331420	443210
011111	123401	230241	342031	404321
022222	134012	241302	303142	410432
033333	140123	202413	314203	421043
044444	101234	213024	320314	432104

Let $S(a, b)$ be string number b from block a (both indexed from 0).
Then $S(a, b)[0] = a$, and for $i > 0$, $S(a, b)[i] = (a \cdot i + b) \bmod 5$

Claim: $S(a_1, b_1)$ has exactly one position equal to any other $S(a_2, b_2)$.

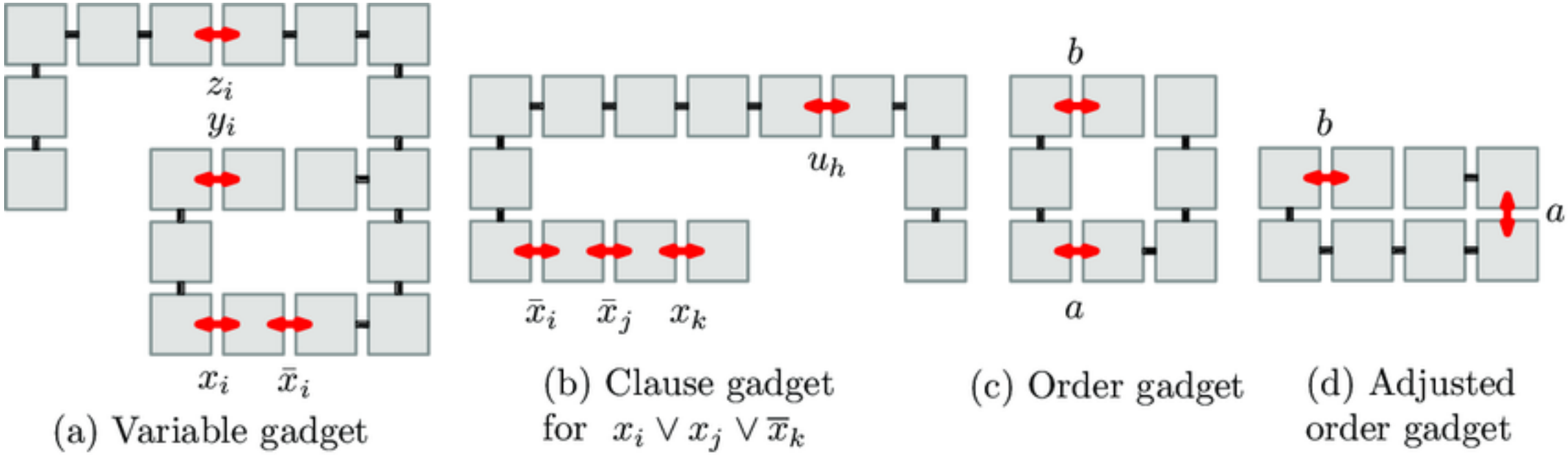
- *If* $a_1 = a_2$, then that's 1 match, and any other would imply $(a_1 \cdot i + b_1) \equiv_5 (a_1 \cdot i + b_2)$, and thus $b_1 = b_2$.
- Otherwise, a match means $(a_1 \cdot i + b_1) \equiv_5 (a_2 \cdot i + b_2)$, and thus $(a_1 - a_2) \cdot i \equiv_5 (b_2 - b_1)$ which has unique solution $i = (a_1 - a_2)^{-1} (b_2 - b_1)$ since 5 is prime and $(a_1 - a_2) \neq 0$.

