Algorithms (III)

NP Problem

Guoqiang Li

School of Software, Shanghai Jiao Tong University

Homework

- Assignment 1 is announced! (deadline Mar. 23)

NP-Complete Problems

Set Cover

Set Cover

- Input: A set of elements B, sets $S_1, \ldots, S_m \subseteq B$
- Output: A selection of the S_i whose union is B.
- Cost: Number of sets picked.

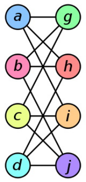
Graph Isomorphism

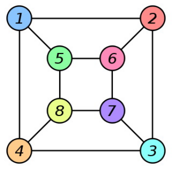
Graph Isomorphism

An isomorphism of graphs G and H is a bijection between the vertex sets of G and H

$$f : V(G) \to V(H)$$

such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H.





Hard Problems, Easy Problems

Hard problems (NP-complete)	Easy problems (in P)
3SAT	2SAT, Horn SAT
Traveling salesman problem	Minimum spanning tree
Longest path	Shortest path
3D matching	Bipartite matching
Knapsack	Unary knapsack
Independent set	Independent set on trees
Integer linear programming	Linear programming
Rudrata path	Euler path
Balanced cut	Minimum cut

NP

- Recall a search problem is defined by:
  1. There is an efficient checking algorithm C that takes as input the given `instance` I, as well as the proposed `solution` S, and outputs `true` if and only if S really is a `solution` to `instance` I.
  2. Moreover, the running time of $C(I, S)$ is bounded by a polynomial in $|I|$, the length of the `instance`.
- We denote the class of all search problems by NP.

P

- We've seen many examples of NP search problems that are solvable in polynomial time.
- In such cases, there is an algorithm that takes as input an `instance` I and has a running time polynomial in $|I|$.
  - If I has a `solution`, the algorithm returns such a `solution`;
  - and if I has no `solution`, the algorithm correctly reports so.
- The class of all search problems that can be solved in polynomial time is denoted P.

Why P and NP

- P: polynomial time
- NP: nondeterministic polynomial time

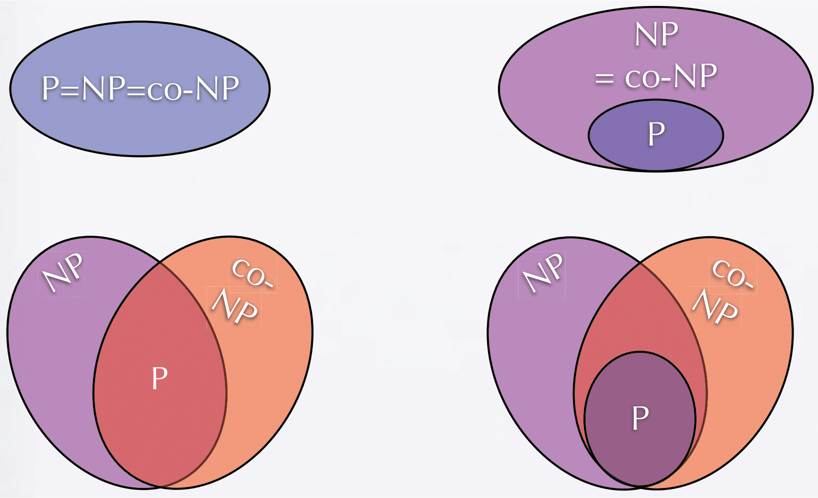
Complementation

- A class of problems $\mathcal{C}$ is closed under complementation if for any problem in $\mathcal{C}$, its complement is also in $\mathcal{C}$.
- P: is closed under complementation.
- NP ?

Example (Complementation of TSP)

Given n cities with their intercity distances, is it the case that there does not exist any tour length k or less?

Conjectures



$\mathrm{P} \neq \mathrm{NP}$

Theorem Proving

- Input: A mathematical statement φ and n.
- Problem: Find a proof of φ of length $\leq n$ if there is one.

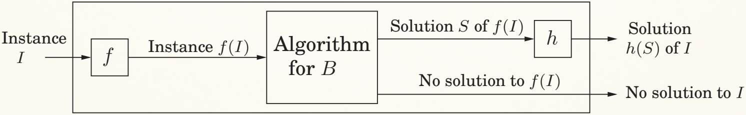
- The task of finding a proof for a given mathematical assertion is a search problem and is therefore in NP.
- Because a formal proof of a mathematical statement is written out in excruciating detail, it can be checked mechanically, line by line, by an efficient algorithm.
- So if $\mathrm{P} = \mathrm{NP}$, there would be an efficient method to prove any theorem, thus eliminating the need for mathematicians!

Solve One and All Solved

- Even if we believe $P \neq NP$, what about the specific problems on the left side of the table?
- On the basis of what evidence do we believe that these particular problems have no efficient algorithm?
- Such evidence is provided by reductions, which translate one search problem into another.
- We will show that the problems on the left side of the table are all, in some sense, exactly the same problem, the hardest search problems in NP.
- If even one of them has a polynomial time algorithm, then every problem in NP has a polynomial time algorithm.

