Computational Complexity; slides 9, HT 2019
NP search problems, and total search problems

Prof. Paul W. Goldberg (Dept. of Computer Science,
University of Oxford)

HT 2019

Examples of total search problems in NP

- FACTORING
- NASH: the problem of computing a Nash equilibrium of a game (comes in many versions depending on the structure of the game)
- PIGEONHOLE CIRCUIT:
  Input: a boolean circuit with n input gates and n output gates
  Output: either input vector x mapping to $\mathbf{0}$ or vectors x, x' mapping to the same output
- NECKLACE SPLITTING
- SECOND HAMILTONIAN CYCLE (in 3-regular graph)
- HAM SANDWICH: search for ham sandwich cut
- Search for *local optima* in settings with neighbourhood structure

The above *seem* to be hard. (of course, many search probs are in P, e.g. input a list L of numbers, output L in increasing order)

Search problems as poly-time checkable relations

NP search problem is modelled as a relation $R(\cdot,\cdot)$ where

- $R(x,y)$ is checkable in time polynomial in $|x|$, $|y|$
- input x, find y with $R(x,y)$ (y as certificate)
- *total* search problem: $\forall x \exists y \quad (|y| = poly(|x|), R(x,y))$

SAT: x is boolean formula, y is satisfying bit vector.
Decision version of SAT is polynomial-time equivalent to search for y.

FACTORING: input (the "x" in $R(x,y)$) is number N, output (the "y") is prime factorisation of N. No decision problem!

NECKLACE SPLITTING (k thieves): input is string of n beads in c colours; output is a decomposition into $c(k-1)+1$ substrings and allocation of substrings to thieves such that they all get the same number of beads of each colour.
(assume the number of beads of each colour is a multiple of k)

contrast with "promise problems"

Reducibility among search problems

FP, FNP: search (or, function computation) problems where output of function is computable (resp., checkable) in poly time.

Any NP problem has FNP version "find a certificate".

Definition

Let R and S be search problems in FNP. We say that R (many-one) reduces to S, if there exist polynomial-time computable functions f, g such that

$$(f(x), y) \in S \implies (x, g(x, y)) \in R.$$

Observation: If S is polynomial-time solvable, then so is R. We say that two problems R and S are (polynomial-time) equivalent, if R reduces to S and S reduces to R.

Theorem: FSAT, the problem of finding a s.a. of a boolean formula, is FNP-complete.

Example

(This slide added subsequently to the lecture)
Consider 2 versions of FACTORING: one using base-10 numbers, and the other version using base-2 numbers. Intuitively, these two problems have the same difficulty: there is a fast algorithm to factor in base 2, if and only if there is a fast algorithm to factor in base 10.

In trying to make that intuition mathematically precise, we get the definition of the previous slide.

Some total search problems seem hard. (F)NP-hard?

Some total search problems seem hard. (F)NP-hard?

Theorem

There is an FNP-complete problem in TFNP if and only if NP=co-NP.

Proof: "if": if NP=co-NP, then any FNP-complete problem is in TFNP (which is F(NP∩co-NP)).
"only if": Suppose $X \in TFNP$ is FNP-complete, and R is the binary relation for X.
Consider problem FSAT (given formula φ, find a satisfying assignment.)
Any unsatisfiable φ has a certificate of unsatisfiability, namely the string y with $(f(\varphi), y) \in R$ and $g(y) =$"no" (or generally, anything other than a satisfying assignment).

N. Megiddo and C.H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, **81**(2) pp. 317–324 (1991).

So what can we about the hardness of FACTORING, and others?

FACTORING (for example) cannot be NP-hard unless
NP = co-NP.
Unlikely! So FACTORING is in strong sense "NP-intermediate".

[1] Try to describe "generic" problem/language X in NP∩co-NP as pair of NTMs that accept X and $\overline{X}$: what goes wrong?

So what can we about the hardness of FACTORING, and others?

FACTORING (for example) cannot be NP-hard unless NP = co-NP.
Unlikely! So FACTORING is in strong sense "NP-intermediate".

$\rightsquigarrow$ task of classifying "hard" NP total search problems.
FACTORING is a very important NP total search problem; others are interesting (and somewhat important)

OK can we have, say, FACTORING is TFNP-complete?

[1] Try to describe "generic" problem/language X in NP$\cap$co-NP as pair of NTMs that accept X and $\overline{X}$: what goes wrong?

So what can we about the hardness of FACTORING, and others?

FACTORING (for example) cannot be NP-hard unless NP = co-NP.
Unlikely! So FACTORING is in strong sense "NP-intermediate".

