Polynomial transformations and reductions

Definition: Let $X_1 = (D_1, F_1)$, $X_2 = (D_2, F_2)$ be two feasibility problems.

Assume $\exists$ function $g: D_1 \longrightarrow D_2$ such that $\forall d_1 \in D_1$, $g(d_1) \in F_2 \Leftrightarrow d_1 \in F_1$

If g is computable in time that is polynomial in the length of the encoding of d_1 then X_1 is polynomially transformable to X_2.

Proposition: If X_1 is polynomially transformable to X_2 and $X_2 \in P$ then $X_1 \in P$.

Proof: Apply the algo for X_1: Compute g, then apply algo for X_2. //

Eg: i) X_1 is perfect matching feasibility problem on a bipartite graph, with each side of bipartition having same number of nodes

X_2 is max flow problem with lower bound.

Definition: Let $X_1 = (D_1, F_1)$, $X_2 = (D_2, F_2)$ be two feasibility problems.

ii) X_1 is same as example (i),

X_2 is perfect matching feasibility problem.

Then g is identity map.

Reducibility

X_1, X_2 feasibility problems

X_1 is polynomially reducible to X_2 if ∃ an algo A_1 for X_1 that uses an algo A_2 for X_2 as a subroutine, and ~~the algo~~ A_1 runs in poly time under the assumption that ~~algo A_2 takes~~ each call of the subroutine takes unit time.

Eg 1) Any transformation + subroutine is only ~~run once~~, on the transformed data $g(d_1)$.

~~2) X_1 : Optimization problem~~

Prop If X_1 is polynomially reducible to X_2 and $X_2 \in P$ then $X_1 \in P$.

Proof Real complexity of algo to solve X_1 is at most

$$p_1(n) \cdot p_2(p_1(n)),$$

where ~~$p_1(n)$ calculate~~ $p_1(n)$ = complexity of A_1 with assumption that each call of subroutine takes unit time

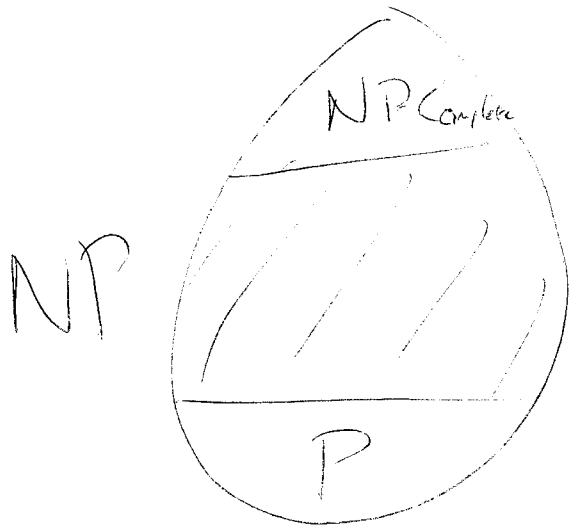
$p_2(n)$ is time required by A_2.

We invoke A_2 at most $p_1(n)$ times, each call of A_2 has input of length at most $p_1(n)$ since ~~that~~ we have to write the input down and that is an upper bound on how much we can write //.

Defn A feasibility problem $X \in NP$ is said to be **NP-complete** if all other problems in NP polynomially reduce to X.
(Note direction!)

Theorem If X is NP-complete and $X \in P$ then $P = NP$.

Proof Any problem in NP is polynomially reducible to X, and $X \in P$
$\therefore$ any problem in NP is in P, i.e. $NP \subseteq P$ //.



Do there exist NP-complete problems? Yes.

Def. The problem SATISFIABILITY (SAT):

A Boolean variable x, is a variable that can assume only the values true, false.

Boolean variables can be combined using or (denoted by $+$), and (by $\cdot$) not (denoted by $\bar{x}$) to form Boolean formulas:

eg $\bar{x}_3 \cdot (x_1 + \bar{x}_2 + x_3)$ For this to be true: need $\bar{x}_3$ true and (x_1 or $\bar{x}_2$ or x_3 true)

if $\begin{cases} x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false}. \\ \text{expression has value true} \end{cases}$

$\begin{cases} x_1 = \text{false}, x_2 = \text{true}, x_3 = \text{false}. \\ \text{expression has value false.} \end{cases}$

This formula is satisfiable since $\exists$ assignment of variables such that expression is true

Consider $x_1 \cdot \bar{x}_1$. This is not satisfiable.