Reduction Between Search Problems

- A reduction from search problem A to search problem B is a polynomial time algorithm f that transforms any `instance` I of A into an `instance` $f(I)$ of B,
- Together with another polynomial time algorithm h that maps any solution S of $f(I)$ back into a solution $h(S)$ of I .
- If $f(I)$ has no solution, then neither does I.
- These two translation procedures f and h imply that any algorithm for B can be converted into an algorithm for A by bracketing it between f and h.



The Two Ways to Use Reductions

- Assume there is a reduction from a problem A to a problem B.

$$A \to B$$

  - If we can solve B efficiently, then we can also solve A efficiently.
  - If we know A is hard, then B must be hard too.
- Reductions also have the convenient property that they compose.
  - If $A \to B$ and $B \to C$, then $A \to C$.

NP-Completeness

Definition

A NP problem is NP-complete if all other NP problems reduce to it.

- For a problem to be NP-complete, it must be useful in solving every NP problem in the world!
- It is remarkable that such problems exist.
- But they do, and the first column of the table we saw earlier is filled with the most famous examples.

Reductions to NP-Complete

- We know NP-complete problems are hard: all other search problems reduce to them.
- Each NP-complete problems contains the complexity of all search problems.
- If even one NP-complete problem is in P, then P = NP.
- If a problem A is NP-complete, we can use it to prove that a new search problem B is also NP-complete, simply by reducing A to B.
- Such a reduction establishes that all problems in NP reduce to B, via A.

Co-NP-Completeness

Definition

A co-NP problem is co-NP-complete if all other co-NP problems reduce to it.

A problem is NP-complete if and only if its complement is co-NP-complete.

If a problem and its complement are NP-complete then co-NP = NP.

Tautology

A CNF formula f is unsatisfiable if and only if its negation is a tautology. The negation of a CNF formula can be converted into a DNF formula. The resulting DNF formula is a tautology if and only if the negation of the CNF formula is a tautology.

The problem Tautology: Given a formula f in DNF, is it a tautology?

- Tautology is in P if and only if co-NP = P, and
- Tautology is in NP if and only if co-NP = NP.

Factoring

- The difficulty of Factoring is of a different nature than that of the other hard search problems we have just seen.
- For example, nobody believes that Factoring is NP-complete.
- One evidence is that a number can always be factored into primes.
- Another difference: Factoring succumbs to the power of quantum computation, while SAT, TSP and the other NP-complete problems do not seem to.
- Q: Is Factoring a NP?
- Q: How about its complementation?

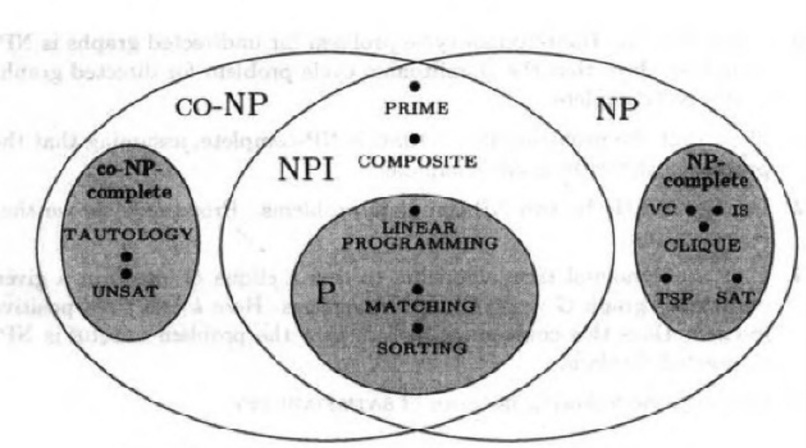
NPI

Primality
Given an integer $k \geq 2$, is k a prime number?

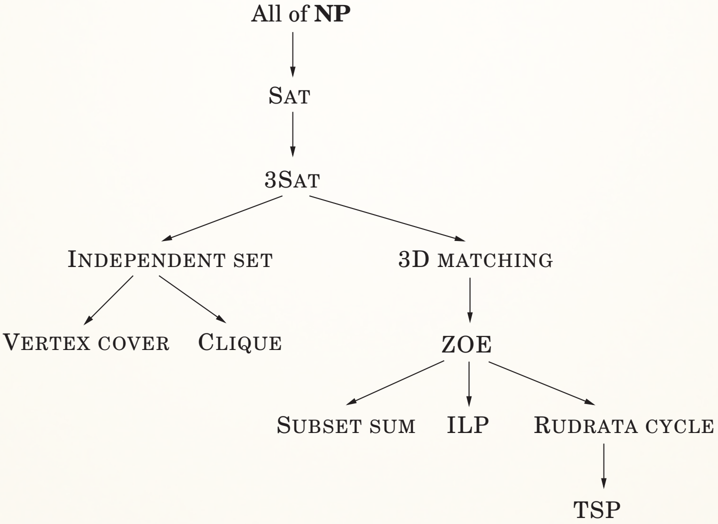
Composite
Given an integer $k \geq 4$, are there two integers $p, q \geq 2$ such that $k = pq$?