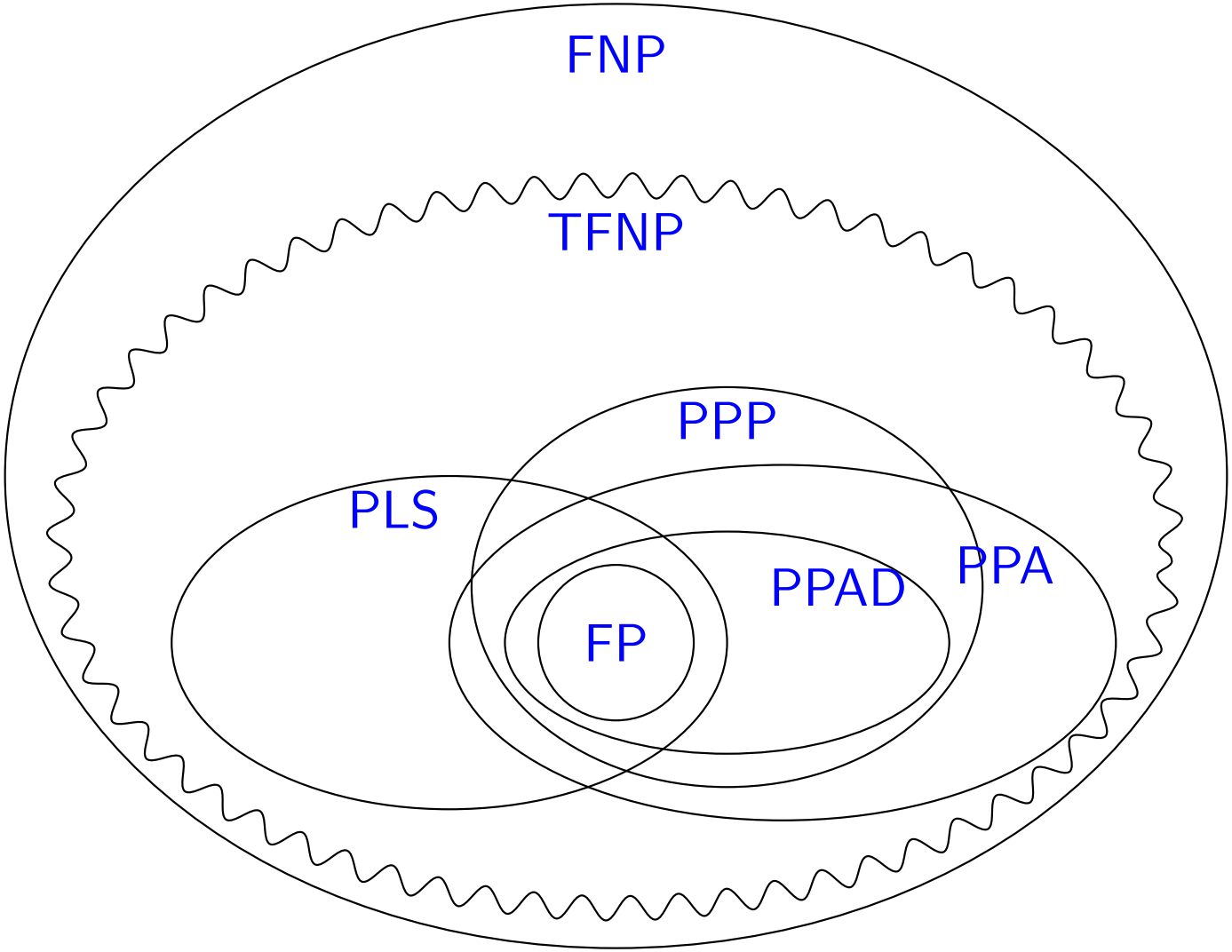
$\rightsquigarrow$ task of classifying "hard" NP total search problems.
FACTORING is a very important NP total search problem; others are interesting (and somewhat important)

OK can we have, say, FACTORING is TFNP-complete?
Good question! TFNP-completeness is as much as we can hope for, hardness-wise

TFNP doesn't (seem to) have complete problems (which needs syntactic description of "fully general" TFNP problem). (Similarly, RP, BPP, NP$\cap$co-NP don't have complete problems[1])

[1] Try to describe "generic" problem/language X in NP$\cap$co-NP as pair of NTMs that accept X and $\overline{X}$: what goes wrong?

Some syntactic classes



Johnson, Papadimitriou, and Yannakakis. How easy is local search? *JCSS*, 1988.
C.H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *JCSS*, 1994.

