UNIVERSITY OF TRENTO

Complexity, Cryptography, and Financial Technologies

Lecture 5 – Introduction to Computational Complexity

Fabio Massacci

What is Computational Complexity?

UNIVERSITY OF TRENTO

- **Computability Theory: Can we potentially solve a class of problems with a finite amount of computation?**
  - Elementary computation: $\left.2^{2^{\cdot^{\cdot^{2}}}}\right\}n$ is finite for any n
- **Computational Complexity Theory: Can we actually solve a class of problems i.e. with realistically limited computational resources?**
  - TIME - Execution steps,
  - SPACE - memory cells,
  - RANDOMNESS - random numbers
    - A supply of "good" random numbers is critical for cryptography

Model of Computations

UNIVERSITY OF TRENTO

- **To specify a computational model we need:**
  - the set of possible environments
  - the set of machines (computational rules)
  - the effect of applying such rules on an environment
- **Several models exist**
  - Turing Machine → most famous one
  - Random Access Memory → mostly equivalent to a TM
  - Quantum Machine → can do some operations in parallel
- **Mostly equivalent from computability perspective, some difference from complexity perspective**

Turing Machines vs RAM

UNIVERSITY OF TRENTO

- **Main component in the environment of a TM**
  - an infinite sequence of cells (a tape),
    - each cell hold a single symbol or blank, extending infinitely to the right
  - a transition function based on current state of machine and content of current cell determines new symbol state of the machine, movement instruction (L or R or S)
    - The machine modifies content of current cell and its internal state, and moves as directed
  - Description typically includes some special states called accepting states
  - Some versions had more tapes for parallel operations
- **Main Component in the environment of a RAM**
  - A infinite vector of registers
  - Classical operation on registers
    - loading a value into a register, adding the value of two registers, jumping to a location specified into a register if another register is zero
  - Possibility to refer to a register directly or indirectly (a register whose number is identified by the value specified in another register)
    - This property is important to be equivalent to a Turing Machine
- **The two models are equivalent so we use a RAM for simplicity**

More "Powerful" Machines

- **Oracle Machine**
  - additional data structure (to make queries and read its answers) and two special state (oracle invocation and oracle spoke).
    - For turing machine is a tape, for RAM some other registers
  - Computation of oracle machine M_f on input x and access to the oracle $f:\{0,1\}^* \rightarrow \{0,1\}^*$ is essentially identical
    - If a machine makes a query q then the answer it obtains is f(q).
    - $M_f(x)$ is the output of M on input x when given access to oracle f.
  - Intuitively, with an oracle computing f costs 1 = nothing
    - Either in time or in space
- **Universal Machine**
  - Basically a machine that can read its own program to execute from input data structure i.e. a normal computer

Uncomputable Functions

- **Not all functions are computable.**
  - Not every well-defined task can be solved by a "reasonable" automated procedure.
  - Theorems hold for any reasonable model of computation (See Goldreich book)
  - Only assumption: each machine/function M etc. in the model has a finite description <M> (i.e., can be described by a string)
- **Theorem 1.4: Most functions are uncomputable.**
  - The set of computable functions is countable (set of integers), whereas the set of all functions (from string to string) has cardinality of reals
    - Each string describing a program can be be described by the integer of its binary representation
    - Each real in [0,1] can be described by a 0/1 function over strings as f(n)=n-th decimal digit of the number
- **Theorem 1.5: The halting function is not computable.**
  - Halting function h : $\{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}$:
    - h(<M>, x) = 1 iff M halts on input x
  - No algorithm given a arbitrary pair (<M>, x), can decide whether M halts on input x)
    - Technique is diagonalization: construct a new machine M* that reads a machine description <M>, calls h and if h(M,x)=1 then loops for ever (else stops). When M* reads <M*> it runs into troubles…
  - This has to be true for arbitrary <M>. For some particular M this is well possible
    - (e.g. <M> corresponding to context free grammars written in a particular form)

Rice's Theorem

UNIVERSITY OF TRENTO

- **Theorem 1.6 (Rice's Theorem):**
  - Let F be any non-trivial subset of the set of all computable partial functions, and let SF be the set of strings that describe machines that compute functions in F. Then deciding membership in SF cannot be solved by an algorithm.
- **Rice's Theorem means:**
  - no algorithm can determine any non-trivial property of the function computed by a given computer program (written in any programming language)
- **Practical example**
  - It is impossible to design an algorithm that automatically distinguishes an arbitrary program from the set of functionally identical programs with some vulnerability

Complexity concerns "efficient "computation

UNIVERSITY OF TRENTO

- **If most functions (i.e. problems) are not computable, what about the relation between "efficiently computable" functions and "just computable"?**
- **We try to characterize the problems into classes**
  - Can we solve them efficiently in time?
  - Do we need a lot of memory?
  - Do we need a lot of "good" random numbers?
    - "Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin." John von Neumann
- **What can (or cannot) be solved by**
  - Making lucky guesses
  - Making random moves
  - Asking advice
  - Leaving no trace behind
- **Efficient → Polynomial**
  - In practice: if input dataset is LARGE then efficient → poly logarithmic