Such class is called NPI, standing for NP-Intermediate.

NPI (A Problematic Category)



The Reductions

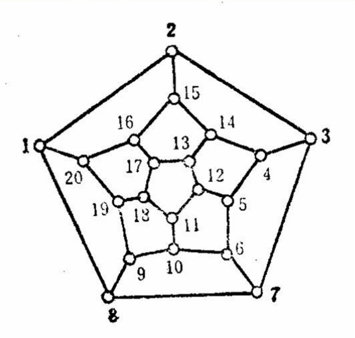


All of NP
SAT
3SAT
INDEPENDENT SET
3D MATCHING
VERTEX COVER
CLIQUE
ZOE
SUBSET SUM
ILP
RUDRATA CYCLE
TSP

1. Rudrata path $\rightarrow$ Rudrata cycle

Rudrata Cycle

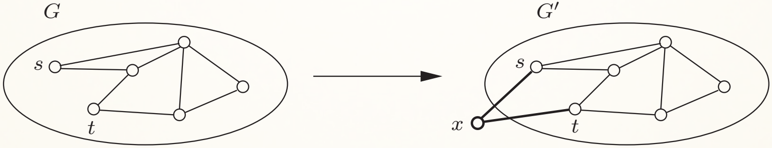
Rudrata Cycle: Given a graph, find a cycle that visits each vertex exactly once.



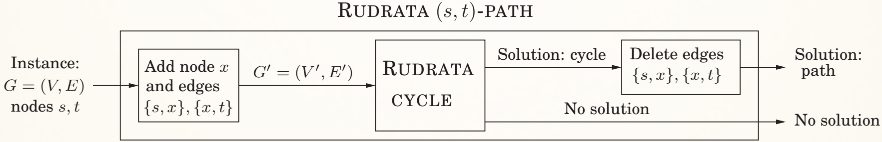
Rudrata (s, t)-path $\rightarrow$ Rudrata cycle

- We can also formulate the closely related Rudrata (s, t)-path problem, in which two vertices s and t are specified, and we want a path starting at s and ending at t that goes through each vertex exactly once.
- Q: Is it possible that Rudrata cycle is easier than Rudrata (s, t)-path?
    - We will show by a reduction that the answer is no.
- The reduction maps an `instance` ($G = (V, E)$) of Rudrata (s, t)-Path into an `instance` $G' = (V', E')$ of Rudrata cycle as follows: G' is simply G with an additional vertex x and two new edges $\{s, x\}$ and $\{x, t\}$.

Rudrata (s, t)-path $\rightarrow$ Rudrata cycle



Rudrata (s, t)-path $\rightarrow$ Rudrata cycle



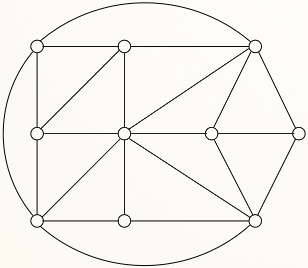
2. 3SAT $\rightarrow$ Independent set

3 SAT

The instances of 3SAT, is set of clauses(a Boolean formula in conjunctive normal form (CNF)), each with three or fewer literals,

$$(x \vee y \vee z)(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{x})(\overline{x} \vee \overline{y} \vee \overline{z})$$

Independent Set



Independent set: Given a graph G and an integer g, find g vertices, no two of which have an edge between them.

True Assignment

- To form a satisfying truth assignment we must pick one literal from each clause and give it the value `true`.
- But our choices must be consistent, if we choose $\bar{x}$ in one clause, we cannot choose x in another.
- Solution: put an edge between any two vertices that correspond to opposite literals.

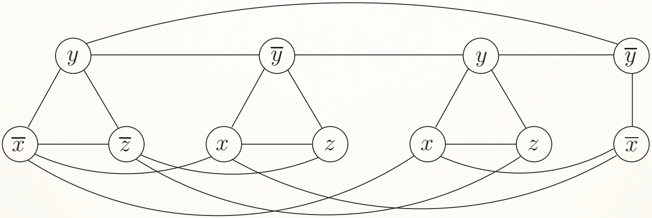
Clause

- Let us represent a clause, say $(x \vee \bar{y} \vee z)$, by a triangle, with vertices labeled $x, \bar{y}, z$.
- Because a triangle has its three vertices maximally connected, and thus forces us to pick only one of them for the independent set.