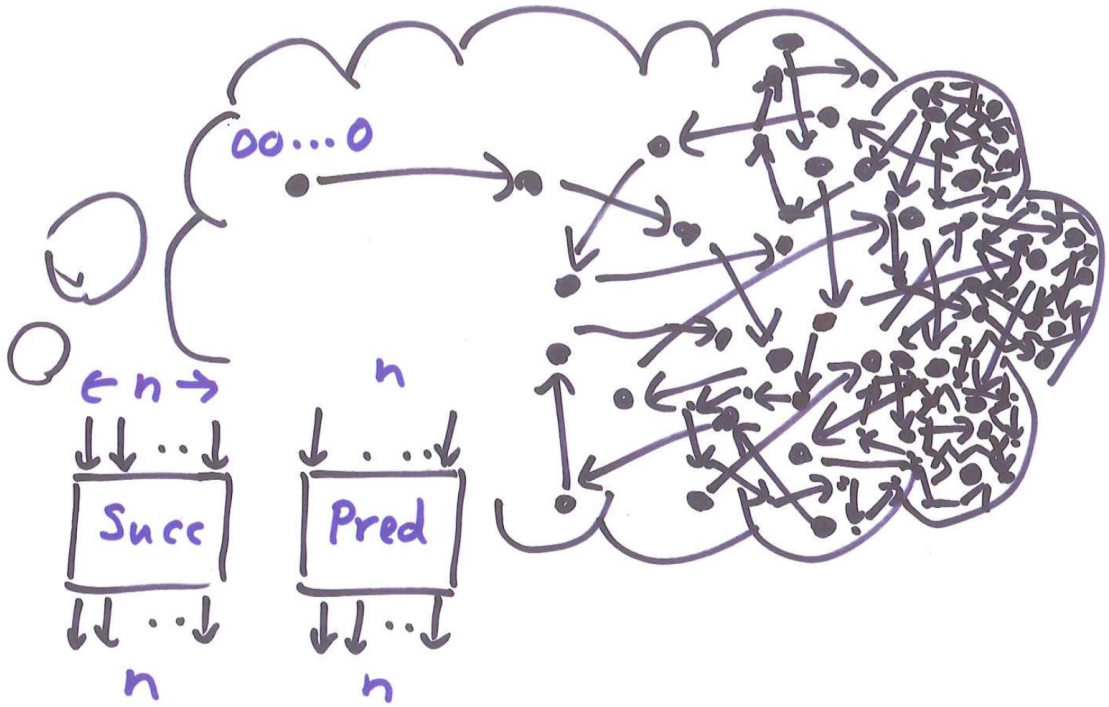
FACTORING, NASH, and the others, as "NP-intermediate"

Advantage of (problems arising in) Ladner's theorem: you just have to believe P≠NP, to have NP-intermediate. For us, we have to believe that FACTORING (say) is not in FP, also that NP≠co-NP.

Disadvantage of Ladner's theorem: the NP-intermediate problems are unnatural (did not arise independently of Ladner's thm; problem definitions involve TMs/circuits)

PPAD

PPAD "given a source in a digraph having in/outdegree at most 1, there's another degree-1 vertex"



The End-of-Line problem

given Boolean circuits S, P with n input bits and n output bits and such that $P(0) = 0 \neq S(0)$, find x such that $P(S(x)) \neq x$ or $S(P(x)) \neq x \neq 0$.

PPA, PPP

PPA: Like PPAD, but the implicit graph is undirected.

The END-OF-UNDIRECTED-LINE problem

given boolean circuit C with n inputs, $2n$ outputs. regard input as one of 2^n vertices, output as 2 neighbouring vertices.

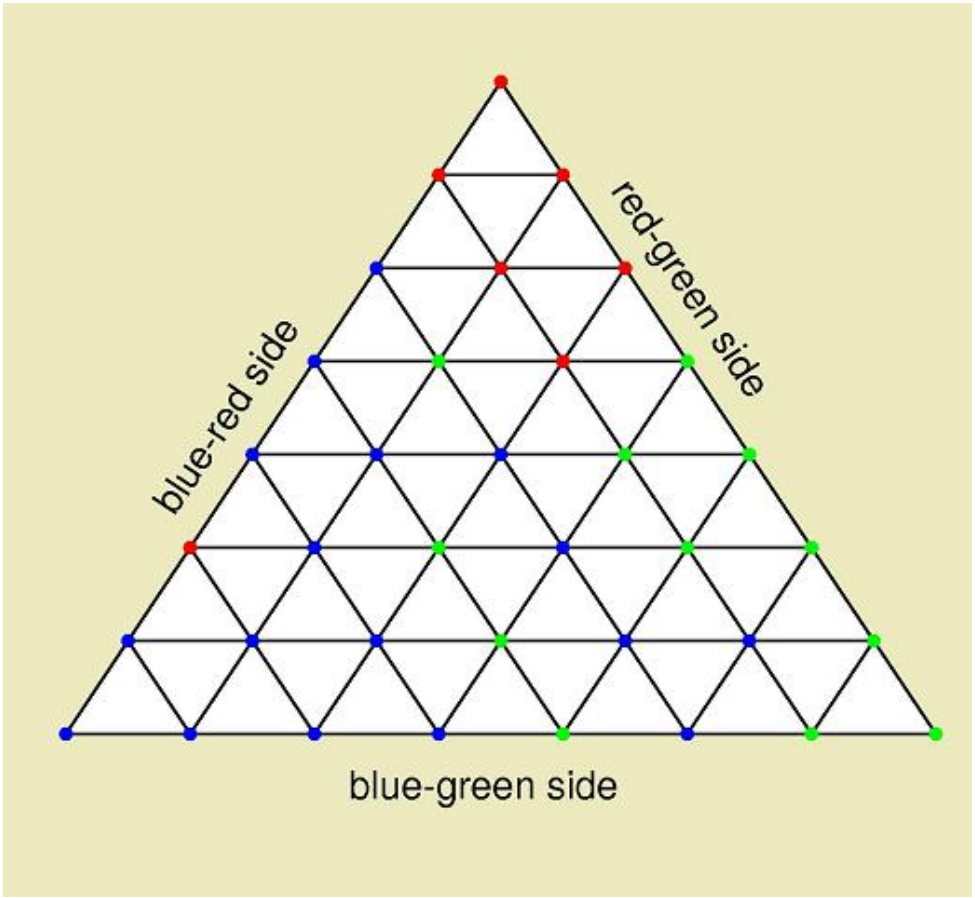
PPP ("polynomial pigeonhole principle"): defined in terms of the PIGEONHOLE CIRCUIT problem.

