

KNAPSACK

Given positive integers v_i and w_i for $i = 1, 2, \dots, n$.
and an positive integers K and W .

Does there exist a subset S of $\{1, 2, \dots, n\}$ such that:

$$\sum_{i \in S} w_i \leq W$$

and $\sum_{i \in S} v_i \geq K$

A special case: SUBSET SUM

Given positive integers v_i for $i = 1, 2, \dots, n$.

and an integer K .

Does there exist a subset S of $\{1, 2, \dots, n\}$ such that:

$$\sum_{i \in S} v_i = K$$

2120				146		2688	1344
2065		2082	1408	2081		769	14
2336	265	1288	258	56		3073	
1160	592	256		3328		4095	

EXACT COVER BY 3-SETS can be reduced to SUBSET SUM

Knapsack problem

- $L_{\text{Knapsack}}^{\text{binary}}$ is **NP-hard**.
- Knapsack can be solved in time $O(n W)$
- (on a random access machine).
- Therefore $L_{\text{Knapsack}}^{\text{unary}}$ is in **P**: “*Knapsack has a pseudopolynomial algorithm*”
- Unless **P=NP**, any reduction of an NP-hard problem to KNAPSACK has to involve “large numbers”.

Pseudopolynomial algorithms

- Let a problem Q involving integers be given.
- If L_Q^{unary} is in \mathbf{P} , we say that Q has a *pseudopolynomial* algorithm.
- If L_Q^{unary} is \mathbf{NP} -hard, we say that Q is *strongly NP-hard*.
- If a strongly \mathbf{NP} -hard problem has a pseudopolynomial algorithm, then $\mathbf{P}=\mathbf{NP}$.

ILP

- The reduction of 3SAT to ILP only generates coefficients in $\{0, 1\}$.
- $L_{\text{ILP}}^{\text{unary}}$ is **NP**-hard: “*Integer linear programming is strongly **NP**-hard*”.

Bin packing

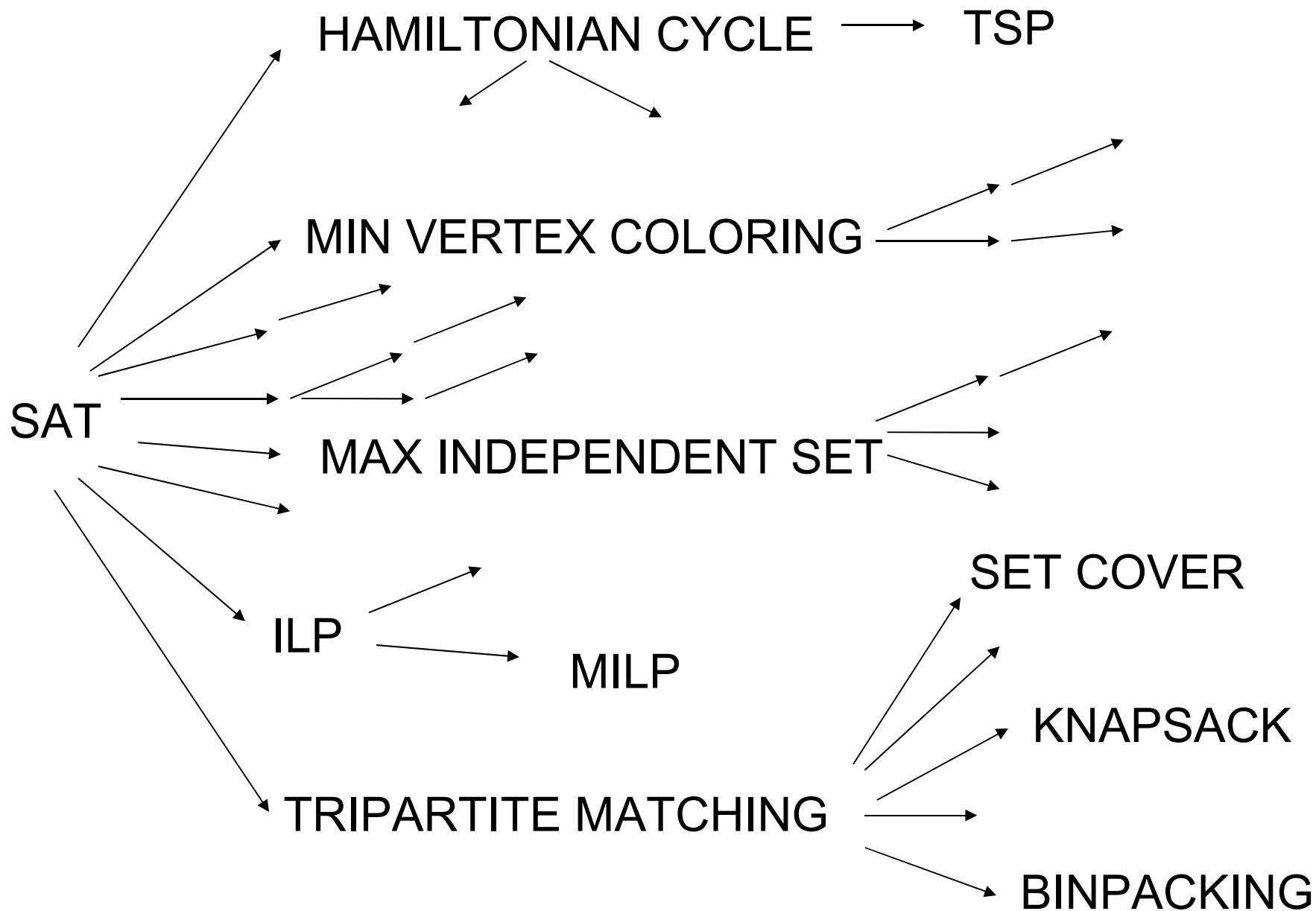
Given n positive integers a_1, a_2, \dots, a_n (items),
a positive integer B (number of bins) and
a positive integer C (the capacity of a bin).