3SAT $\rightarrow$ Independent set

- Given an `instance` I of 3SAT, we create an `instance` (G, g) of Independent set as follows.
  - Graph G has a triangle for each clause (or just an edge, if the clause has two literals), with vertices labeled by the clause's literals, and
  - has additional edges between any two vertices that represent opposite literals.
  - The goal g is set to the number of clauses.

3SAT → Independent set



$(\overline{x} \vee y \vee \overline{z})(x \vee \overline{y} \vee z)(x \vee y \vee z)(\overline{x} \vee \overline{y})$

3. SAT $\rightarrow$ 3SAT

SAT $\rightarrow$ 3SAT

- This is an interesting and common kind of reduction, from a problem to a special case of itself.
- Given an `instance` I of SAT, use exactly the same `instance` for 3SAT, except that any clause with more than three literals, $(a_1 \vee a_2 \vee \ldots \vee a_k)$ (where the a_i's are literals and $k > 3$), is replaced by a set of clauses,

  $$(a_1 \vee a_2 \vee y_1)(\overline{y_1} \vee a_3 \vee y_2)(\overline{y_2} \vee a_4 \vee y_3)\ldots(\overline{y_{k-3}} \vee a_{k-1} \vee a_k)$$

  where the y_i's are new variables.
- Call the resulting 3SAT `instance` I'. The conversion from I to I' is clearly polynomial time.
- I' is equivalent to I in terms of satisfiability.

SAT $\rightarrow$ 3SAT

$$\left\{ \begin{array}{c} (a_1 \vee a_2 \vee \cdots \vee a_k) \\ \text{is satisfied} \end{array} \right\} \Longleftrightarrow \left\{ \begin{array}{c} \text{there is a setting of the } y_i\text{'s for which} \\ (a_1 \vee a_2 \vee y_1)\ (\overline{y}_1 \vee a_3 \vee y_2)\ \cdots\ (\overline{y}_{k-3} \vee a_{k-1} \vee a_k) \\ \text{are all satisfied} \end{array} \right\}$$

- Suppose that the clauses on the right are all satisfied. Then at least one of the literals $a_1, \ldots, a_k$ must be `true`.
- Otherwise y_1 would have to be `true`, which would in turn force y_2 to be `true`, and so on.
- Conversely, if $(a_1 \vee a_2 \vee \ldots \vee a_k)$ is satisfied, then some a_i must be `true`. Set $y_1, \ldots, y_{i-2}$ to `true` and the rest to `false`. This ensures that the clauses on the right are all satisfied.

SAT $\rightarrow$ 3SAT

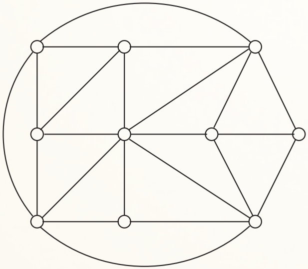
- In fact, 3SAT remains hard even under the further restriction that no variable appears in more than three clauses.
- Suppose that in the 3SAT `instance`, variable x appears in $k > 3$ clauses. Then replace its first appearance by x_1, its second appearance by x_2, and so on, replacing each of its k appearances by a different new variable.
- Finally, add the clauses

$$(\overline{x_1} \vee x_2)(\overline{x_2} \vee x_3) \ldots (\overline{x_k} \vee x_1)$$

- In the new formula no variable appears more than three times (and in fact, no literal appears more than twice).

4. Independent set $\rightarrow$ Vertex cover

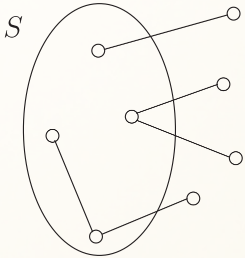
Vertex Cover



Vertex cover: Given a graph G and an integer b, find b vertices cover (touch) every edge.

Independent set $\rightarrow$ Vertex cover

- To reduce Independent set to Vertex cover we just need to notice that a set of nodes S is a vertex cover of graph $G = (V, E)$ (that is, S touches every edge in E) if and only if the remaining nodes, $V - S$, are an independent set of G.

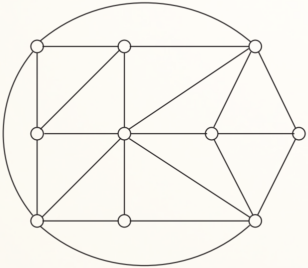


Independent set $\rightarrow$ Vertex cover

- Therefore, to solve an `instance` (G, g) of Independent set, simply look for a Vertex cover of G with $|V| - g$ nodes.
- If such a vertex cover exists, then take all nodes not in it. If no such vertex cover exists, then G cannot possibly have an independent set of size g.

5. Independent set $\rightarrow$ Clique

Clique



Clique: Given a graph G and an integer g, find g vertices such that all possible edges between them are present.