Clauses are subformulas of the expression containing only ors and negations

eg: first expression $\bar{x}_3$ is a clause, $x_1 + \bar{x}_2 + x_3$ is a clause.

Literals: clauses are made up of literals; a literal is a variable or its negation.

Satisfiability problem:

Given m clauses $C_1, \ldots, C_m$ involving the variables $x_1, \ldots, x_n$, is the formula $C_1 \cdot C_2 \cdots C_m$ satisfiable?

Theorem (Cook)

Satisfiability is NP-complete

Idea of proof: Build a Turing machine that solves all ~~NP~~ problems in NP.

Then polynomially transform Turing machine into SAT.

Theorem If $X_1 \in$ is NP-complete and X_1 is polynomially reducible to $X_2 \in NP$, then X_2 is NP-complete.

Proof Obvious, since polynomially reducibility is a transitive property. //

Theorem The 0-1 integer programming feasibility problem is NP-complete.

"SAT is easier than 0-1 IP."

Proof We reduce SAT to 0-1 IP feas: "Show that any instance of SAT is equivalent to a 0-1 IP."

Given an instance of SAT:

m clauses $C_1, \ldots, C_m$

n Boolean variables $x_1, \ldots, x_n$

Define $y_i = \begin{cases} 1 & \text{if } x_i = \text{true} \\ 0 & \text{o/w} \end{cases}$ $\quad i = 1, \ldots, n$

Now $C_j = \sum_{i: x_i \text{ would be true to make } C_j \text{ true} \equiv C_j^+} x_i + \sum_{i: x_i \text{ would be false to make } C_j \text{ true} \equiv C_j^-} \bar{x}_i$

Consider constraint $\sum_{i \in C_j^+} y_i + \sum_{i \in C_j^-} (1 - y_i) \geq 1$.

Σ ... is bounded to a constraint //

Let 3-SAT be the problem where each instance is a satisfiability problem with each clause containing exactly 3 literals.

Thm 3-SAT is NP-complete (Leave proof as exercise) POS p. 359

Proof 3-SAT is in NP since it is a special case of SAT.

To show 3-SAT is NP-complete, we reduce SAT to 3-SAT:

Consider a clause $C_i = \lambda_1 + \lambda_2 + \cdots + \lambda_k$, each λ_j is a literal i.e. x_l or $\bar{x}_l$ for some l.

Assume $k > 3$.

Consider the clauses

$$\begin{aligned} &\lambda_1 + \lambda_2 + x_1 \\ &\bar{x}_1 + \lambda_3 + x_2 \\ &\bar{x}_2 + \lambda_4 + x_3 \\ &\vdots \\ &\bar{x}_{k-3} + \lambda_{k-1} + \lambda_k \end{aligned}$$

C_i holds iff we can choose $x_1, \ldots, x_{k-3}$ such that all these clauses hold

Eg if C_i holds because λ_5 is true set x_1, x_2, x_3 true, $x_4, \ldots, x_{k-3}$ false

If $k=3$: Use old clause in new formulation.

If $k=2$: Replace $C_i = \lambda + \lambda'$ by $\lambda + \lambda' + y$

If $k=1$: Replace $C_i = \lambda$ by $\lambda + y + z$

For the $k=2$, $k=1$ cases, need to force y and z to be false, so add clauses:

$$\begin{aligned} &\bar{z} + \alpha + \beta \\ &\bar{z} + \alpha + \bar{\beta} \\ &\bar{z} + \bar{\alpha} + \beta \\ &\bar{z} + \bar{\alpha} + \bar{\beta} \end{aligned}$$