Can the items fit into the bins?

That is, can the items be partitioned into B subsets such that the sum of the items in each of the subset is at most C ?

TRIPARTITE MATCHING can be reduced to BIN PACKING with all integers in the output being $O(n^4)$.

BIN PACKING is strongly **NP**-hard: A pseudopolynomial algorithm would imply **P=NP**.



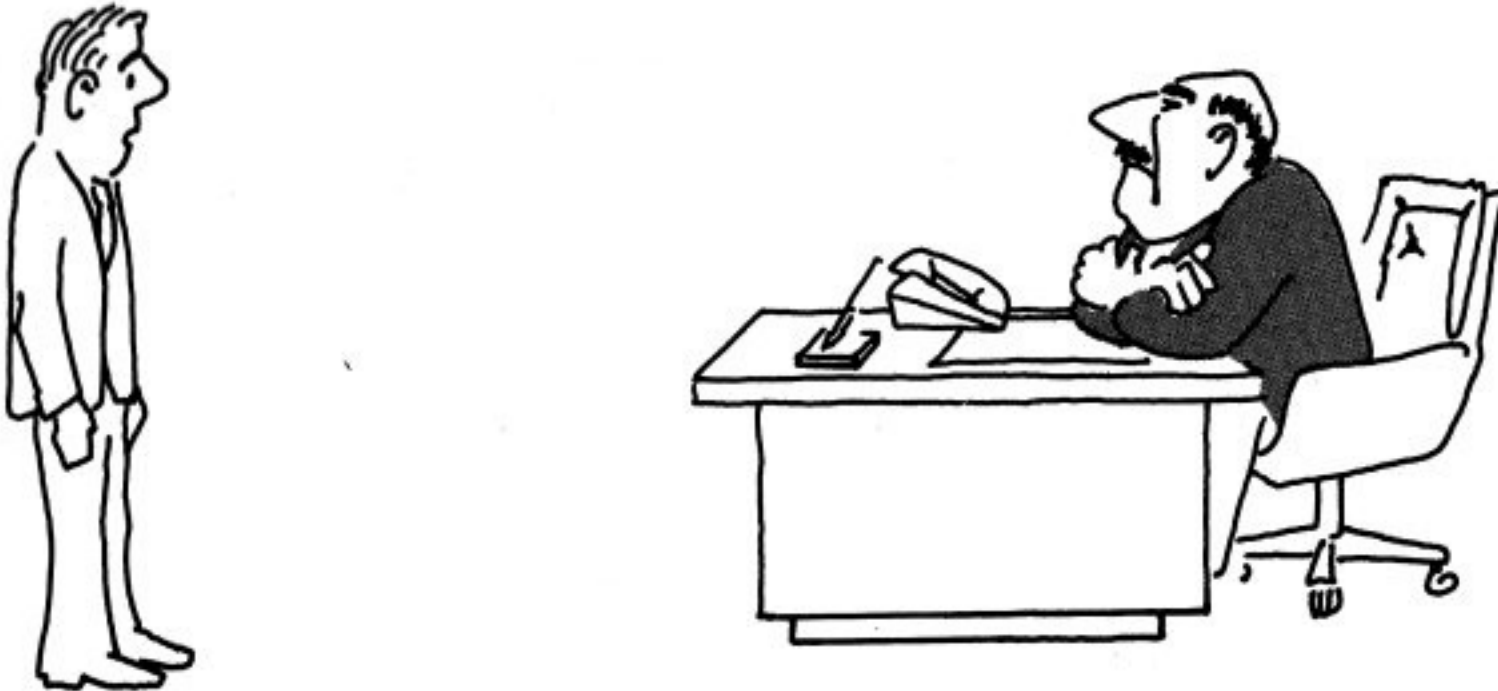
How to establish a problem to be NP-hard