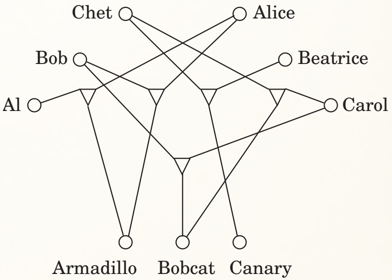
Independent set $\to$ Clique

- Define the complement of a graph $G = (V, E)$ to be $\overline{G} = (V, \overline{E})$, where $\overline{E}$ contains precisely those unordered pairs of vertices that are not in E.
- Then a set of nodes S is an independent set of G if and only if S is a clique of $\overline{G}$.
- To paraphrase, these nodes have no edges between them in G if and only if they have all possible edges between them in $\overline{G}$.
- Therefore, we can reduce Independent set to Clique by mapping an `instance` (G, g) of Independent set to the corresponding `instance` $(\overline{G}, g)$ of Clique.
- The solution to both is identical.

6. 3SAT $\rightarrow$ 3D matching

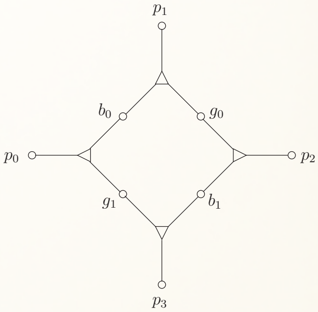
Three-Dimensional Matching

- 3D matching: There are n boys and n girls, but also n pets, and the compatibilities among them are specified by a set of triples, each containing a boy, a girl, and a pet.
- Intuitively, a triple (b, g, p) means that boy b, girl g, and pet p get along well together.
- We want to find n disjoint triples and thereby create n harmonious households.



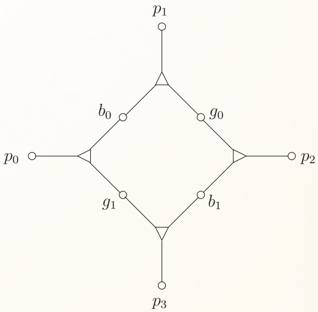
3SAT → 3D matching

- Consider a set of four triples, each represented by a triangular node joining a boy, girl, and pet:
  - Suppose that the two boys b_0 and b_1 and the two girls g_0 and g_1 are not involved in any other triples.
  - The four pets $p_0, \ldots, p_3$ will of course belong to other triples as well.
- Then any matching must contain either the two triples $(b_0, g_1, p_0), (b_1, g_0, p_2)$ or the two triples $(b_0, g_0, p_1), (b_1, g_1, p_3)$, because these are the only ways in which these two boys and girls can find any match.



3SAT → 3D matching

- Therefore, this "gadget" has two possible states: it behaves like a Boolean variable!
- To then transform an `instance` of 3SAT to one of 3D matching, we start by creating a copy of the preceding gadget for each variable x.
- Call the resulting nodes p_{x1}, b_{x0}, g_{x1}, and so on. The intended interpretation is that boy b_{x0} is matched with girl g_{x1} if $x = \texttt{true}$, and with girl g_{x0} if $x = \texttt{false}$.



3SAT $\rightarrow$ 3D matching

- Next we must create triples that somehow mimic clauses.
- For $c = (x \vee \bar{y} \vee z)$ we introduce a new boy b_c and a new girl g_c.
- They will be involved in three triples, one for each literal in the clause.
- And the pets in these triples must reflect the three ways whereby the clause can be satisfied:
  1. $x = \texttt{true}$,
  2. $y = \texttt{false}$,
  3. $z = \texttt{true}$.

3SAT $\rightarrow$ 3D matching

- For (1), we have the triple (b_c, g_c, p_{x1}), where p_{x1} is the pet p_1 in the gadget for x.
  - If $x = \texttt{true}$, then b_{x0} is matched with g_{x1} and b_{x1} with g_{x0}, and so pets p_{x0} and p_{x2} are taken. In which case b_c and g_c can be matched with p_{x1}.
  - But if $x = \texttt{false}$, then p_{x1} and p_{x3} are taken, and so g_c and b_c cannot be accommodated this way.
- We do the same thing for the other two literals of the clause, which yield triples involving b_c and g_c with either p_{y0} or p_{y2} (for the negated variable y) and with either p_{z1} or p_{z3} (for variable z).

3SAT $\rightarrow$ 3D matching

- We have to make sure that for every occurrence of a literal in a clause c there is a different pet to match with b_c and g_c.
- But this is easy: by an earlier reduction we can assume that no literal appears more than twice, and so each variable gadget has enough pets, two for negated occurrences and two for unnegated.
- The reduction now seems complete:
  - From any matching we can recover a satisfying truth assignment by simply looking at each variable gadget and seeing with which girl b_{x0} was matched.
  - From any satisfying truth assignment we can match the gadget corresponding to each variable x so that triples (b_{x0}, g_{x1}, p_{x0}) and (b_{x1}, g_{x0}, p_{x2}) are chosen if $x = \texttt{true}$, and Triples (b_{x0}, g_{x0}, p_{x1}) and (b_{x1}, g_{x1}, p_{x3}) are chosen if $x = \texttt{false}$;
  - For each clause c match b_c and g_c with the pet that corresponds to one of its satisfying literals.

