8 July

## Still more NP-complete problems [P&S §15.6, §15.7]

[P&S Thm. 15.6] **Theorem.** HAMILTONIAN CIRCUIT is NP-complete.

**Proof.** We previously showed that HAMILTONIAN CIRCUIT is in NP (lecture notes, July 2).

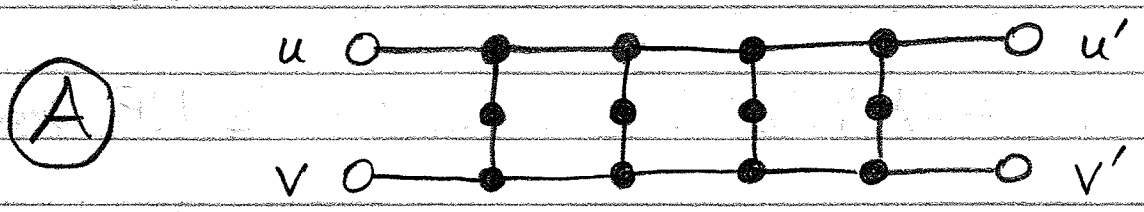
Now we show that 3-SAT polynomially transforms to HAMILTONIAN CIRCUIT.

Let $F$ be a 3-SAT formula on the Boolean variables $x_1, \ldots, x_n$ with clauses $C_1, \ldots, C_n$.

Our goal is to construct a graph $G = (V, E)$ that has a Hamiltonian circuit if and only if $F$ is satisfiable.

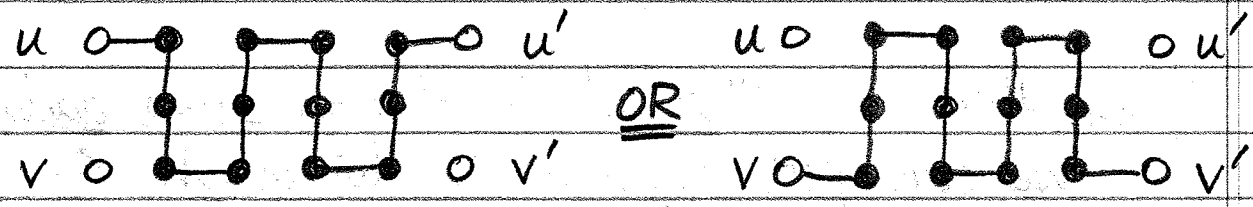
To do this, we will need to design some gadgets that we can put together in the graph $G$ to simulate parts of the formula $F$. This technique is called component design (gadgets are also called components).

The first gadget, which we will call A, looks like this:

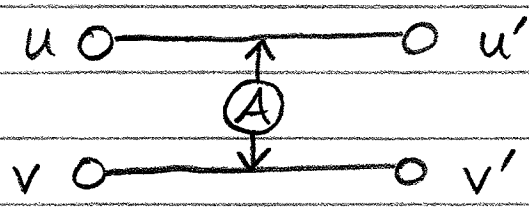



The filled vertices are not to be joined to any other vertices in the graph (so the only vertices that may have neighbors not shown here are u, u′, v, and v′).

The usefulness of gadget A comes from the fact that if a graph containing A has a Hamiltonian circuit, there are exactly two possible ways that circuit could pass through A:



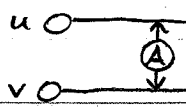

As a shorthand notation, we will draw




to represent this gadget.

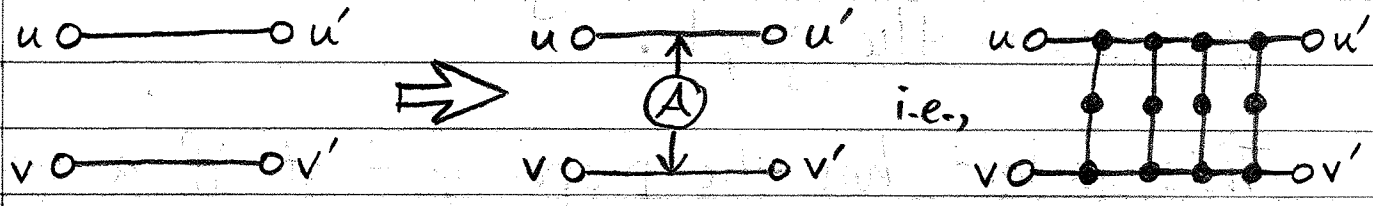
HAM CIRCUIT is NP-complete — ②

8 July




If we have in a graph, then any Hamiltonian circuit must enter this portion of the graph at $u$ and exit at $u'$, or else enter at $v$ and exit at $v'$, but not both.