- Reduce from known problem
(very often special case in disguise)
- Modify known reduction
- Google is your friend!!

Life as an algorithms designer

- Suppose your boss gives you the job of designing an efficient algorithm to solve a specific computational problem.
- For example, given a set of specifications for a task the company considers to bid for, can the company meet these requirements?
- What if, using all your skills learned from algorithms courses you can't come up with an efficient algorithm?

Wrong way to approach boss



“I can’t find an efficient algorithm, I guess I’m just too dumb.”

Correct way to approach boss



“I can’t find an efficient algorithm, because no such algorithm is possible!”

Unfortunately, often proving intractability is also not possible

Best possible way to approach boss



“I can’t find an efficient algorithm, but neither can all these famous people.”

How (not) to use the theory of **NP**-completeness

Common(!) Misuse!

Arthur: I have a 150 cities on a map. I want to find the shortest cycle visiting all of them.

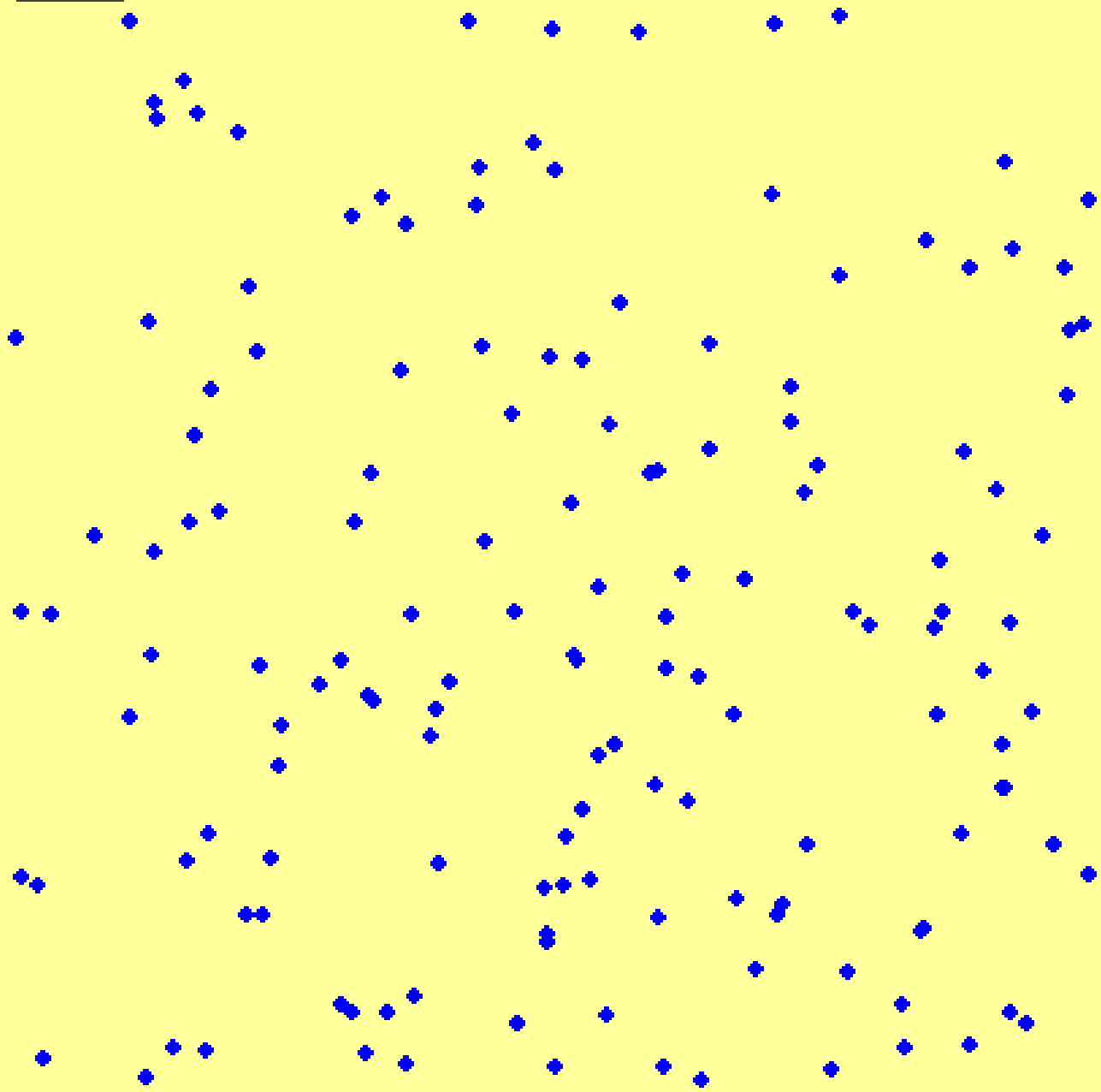


~~**Bill:** Wait a sec.. That's the TSP-problem! It's **NP**-hard! You'll never be able to do it!~~