3SAT $\rightarrow$ 3D matching

- But one last problem remains: In the matching defined so far, some pets may be left unmatched.
- In fact, if there are n variables and m clauses, then exactly $2n - m$ pets will be left unmatched.
- But this is easy to fix: Add $2n - m$ new boy-girl couples that are "generic animal-lovers", and match them by triples with all the pets!

7. 3D matching $\to$ ZOE

Zero-One Equations

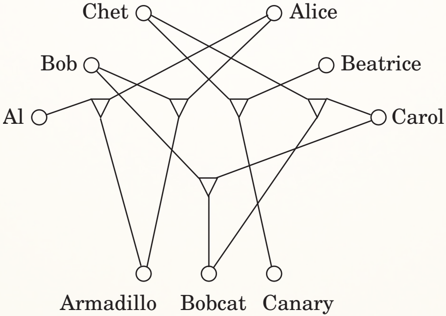
In ZOE (Zero-one equations) we are given an $m \times n$ matrix A with $0 - 1$ entries, and we must find a $0 - 1$ vector $\mathbf{x} = (x_1, \ldots, x_n)$ such that the m equations $A\mathbf{x} = 1$; are satisfied.

3D matching $\rightarrow$ ZOE

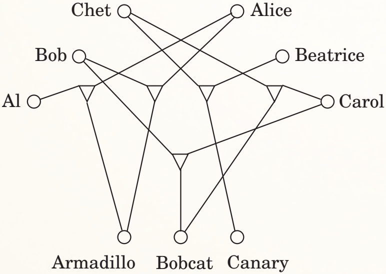
- Here is how we express an `instance` of 3D matching (m boys, m girls, m pets, and n boy-girl-pet triples) in the language of ZOE.
- We have $0 - 1$ variables, $x_1, \ldots, x_n$, one per triple, where $x_i = 1$ means that the i-th triple is chosen for the matching, and $x_i = 0$ means that it is not chosen.
- Now all we have to do is write equations stating that the solution described by the x_i's is a legitimate matching.
- For each boy (or girl, or pet), suppose that the triples containing him (or her, or it) are those numbered $j_1, j_2, \ldots, j_k$; the appropriate equation is then

$$x_{j_1} + x_{j_2} + \ldots + x_{j_k} = 1$$

3D matching → ZOE



3D matching → ZOE



$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

8. ZOE $\rightarrow$ Subset Sum

Subset Sum

Subset sum: Find a subset of a given set of integers that adds up to exactly W.

ZOE $\rightarrow$ Subset Sum

- This is a reduction between two special cases of ILP:
  - One with many equations but only $0-1$ coefficients;
  - The other with a single equation but arbitrary integer coefficients.
- The reduction is based on a simple and time-honored idea: $0-1$ vectors can encode numbers!
- We are looking for a set of columns of A that, added together, make up the all-1's vector.
- But if we think of the columns as binary integers (read from top to bottom), we are looking for a subset of the integers corresponding to the columns of A that add up to the binary integer $11\ldots1$.
- And this is an `instance` of Subset sum. The reduction seems complete!

An Example

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

ZOE $\to$ Subset Sum

- Except for one detail: carry.
- Because of carry, say 5-bit binary integers can add up to $11111 = 31$, for example, $5 + 6 + 20 = 31$ or, in binary,

$$00101 + 00110 + 10100 = 11111$$

  even when the sum of the corresponding vectors is not $(1, 1, 1, 1, 1)$.
- But this is easy to fix: Think of the column vectors not as integers in base 2, but as integers in base $n + 1$, one more than the number of columns.
- This way, since at most n integers are added, and all their digits are 0 and 1, there can be no carry, and our reduction works.

9. ZOE $\rightarrow$ ILP

Special Cases

- 3SAT is a special case of SAT, or, SAT is a generalization of 3SAT.
- By special case we mean that the `instances` of 3SAT are a subset of the `instances` of SAT.
- Consequently, there is a reduction from 3SAT to SAT, in which the input undergoes no transformation, and the solution to the target instance is also kept unchanged.
- In other words, functions f and h from the reduction diagram are both the identity.
- It is a very useful and common way of establishing that a problem is NP-complete: Simply notice that it is a generalization of a known NP-complete problem. For example, the Set cover problem is NP-complete because it is a generalization of Vertex cover.

ZOE $\rightarrow$ ILP

- In ILP we are looking for an integer vector $\mathbf{x}$ that satisfies $A\mathbf{x} \leq b$, for given matrix A and vector b.
- To write an `instance` of ZOE in this precise form, we need to rewrite each equation of the ZOE `instance` as two inequalities, and to add for each variable x_i the inequalities $x_i \leq 1$ and $-x_i \leq 0$.