Classical Formulation: P vs NP

UNIVERSITY OF TRENTO

- **The students' accommodation problem**
  - Suppose that you are organizing housing accommodations for a group of 400 university students. Space is limited and only 100 students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice.
- **Wanna be a millionaire?**
  - we don't know if problem above admits a solution that can be found in polynomial time
    - In any reasonable model of computation
  - http://www.claymath.org/millennium/P_vs_NP/

P vs NP Questions

UNIVERSITY OF TRENTO

- **The P vs NP questions has been at a core of CS**
- **However most people make a great mistake**
  - P stand for Polynomial Time
  - N in NP stand for Non-Polynomial Time → ERROR!!!
- **What acronyms really stand for**
  - P = "Solvable in Deterministic Poly Time"
  - NP = "Solvable in NON-Deterministic Poly Time"
- **NP complete = hardest problem in NP**
  - If we can solve any of them then you can use the solution to solve any problem in NP
- **Non-Deterministic "intuitive" meaning**
  - IF you make a lucky guess OR somebody gives you an hint then the problem is solvable in polynomial time OTHERWISE though luck
  - The whole existence of modern crypto is based on this intuition

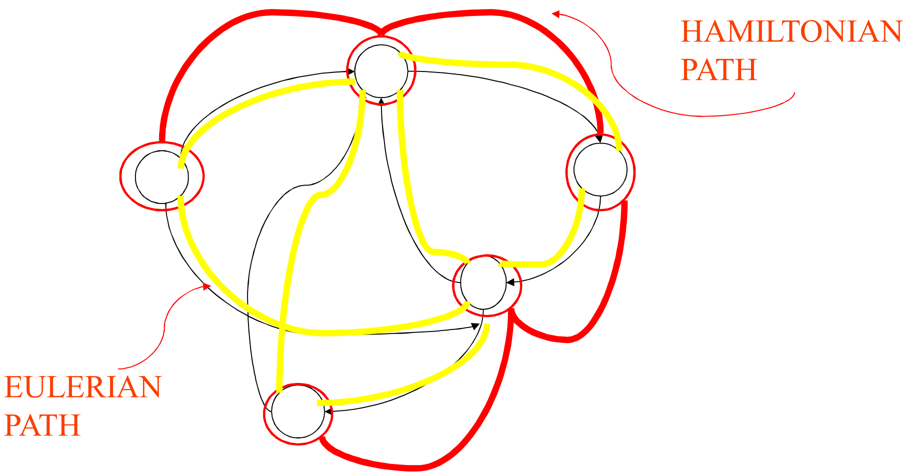
An example

UNIVERSITY OF TRENTO

- **The old Prussian city of Konigsberg (now Kaliningrad in Russia) had seven bridges. Can citizens stroll along every bridge and return to the same point?**
  - Formulated by Euler, a famous mathematician
- **Admit two formulations as a graph**
  - Crossroads are nodes and bridges are edges between them → P
  - Bridges are nodes and roads connecting them are edges → NP-complete
- **As you see formulation is everything…**

Hamilton – Eulerian Path

UNIVERSITY OF TRENTO



Problem Complexity?

- Complexity Class = set of problems with "same" computational complexity

Problem Instance	Efficient Solution?
cannot be solved efficiently	Solution exp. long wrt the problem's instances
could potentially be solved efficiently	Solution comparable to problem's instances BUT we are not able to find it quickly
can be solved efficiently	Solution comparable to problem instances AND we are able to find it quickly

Solvable Search Problems?

- **Informally:**
  - to be potentially solvable a problem must have a short solution
  - whether we are able to find it, that's another story
- **Formally:**
  - search problems must have a solution whose length is bounded by a polynomial in the size of the instance
- **Def. 2.1 (Polynomially bounded relations):**
  - $R \subseteq \{0,1\}^* \times \{0,1\}^*$ is polynomially bounded iff
  - there exists a polynomial p s.t.
  - for every $(x, y) \in R$ $|y| \leq p(|x|)$..

Graph Isomorphism

- Two graphs (V1,E1) and (V2,E2) are isomorphic if we can find a mapping between the edges of one and the other and viceversa
- Define the input/output

Problem Representation of Graph Isomorphism

$R=\{((\underbrace{\langle V_1,E_1\rangle\ \langle V_2,E_2\rangle}_{\text{THIS IS X}}),\ \underbrace{map}_{\text{THIS IS Y}})\ |$

$\langle V_1,E_1\rangle$ is a graph,

$\langle V_2,E_2\rangle$ is a graph,

map: $V_1 \rightarrow V_2$ must be injective & surjective etc. }

- n nodes V_1 & $|E_1| < n^2$
- n nodes V_2 & $|E_2| < n^2$

We have:

$\boxed{\begin{array}{l} |x| \leq O(2n^2+ 2n) \sim O(n^2) \\ |y| \leq O(n) \end{array}}$

What we want: $|y| \sim O(p(|x|))$

Polynomially bounded $R_{ISOMORPHISM}$

- $|x| \sim O(n^2)$
- $|y| \sim O(n)$
- To conclude that $R_{ISOMORPHISM}$ is polynomially bounded
  - $|y| < p(|x|)$
  - $n \sim \sqrt{|x|} \rightarrow |y| \leq O(\sqrt{|x|}) \sim O(|x|^{1/2})$
- Graph isomorphism is polynomially bounded
- This reasoning is (partly) wrong!