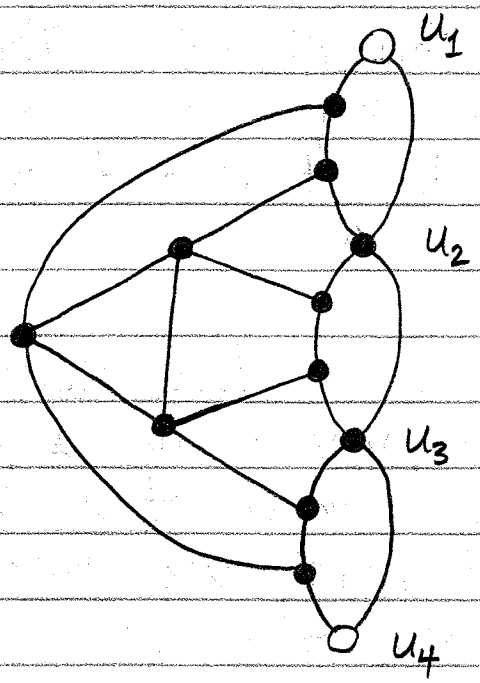
Therefore, if we have two nonadjacent edges $\{u,u'\}$ and $\{v,v'\}$ in a graph, and we want to enforce that any Hamiltonian circuit must use <u>exactly one</u> of those two edges, then we can simulate this requirement by inserting an (A) between them:




So now we can design graphs that have pairs of "edges" $\{u,u'\}$ and $\{v,v'\}$ with the property that any Hamiltonian path must pass through <u>exactly one</u> of them.

The next gadget, which we shall imaginatively call B, looks like this:

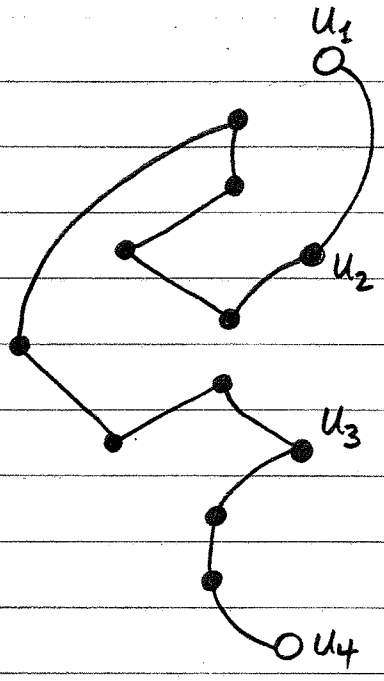
(B)



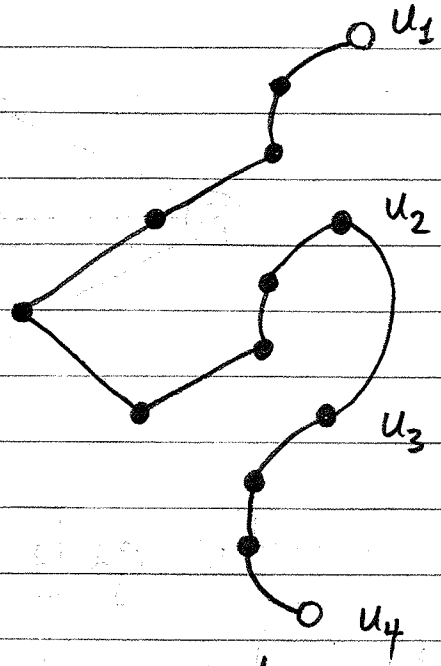

As before, the filled vertices (all except $u_1$ and $u_4$) are not to be joined to any other vertices in the graph.

This gadget has the property that if a graph containing B has a Hamiltonian circuit, then the circuit <u>cannot include all three</u> of the edges $\{u_1, u_2\}$, $\{u_2, u_3\}$, and $\{u_3, u_4\}$, because in that case the other filled vertices would be bypassed, but <u>all proper subsets</u> of these three edges are possibilities, as shown on the next page. (Note that we are assuming that B is part of a larger graph, so the portion of the Hamiltonian circuit within B is a <u>path</u> from $u_1$ to $u_4$.)