NP-hardness Gadgets



Vertex partitions of (C_3, C_4, C_6)-free planar graphs *(Dross and Ochem 2017)*

NP-hardness Gadgets



(a) Variable gadget

(b) Clause gadget for $x_i \vee x_j \vee \overline{x}_k$

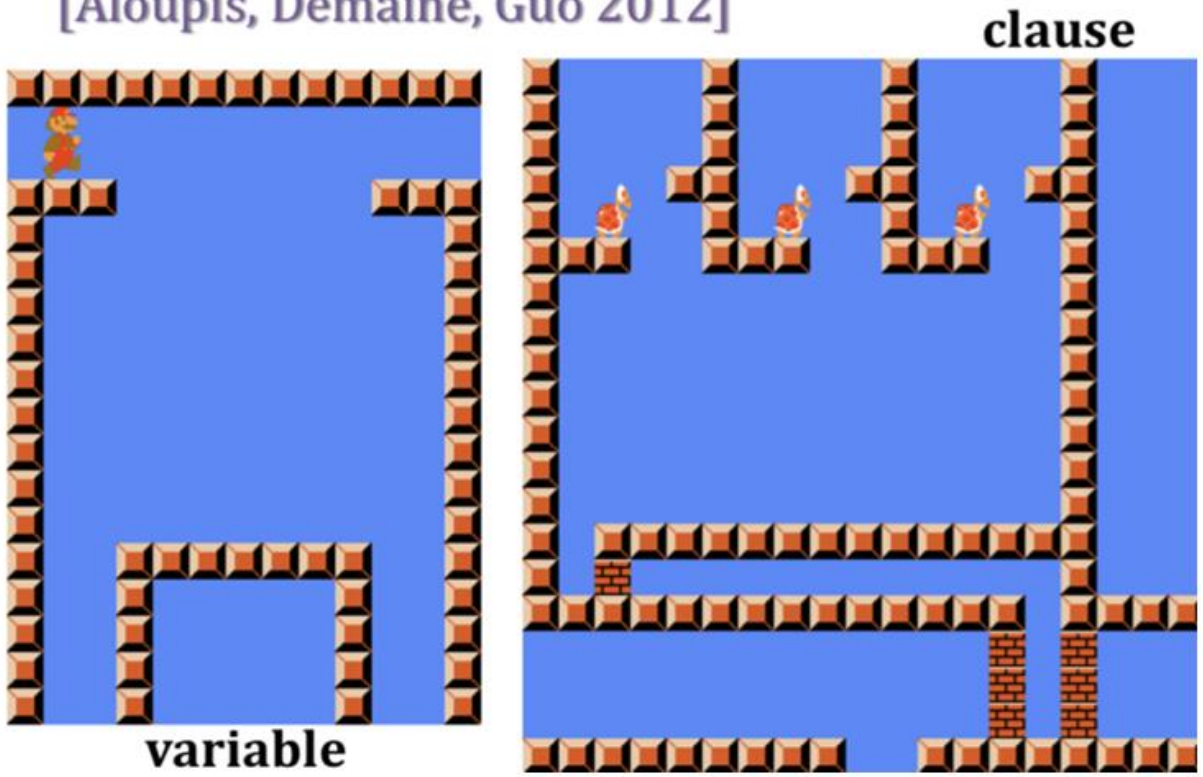
(c) Order gadget

(d) Adjusted order gadget

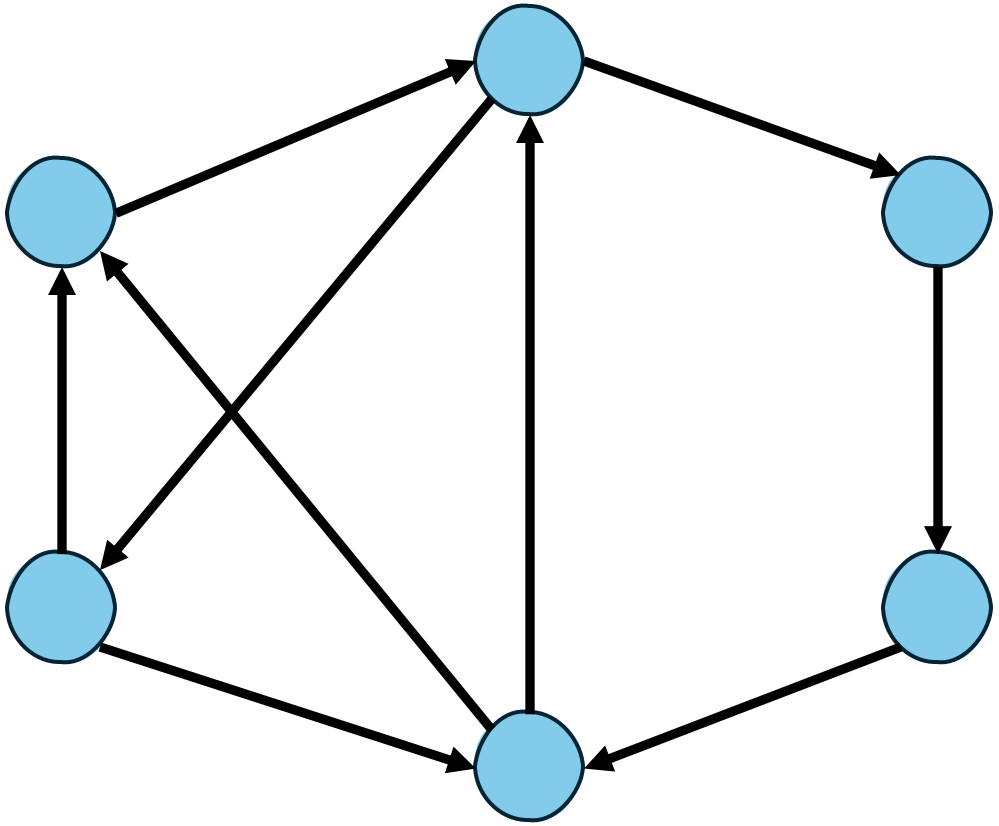
Collision Detection for Modular Robots -- it is easy to cause collisions and hard to avoid them
(Gupta, Kreveld, Michail, Padalkin 2013)

NP-hardness Gadgets

- **Super Mario Bros. is NP-Hard**
  [Aloupis, Demaine, Guo 2012]

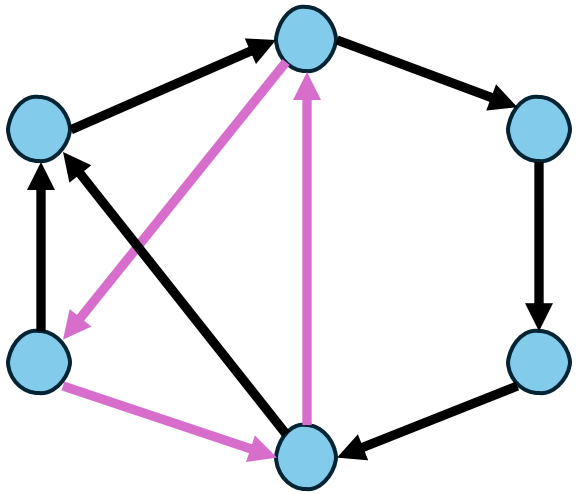


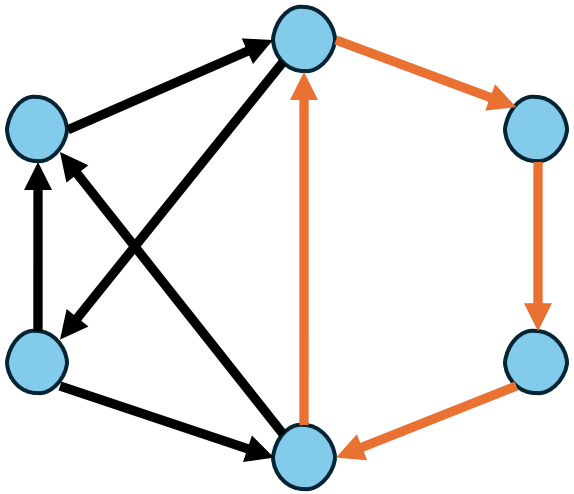
Feedback Vertex Set (NP-hard)

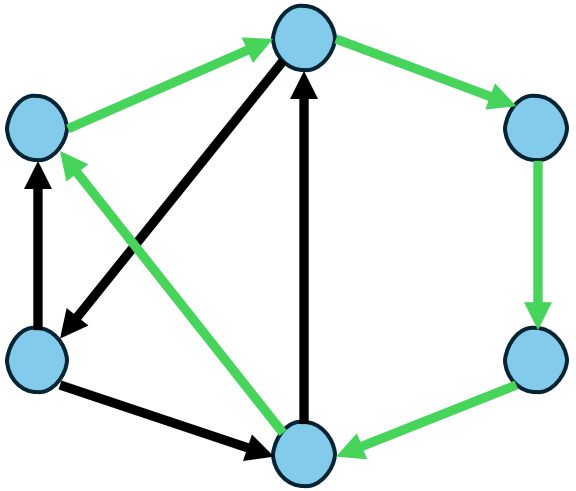


Remove minimum number of vertices so that the resulting graph has no directed cycles

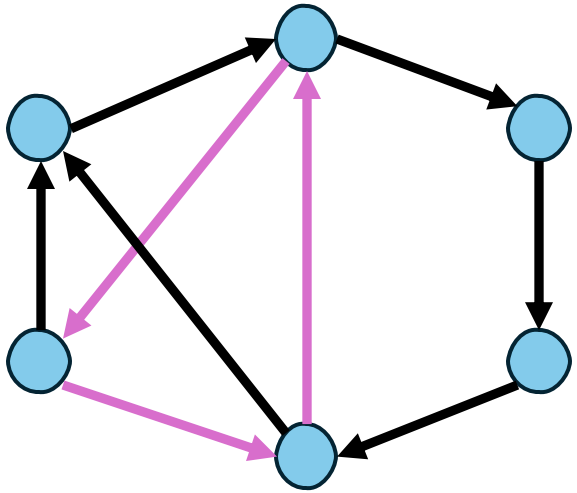
Feedback Vertex Set (NP-hard)