We have

- END-OF-LINE $\leq_p$ END-OF-UNDIRECTED-LINE (hence PPAD$\subseteq$PPA)
- END-OF-LINE $\leq_p$ PIGEONHOLE CIRCUIT (hence PPAD$\subseteq$PPP)

Next: SPERNER, a PPAD-complete problem

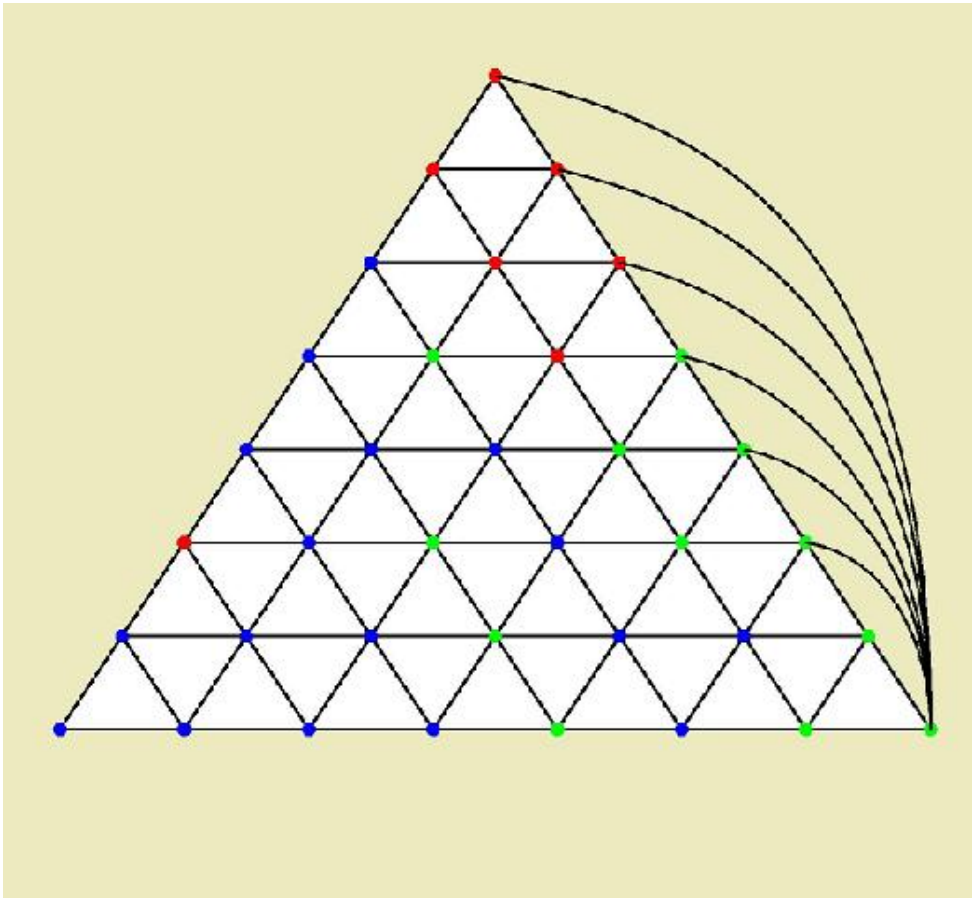
Sperner's Lemma



SPERNER: given a triangular grid coloured subject to certain boundary conditions. Find a "trichromatic triangle". (in "exponentially-fine" resolution, assume some circuit colours each point.)