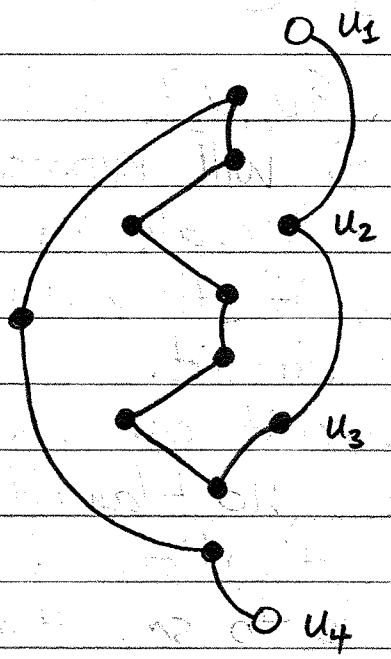
8 July



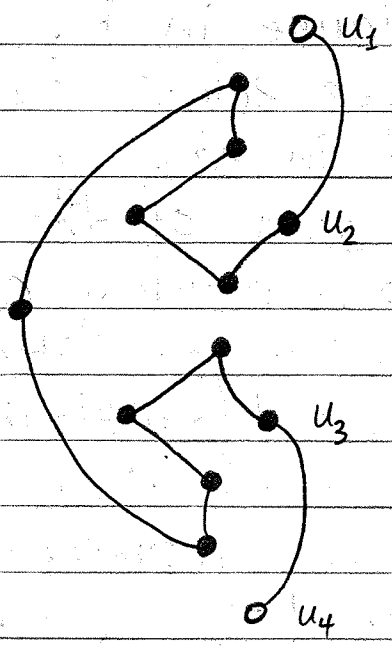

$u_1u_2$ only.
(Symmetrically for $u_3u_4$ only.)



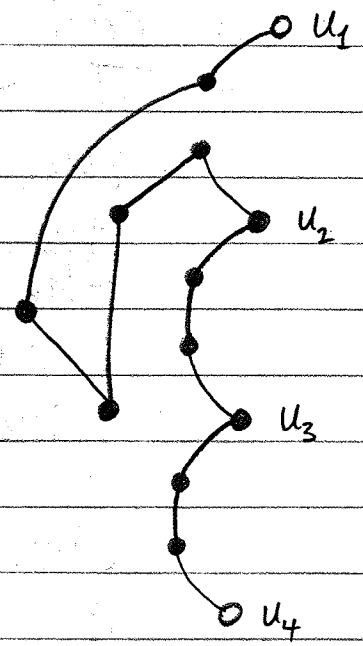

$u_2u_3$ only.




$u_1u_2$ and $u_2u_3$.
(Symmetrically for $u_2u_3$ and $u_3u_4$.)

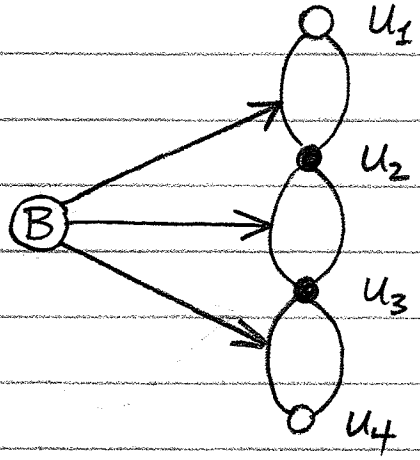



$u_1u_2$ and $u_3u_4$.




None of the three edges $u_1u_2$, $u_2u_3$, $u_3u_4$.

As a shorthand notation, we will draw




for this gadget.

Now that we have the gadgets A and B, we can construct the desired graph G.

We begin with m copies of gadget B, one for each clause in F.

— The three edges $\{u_1, u_2\}$, $\{u_2, u_3\}$, and $\{u_3, u_4\}$ in each gadget B will represent <u>falsification</u> of the three literals in the corresponding clause, if they are used in a Hamiltonian circuit.
Since no Hamiltonian circuit can use all three of these edges, no Hamiltonian circuit will represent falsification of all three literals, which is to say that at least one literal will be true, and therefore the clause will be satisfied.

