

Limits of Computation + Course Recap

ORF 363/COS 323

Instructor: Amir Ali Ahmadi
TAs: Y. Chen, G. Hall, J. Ye
Fall 2014

Reminder: NP-hard and NP-complete problems

Definition.

- A decision problem is said to be **NP-hard** if every problem in NP reduces to it via a polynomial-time reduction.
(roughly means “harder than all problems in NP.”)

Definition.

- A decision problem is said to be **NP-complete** if
 - (i) It is NP-hard
 - (ii) It is in NP.(roughly means “the hardest problems in NP.”)

Remarks.

- NP-hardness is shown by a reduction from a problem that’s already known to be NP-hard.
- Membership in NP is shown by presenting an easily checkable certificate of the YES answer.
- NP-hard problems may not be in NP (or may not be known to be in NP as is often the case.)

The complexity class NP

NP

ORFEO

EP

NP-complete

- ADDITION
- MULTIPLICATION
- LINEQ
- LP
- MAXFLOW
- MINCUT
- MATRIXPOS
- SHORTEST PATH
- SDP ϵ
- PRIMES
- ZEROSUMNASH
- PENONPAPER,...

- TSP
- MAXCUT
- STABLE SET
- SAT
- 3SAT
- PARTITION
- KNAPSACK
- IP
- COLORING
- VERTEXCOVER
- 3DMATCHING
- SUDOKU,...

Reductions

- A reduction from a decision problem A to a decision problem B is
 - a “general recipe” (aka an algorithm) for taking any instance of A and explicitly producing an instance of B, such that
 - the answer to the instance of A is YES if and only if the answer to the produced instance of B is YES.

▪ This enables us to answer A by answering B.

▪ Using reductions for showing NP-hardness:

- If A is known to be hard, then B must also be hard.

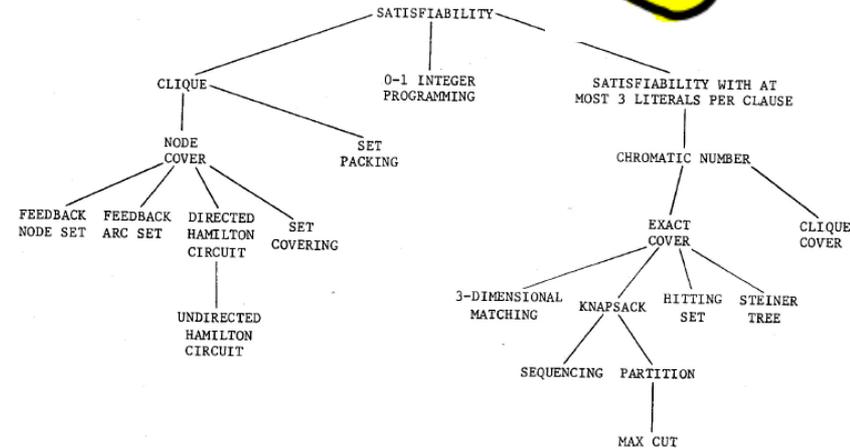
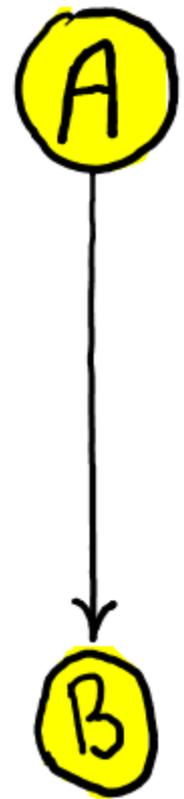


FIGURE 1 - Complete Problems

P versus NP

- All NP-complete problems reduce to each other!
- If you solve one in polynomial time, you solve ALL in polynomial time!



- Assuming $P \neq NP$, no NP-complete problem can be solved in polynomial time.
- This shows *limits of efficient computation (under a complexity theoretic assumption)*

- **Today:** limits of computation in general (and under no assumptions)

Matrix mortality

Consider a collection of m $n \times n$ matrices $\{A_1, \dots, A_m\}$.

We say the collection is **mortal** if there is a finite product out of the matrices (possibly allowing repetition) that gives the zero matrix.