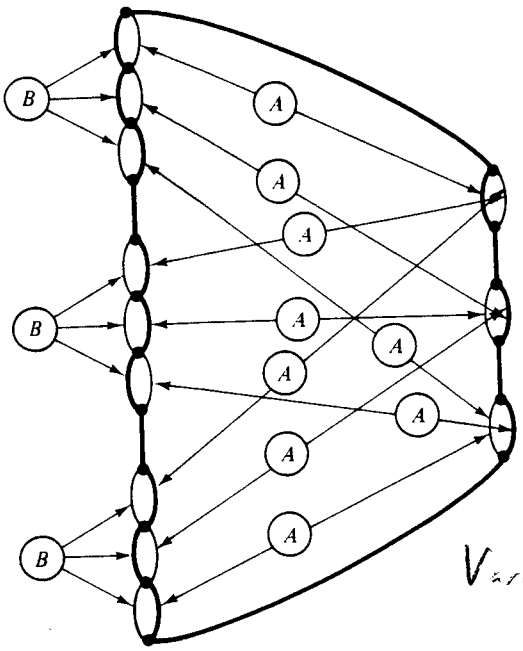
and

$$\begin{aligned} &\bar{y} + \alpha + \beta \\ &\bar{y} + \alpha + \bar{\beta} \\ &\bar{y} + \bar{\alpha} + \beta \\ &\bar{y} + \bar{\alpha} + \bar{\beta} \end{aligned}$$

//

Reducing 3-SAT to Hamiltonian Circuit

(Papadimitriou & Steiglitz, p. 366)



Variables

Clauses

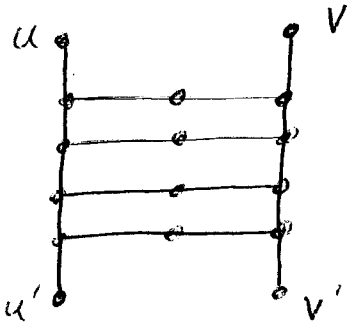
$F = (x_1 + \bar{x}_2 + x_3)(\bar{x}_1 + x_2 + \bar{x}_3)(\bar{x}_1 + \bar{x}_2 + x_3)$

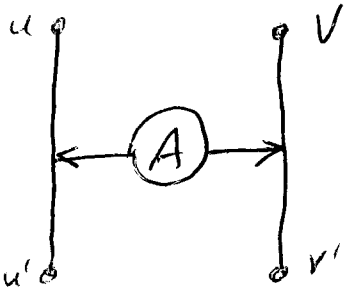
The Hamilton circuit shown corresponds to:

$t(x_1) = true$
$t(x_2) = false$
$t(x_3) = false$

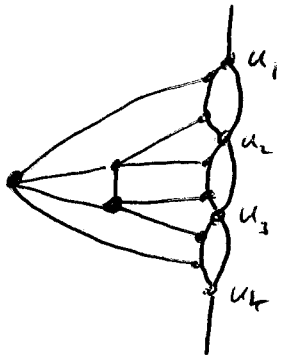
Figure 15–10

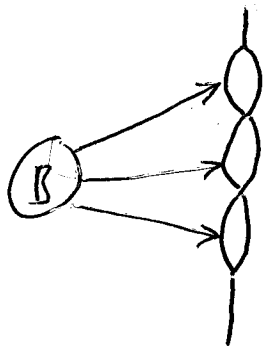
Subgraph A:





Subgraph B:





Use subgraph B for each clause.

Connect a literal in a clause to the literal on the variable side, using the A subgraph.

Hamiltonian circuit

Given a graph G, is there a circuit visiting each node exactly once

Th. Hamiltonian circuit is NP-complete

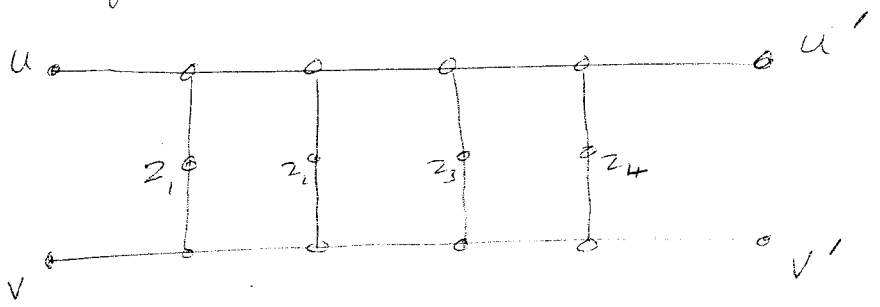
Proof Clearly in NP.