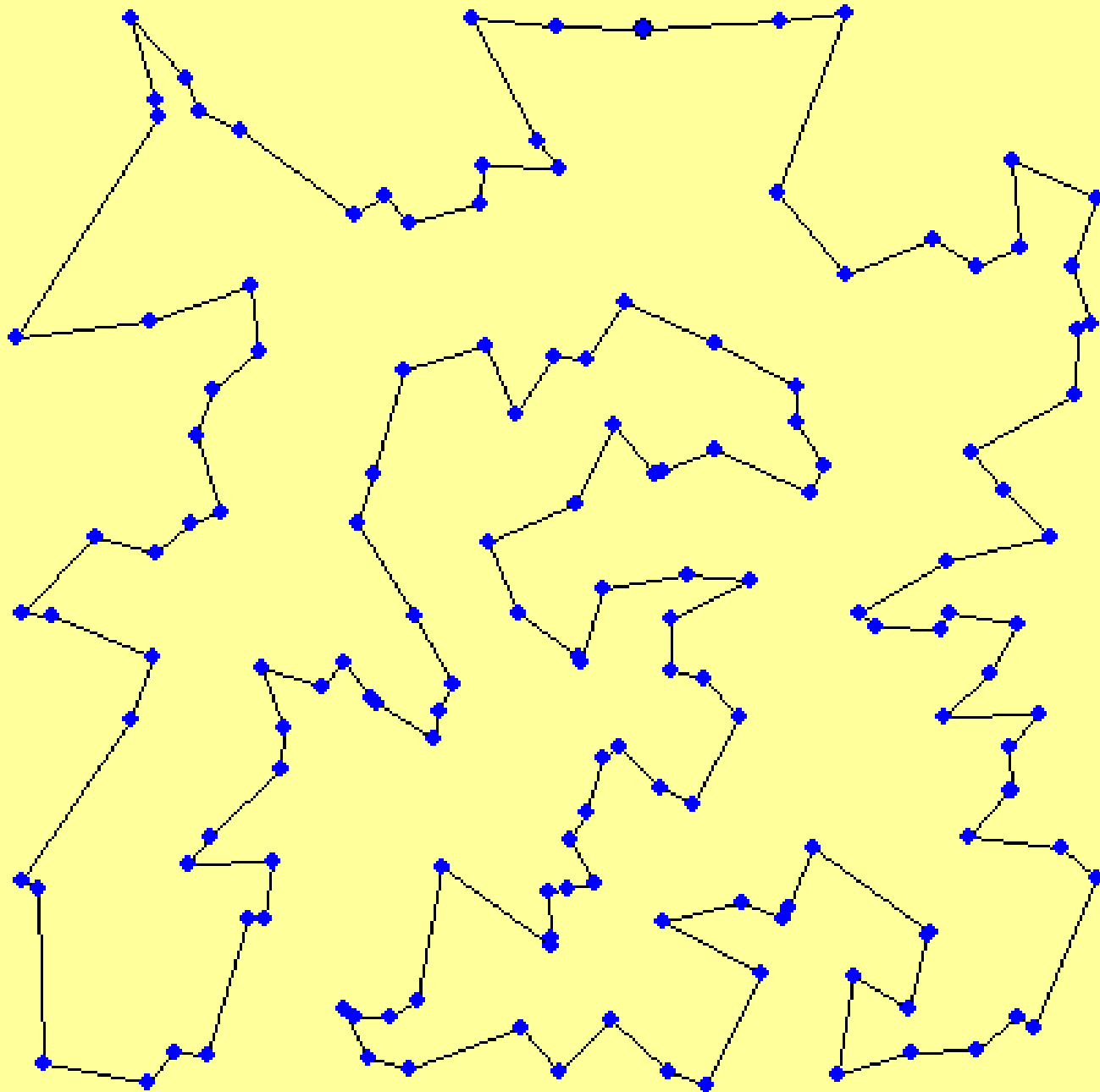
break

150 random cities seed=-1218878044



demos

150 random cities seed=-1218878044



What went wrong?



- Bill **only** knows that no algorithm solving TSP has worst case polynomial complexity (assuming **P** is not **NP**).
- This says **absolutely nothing** about how hard it is to solve instances of size 150. Or even instances of size 10000000.
- Even if it did, it would say **absolutely nothing** about the particular instances of size 150 that Arthur wants to solve!