Polynomially bounded $R_{ISOMORPHISM}$

- $|x| \sim O(n^2) \leftarrow$ Worst case
- $|y| \sim O(n) \leftarrow$ Worst case
- To conclude that $R_{ISOMORPHISM}$ is poly bounded

For all $|x|$, $|y|$

(Best case, Best case)
(Best case, Worst case)
(Worst case, Best case)
(Worst case, Worst case)
(etc, etc)

Smallest input $O(n)$ — Largest output $O(n)$ $\rightarrow$ $|y| \leq O(|x|)$

- Graph isomorphism is polynomially bounded

Discrete Logarithm modulo a prime

UNIVERSITY OF TRENTO

- **Group Z_p**
  - Integers from 0… p-1
  - Multiplication and Addition are defined modulo p
    - 6+6 mod 7 = 7 + 5 mod 7 = 5
    - 7*5 mod 7 = 7 *1 mod 7 = 0
  - If p is a prime → inverse of addition and (non-zero) multiplication always exists
    - 6 + 1 mod 7 = 0 → 6 and 1 are additive inverse
    - 4 * 2 mod 7 = 1 → 4 and 2 are multiplicative inverse
- **Generator of a Group**
  - Exists number g s.t. ForAll $n \in Z_p$ Exists $k \in Z_p$ s.t. $n = g^k \bmod p$
    - 2^2 mod 7 = 4 , 2^3 mod 7 = 1 → no
    - 3^1 mod 7 = 3, 3^2 mod 7 = 2, 3 mod 7 = 6, 3^4 mod 7 = 4, 3^5 mod 7 = 5→yes

Discrete Logaritm Modulo a prime (II)

UNIVERSITY OF TRENTO

- **Discrete Log**
  - Given a prime p, a generator g and a number x < p
  - Find y s.t g^y mod p = x
- **A solution always exists for all input problem x**
  - By definition of generator → $R_{DLOG}(x) \neq \emptyset$
- **How is $R_{DLOG\text{-}known\text{-}g}$ defined? What is x? what is y?**

Discrete Logaritm Modulo a prime (III)

UNIVERSITY OF TRENTO

- **Previous formulation deliberately misleading**
- **Discrete Log**
  - Given a prime p, a generator g and a number n < p
  - Find k s.t g^k mod p = n
- **How is $R_{DLOG\text{-}known\text{-}g}$ defined**
  - $R_{DLOG} = \{<\underbrace{(p,g,n)}_{\text{THIS IS X}},\underbrace{k}_{\text{THIS IS Y}}>\} \mid g^k = n \pmod p$
    AND p is prime
    AND g is a generator of Z_p }
- **A solution always exists for all input problem x**
  - $R_{DLOG\text{-}knonw\text{-}g}(x) \neq \emptyset$

Discrete Logaritm Modulo a prime (IV)

UNIVERSITY OF TRENTO

- **Discrete Log-known-g**
  - Given a prime p, a generator g and a number n < p
  - Find k s.t g^k mod p = n
- **How is $R_{DLOG\text{-}known\text{-}g}$ defined**
  - $R_{DLOG\text{-}known\text{-}g} = \{<\underbrace{(p,g,n)}_{\text{THIS IS X}},\underbrace{k}_{\text{THIS IS Y}}>\} \mid g^k = n \pmod p$
    AND p is prime
    AND g is a generator of Z_p }
- **Which are the dimensions of $R_{DLOG\text{-}known\text{-}g}$?**
  - |x| ~ O(…), |y| ~ O(...)

Discrete Logaritm Modulo a prime (IV)

UNIVERSITY OF TRENTO

- **Discrete Log**
  - Given a prime p, a generator g and a number n < p
  - Find k s.t g^k mod p = n
- **How is $R_{DLOG\text{-}known\text{-}g}$ defined**
  - $R_{DLOG} = \{<(\underbrace{p,g,n}_{\text{THIS IS X}}),\underbrace{k}_{\text{THIS IS Y}}>\} \mid g^k = n \pmod p$ AND p is prime AND g is a generator of Z_p }
- **Which are the dimensions of $R_{DLOG\text{-}known\text{-}g}$?**
  - |x| ~ O(log p), |y| ~ O(log p) = O(|x|)

24/09/2018 Massacci, Ngo - Complexity, Crypto, and FinTech ► 23

Discrete Logaritm Modulo a prime (IV)

UNIVERSITY OF TRENTO

- **Discrete Log (without knowing g)**
  - Given a prime p~~, a generator g~~ and a number n < p
  - Find g, k s.t g^k mod p = n
- **How is R_{DLOG} defined**
  - $R_{DLOG} = \{<(\underbrace{p,n}_{\text{THIS IS X}}),\underbrace{(g,k)}_{\text{THIS IS Y}}>\} \mid g^k = n \pmod p$ AND p is prime AND g is a generator of Z_p }
- **Which are the dimensions of R_{DLOG}?**
  - |x| ~ O(log p), |y| ~ O(log p) = O(|x|)