Example 1:

$A_1 =$

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$A_2 =$

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

```
>> A1*A2
```

```
ans =
```

$$\begin{bmatrix} 0 & 0 \\ -1 & 0 \end{bmatrix}$$

```
>> A1*A2*A1*A2
```

```
ans =
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Example from [W11].

Mortal.

Matrix mortality

Consider a collection of m $n \times n$ matrices $\{A_1, \dots, A_m\}$.

We say the collection is **mortal** if there is a finite product out of the matrices (possibly allowing repetition) that gives the zero matrix.

Example 2:

$$A_1 = \begin{pmatrix} 1 & -2 \\ 3 & 0 \end{pmatrix} \quad A_2 = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix} \quad A_3 = \begin{pmatrix} 1 & 2 \\ 0 & -1 \end{pmatrix}$$

Not mortal. (How to prove that?)

- In this case, can just observe that all three matrices have nonzero determinant.
- Determinant of product = product of determinants.

But what if we aren't so lucky?

```
>> A1*A2*A3
ans =
     2     5
     0     3
>> A1*A2*A3*A1*A3
ans =
    17    38
     9    18
>> A2*A2*A3*A1*A3
ans =
     7    16
    -3    -6
>> A2*A2*A1*A3
ans =
     1     4
     3     6
>> ...
```

Matrix mortality

▪ MATRIX MORTALITY

- **Input:** A set of m $n \times n$ matrices with integer entries.
- **Question:** Is there a finite product that equals zero?

Thm. MATRIX MORTALITY is **undecidable** already when

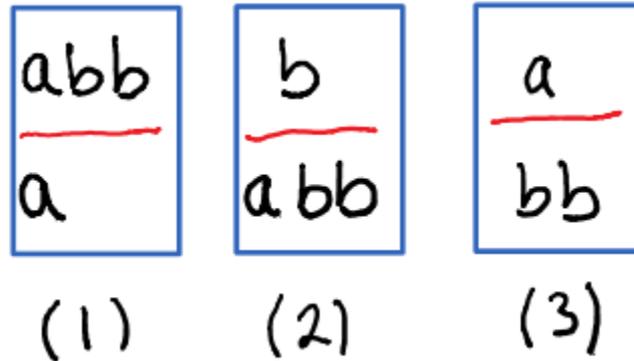
- $n = 3, m = 7,$

or

- $n = 21, m = 2.$

- This means that **there is no finite time algorithm** that can take as input two 21x21 matrices (or seven 3x3 matrices) and always give the correct yes/no answer to the question whether they are mortal.
- This is a definite statement.
(It doesn't depend on complexity assumptions, like P vs. NP or alike.)