The Eternity Puzzle



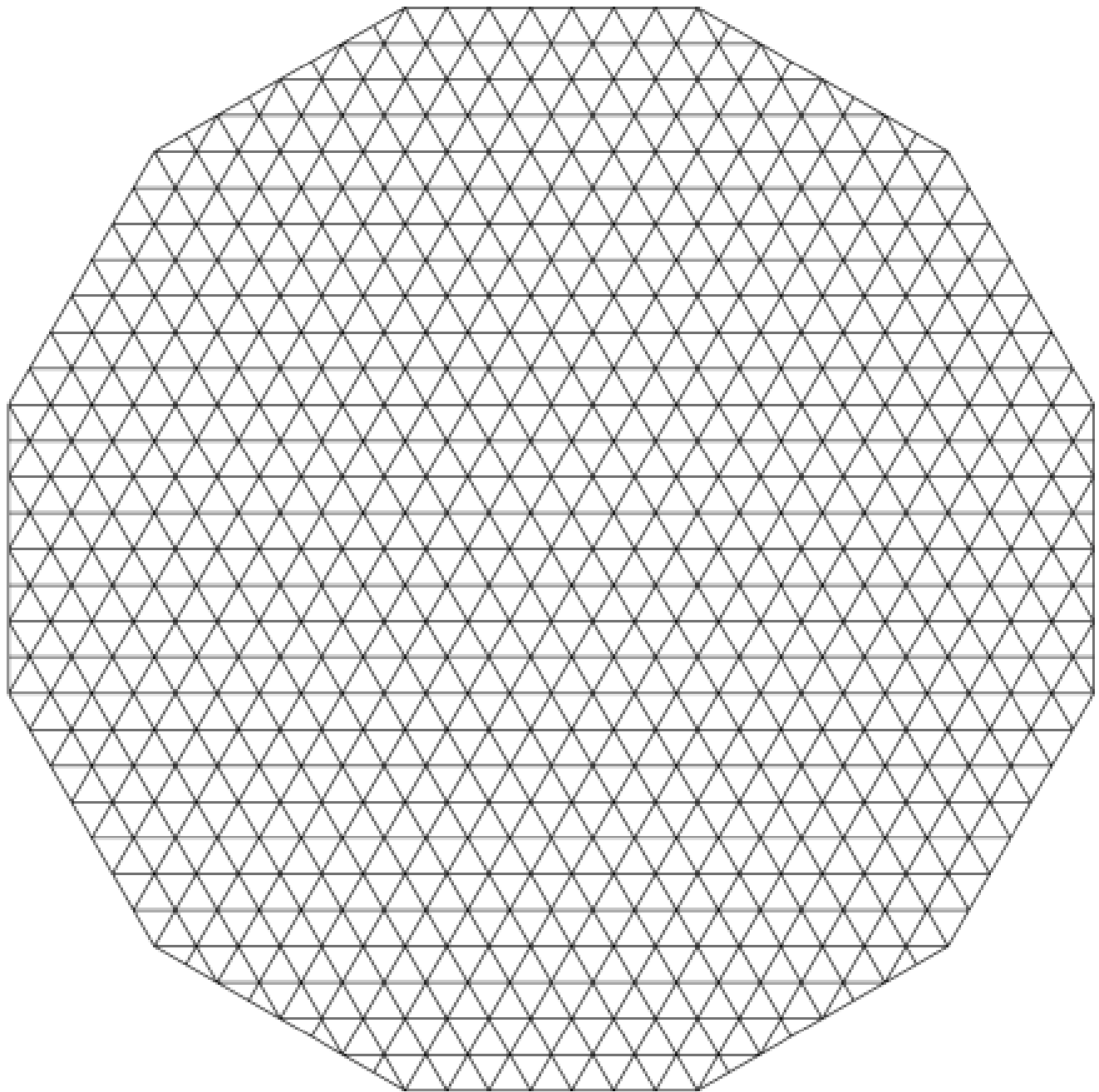
- **June 1999:** Christopher Monckton releases the **Eternity Puzzle**.
- The first person to solve the puzzle before 30 September 2003 wins £ 1 Million.

Pieces in original order



Sorry, for copyright reasons the full Eternity piece set cannot be displayed here.





Big Prize, but...

He reckoned that by the time the world's puzzle-solving community had unlocked Eternity's secret (at £29.99 a go), he would have earned substantially more than £1 million in sales.

Why?