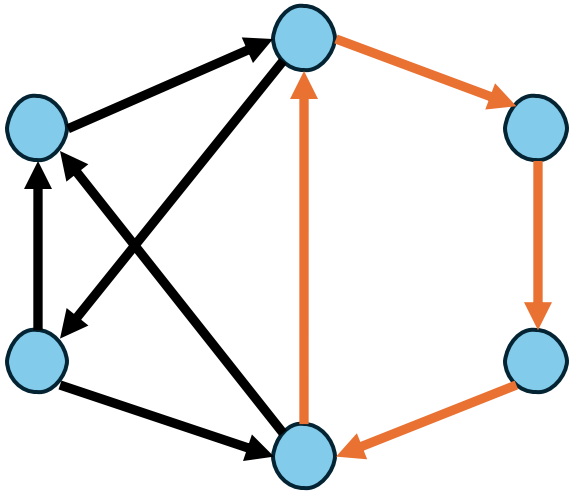


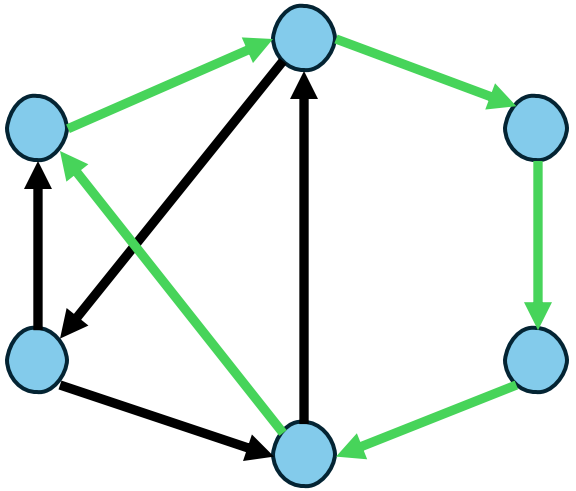


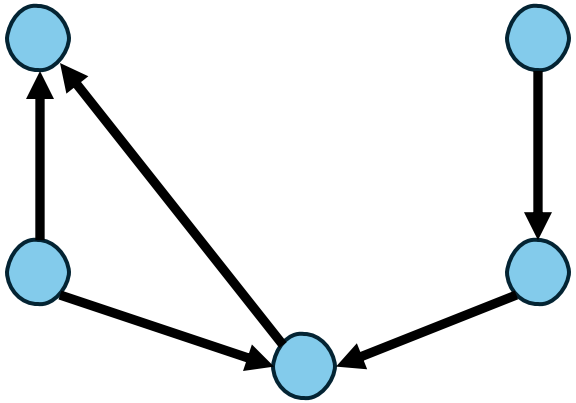


Feedback Vertex Set (NP-hard)