 - How in the world would someone prove something like this?
 - By a **reduction** from another undecidable problem!

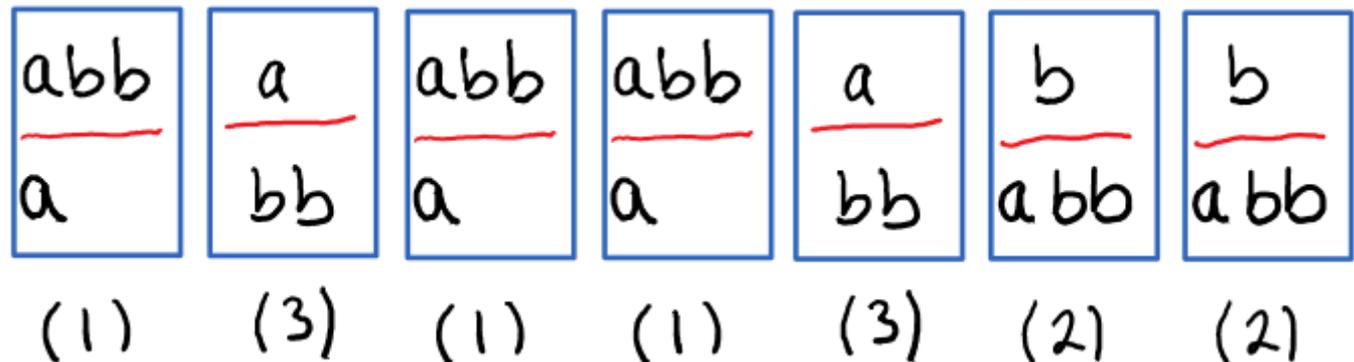
The Post Correspondence Problem (PCP)



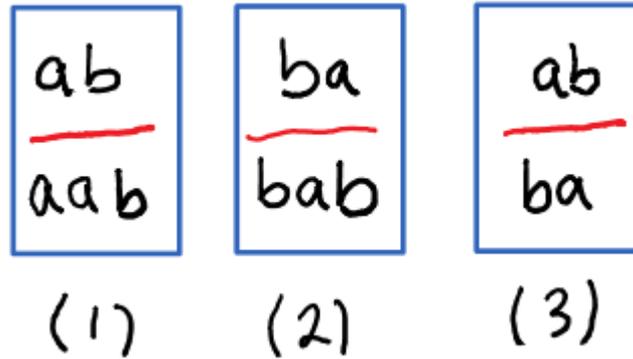
Emil Post
(1897-1954)

Given a set of dominos such as the ones above, can you put them next to each other (repetitions allowed) in such a way that the top row reads the same as the bottom row?

Answer to this instance is **YES**:



The Post Correspondence Problem (PCP)



Emil Post
(1897-1954)

What about this instance?

Answer is **NO**. Why?

There is a length mismatch, unless we only use (3), which is not good enough.

But what if we aren't so lucky?

The Post Correspondence Problem (PCP)

▪PCP

- Input:** A finite set of m domino types with letters a and b written on them.
- Question:** Can you put them next to each other (repetition allowed) to get the same word in the top and bottom row?



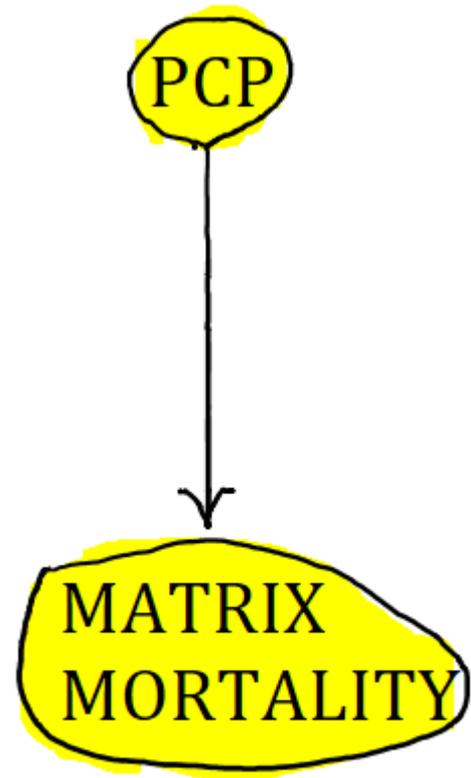
Emil Post
(1897-1954)

Thm. PCP is **undecidable** already when $m = 7$.

- Again, we are **ruling out any finite time algorithm.**
- PCP is decidable for $m = 2$.
- Status unknown for $2 < m < 7$.

Reductions

- There is a rather simple reduction from PCP to MATRIX MORTALITY; see, e.g., [Wo11].
- This shows that if we could solve MATRIX MORTALITY in finite time, then we could solve PCP in finite time.
- It's impossible to solve PCP in finite time (because of another reduction!)
- Hence, it's impossible to solve MATRIX MORTALITY in finite time.
- Note that these reductions only need to be finite in length (not polynomial in length like before).



Integer roots of polynomial equations

- Can you give me three positive integers x, y, z such that

$$x^2 + y^2 = z^2?$$