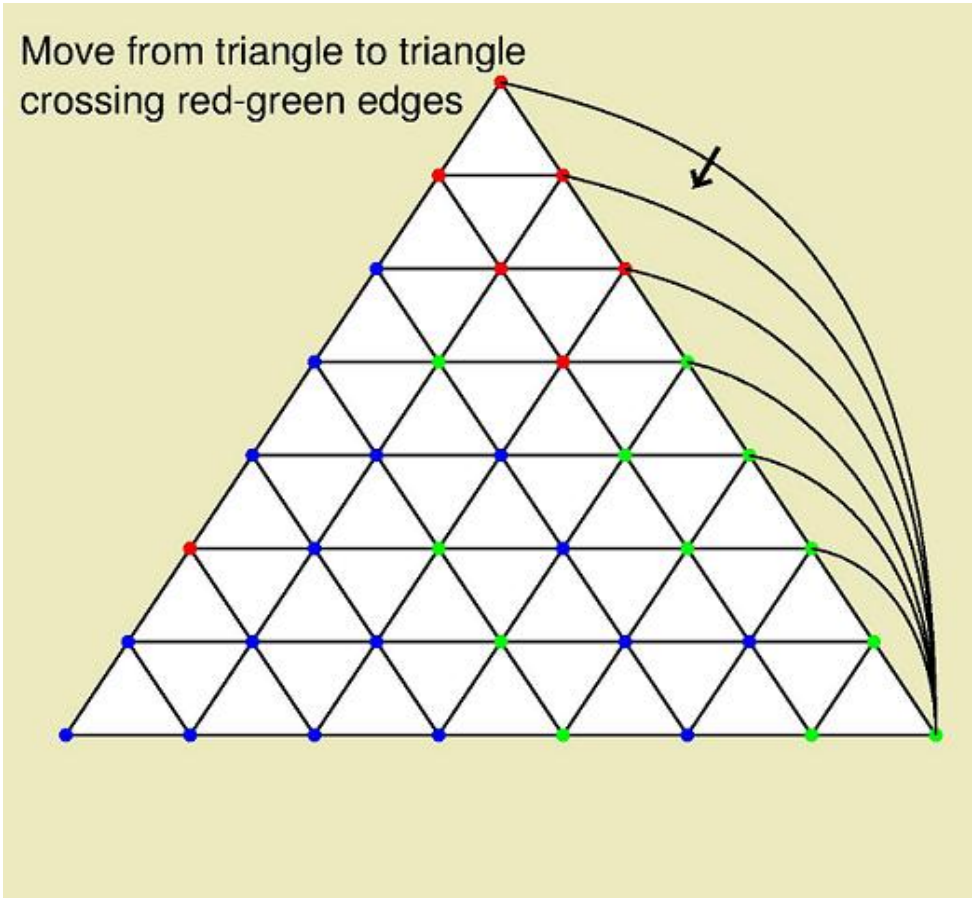
Suppose we color the grid points under the constraint shown in the diagram. Why can we be *sure* that there is a trichromatic triangle?

Reduction to End-of-Line



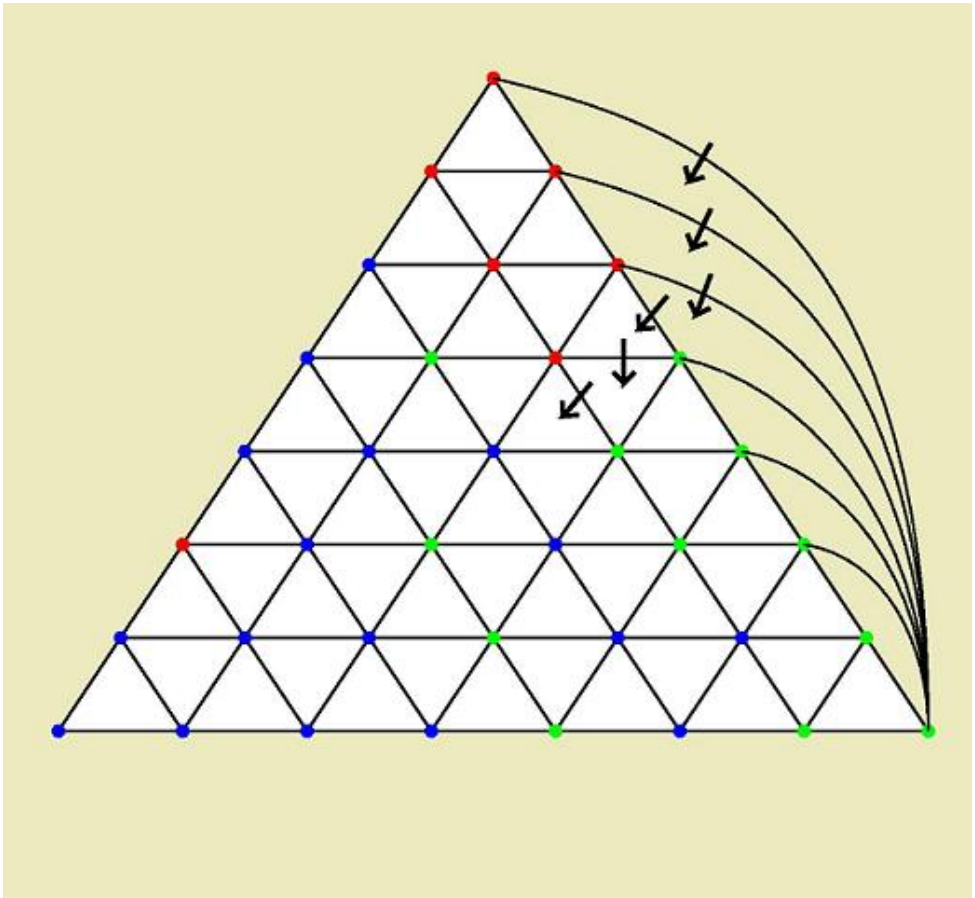
Add some edges such that only one red/green edge is open to the outside