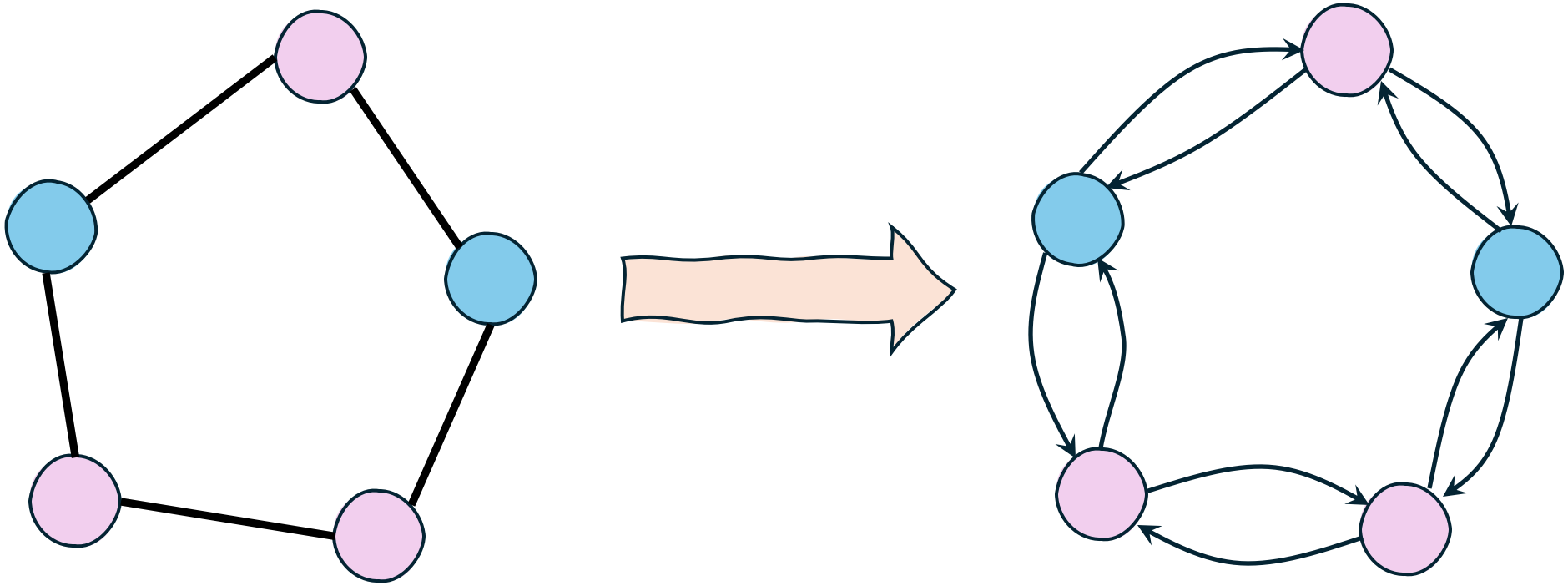




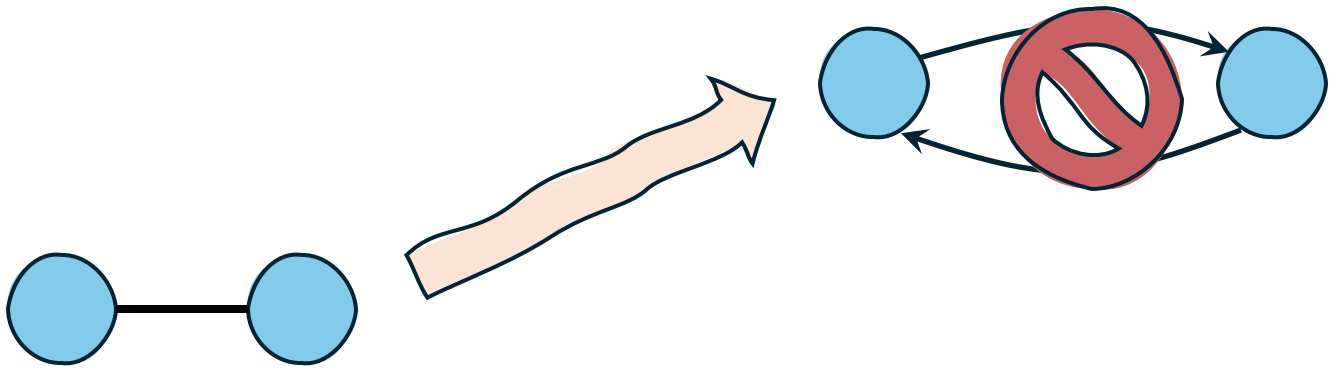




Vertex Cover reduces to FVS

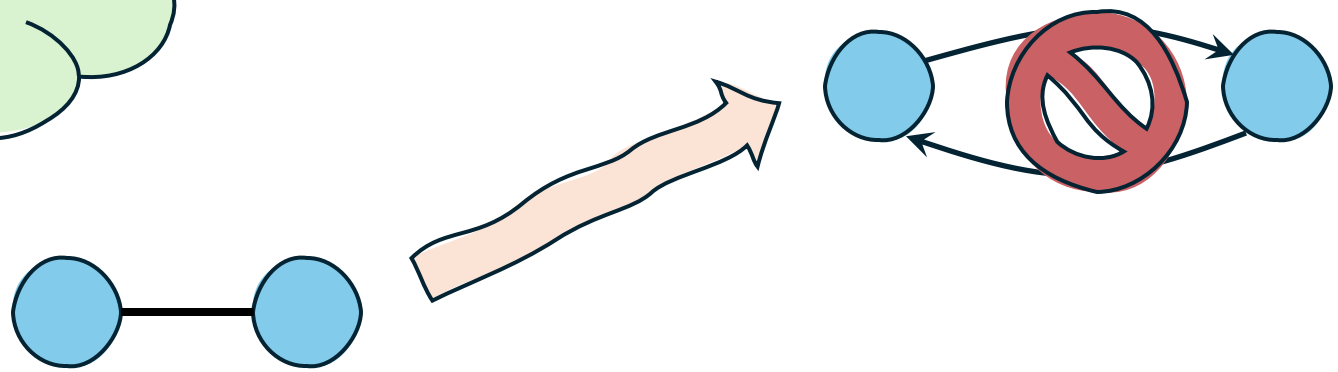


I needed to prove that FVS is NP-hard On 3-out-regular graphs without $\mathbf{C_2}$



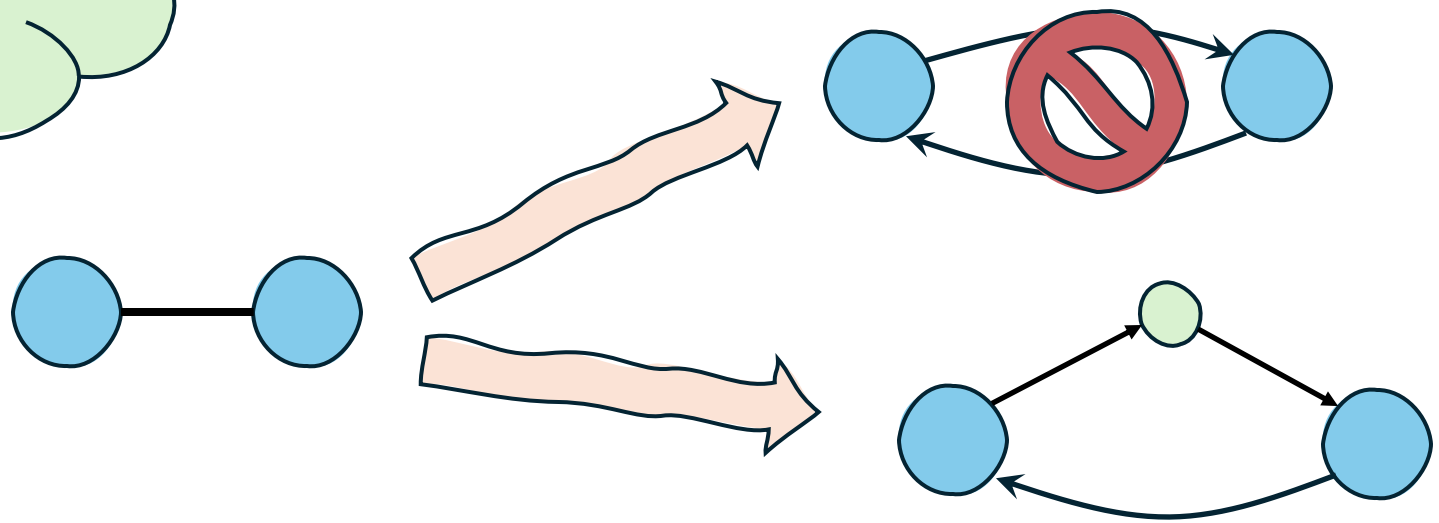
I needed to prove that FVS is NP-hard On 3-out-regular graphs without $\mathbf{C_2}$

Note: Vertex-cover is NP-hard on 3-regular graphs



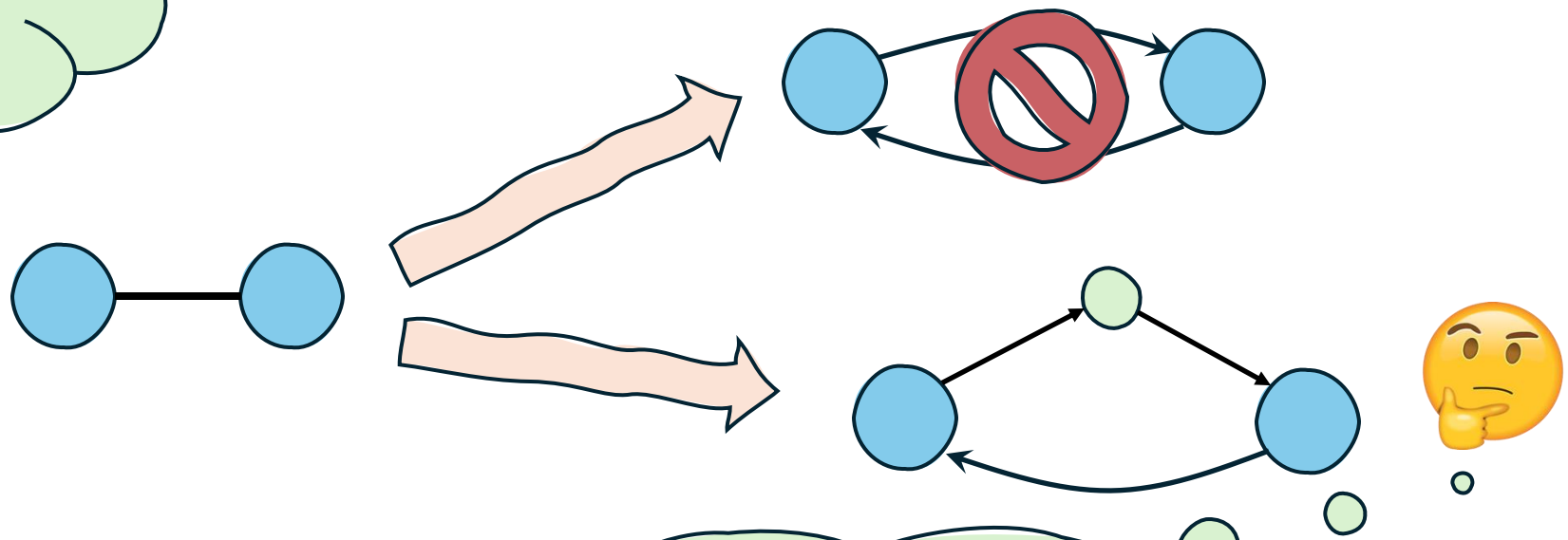
I needed to prove that FVS is NP-hard On 3-out-regular graphs without $\mathbf{C_2}$