24/09/2018 Massacci, Ngo - Complexity, Crypto, and FinTech ► 24

Graph Reachability

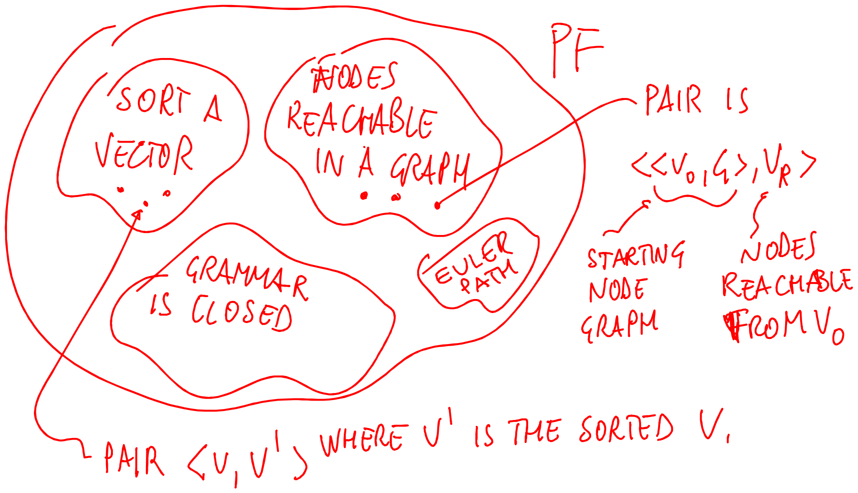
- **Examples of Graph reachability Problems**
  - Does a web site have dangling links?
  - Can a distributed system enter a deadlock state?
  - Can an embedded system controller reach an unwanted state?
- **Explicit Representation**
  - R={<v0,<V,E>, Vr> | <V,E> is a graph, $v_0 \in V_r \subseteq V$ s.t V_r is the set of nodes reachable from v0}
  - $|x| = |<v_0,<V,E>>| = O(n^2)$
  - $|y| = |V_r| \leq |V| = O(n)$
- **Is this polynomially bounded? Yes**
- **How to find it? How to check if solution is correct?**

PF (Polynomial-time Find)

- **PF: class of efficiently solvable search problems**
- **R ∈ PF iff**
  - R is polynomially bounded (Def 2.1) and
  - there is a algorithm that given x efficiently finds y s.t. (x, y) ∈ R (or asserts no such y exists).
- **Def. 2.2 (efficiently solvable search problems): Search problem $R \subseteq \{0,1\}^* \times \{0,1\}^*$ is efficiently solvable iff**
  - R is a polynomially bounded relation and
  - there exists a polynomial time algorithm A s.t.
  - for every x, $A(X) \in R(x)$ if $R(x) = \{y \mid (x,y) \in R\}$ is not empty
  - $A(x) = \perp$ If $R(x)=\emptyset$ (x has no solution).

Instances vs Classes

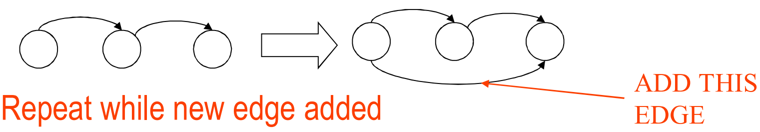
UNIVERSITY OF TRENTO



Graph Reachability - Algorithm

UNIVERSITY OF TRENTO

- **The transitive closure is the graph where all nodes reachable from another node also have a direct edge among this latter node**

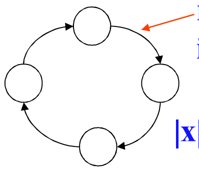


- Repeat while new edge added
- For all $<u,v> \in E$
- For all $<v,w> \in E$
- If $<u,w> \notin E$ then $E \leftarrow E \cup \{<u,w>\}$
- $Vr \leftarrow \{ w \mid <v_0,w> \in E\}$
- Easy upper bound is $O(n^5)$ – can be improved to $O(n^4)$

Transitive Closure of a Graph

UNIVERSITY OF TRENTO

- **The set V_r is expensive to verify**
  - Essentially we need to re-run the algorithm
- **The transitive closure itself could be a y**
  - Easy to verify (run loop from previous slide once, reject if transitive edge not among edges)
  - But larger size