8 July

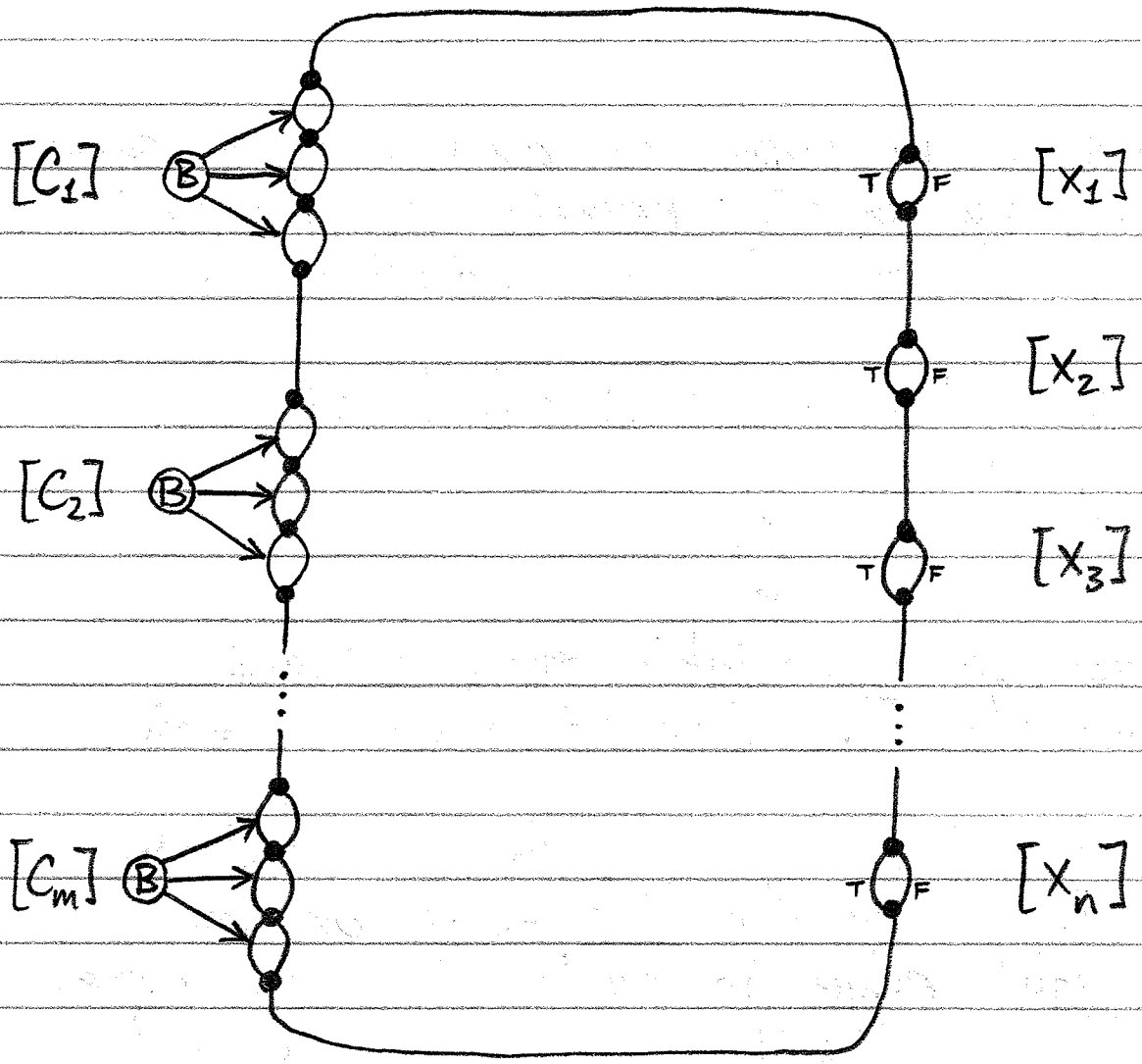
Connect the m copies of gadget B in series, that is, put an edge between $u_4$ of copy i and $u_1$ of copy i+1 for $1 \le i \le m-1$.

Now we also create n copies of the (multi-) graph

one for each variable that appears in the formula F. Connect these in series too.

— Any Hamiltonian path will need to use either the left edge or the right edge in each of these copies (and it can't use both). The left edge will represent an assignment of TRUE to the corresponding variable, and the right edge FALSE.

Join $u_1$ of the first copy of gadget B to the top vertex of the subgraph for $x_1$, and join $u_4$ of the last copy of gadget B to the bottom vertex of the subgraph for $x_n$. The result is shown on the next page.




m copies of gadget B, one for each clause in F.