Note: Vertex-cover is NP-hard on 3-regular graphs



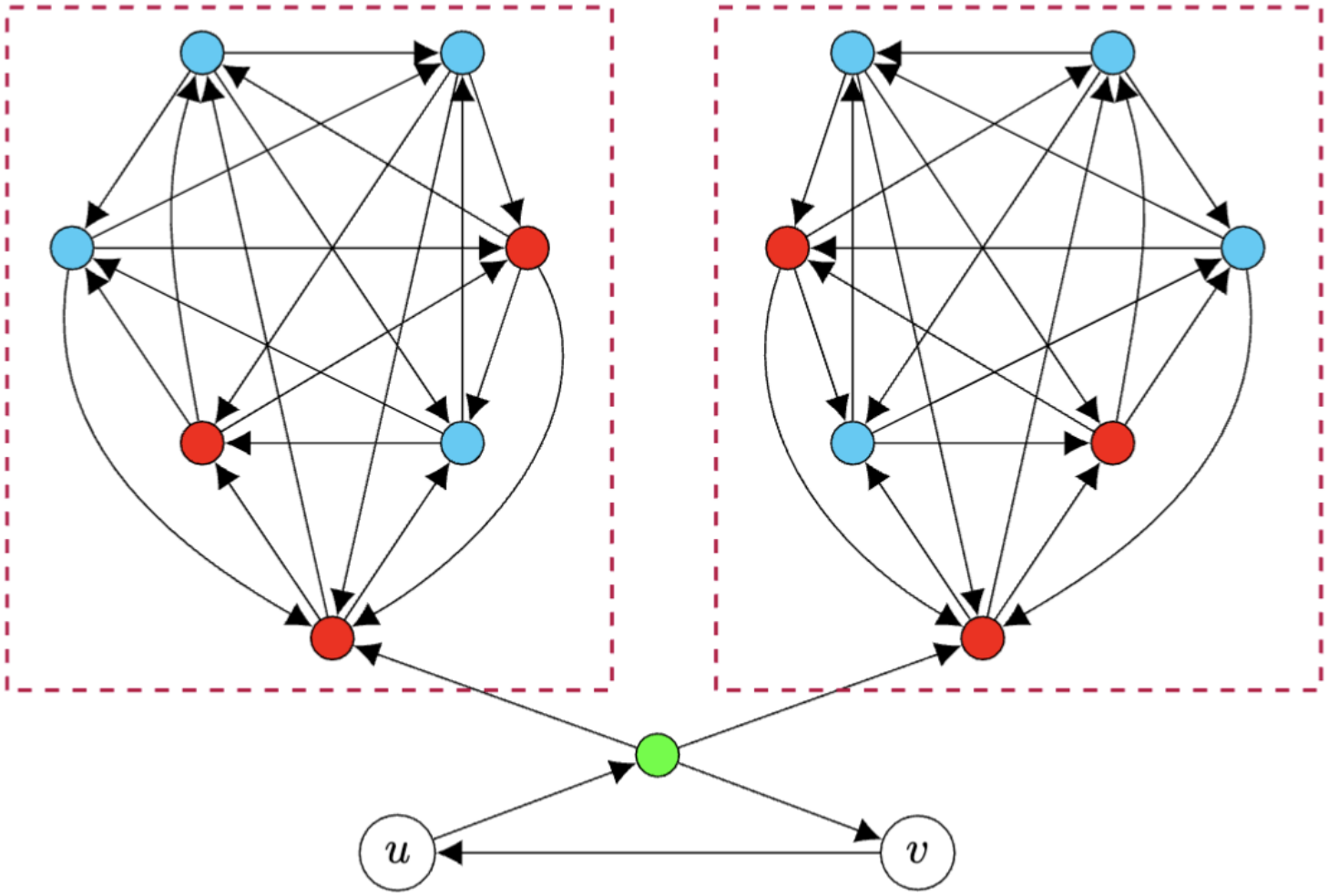
I needed to prove that FVS is NP-hard On 3-out-regular graphs without C_2

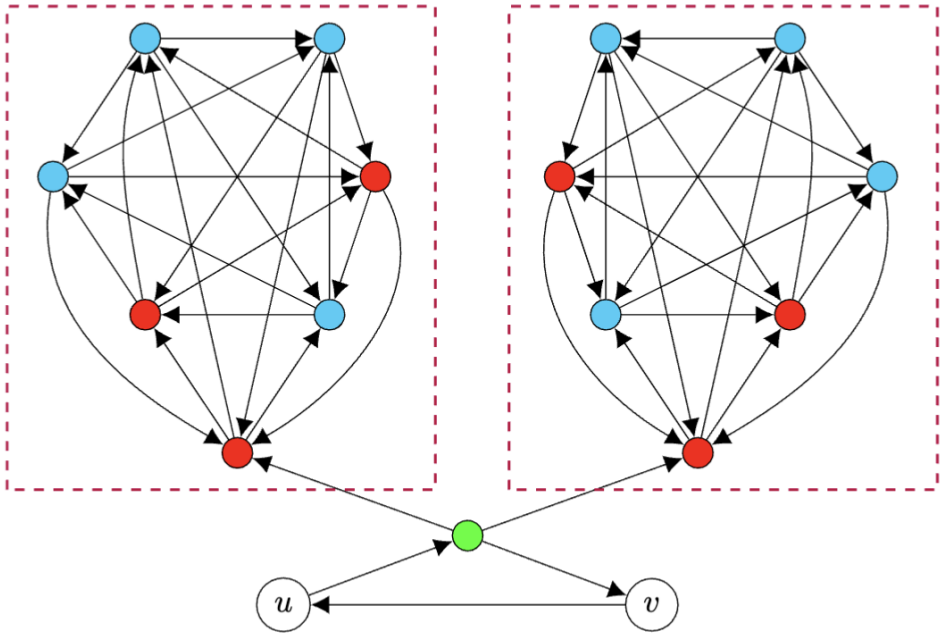
Note: Vertex-cover is NP-hard on 3-regular graphs



