CS21
Decidability and Tractability

Lecture 21

February 26, 2014

Outline

- the class NP
  - NP-complete problems: Hamilton path and cycle,Traveling Salesperson Problem
  - NP-complete problems: subset sum

Hamilton Path

- Definition: given a directed graph G = (V, E), a Hamilton path in G is a directed path that touches every node exactly once.

- A language (decision problem):

  HAMPATH = {(G, s, t) : G has a Hamilton path from s to t}

HAMPATH is NP-complete

Theorem: the following language is NP-complete:

HAMPATH = {(G, s, t) : G has a Hamilton path from s to t}

- Proof:
  - Part 1: HAMPATH $\in$ NP. Proof?
  - Part 2: HAMPATH is NP-hard.
    - reduce from?

HAMPATH is NP-complete

- We are reducing from the language:

3SAT = { φ : φ is a 3-CNF formula that has a satisfying assignment }

to the language:

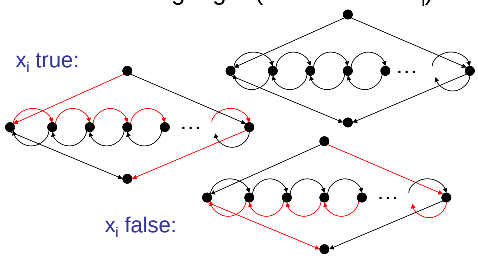
HAMPATH = {(G, s, t) : G has a Hamilton path from s to t}

HAMPATH is NP-complete

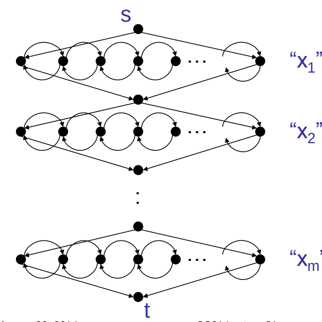
- We want to construct a graph from φ with the following properties:
  - a satisfying assignment to φ translates into a Hamilton Path from s to t
  - a Hamilton Path from s to t can be translated into a satisfying assignment for φ
- We will build the graph up from pieces called gadgets that "simulate" the clauses and variables of φ.

HAMPATH is NP-complete

- The variable gadget (one for each x_i):



HAMPATH is NP-complete



- path from s to t translates into a truth assignment to $x_1 \ldots x_m$
- why must the path be of this form?

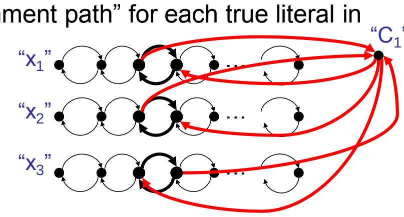
HAMPATH is NP-complete

$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots \wedge (\ldots)$

- How to ensure that all k clauses are satisfied?
- need to add nodes
  - can be visited in path if the clause is satisfied
  - if visited in path, implies clause is satisfied by the assignment given by path through variable gadgets

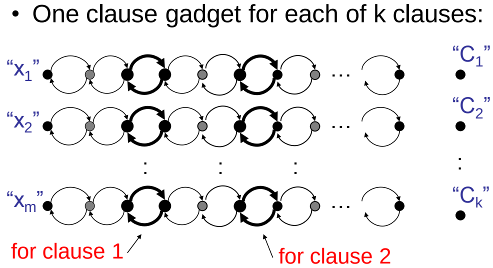
HAMPATH is NP-complete

- $\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots \wedge (\ldots)$
- Clause gadget allows “detour” from “assignment path” for each true literal in clause



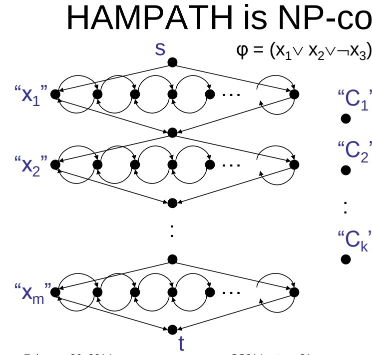
HAMPATH is NP-complete

- One clause gadget for each of k clauses:



HAMPATH is NP-complete

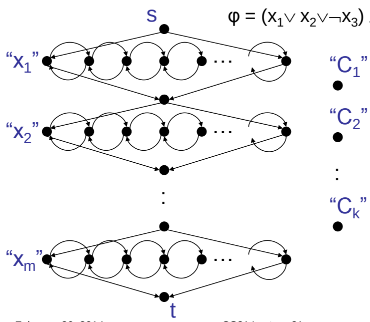
$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots$



- $f(\varphi)$ is this graph (edges to/from clause nodes not pictured)
- f poly-time computable?
  - # nodes = O(km)