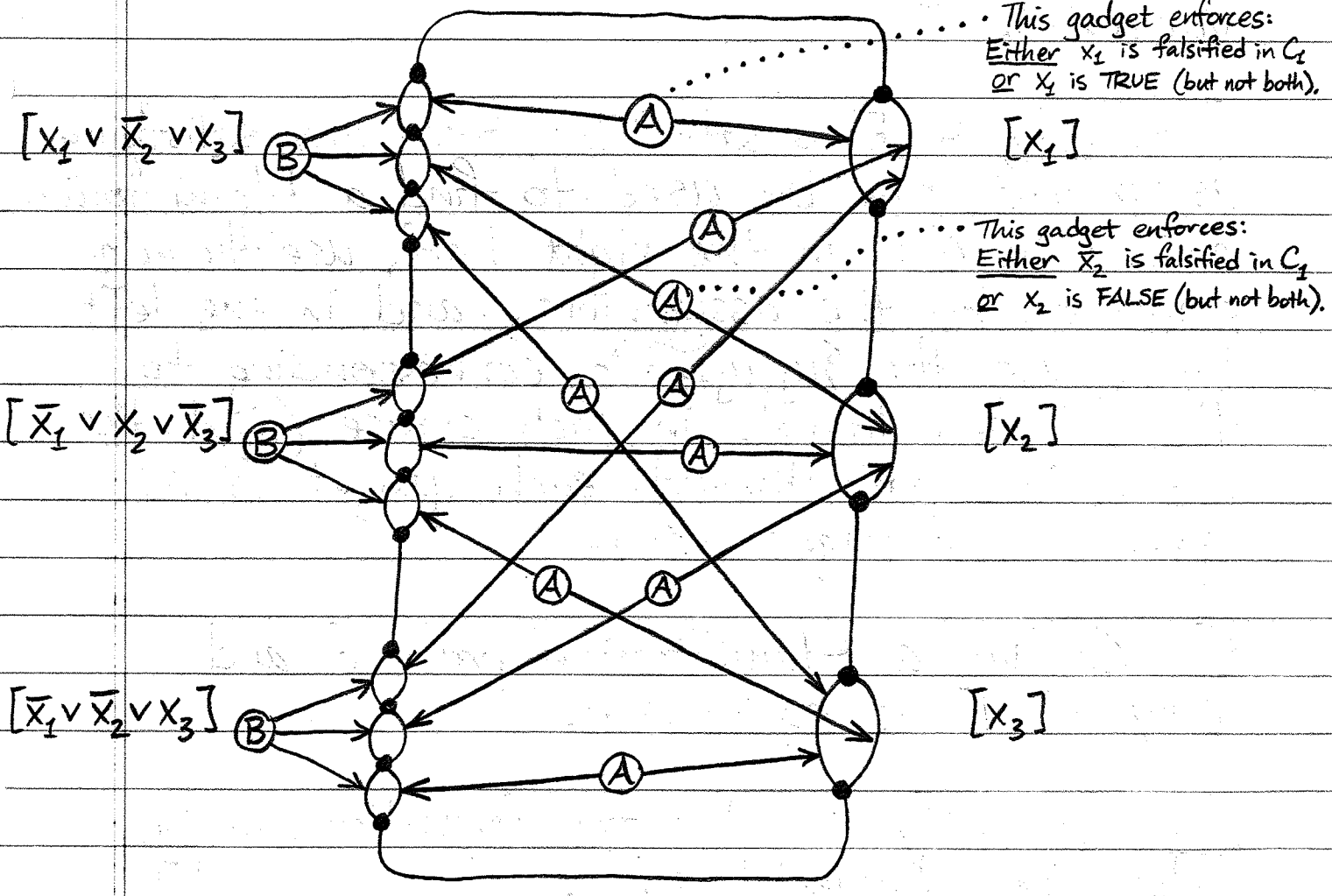
n copies of Q, one for each variable in F.

What remains is to ensure that the assignments of truth values to the variables $x_1, \ldots, x_n$ represented by a Hamiltonian circuit's route through the right half of this graph agree with the falsifications of literals represented by the route through the left half. This is where we use gadget A.

8 July

For each clause $C_i$ and each literal $\lambda_j$ in that clause ($j \in \{1,2,3\}$; $\lambda_j = x_k$ or $\lambda_j = \bar{x}_k$ for some $k$), put a copy of gadget A between the edge $\{u_j, u_{j+1}\}$ in copy $i$ of gadget B and one of the edges in the $k$th copy of ⬯. That edge of ⬯ should be the left (TRUE) edge if $\lambda_j = x_k$, or the right (FALSE) edge if $\lambda_j = \bar{x}_k$.

Example. $F = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$.