To show problem is NP-complete:

Transform 3-SAT to Hamiltonian circuit:

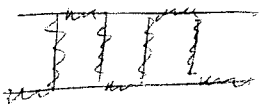
So given an instance of 3-SAT, we construct a graph such that there is a Hamiltonian tour in the graph iff the instance of 3-SAT is feasible.

Consider graph:

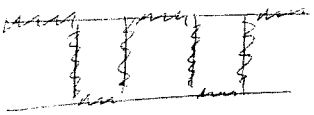


No other edges are incident on any vertex of A except u, u', v, v'.

Any tour of A has to traverse it either in the form

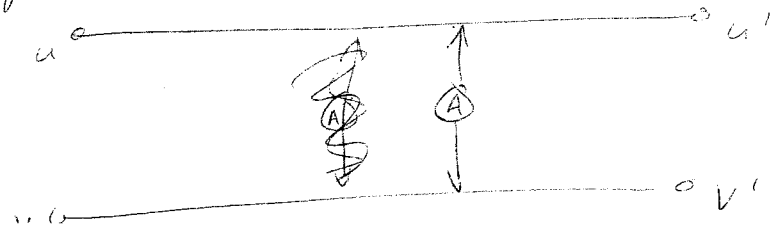


or in the form



So any tour traverses either vv' or uu' but not both.

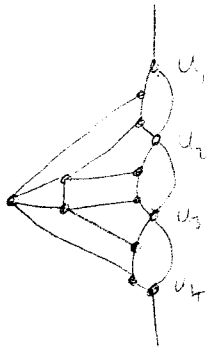
Write graph as



A - connector.

Consider graph

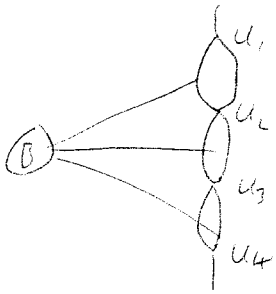
B:



Outside Edges only connect at u_1 and u_4.

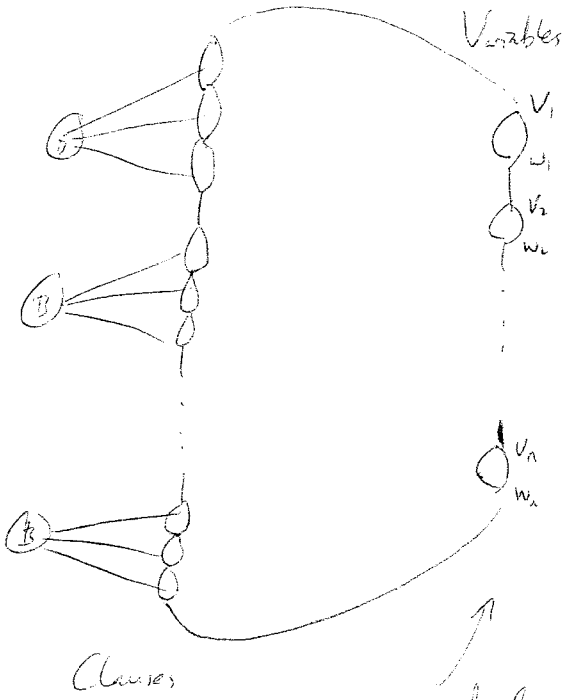
Note that any Hamiltonian circuit can not traverse all three of edges (u_1, u_2), (u_2, u_3), (u_3, u_4), but it can traverse any other combination of them.

Write as



So think: if traverse edge $u_j u_{j+1}$, regard literal $\bar{x}_j$ as not holding. Since don't traverse one edge, must have clause holding.