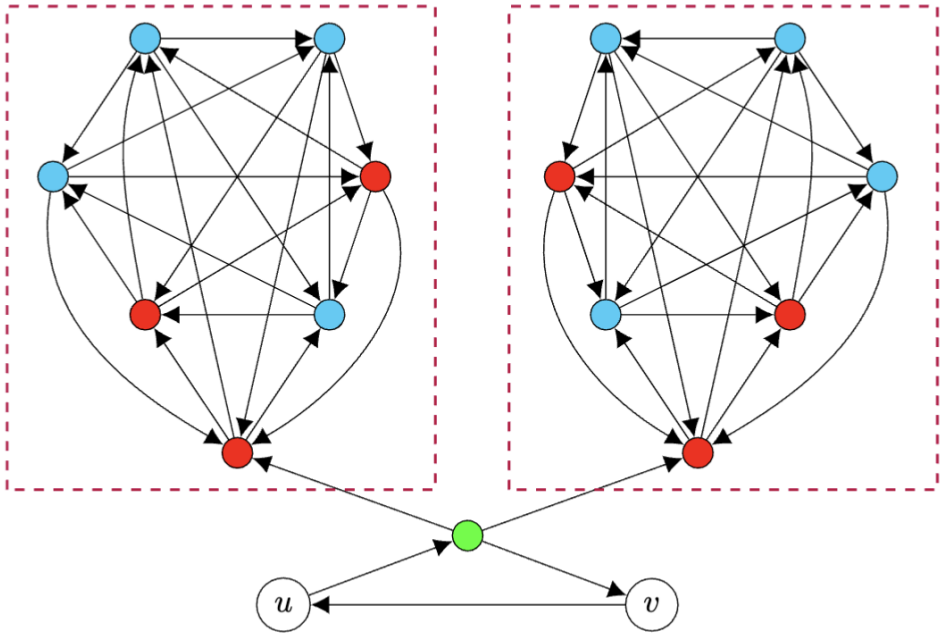
Can we attach to the green vertex a 3-out-regular C_2-free graph (except that green vertex has outdegree 2)?

I needed to prove that FVS is NP-hard On 3-out-regular graphs without $\mathbf{C_2}$





The solutions I was getting were orientations of the complete graph…



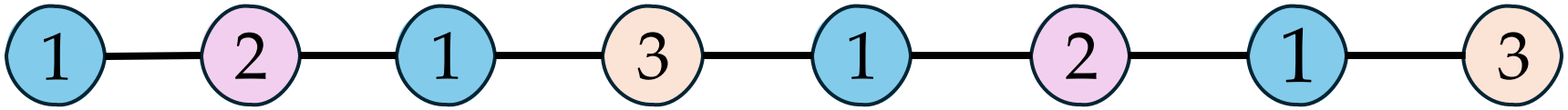
The solutions I was getting were orientations of the complete graph…

General construction for *k*-out-regular C_2 –free:
Take complete graph on ***2k+1*** vertices.
As every vertex has degree ***2k,*** there exists an Eulerian circuit w (visits each edge exactly once).

Now orient each edge according to the direction in which w takes it.

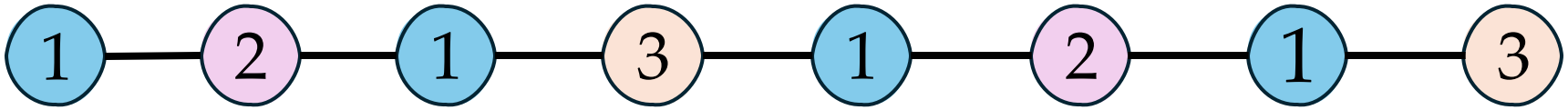
4-Packing Colorings

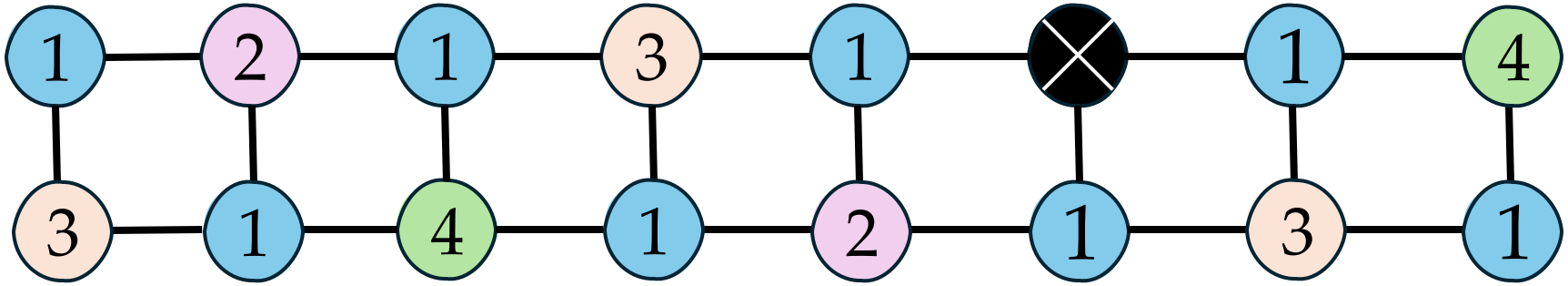
Color the vertices of a graph with {1, 2, 3, 4} so every pair of vertices receiving color *i* are at distance at least *i+1*



4-Packing Colorings

Color the vertices of a graph with {1, 2, 3, 4} so every pair of vertices receiving color *i* are at distance at least *i+1*





4-Packing Colorings

Color the vertices of a graph with {1, 2, 3, 4} so every pair of vertices receiving color *i* are at distance at least *i+1*

Reduction from Not-All-Equal-3SAT

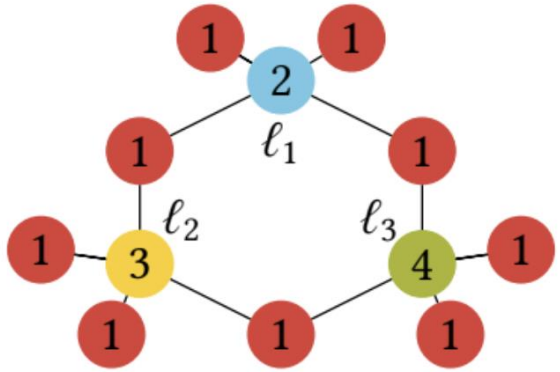


Fig. 7. Illustration of the coloring for a clause gadget.

4-Packing Colorings

Color the vertices of a graph with {1, 2, 3, 4} so every pair of vertices receiving color *i* are at distance at least *i+1*

Reduction from Not-All-Equal-3SAT

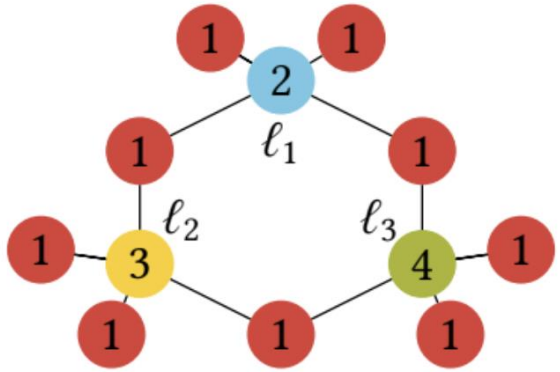


Fig. 7. Illustration of the coloring for a clause gadget.