Reduction to End-of-Line



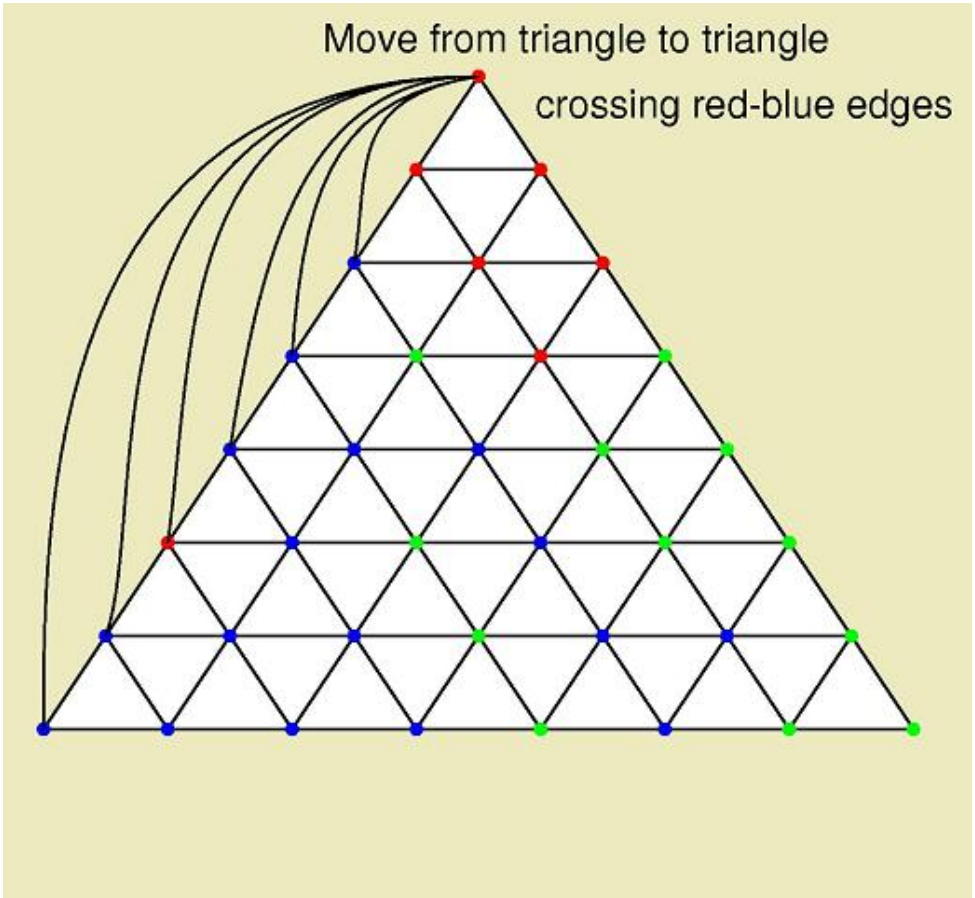
red/green edges are "doorways" that connect the triangles

Reduction to End-of-Line



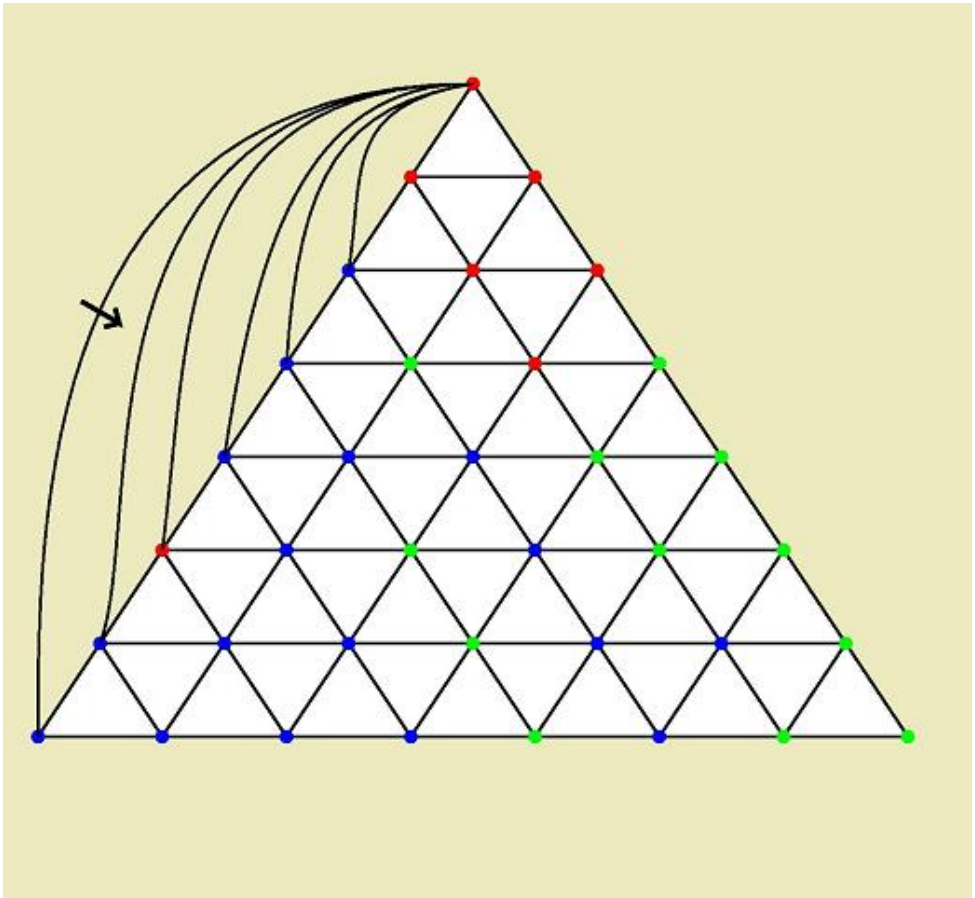
Keep going — we end up at a trichromatic triangle!