- Sure:

(3, 4, 5)	(5, 12, 13)	(8, 15, 17)	(7, 24, 25)
(20, 21, 29)	(12, 35, 37)	(9, 40, 41)	(28, 45, 53)

And there are infinitely many more...

- How about $x^3 + y^3 = z^3?$

- How about $x^4 + y^4 = z^4?$

- How about $x^5 + y^5 = z^5?$

Fermat's last theorem tells us the answer is NO to all these instances.

Integer roots to polynomial equations

What about integer solutions to $x^3 + y^3 + z^3 = 29$?

YES: (3,1,1)

What about $x^3 + y^3 + z^3 = 30$?

Looped in MATLAB over all $|x, y, z|$ less than 10 million \rightarrow no solution!

But answer is YES!! $(-283059965, -2218888517, 2220422932)$

What about $x^3 + y^3 + z^3 = 33$?

No one knows!

Integer roots of polynomial equations

■ POLY INT

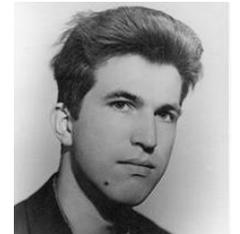
- **Input:** A polynomial p in n variables and of degree d .
- **Question:** Does it have an integer root?

- **Hilbert's 10th problem (1900):** Is there an algorithm for POLY INT?

- **Matiyasevich (1970)** – building on earlier work by Davis, Putnam, and Robinson:

No! The problem is undecidable.

- It's undecidable even in fixed degree and dimension (e.g., $d = 4, n = 58$).



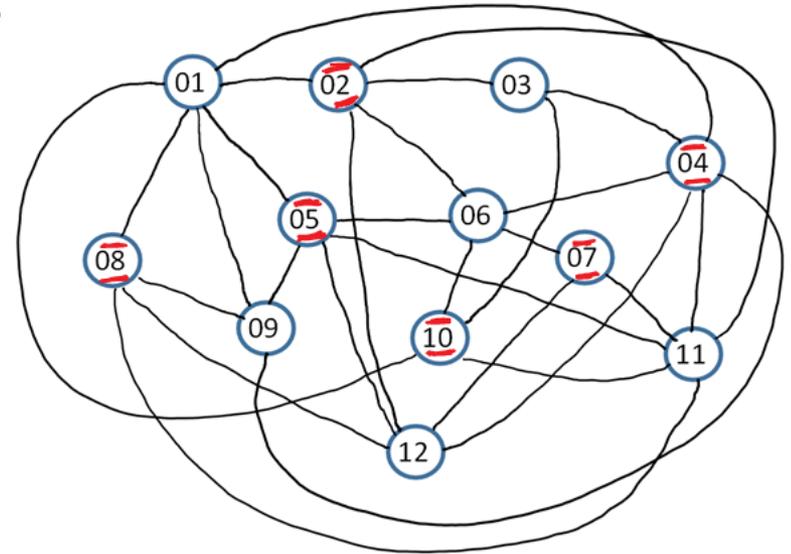
From
Logicomix

Real/rational roots of polynomial equations

- If instead of integer roots, we were testing existence of **real roots**, then the problem would become decidable.
 - Such finite-time algorithms were developed in the past century (Tarski–Seidenberg)
- If instead we were asking for existence of **rational roots**,
 - We currently don't know if it's decidable!
- Nevertheless, both problems are **NP-hard**. For example for
 - A set of equations of degree 2
 - A single equation of degree 4.
 - Proof on the next slide.

A simple reduction