n nodes in a circle

|x|= n nodes + n edges

|y|: all possible n^2 connections
$|y| < O(|x|^2)$

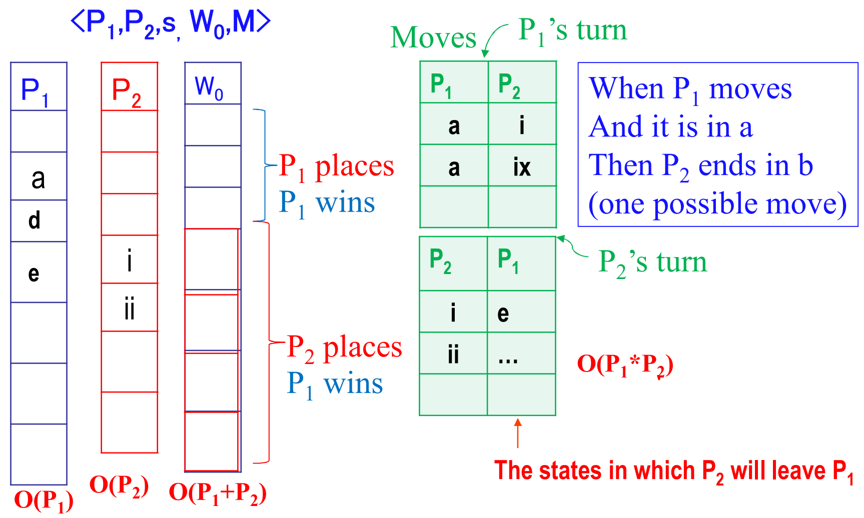
Problems in PF

UNIVERSITY OF TRENTO

- **Find Eulerian Path in a Graph**
  - *Input: Graph <V,E,s,t> where $E \subseteq V \times V$ and*
    - *$s \in V$ – source vertex, $t \in V$ – target vertex*
  - *Output: sequence $<e_1,\dots,e_n> \in E^*$ s.t.*
    - *n=|E|* Only n
    - *$\cup_{i=1}^{n} \{e_i\} = E$* All n
    - *e_i=<u,v> and e_{i+1}=<v,w> for all i=1…n for some u,v,w* $e_1, \dots, e_n$ is a path
    - *e_1=<s,v> and e_n=<u,t> for some v, u*
- **Find winning strategy in 2-player game (explicit moves)**
  - *Input: Game $<P_1, P_2, s, W_0, M>$ where*
    - *$s \in P_1$ – the initial position of Player 1*
    - *$W_0 \subseteq P_1 \cup P_2$ – the winning positions of Player 1*
    - *$M \subseteq P_1 \times P_2 \cup P_2 \times P_1$ – the possible moves*
  - *Output: a winning strategy for Player 1*
  - *(Question: How do we represent the output efficiently?)*

Representation of Problems in PF

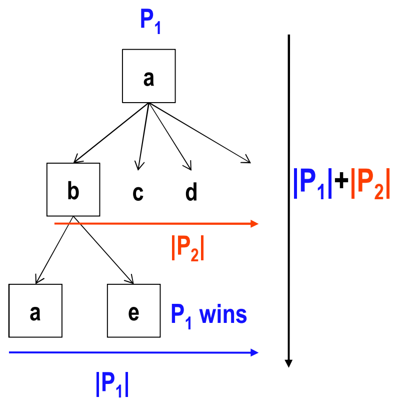
UNIVERSITY OF TRENTO



How to represent a strategy?

UNIVERSITY OF TRENTO

- Naïve representation of strategy so that P_1 always wins



Should be +, not * (why?)

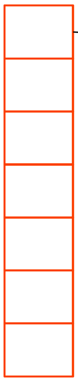
$|y| \approx O(\max(|P_1|,|P_2|)^{|P_1|+|P_2|})$ $|x| \geq O(|P_1|*|P_2|)$

How to represent a strategy?

UNIVERSITY OF TRENTO

More efficient Strategy: only store the best move

Indexed by P_1 states $O(P_1)$ + P_1 initial move $O(1)$



Only store the best P_1 move that responds to the P_2 move that bring P_1 in that state

$$\underbrace{O(|P_1|+|P_2|)}_{\text{size of x}}^{\text{Moves}} \quad \text{vs} \quad \underbrace{O(P_1)}_{\text{size of y}}$$

24/09/2018 Massacci, Ngo - Complexity, Crypto, and FinTech ► 33

Bottom-up construction

UNIVERSITY OF TRENTO

- **Idea behind the proof**
  - If P_2 brings me in this state, I should do this
  - When this happens is immaterial in this representation
- **The algorithm works bottom up:**
  - Start from P_2 positions where P_1 wins (0-wins)
  - Find P_1 moves that bring us to 0-steps wins
  - Mark P_1 departing states as winning states for P_1 (1-wins)
  - Find P_2 states where every P_2 moves goes to 1-wins
  - mark those P_2 states as winning states (2-wins)
  - Etc.

24/09/2018 Massacci, Ngo - Complexity, Crypto, and FinTech ► 34

Bottom-up construction

- **Usually best idea to find polytime solution**
  - Start from "local" solution expanding to a global solution
  - Another example is Dijkstra shortest path
- **Algorithm in the general form**
  - Wins = W_0 // sol = $\varnothing$
  - while Wins changes
  - for each $<p_1,p_2>\in$Moves // your moves
  - if $p_2\in$Wins then Wins $\leftarrow$Wins $\cup$ $\{p_1\}$
  - // sol $\leftarrow$ sol$\cup\{<p_1,p_2>\}$
  - for each p_2 // positions of the adversary
  - if (for each $<p_2,p_1>\in$ Moves. $p_1\in$Wins) then
  - Wins$\leftarrow$Wins $\cup$ $\{p_2\}$
  - return sol iff s$\in$Wins

Problem Representation is critical

- **Problem Representation makes a difference between efficiently solvable and NOT efficiently solvable**

Problem	Explicit	Implicit
Winning Strategy in 2-player Game	Positions are integers Moves as a table of pairs of positions Winning Positions as a list of integers	Positions are binary Circuit tells if move between two positions valid Circuit tells if position is winning
Finding an Eulerian path	Vertices are integers Edges is a table of pairs of vertices	Vertices are binary Circuit tells if two vertices are connected by an edge

Checkable Search Problems?