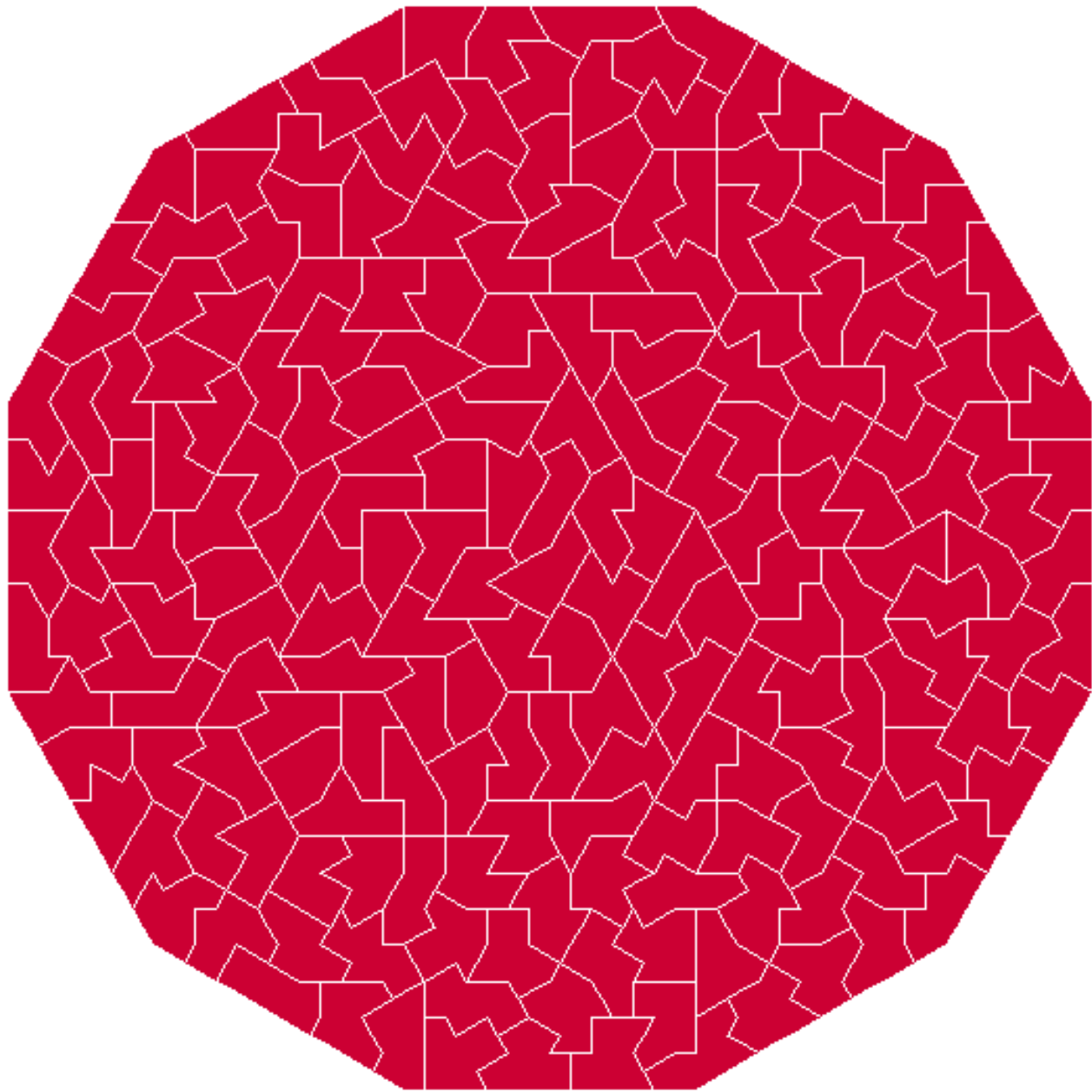
Eternity seems hard....

- Solving jigsaw puzzles is **NP**-complete.
- Monckton:

”I worked out that there were about as many possible combinations of these 209 pieces as there were particles in the known universe, and that is a figure which is at least 621 digits long. As for a computer, I found it could solve a 12-piece puzzle in one-hundredth of a second, but from that point on the progress was exponential - which meant that even with quite sophisticated software, it would take one million billion years to solve a 209-piece puzzle.”

... but was easier than expected

- A solution to Eternity was found on 15th May 2000 by Alex Selby and Oliver Riordan.
- Another solution was found independently by Günter Stertenbrink on 1st July 2000.



Method used

- Branch and Bound tailored to the particular instance.

June 2001: Monckton sells his home Crimonmogate, a nineteenth-century mansion in Crimond, Aberdeenshire, for an estimated £1.2m.



Poor Bill ...

So, I took this course and
had to learn this theory that
can only be misused!?



Excellent use!

Arthur: Here is my suggestion for solving my problem. I'm not sure how efficient it is, but at least it's correct. I'll write down this linear program and...



Bill: Wait a sec..This method would work in general and create a polynomial sized LP. It cannot be correct unless **$P=NP$** . *I don't even have to try and understand it!*



Excellent use!

Arthur: Okay, here is another suggestion. This clearly may take exponential time in the worst case and now I'm positive it's correct. It's a local search approach where you (*bla bla*)



Bill: Wait a sec..This local search approach has polynomial sized neighborhoods. It cannot solve an **NP**-hard optimization Problem unless **NP=coNP**. ***And I don't even have to try and understand it!***



How to solve NP-hard problems

- **Irony of NP-hardness**: It is often *easier* to find the *best* way of solving an NP-hard problem than the *best* way of solving a problem in P, because *you know which approaches (not) to try*.
- Algorithmic patterns for NP-hard problems: *branch-and-bound, branch-and-cut, branch-and-reduce*.
- Alternative approach if exact solutions are too hard to obtain: Solve the problem *approximately*.