- We give a simple reduction from STABLE SET to show that testing existence of a real (or rational or integer) solution to a set of quadratic equations is NP-hard.
- Contrast this to the case of linear equations which is in P.

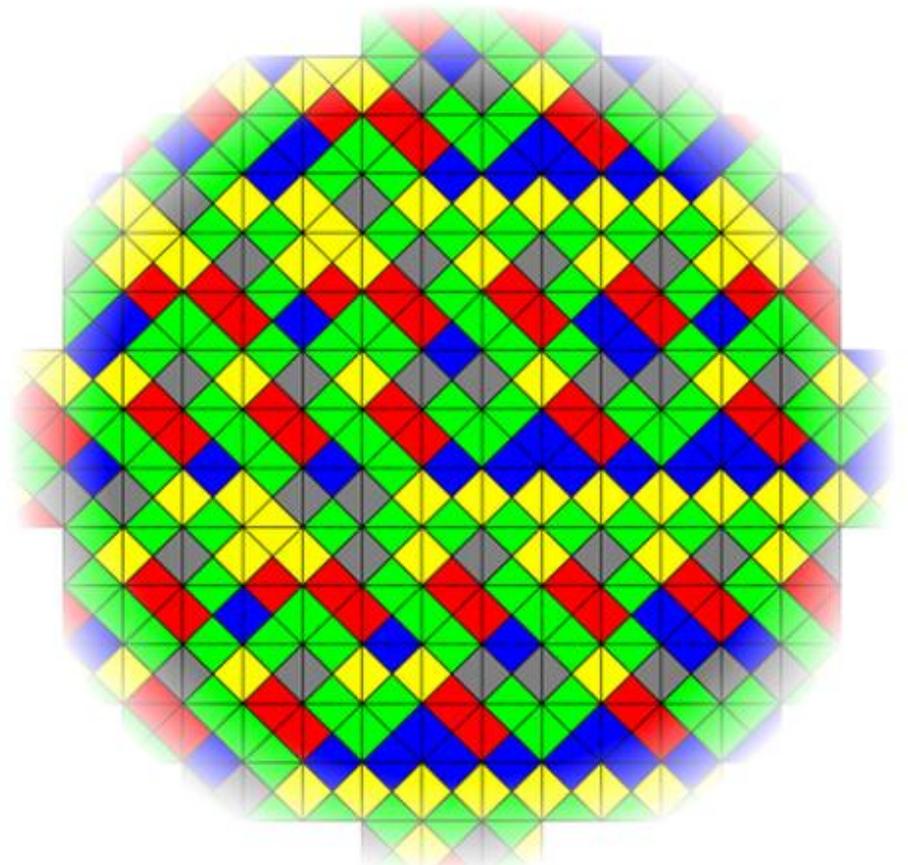
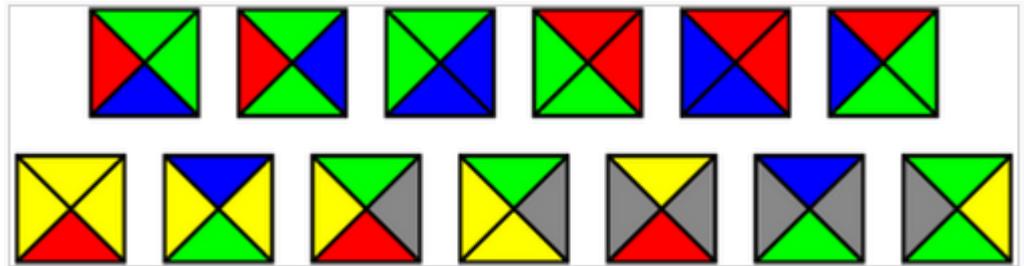


$$\begin{array}{l}
 \exists \text{ stable} \\
 \text{set of} \\
 \text{size } k
 \end{array}
 \Leftrightarrow
 \begin{array}{l}
 \exists x \text{ s.t.} \\
 \left[\begin{array}{l}
 x_1 + \dots + x_n = k \\
 x_i + x_j \leq 1 \quad i, j \in E \\
 x_i \in \{0, 1\}
 \end{array} \right]
 \end{array}
 \Leftrightarrow
 \begin{array}{l}
 \exists x, z \text{ s.t.} \\
 \left[\begin{array}{l}
 (x_1 + \dots + x_n - k)^2 = 0 \\
 1 - x_i - x_j = z_{ij}^2 \quad i, j \in E \\
 x_i(1 - x_i) = 0 \quad i = 1, \dots, n
 \end{array} \right]
 \end{array}$$

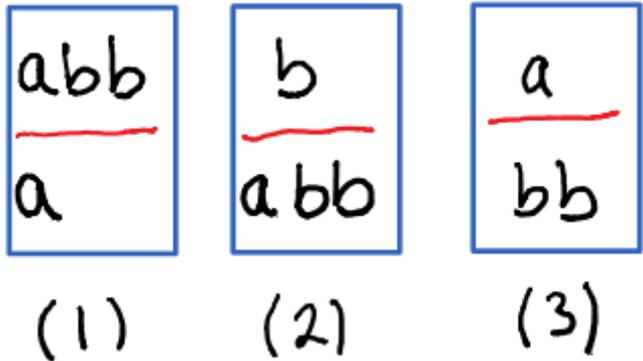
- How would you go from here to a single equation of degree 4?

Tiling the plane

- Given a finite collection of tile types, can you tile the 2-dimensional plane such that the colors on all tile borders match.
- Cannot rotate or flip the tiles.
- The answer is YES, for the instance presented.
- But in general, the problem is undecidable.

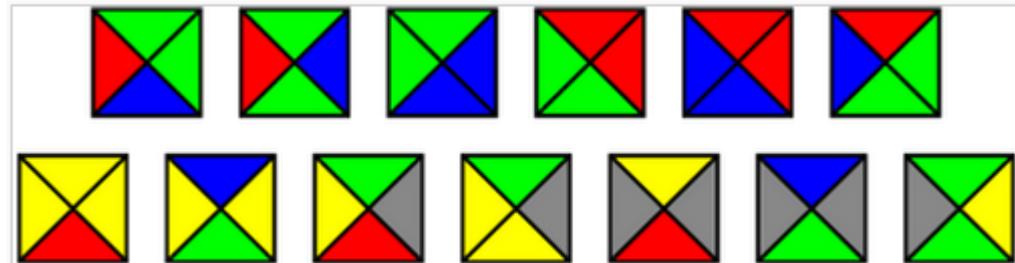


All undecidability results are proven via reductions



$$A1 = \begin{matrix} 1 & -2 & 0 \\ 3 & 0 & -1 \end{matrix} \quad A2 = \begin{matrix} -1 & 1 \\ 0 & -1 \end{matrix} \quad A3 = \begin{matrix} 1 & 2 \\ 0 & -1 \end{matrix}$$

$$x^3 + y^3 + z^3 = 33?$$



But what about the first undecidable problem?

The halting problem

▪ HALTING

▪ **Input:** A file containing a computer program p and a file containing an input x to the computer program.

▪ **Question:** Does p ever terminate (aka halt) when given input x ?

An instance of HALTING:

```
1 function gradient_descent(x)
2
3 %gradient descent with exact line search for minimizing a quadratic
4 %function.
5 Q=[8 0;0 17];
6 b=[136;154];
7 xvec=[];
8 while norm(Q*x-b,2)>10^-5
9     alpha=((Q*x-b)'*(Q*x-b))/((Q*x-b)'*Q*(Q*x-b));
10    x=x-alpha*(Q*x-b);
11    xvec=[xvec x];
12 end
```

→ Program p

$x = [3; 63];$

The halting problem

An instance of HALTING:

```
1 function gradient_descent(x)
2
3 %gradient descent with exact line search for minimizing a quadratic
4 %function.
5 Q=[8 0;0 17];
6 b=[136;154];
7 xvec=[];
8 while norm(Q*x-b,2)>10^-5
9     alpha=((Q*x-b)'*(Q*x-b))/((Q*x-b) '*Q*(Q*x-b));
10    x=x-alpha*(Q*x-b);
11    xvec=[xvec x];
12 end
```

→ Program p

$x = [3; 63];$

- Both the program p and the input x can be represented with a finite number of bits.
- Can there be a program --- call it **terminates(p,x)** --- that takes p and x as input and always outputs the correct yes/no answer to the question: does p halt on x ?
 - We'll show that the answer is no!
 - This will be a proof by contradiction.

The halting problem is undecidable

Proof.

- Suppose there was such a program `terminates(p,x)`.
- We'll use it to create a new program `paradox(z)`:

```
function paradox(z)
1: if terminates(z,z)==1 goto line 1.
```

- The input z to `paradox` is a computer program.
- As a subroutine, `paradox` asks `terminates` to check whether a given computer program z halts when given itself as input. (This is perfectly legal as any program is just a finite number of bits.)
- Note that `paradox` halts on z if and only if z does not halt when given itself as input.
 - What happens if we run `paradox(paradox)` ?!
 - If `paradox` halts on itself, then `paradox` doesn't halt on itself.
 - If `paradox` doesn't halt on itself, then `paradox` halts on itself.
 - This is a contradiction → `terminates` can't exist.

The halting problem (1936)



**Alan Turing
(1912-1954)**

A simpler story to tell strangers at a bar...

(aka Russell's paradox)



The power of reductions (one last time)

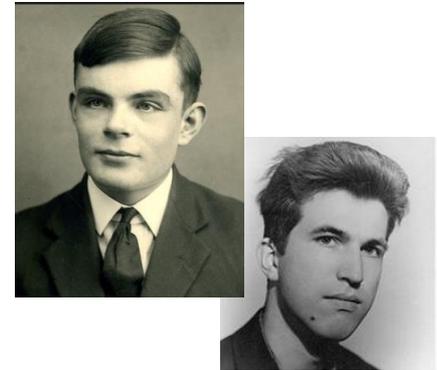
A simple paradox/puzzle:



function $\text{paradox}(z)$
1: if $\text{terminates}(z,z) == 1$ goto line 1.



(lots of nontrivial mathematics, including the formalization of the notion of an “algorithm”)



A fundamental algorithmic question:

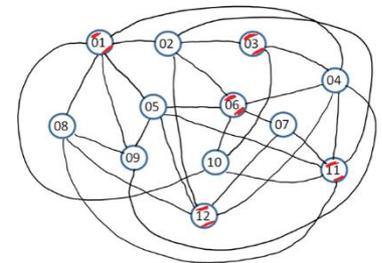
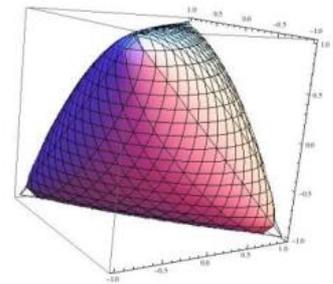
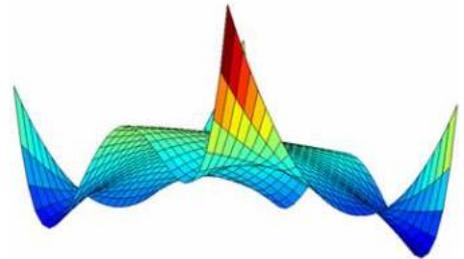
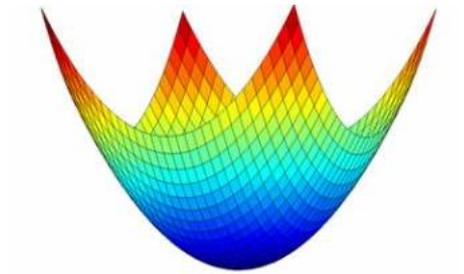
- **POLY INT**
- **Input:** A polynomial p in n variables and degree d .
- **Question:** Does it have an integer root?

A look back at ORF 363/COS 323 ---- Fall 2014



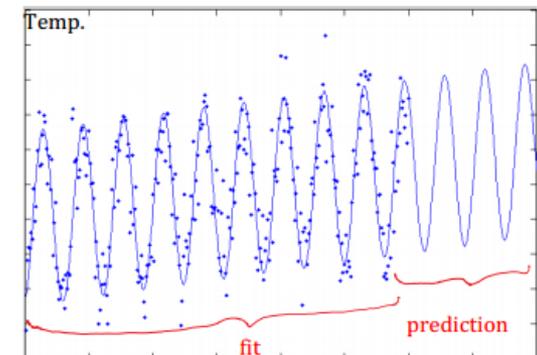
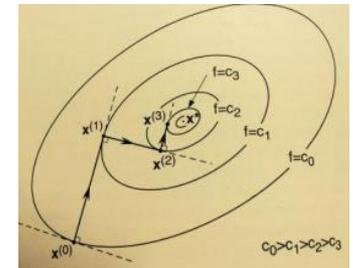
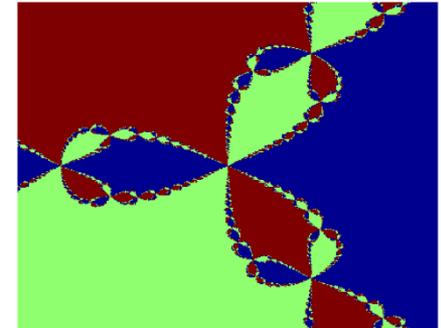
Topics we covered in optimization

- Optimality conditions for unconstrained optimization
- Convex analysis
 - Convex sets and functions
 - Optimality conditions for constrained convex problems
 - Convexity detection and convexity-preserving rules
- Modeling a problem as a convex program
 - Solving it in CVX
- Algorithms for convex unconstrained optimization
- Algorithms for constrained linear optimization
- Duality and robust linear programming
- Semidefinite programming
- Convex relaxations for non-convex and combinatorial optimization
- Theory of NP-completeness



Topics we covered in numerical computing

- Least squares
 - Optimality conditions and normal equations
- Singular value decomposition
- Solving linear systems
- Conjugate gradient methods
- Root finding
 - Bisection, the secant method
 - The Newton method
- Nonlinear least squares
 - The Gauss-Newton method
- Iterative descent algorithms
 - Convergence rates of gradient descent and Newton
 - Condition number
- Approximation and fitting

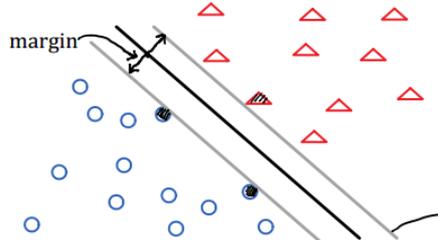


Some application areas we covered

Hey man,
I'm tired of this homework for ORF 363. Let's go party tonight. We can always ask for an extension.

-J

Spam



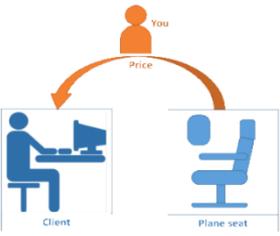
Support vector machines



Image compression

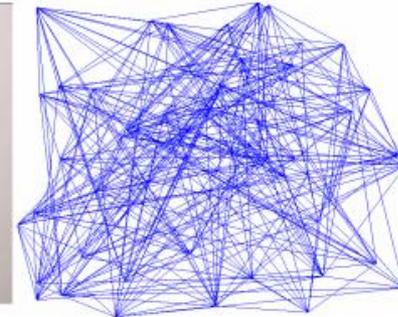


Optimal facility location



LASSO

$$\min_{\beta} \left\| y - \beta_0 - \sum_{k=1}^p \beta_k x_k \right\|_2 + \gamma \|\beta\|_1$$



Event planning

Doodle

	December 2014						
	Mon	Tue	Wed	Thu	Fri	Sat	Sun
# participants	108 PM	118 PM	120 PM	140 PM	150 PM	155 PM	165
Woodrow Wilson	✓	✓					
F. Scott Fitzgerald						✓	
Richard Feynman							
Michelle Obama		✓					
Paul Volcker							✓
John Nash							
Tarek El Moussa				✓			
Ben Brannen					✓		
Paul Krugman						✓	
Andrew Weil	✓						
Steve Forbes					✓		
John Miller	✓						
Katya Sejna							✓
Albert Einstein			✓				
George A. Miller			✓				
Ben Topik			✓	✓			
Mag Wilman							
Donald Rumsfeld							
Eugene O'Neill							✓

Scheduling

Leontief input-output economy

	Transportation	Agriculture	Services	Manufacturing
Transportation	.2	.3	.5	.3
Agriculture	.5	.3	.1	.0
Services	.1	.2	.2	.5
Manufacturing	.1	.1	.1	.1

Minimum fuel control

Minimum intensity radiation therapy

...

How to check if an optimization problem is easy?

- Check if it's convex!
- The functional form of convexity meant:
 - Objective a convex function (if you are minimizing)
 - Constraints: “Convex \leq Concave”, “Affine $=0$ ”.
- If it is, then (most of the time) CVX can already solve it for you up to a reasonably large size.