UNIVERSITY OF TRENTO

- **Informally:**
  - valid solutions can be efficiently recognized.
- **Formally**
  - Given an instance x of the problem R and a candidate solution y, efficiently determine whether or not y is a valid solution for x (i.e. $y \in R(x)$ i.e. $(x,y) \in R$)
- **Important Note**
  - we decide membership of given pairs of the form (x, y) in a fixed relation R
  - Different from deciding membership of x in the set
  - $SR = \{x : R(x) \neq \emptyset\}$

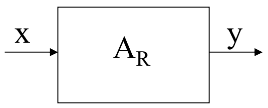
PC (Polynomial-time Check)

UNIVERSITY OF TRENTO

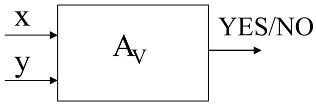
- **PC: class of efficiently checkable search problems**
- **$R \in PC$ if the following two conditions hold:**
  - Every x that has a solution in R only has short solutions
  - There exists an efficient algorithm that given an input x and a solution y determines whether or not $(x, y) \in R$.
- **Def. 2.3 (search prob. with efficiently checkable sol.) – Goldreich Book**
- **Search problem $R \subseteq \{0,1\}^* \times \{0,1\}^*$ has efficiently checkable solutions iff**
  - R is a polynomially bounded relation
  - there exists a polynomial time algorithm A s. t.,
  - for every x and y, $A(x,y) = 1$ iff $(x,y) \in R$

PC vs PF as "black-boxes"

UNIVERSITY OF TRENTO



R is in PF
IF for all x,
exists A_R working in poly time
outputting <u>some</u> y
(or $\perp$ if no solution exists)



R is in PC
IF for all x, <u>all</u> y
exists A_V working in poly time
telling me if y is really a solution

Problems in PC

UNIVERSITY OF TRENTO

- **Find an Hamiltonian Path in a Graph**
  - Input: Graph <V,E,s,t> where E ⊆ V×V and
    - s∈ V – source vertex, t ∈ V – target vertex
  - Output: sequence <v1,…,vn> ∈ V* s.t.
    - n=|V| and ∪i=1n {vi} = V and
    - <vi,vi+1> ∈ E for all i=1…n-1
    - v1=s and vn =t
- **Find a Coloring of a Graph with at most k Colors**
  - Input: Graph <V,E,k> where E⊆ V×V and k – number of colors
  - Output: association [<v1, c1>,…, <vn, cn>] ⊆ V×{0…k-1} s.t.
    - n=|V| and ∪i=1n {vi} = V and
    - If <vi,vj> ∈ E then ci≠ cj for all i,j

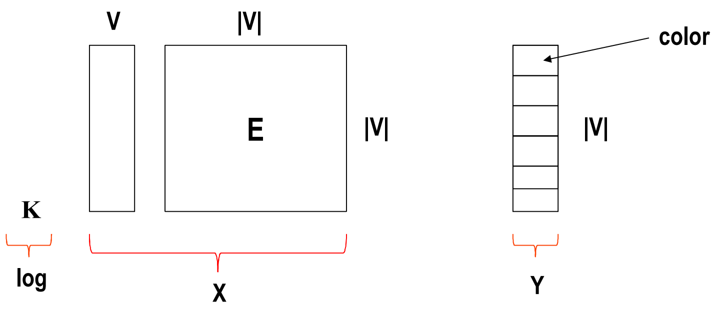
Discrete Logaritm Modulo a prime (IV)

- **How is R_{DLOG} defined**
  - {<(p,g,n),k>} | g^k = n (mod p) AND p is prime AND g is a generator of Z_p }
- **Is R_{DLOG} in PC? Simple algorithm**
  - x=1
  - For i=1 to k
    x=x*g;
  - If x=n return (0) else return (0);
- **Does it work?**

Discrete Logaritm Modulo a prime (IV)

- **How is R_{DLOG} defined**
  - {<(p,g,n),k>} | g^k = n (mod p) AND p is prime AND g is a generator of Z_p }
- **Is R_{DLOG} in PC? Simple algorithm**
  - x=1
  - For i=1 to k
    x=x*g;
  - If x=n return (0) else return (0);
- **Does it work? NOT really**
  - takes O(k)=O(p) but input is O(log p) → exponential!
  - Actual algorithm uses square and multiply → see lecture