A general issue with gadgets like these is creating "consistency" gadgets that guarantee that all occurrences of a variable get the same color

4-Packing Colorings

Color the vertices of a graph with {1, 2, 3, 4} so every pair of vertices receiving color i are at distance at least $i+1$

Reduction from Not-All-Equal-3SAT

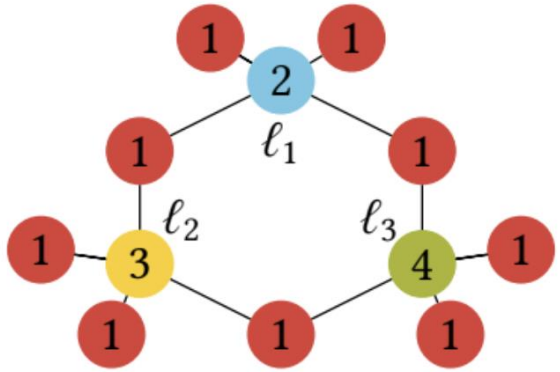


Fig. 7. Illustration of the coloring for a clause gadget.

SAT-constructed Gadget

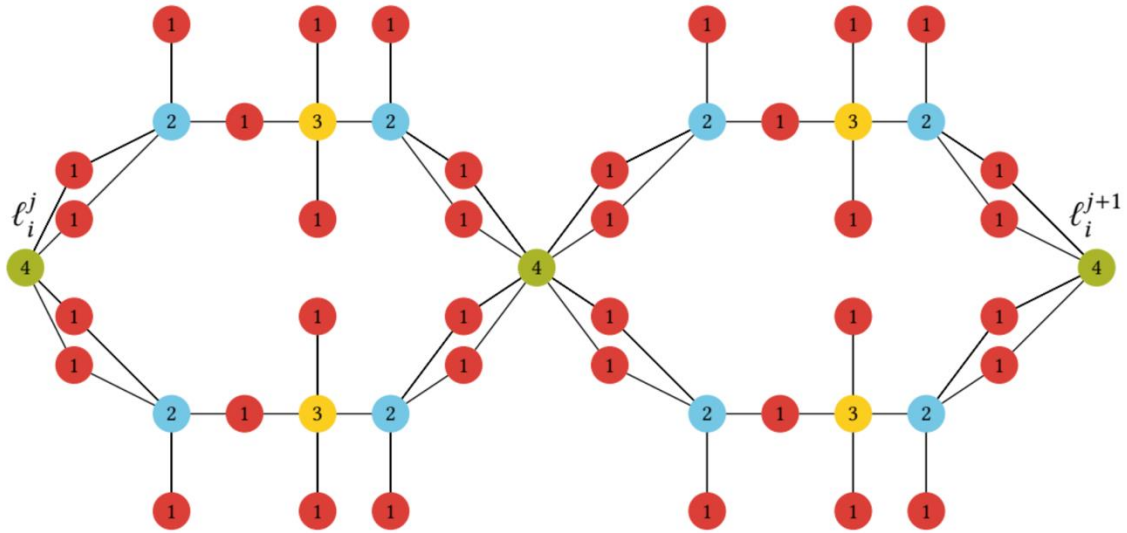


Fig. 8. Coloring of a propagation gadget between positive colors.

How can we trust SAT solvers?

When the solver outputs a positive solution, we can just check in linear time that all constraints are satisfied

$\rightarrow F_1 \rightarrow F_2 \ldots \rightarrow \{\}$ (the empty clause)

How can we trust SAT solvers?

When the solver outputs a positive solution, we can just check in linear time that all constraints are satisfied

But what if there's no solution?

$\rightarrow F_1 \rightarrow F_2 \ldots \rightarrow \{\}$ (the empty clause)

How can we trust SAT solvers?

When the solver outputs a positive solution, we can just check in linear time that all constraints are satisfied

But what if there's no solution?

- Then the solvers can emit a proof in a standardized format (e.g., DRAT)

$\rightarrow F_1 \rightarrow F_2 \ldots \rightarrow \{\}$ (the empty clause)

How can we trust SAT solvers?

When the solver outputs a positive solution, we can just check in linear time that all constraints are satisfied

But what if there's no solution?

- Then the solvers can emit a proof in a standardized format (e.g., DRAT)

- Each step of the proof is basically a transformation of the formula: $\rightarrow F_1 \rightarrow F_2 \ldots \rightarrow \{\}$ (the empty clause)

How can we trust SAT solvers?

When the solver outputs a positive solution, we can just check in linear time that all constraints are satisfied

But what if there's no solution?

- $F\ 1\ FF\ F\ 1\ 1\ F\ 1 \rightarrow F\ 2\ FF\ F\ 2\ 2\ F\ 2 \ldots \rightarrow \{\}$ (the empty clause)
- Then the solvers can emit a proof in a standardized format (e.g., DRAT)

- Each step of the proof is basically a transformation of the formula: $F \rightarrow F_1 \rightarrow F_2 \ldots \rightarrow \{\}$ (the empty clause)

How can we trust SAT solvers?

When the solver outputs a positive solution, we can just check in linear time that all constraints are satisfied

But what if there's no solution?

- $F\ 1\ FF\ F\ 1\ 1\ F\ 1 \rightarrow F\ 2\ FF\ F\ 2\ 2\ F\ 2 \ldots \rightarrow \{\}$ (the empty clause)
- Then the solvers can emit a proof in a standardized format (e.g., DRAT)

- Each step of the proof is basically a transformation of the formula: $F \rightarrow F_1 \rightarrow F_2 \ldots \rightarrow \{\}$ (the empty clause)
- There are verified checkers for these proof formats

How can we trust SAT solvers?

When the solver outputs a positive solution, we can just check in linear time that all constraints are satisfied

But what if there's no solution?

- $F\ 1\ FF\ F\ 1\ 1\ F\ 1 \rightarrow F\ 2\ FF\ F\ 2\ 2\ F\ 2 \ldots \rightarrow \{\}$ (the empty clause)
- Then the solvers can emit a proof in a standardized format (e.g., DRAT)

- Each step of the proof is basically a transformation of the formula: $F \rightarrow F_1 \rightarrow F_2 \ldots \rightarrow \{\}$ (the empty clause)
- There are verified checkers for these proof formats

Lam's problem:

"is there a projective plane of order 10"?

Lam's problem:
"**is there a projective plane of order 10**"?

- Proven **NO** by custom computer programs in 1989

Lam's problem:

"is there a projective plane of order 10"?

- Proven **NO** by custom computer programs in 1989
- These custom algorithms are very hard to verify!

Lam's problem:

"is there a projective plane of order 10"?