Reduction to End-of-Line



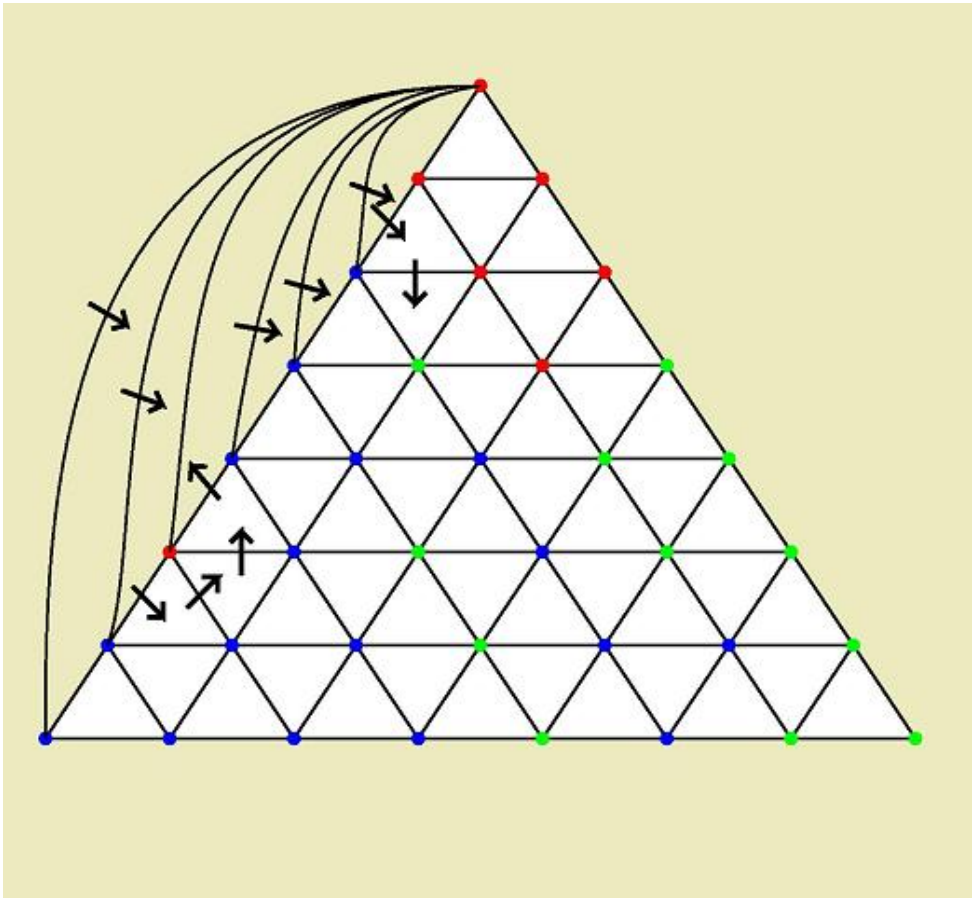
We can do the same trick w.r.t. the red/blue edges