Have m clauses, so put n copies of B in series



Connect tops and bottoms

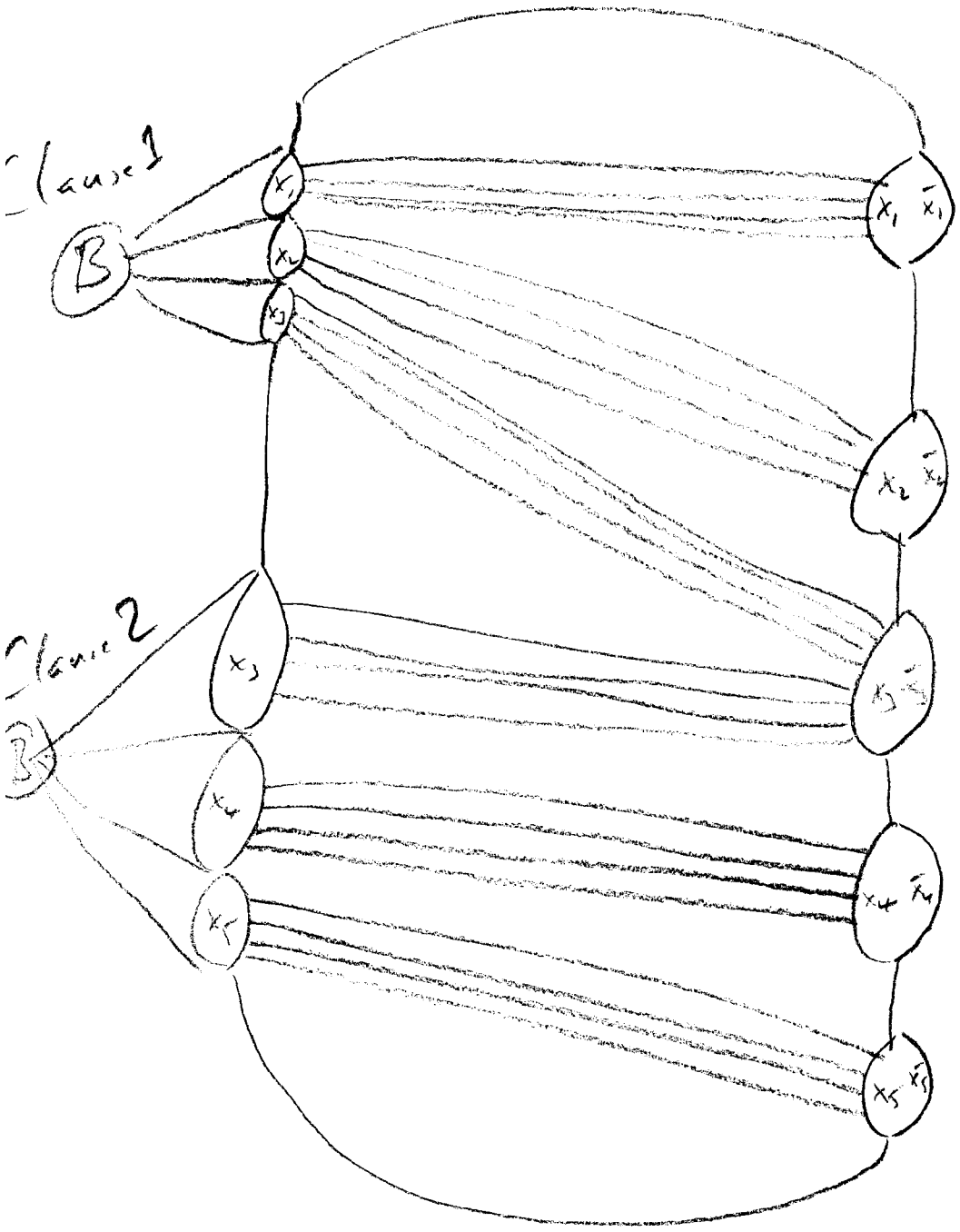
Use A-components to ensure consistency, so always take x_i true (or false) in every clause.

Connect u_j, u_{j+1} in clause i to left part of V_k, W_k if $\neg x_k$ is literal of C_i; to right part if it is $\bar{x}_k$.

Pairs of edges for variables

Cross left edge if variable true, right — — false

C_1: $x_1 + x_2 + x_3$ C_2: $x_3 + x_4 + x_5$



Clauses

Variables.

Thm TSP is NP-complete.

Pf We know it is in NP.

To show NP-complete:

Reduce ~~TSP~~ Hamiltonian circuit to TSP:

Map $G = (V, E)$ to complete graph on $|V|$ vertices.

If $(i, j) \in E$ then $d_{ij} = 1$

$(i, j) \notin E$ $d_{ij} = 2$.

Find ~~TSP~~ tour of length $\leq |V|$ //