HAMPATH is NP-complete

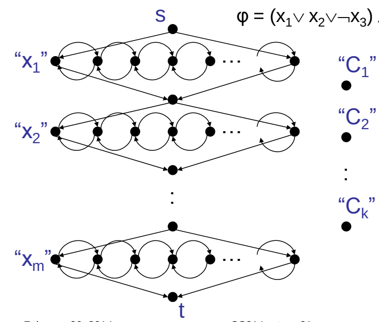
$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots$



- YES maps to YES?
  - first form path from satisfying assign.
  - pick true literal in each clause and add detour

HAMPATH is NP-complete

$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots$

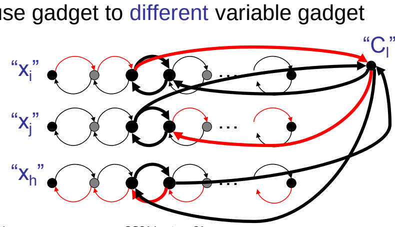


- NO maps to NO?
  - try to translate path into satisfying assignment
  - if path has "intended" form, OK.

HAMPATH is NP-complete

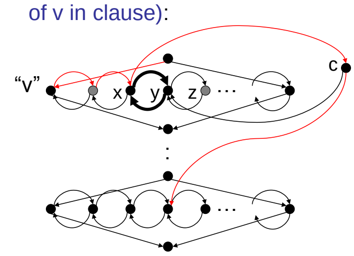
- What can go wrong?
  - path has "intended form" unless return from clause gadget to different variable gadget

we will argue that this cannot happen:



HAMPATH is NP-complete

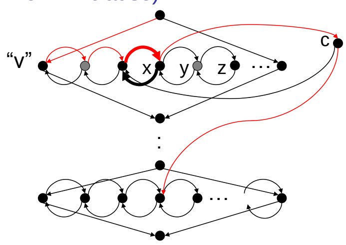
Case 1 (positive occurrence of v in clause):



- path must visit y
- must enter from x, z, or c
- must exit to z (x is taken)
- x, c are taken. can't happen

HAMPATH is NP-complete

Case 2 (negative occurrence of v in clause):



- path must visit y
- must enter from x or z
- must exit to z (x is taken)
- x is taken. can't happen

Undirected Hamilton Path

- HAMPATH refers to a directed graph.
- Is it easier on an undirected graph?
- A language (decision problem):

  UHAMPATH = {(G, s, t) : undirected G has a Hamilton path from s to t}

UHAMPATH is NP-complete

Theorem: the following language is NP-complete:

UHAMPATH = {(G, s, t) : undirected graph G has a Hamilton path from s to t}

- Proof:
  - Part 1: UHAMPATH $\in$ NP. Proof?
  - Part 2: UHAMPATH is NP-hard.
    - reduce from?

UHAMPATH is NP-complete

- We are reducing from the language:

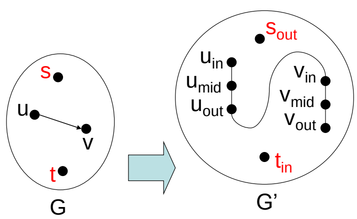
HAMPATH = {(G, s, t) : directed graph G has a Hamilton path from s to t}

to the language:

UHAMPATH = {(G, s, t) : undirected graph G has a Hamilton path from s to t}

UHAMPATH is NP-complete

- The reduction:



- replace each node with three (except s, t)
- (u_{in}, u_{mid})
- (u_{mid}, u_{out})
- (u_{out}, v_{in}) iff G has (u,v)

UHAMPATH is NP-complete

- Does the reduction run in poly-time?
- YES maps to YES?
  - Hamilton path in G: $s, u_1, u_2, u_3, \ldots, u_k, t$
  - Hamilton path in G':

$s_{out}, (u_1)_{in}, (u_1)_{mid}, (u_1)_{out}, (u_2)_{in}, (u_2)_{mid}, (u_2)_{out}, \ldots (u_k)_{in}, (u_k)_{mid}, (u_k)_{out}, t_{in}$

UHAMPATH is NP-complete

- NO maps to NO?
  - Hamilton path in G':

    $s_{out}, v_1, v_2, v_3, v_4, v_5, v_6, \ldots, v_{k-2}, v_{k-1}, v_k, t_{in}$
  - $v_1 = (u_{i1})_{in}$ for some i_1 (only edges to ins)
  - $v_2 = (u_{i1})_{mid}$ for some i_1 (only way to enter mid)
  - $v_3 = (u_{i1})_{out}$ for some i_1 (only way to exit mid)
  - $v_1 = (u_{i2})_{in}$ for some i_2 (only edges to ins)
  - ...
  - Hamilton path in G: $s, u_{i1}, u_{i2}, u_{i3}, \ldots, u_{ik}, t$

Undirected Hamilton Cycle

- Definition: given a undirected graph G = (V, E), a Hamilton cycle in G is a cycle in G that touches every node exactly once.
- Is finding one easier than finding a Hamilton path?
- A language (decision problem):

  UHAMCYCLE = {G : G has a Hamilton cycle}

UHAMCYCLE is NP-complete

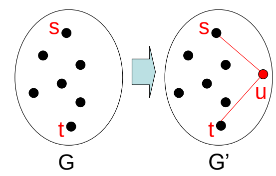
Theorem: the following language is NP-complete:

UHAMCYCLE = {G: G has a Hamilton cycle}

- Proof:
  - Part 1: UHAMCYCLE $\in$ NP. Proof?
  - Part 2: UHAMCYCLE is NP-hard.
    - reduce from?