Reduction to End-of-Line



Now the red/blue edges are doorways

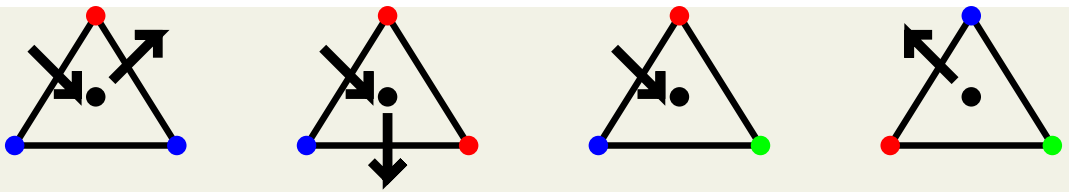
Reduction to End-of-Line

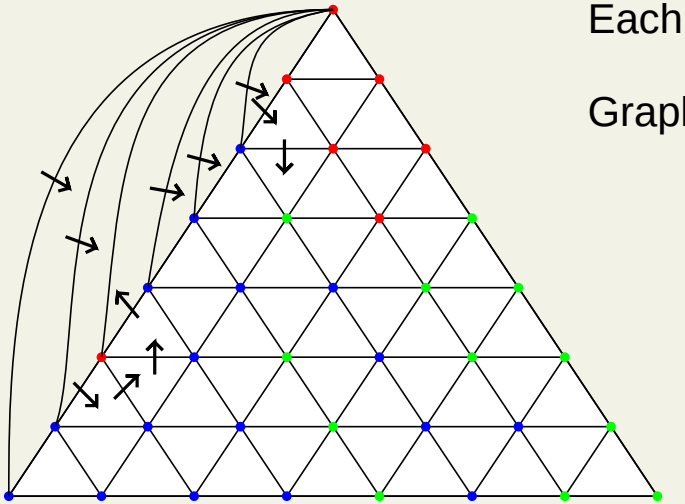


Keep going through them — eventually find a panchromatic triangle!

Reduction to End-of-Line

Degree-2 Directed Graph





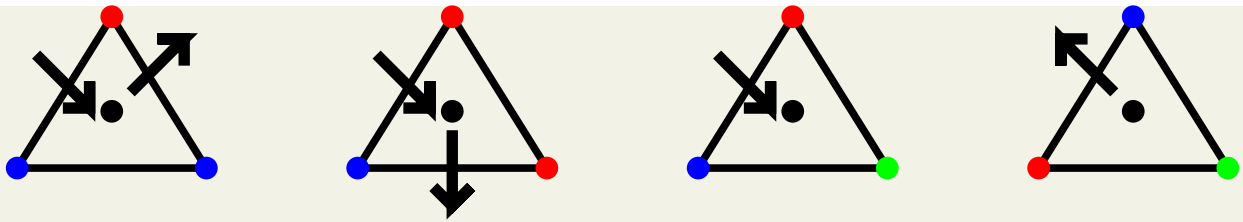
Each little triangle is a vertex

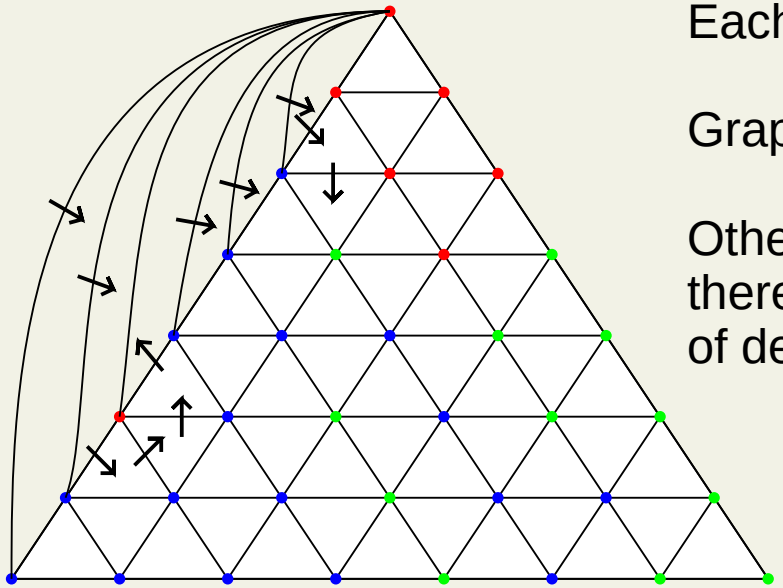
Graph has one known source

Essentially, Sperner's lemma converts the colouring function into an End of line graph!

Reduction to End-of-Line

Degree-2 Directed Graph





Each little triangle is a vertex

Graph has one known source

Other than the known source, there must be an odd number of degree-1 vertices.

Classifying some of these problems

PPAD-completeness:

The Complexity of Computing a Nash Equilibrium. Daskalakis, G, and Papadimitriou (STOC 2006, SICOMP 2009)

Settling the Complexity of Computing Two-Player Nash Equilibria. Chen, Deng, and Teng (FOCS 2006, JACM 2009)

Settling the Complexity of Arrow-Debreu Equilibria in Markets with Additively Separable Utilities Chen, Dai, Du, Teng (FOCS 2009)

PPA-completeness

The Complexity of Splitting Necklaces and Bisecting Ham Sandwiches. Filos-Ratsikas and G (STOC 2019)

PPP-completeness

PPP-Completeness with Connections to Cryptography. Sotiraki, Zampetakis, Zirdelis (FOCS 2018)

Various problems involving circuits are complete for these classes. "Integer Factoring and Modular Square Roots" (Jeřábek, JCSS 2016): randomised reduction from FACTORING to PPA.