- There are occasional exceptions:
- Nonconvex problems can be easy:
 - Singular value decomposition (best rank r approximation to a given matrix)
 - One can argue that there is “hidden convexity” (e.g., the dual is an SDP)
- Convex problems can be hard:
 - Optimizing over the set of nonnegative polynomials or copositive matrices
 - Not quite in functional form, but they can be made as such.
- Checking convexity may not be easy
- But the calculus of convex functions and convexity-preserving rules often suffice.

How to check if an optimization problem is easy (formally)?

- Can you reduce it to a problem in P?
- If yes, then it's often easy
 - Unless the polynomial in the running time has high degree or large constants—often rare
 - Unless your input size is massive --- not so rare these days in the era of “big data” (we almost finished the course without using the term “big data”....damn.)
- Can you show it's NP-hard?
- You must reduce a different NP-hard problem to it.
 - If you succeed, an exact efficient algorithm is out of the picture (unless $P=NP$)
- NP-hard problems still routinely solved in practice.
- The message of our guest lecture. Heuristics, branch and bound, etc.
- Convex optimization is often a powerful tool for approximating non-convex and NP-hard problems.
- We saw many examples in recent weeks; e.g., LP and SDP relaxations.

Slide from lecture 1: Course objectives

- The skills I hope you acquire:
 - Ability to view your own field through the lens of optimization and computation
 - To help you, we'll draw applications from operations research, statistics, finance, machine learning, engineering, ...
 - Learn about several topics in scientific computing
 - More mathematical maturity and ability for rigorous reasoning
 - There will be some proofs in lecture. Easier ones on homework.
 - Enhance your coding abilities (nothing too fancy, simple MATLAB)
 - There will be a MATLAB component on every homework and on the take-home final.
 - Ability to recognize hard and easy optimization problems
 - Ability to use optimization software
 - Understand the algorithms behind the software for some easier subclass of problems.

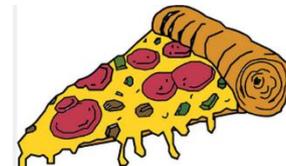
The final exam!

- Take-home exam.
- To be given out on Wednesday, January 14, 9AM.
- Will be due Friday, January 16, 12PM (in my office → Sherrerd 329).
- You get one extra day per conflicting exam. Contact Georgina.
- Don't forget pset 8 is now due Wednesday, January 7, 12 PM.
 - Can drop it ORFE undergraduate lounge (Sherrerd 123) – special box for ORF 363.
- **What to study for the final?**
- All the lecture notes I post. I plan to post a single PDF file with everything.
- Psets 1-8.
- The last page of the notes points you to certain sections of the book for additional reading.
 - [DPV] Chapters 7 and 8 are particularly good. (You can skip 7.2 and 7.5.)
- Be comfortable with MATLAB and CVX.

Some good news

- Undecidability from today's lecture won't be on the final.
- Our guest lecture won't be on the final.
- The course just got approved for the certificate program in statistics and machine learning (SML).
 - It was already part of the certificates for applied computing, and engineering and management systems.
- The TAs and I will hold office hours during reading period. I will hold an extra one for pset 8.
- There will be a 3 hour review session.
 - I proposed Tuesday, January 13, 12-3PM in Friend 006, but some students objected because of Dean's date. I will do a poll on Piazza and reschedule subject to room availability.

- There will be pizza!



Last but not least...

- It's been a pleasure having you as my very first students!
- Go make optimal decisions in your lives!
- And keep in touch!

AAA.
December 11, 2014

Notes & References

■ Notes:

- Chapter 8 of [DPV08] mentions undecidability and the halting problem. Chapter 9 of [DPV09] is optional but a fun read.

■ References:

- [Wo11] M.M. Wolf. Lecture notes on undecidability, 2011.
- [Po08] B. Poonen. Undecidability in number theory, *Notices of the American Mathematical Society*, 2008.
- [DPV08] S. Dasgupta, C. Papadimitriou, and U. Vazirani. Algorithms. McGraw Hill, 2008.