UHAMCYCLE is NP-complete

- The reduction (from UHAMPATH):



- H. path from s to t implies H. cycle in G'
- H. cycle in G' must visit u via red edges
- removing red edges gives H. path from s to t in G

Traveling Salesperson Problem

- Definition: given n cities $v_1, v_2, \ldots, v_n$ and inter-city distances $d_{i,j}$ a TSP tour in G is a permutation π of {1…n}. The tour's length is $\Sigma_{i = 1 \cdots n} d_{\pi(i),\pi(i+1)}$ (where n+1 means 1).
- A search problem:
  given the $\{d_{i,j}\}$, find the shortest TSP tour
- corresponding language (decision problem):
  TSP = $\{(\{d_{i,j} : 1 \le i<j \le n\}, k)$: these cities have a TSP tour of length $\le k\}$

TSP is NP-complete

Theorem: the following language is NP-complete:

TSP = $\{(\{d_{i,j} : 1 \le i<j \le n\}, k)$: these cities have a TSP tour of length $\le k\}$

- Proof:
  - Part 1: TSP $\in$ NP. Proof?
  - Part 2: TSP is NP-hard.
    - reduce from?

TSP is NP-complete

- We are reducing from the language:

UHAMCYCLE = {G : G has a Hamilton cycle}

to the language:

TSP = $\{(\{d_{i,j} : 1 \le i<j \le n\}, k)$: these cities have a TSP tour of length $\le k\}$

TSP is NP-complete

- The reduction:
  - given G = (V, E) with n nodes

  produce:
  - n cities corresponding to the n nodes
  - $d_{u,v} = 1$ if $(u, v) \in E$
  - $d_{u,v} = 2$ if $(u, v) \notin E$
  - set k = n

TSP is NP-complete

- YES maps to YES?
  - if G has a Hamilton cycle, then visiting cities in that order gives TSP tour of length n
- NO maps to NO?
  - if TSP tour of length ≤ n, it must have length exactly n.
  - all distances in tour are 1. Must be edges between every successive pair of cities in tour.

Subset Sum

- A language (decision problem):

  SUBSET-SUM = {(S = {a_1, a_2, a_3, …, a_k}, B) : there is a subset of S that sums to B}

- example:
  - S = {1, 7, 28, 3, 2, 5, 9, 32, 41, 11, 8}
  - B = 30
  - 30 = 7 + 3 + 9 + 11. yes.

Subset Sum

SUBSET-SUM = {(S = {a_1, a_2, a_3, …, a_k}, B) : there is a subset of S that sums to B}

- Is this problem NP-complete? in P?
- Problem set: in TIME(B · poly(k))

SUBSET-SUM is NP-complete

Theorem: the following language is NP-complete:

SUBSET-SUM = {(S … :
there is a subse…

our reduction had better produce super-polynomially large B (unless we want to prove P=NP)

- Proof:
  - Part 1: SUBSET-SU…
  - Part 2: SUBSET-SUM is NP-hard.
    - reduce from?

SUBSET-SUM is NP-complete

- We are reducing from the language:

3SAT = { φ : φ is a 3-CNF formula that has a satisfying assignment }

to the language:

SUBSET-SUM = {(S = {a_1, a_2, a_3, …, a_k}, B) : there is a subset of S that sums to B}

SUBSET-SUM is NP-complete

- $\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots \wedge (\ldots)$
- Need integers to play the role of truth assignments
- For each variable x_i include two integers in our set S:
  - x_i^{TRUE} and x_i^{FALSE}
- set B so that exactly one must be in sum

SUBSET-SUM is NP-complete

x_1^{TRUE} = 1 0 0 0 … 0
x_1^{FALSE} = 1 0 0 0 … 0
x_2^{TRUE} = 0 1 0 0 … 0
x_2^{FALSE} = 0 1 0 0 … 0
…
x_m^{TRUE} = 0 0 0 0 … 1
x_m^{FALSE} = 0 0 0 0 … 1
B = 1 1 1 1 … 1

- every choice of one from each (x_i^{TRUE},x_i^{FALSE}) pair sums to B
- every subset that sums to B must choose one from each (x_i^{TRUE},x_i^{FALSE}) pair