Graph Coloring – Polynomial Bound



$O(n + n^2)$ if K fixed

$O(n + n^2 + \log K)$ if K variable

$O(n)$

$O(n*\log K)$

Checking Algorithm for Graph Coloring

$O(n^2)$

- for all <u,v> ∈ E
-   if color(u) ≠ color(v)
-     OK
-   else return (0)
- endfor
- return(1)

Checking Algorithm for Graph Coloring

$O(n^2)$

- for all <u,v> $\in$ E
- if color(u) $\neq$ color(v)
- OK
- else return (0)
- endfor
- return (1)

$O(n)$

- for all u $\in$ V
- if color(u) < k
- OK
- else return (0)
- endfor
- return (1)

Discrete Log

Search Problems - summary

- **Poly-bounded relation (i.e. potentially solvable)**
  - R s.t. ∃poly p ∀x. $(x,y) \in R \rightarrow |y| < p(|x|)$
  - problems that could be potentially solved
- **Poly-time FIND PF**
  - R s.t. Poly-bounded + ∃poly algorithm A s.t.
  - $A(x) \in R(x)$ OR $A(x)=\perp$ if $R(x)= \emptyset$

    The solutions for x
  - possible to find at least one solution efficiently
- **Poly-time CHECK PC**
  - R s.t. Poly-bounded + ∃poly algorithm A s.t.
  - (x,y) in R IFF A(x,y) = 1
  - Possible to verify all solutions efficiently

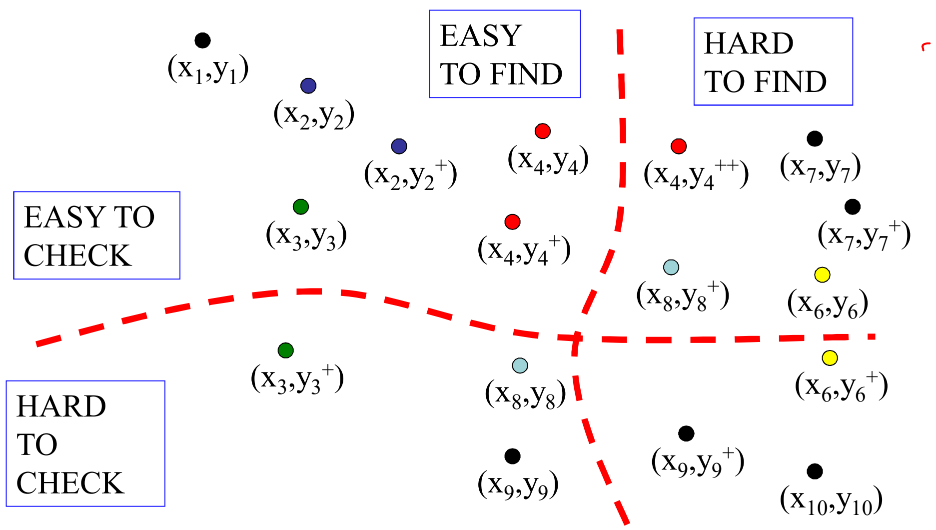
PC vs PF

UNIVERSITY OF TRENTO

- **Is every search problem in PC also in PF?**
  - If it is easy to check correctness of a given solution for a given instance, is it also easy to find a solution to a given instance?
- **If the answer is yes**
  - whenever solutions to given instances can be efficiently checked, such solutions can be efficiently found.
- **Formally what if PC $\subseteq$ PF ?**
  - If one can efficiently check the correctness of solutions wrt some (polynomially-bounded) relation R, then the search problem of R can also be solved efficiently

Exercise in class: draw PF vs PC relations (Colors problems belonging to same class)

UNIVERSITY OF TRENTO



Example of problem in PC\PF

- Real Zero of a polynomial of degree 5
- R={ ($ax^5+bx^4+cx^3+dx^2+ex+f$, ρ) |
  a,b,c,d,e,f ∈ ℝ AND ρ ∈ ℝ
  s.t. $a\rho^5+b\rho^4+c\rho^3+d\rho^2+e\rho+f=0$ }
- Frequency assignments
- Travelling salesman
- Fault detection in circuits
- Real Zero of a polynomial of degree 2
- R={ (ax^2+bx+c, ρ) | a,b,c ∈ ℝ AND ρ ∈ ℝ
  s.t. $a\rho^2+b\rho+c=0$ }

LIKELY IN PC\PF

IN PC and PF

PC ⊆ PF ? continued

- If PC ⊆PF
  - all reasonable search problems (all problems in PC) easy to solve.
  - Contradict the intuition that some reasonable search problems hard to solve.
- If PC\PF≠Ø:
  - exist search problems (in PC) hard to solve.
  - Conform to intuition that some reasonable problems easy to solve whereas others hard to solve.
- Confirm intuitive gap between solving and checking
  - (sometimes "solving" a lot harder than "checking").