10. ZOE $\rightarrow$ Rudrata cycle

ZOE $\rightarrow$ Rudrata cycle

- In Rudrata cycle we seek a cycle in a graph that visits every vertex exactly once.
- In ZOE, we are given an $m \times n$ matrix A with $0 - 1$ entries, and we must find a $0 - 1$ vector $\mathbf{x} = (x_1, \ldots, x_n)$ such that the m equations $A\mathbf{x} = 1$; are satisfied.

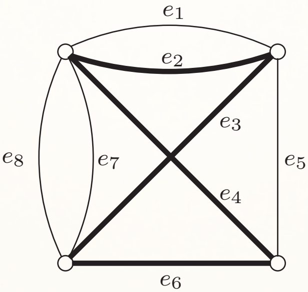
ZOE $\to$ Rudrata cycle

- We shall prove it NP-complete in two stages:
  1. First we will reduce ZOE to a generalization of Rudrata cycle, called Rudrata cycle with Paired edges.
  2. Then we shall see how to get rid of the extra features of that problem and reduce it to the plain Rudrata cycle.

Rudrata Cycle with Paired Edges

- We are given a graph $G = (V, E)$ and a set $C \subseteq E \times E$ of pairs of edges. We seek a cycle that
  1. visits all vertices once,
  2. for every pair of edges (e, e') in C, traverses either edge e or edge e' exactly one of them.
- Notice that we allow two or more parallel edges between two nodes, since now the different copies of an edge can be paired with other copies of edges in ways that do make a difference.

An example

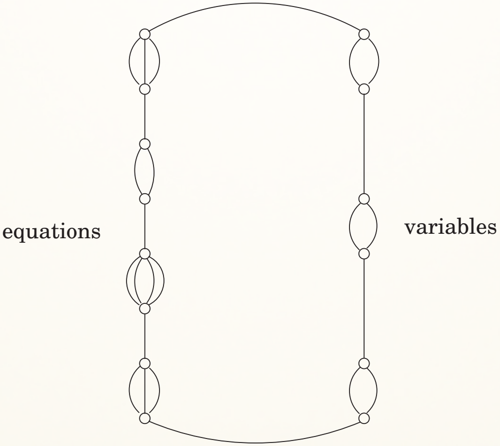


$$C = \{(e_1, e_3), (e_5, e_6), (e_4, e_5), (e_3, e_7), (e_3, e_8)\}$$

ZOE $\rightarrow$ Rudrata cycle with Paired edges

- Given an `instance` of ZOE, $A\mathbf{x} = 1$ (where A is an $m \times n$ matrix with $0 - 1$ entries, and thus describes m equations in n variables), the graph we construct has the very simple structure:
  - A cycle that connects $m + n$ collections of parallel edges.
  - For each variable x_i we have two parallel edges (corresponding to $x_i = 1$ and $x_i = 0$).
  - for each equation $x_{j_1} + \ldots + x_{j_k} = 1$ involving k variables we have k parallel edges, one for every variable appearing in the equation.
- Any Rudrata cycle must traverse the $m + n$ collections of parallel edges one by one, choosing one edge from each collection. This way, the cycle "chooses" for each variable a value 0 or 1 and, for each equation, a variable appearing in it.

ZOE $\rightarrow$ Rudrata cycle with Paired edges



ZOE $\to$ Rudrata cycle with Paired edges

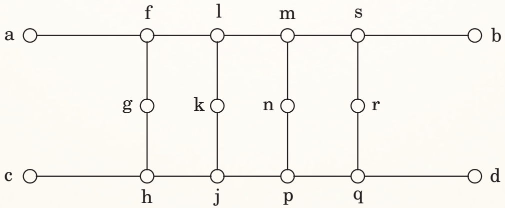
- The structure of the matrix A (and not just its dimensions) must be reflected somewhere, and there is one place left: the set C of pairs of edges such that exactly one edge in each pair is traversed.
- For every equation (recall there are m in total), and for every variable x_i appearing in it, we add to C the pair (e, e') where e is the edge corresponding to the appearance of x_i in that particular equation, and e' is the edge corresponding to the variable assignment $x_i = 0$.
- This completes the construction.

ZOE $\rightarrow$ Rudrata cycle with Paired edges

- Take any `solution` of this `instance` of Rudrata cycle with Paired edges. It picks a value for each variable and a variable for every equation. We claim that the values thus chosen are a `solution` to the original `instance` of ZOE.
- If a variable x_i has value 1, then the edge $x_i = 0$ is not traversed, and thus all edges associated with x_i on the equation side must be traversed, since they are paired in C with the $x_i = 0$ edge.
- So, in each equation exactly one of the variables appearing in it has value 1, i.e., all equations are satisfied.
- The other direction is straightforward as well: from a `solution` to the `instance` of ZOE one easily obtains an appropriate Rudrata cycle.