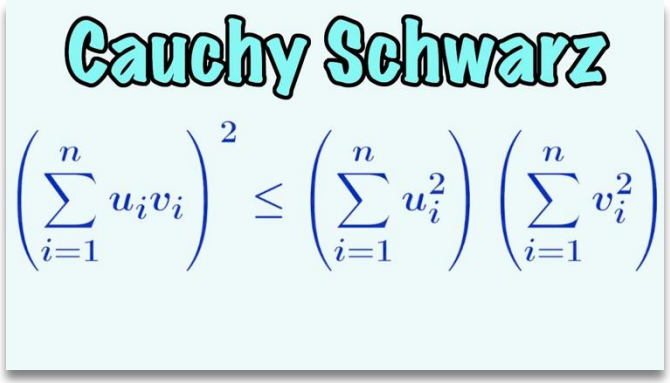
- Proven **NO** by custom computer programs in 1989
- These custom algorithms are very hard to verify!
- SAT refutation in 2020 with public certificates.

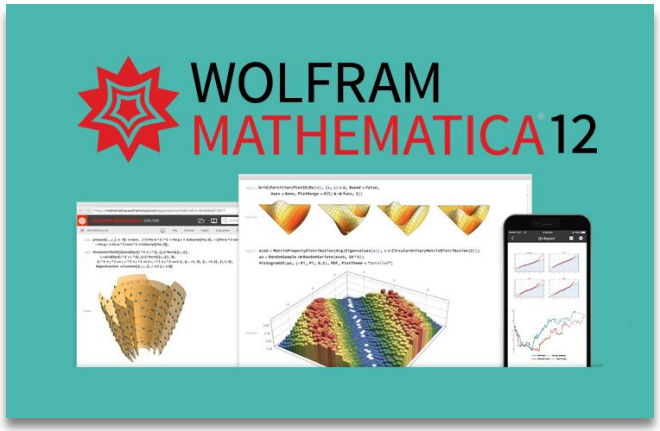
Lam's problem:
"**is there a projective plane of order 10**"?

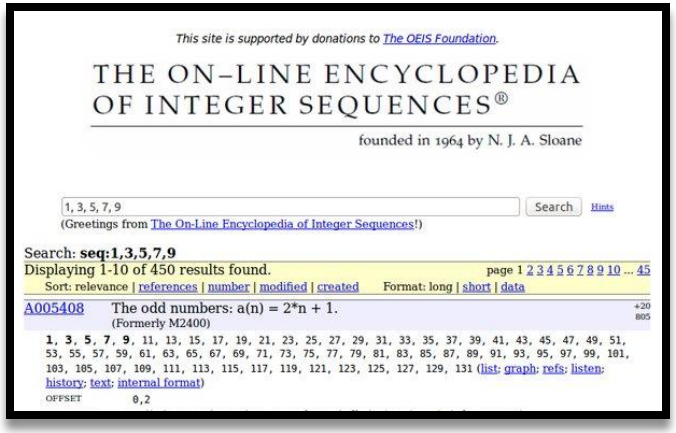
- Proven **NO** by custom computer programs in 1989
- These custom algorithms are very hard to verify!
- SAT refutation in 2020 with public certificates.
- Not verified in Lean yet, but much easier than custom code

(Bright, Cheung, Stevens, Kotsireas, Ganesh; 2020)

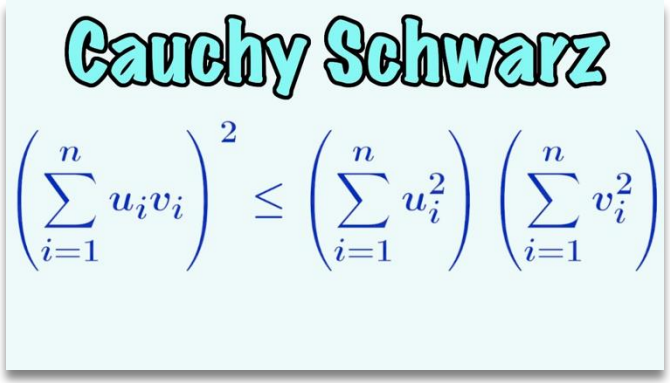
Conclusion: A mathematician's toolbox

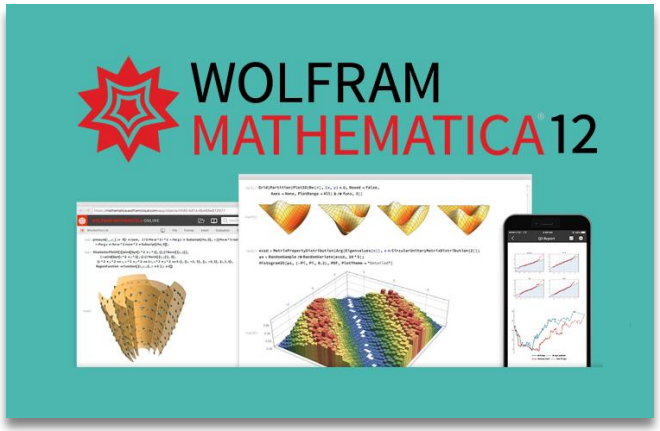


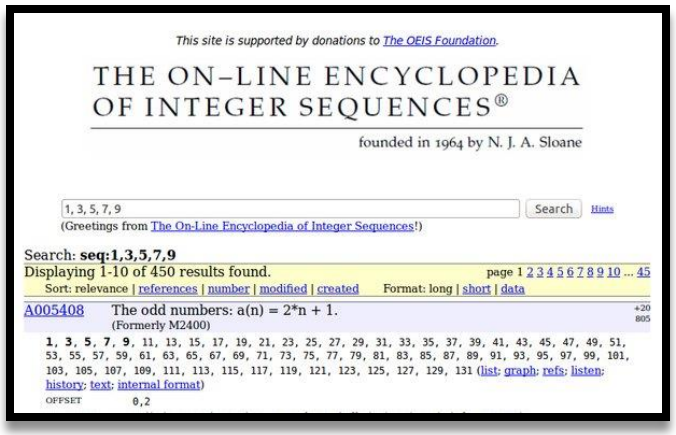


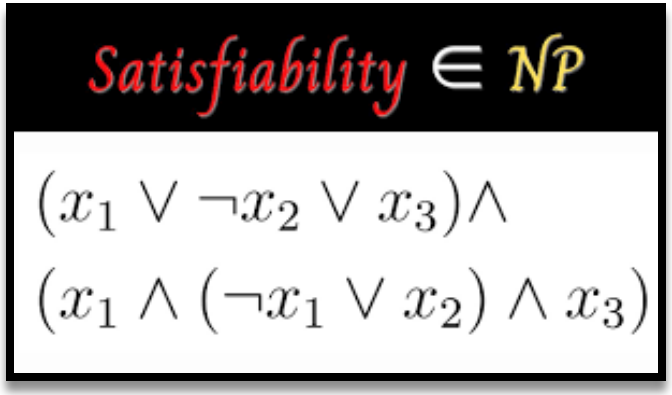


Conclusion: A mathematician's toolbox









SAT Solvers?

Computer-Assisted Intuition: SAT Solvers in Mathematical Discovery

Bernardo Subercaseaux

Carnegie Mellon University