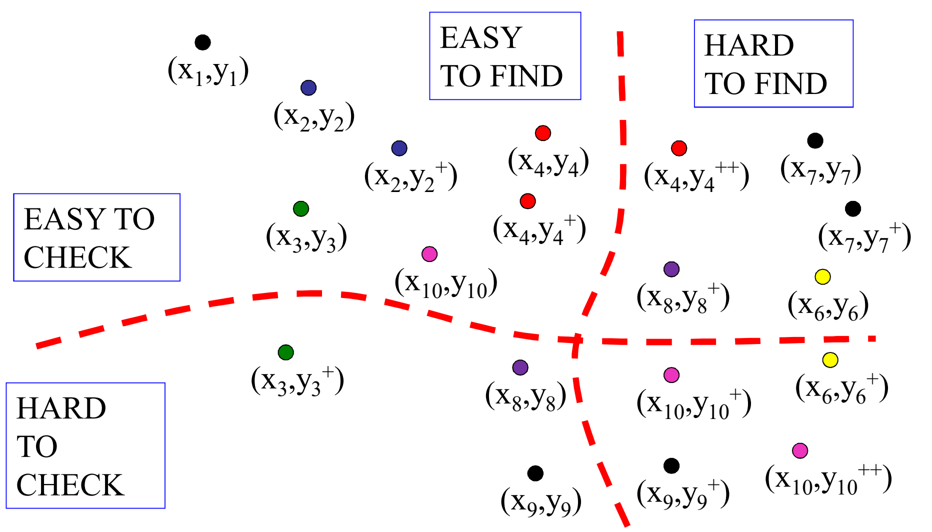
Assessment Exercise

UNIVERSITY OF TRENTO

- From discussion seems likely that PC⊄PF
- What about PF ⊆ PC?
- What if the inclusion is true?
  - Can we say something on how to transform a problem easy to find into a problem easy to check?
- What if the inclusion is false?
  - Can we say something on the problems in PF\PC?
- Discuss the issue

If PF ⊆ PC which points are impossible? If PC ⊆ PF which points are impossible?

UNIVERSITY OF TRENTO



Detour: Quantum Computing

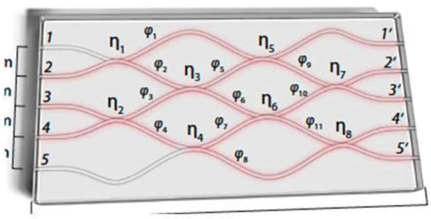
UNIVERSITY OF TRENTO

- Quantum Computing will solve all problems → well, sorry about that…
- More Classes
  - PSPACE = Problem solvable using only polynomial space
  - #P = Problem in which the output is the number of solutions
  - P^C = problems solvable in poly time with oracle access to a solver for problems in C
  - BQP = Problem solvable in Bounded Error Quantum Poly Time
    - Error at most 1/3
- What we know for sure
  - BQP $\subseteq P^{\#P} \subseteq$ PSPACE
  - IF we don't know the problem structure THEN Quantum only shorten time from checking all N possible solutions to $\sqrt{N}$
    - So this is too little to give any exponential speed-up in $N=2^n$ for input size n
  - Quantum Algorithms so far only solved problem in NP not known to be NP-complete
- Most likely
  - BQP *does not include* NP-complete
  - The source of the problem is measurement
    - Photons can interact in all crazy ways and therefore among all possible interactions there could be an answer to our problems BUT we can't observe most of those interactions

Example: Quantum seems super powerful…

UNIVERSITY OF TRENTO

Send identical photons to n x n beam splitter



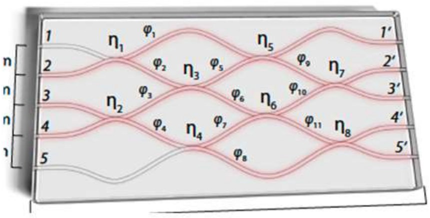
Computational Equivalent

- Probability that a photon exit in output j given input in Is proportional to the Permanent of the interaction matrix A
- $Perm(A) = \Sigma_{\sigma \in Sn} \Pi_{i=1}^{n} A_{i\sigma(i)}$
  - Exactly as the good old determinant except with don't have the ±1 in front
- Perm $\in$ #P but NP $\subseteq$ #P
  - So it is a super hard problem…
- IF nature solves automatically such a hard problem THEN Just build a quantum device…

Example – Harsh Reality

UNIVERSITY OF TRENTO

Send identical photons to n x n beam splitter



Computational Equivalent

- The problem
  - Probability that a photon exit in output j given input in Is proportional to the Permanent of the interaction matrix A
  - Perm ∈ #P ← super hard problem
- IF nature solves automatically such a hard problem …
- Except that it doesn't…
  - Nature does NOT give in output the probability, it only gives us the photons, distributed with this probability
- We cannot "measure" the probability, we can only simulate it with exponentially many trials
  - Send photons and measure where they end out of the box
  - Estimate prob by running (exponentially) many trials (to limit error)

References

UNIVERSITY OF TRENTO

- **Oded Goldreich.**
  - Computational Complexity:A Conceptual Perspective.
  - Cambridge University Press, 2008.
  - Electronic edition once available on the web.
  - Chapter:1.2
- **Sanjeev Arora and Boaz Barak.**
  - Computational Complexity: A Modern Approach.
  - MIT Press, 2008.
  - To appear. Available on the web.
  - Chapter:1.2, 1.3, 1.4