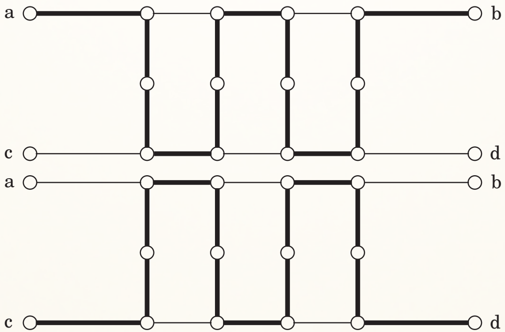
Get Rid of Edge Pairs

- Consider the graph, and suppose that it is a part of a larger graph G in such a way that only the four endpoints a, b, c, d touch the rest of the graph.



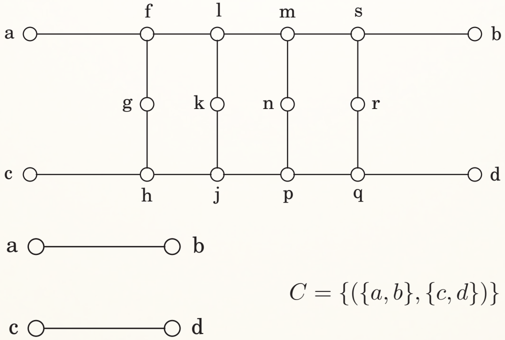
Get Rid of Edge Pairs

- We claim that this graph has the following important property: in any Rudrata cycle of G the subgraph shown must be traversed in one of the two ways.



Get Rid of Edge Pairs

- This gadget behaves just like two edges $\{a, b\}$ and $\{c, d\}$ that are paired up in the Rudrate cycle with Paired edges.

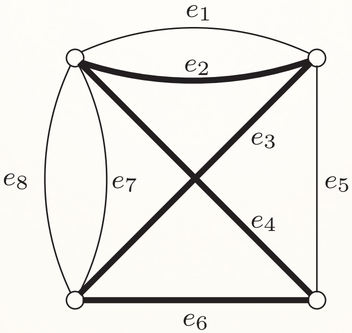


$$C = \{(\{a, b\}, \{c, d\})\}$$

Rudrata cycle with Paired edges$\rightarrow$ Rudrata cycle

- We go through the pairs in C one by one. To get rid of each pair $(\{a,b\},\{c,d\})$ we replace the two edges with the gadget.
- For any other pair in C that involves $\{a,b\}$, we replace the edge $\{a,b\}$ with the new edge $\{a,f\}$, where f is from the gadget: the traversal of $\{a,f\}$ is from now on an indication that edge $\{a,b\}$ in the old graph would be traversed.
- Similarly, $\{c,h\}$ replaces $\{c,d\}$.
- After $|C|$ such replacements (performed in polynomial time, since each replacement adds only 12 vertices to the graph) we are done.
- The Rudrata cycles in the resulting graph will be in one-to-one correspondence with the Rudrata cycles in the original graph that conform to the constraints in C.

An example



$C = \{(e_1, e_3), (e_5, e_6), (e_4, e_5), (e_3, e_7), (e_3, e_8)\}$

11. Rudrata cycle $\rightarrow$ TSP

Rudrata cycle $\rightarrow$ TSP

- Given a graph $G = (V, E)$, construct the following `instance` of the TSP:
  - The set of cities is the same as V.
  - The distance between cities u and v is 1 if $\{u, v\}$ is an edge of G and $1 + \alpha$ otherwise, for some $\alpha > 1$ to be determined.
  - The budget of the TSP `instance` is equal to the number of nodes, $|V|$.
- If G has a Rudrata cycle, then the same cycle is also a tour within the budget of the TSP `instance`.
- Conversely, if G has no Rudrata cycle, then there is no solution: the cheapest possible TSP tour has cost at least $n + \alpha$ (it must use at least one edge of length $1 + \alpha$, and the total length of all $n - 1$ others is at least $n - 1$).
- Thus Rudrata Cycle reduces to TSP.

Rudrata cycle $\to$ TSP

- If $\alpha = 1$, then all distances are either 1 or 2, and so this `instance` of the TSP satisfies the triangle inequality: if i, j, k are cities, then

$$d_{ij} + d_{jk} \geq d_{ik}$$

- This is a special case of the TSP which is of practical importance and which is in a certain sense easier, because it can be efficiently approximated.

Rudrata cycle $\rightarrow$ TSP

- If α is large, then the resulting `instance` of the TSP may not satisfy the triangle inequality, but has another important property:
  - either it has a `solution` of cost n or less,
  - or all its `solutions` have cost at least $n + \alpha$ (which now can be arbitrarily larger than n).
- This important gap property implies that, unless P = NP, no approximation algorithm is possible.

12. Any Problem $\rightarrow$ SAT

home reading!

Referred Materials

- Content of this lecture comes from Section 8.2, 8.3 in [DPV07], and Section 34.2, 34.3 in [CLRS09].