If the resulting graph $G$ has a Hamiltonian circuit, then its route through the right half of $G$ specifies exactly one truth value for each variable, and its route through the left half will include an edge $\{u_j, u_{j+1}\}$ in a copy of gadget $B$ if and only if the corresponding literal in the corresponding clause is falsified. But by the design of gadget $B$, a Hamiltonian circuit cannot include all three edges $\{u_1, u_2\}$, $\{u_2, u_3\}$, and $\{u_3, u_4\}$, so in every clause at least one literal is satisfied, which means that the whole formula $F$ is satisfied by this assignment of truth values to the variables.

Conversely, if $F$ is satisfiable, then any satisfying assignment can be used to find a Hamiltonian circuit in $G$: in the right half, use the edges specified by the assignment, and in the left half, use the $\{u_j, u_{j+1}\}$'s corresponding to the falsified literals in each clause (and the appropriate path through each gadget $B$ as given a few pages ago).

So $G$ has a Hamiltonian path if and only if $F$ is satisfiable.

Exercise: Verify that this transformation can be done in polynomial time. □

8 July

**Corollary.** TSP is NP-complete.

**Proof.** Exercise: TSP $\in$ NP. [Or see P&S Example 15.6, §15.3.] And we have previously shown that HAMILTONIAN CIRCUIT polynomially transforms to TSP (see lecture notes, July 2). □

**Corollary.** HAMILTONIAN PATH is NP-complete.

[Instance: A graph $G=(V,E)$.
Question: Does $G$ contain a Hamiltonian path?
(That is, a spanning path — a path that includes all vertices of $G$.)]

**Proof.** Exercise: HAMILTONIAN PATH $\in$ NP.

Now we show that HAMILTONIAN CIRCUIT polynomially transforms to HAMILTONIAN PATH.

Let $G=(V,E)$ be an instance of HAMILTONIAN CIRCUIT. Our goal is to construct a graph $G'=(V',E')$ such that $G'$ contains a Hamiltonian path if and only if $G$ contains a Hamiltonian circuit.

First, the stupid case: If $|V|<3$, then obviously $G$ cannot contain a Hamiltonian circuit, so take $G'=$  , which does not contain a Hamiltonian path.

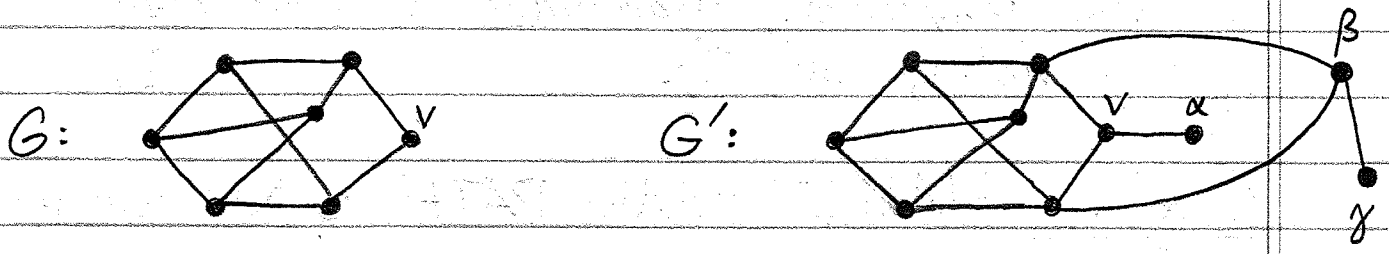
Otherwise $|V| \geq 3$.
Let $v \in V$ be any vertex of $G$. Take

$$V' = V \cup \{\alpha, \beta, \gamma\}$$

where $\alpha$, $\beta$, and $\gamma$ are new vertices, and take

$$E' = E \cup \{\{v,\alpha\}, \{\beta,\gamma\}\} \cup \underbrace{\{\{u,\beta\} : \{u,v\} \in E\}}_{\text{join all neighbors of } v \text{ to } \beta}.$$

Example.




If $G'$ has a Hamiltonian path, then the path must begin at $\alpha$ and end at $\gamma$ (because these vertices have degree 1). So the path must begin $\alpha - v - \cdots$ and end $\cdots - u - \beta - \gamma$ for some vertex $u$, and $u$ must be adjacent to $v$ by the construction of $E'$. Moreover, the edge $\{v,u\}$ must <u>not</u> be in the path, because $|V| \geq 3$ so if the path were $\alpha - v - u - \beta - \gamma$ then it would miss some vertices. Therefore, the portion of this path between $v$ and $u$, together with the edge $\{v,u\}$, forms a Hamiltonian circuit in $G$.

Conversely, suppose $G$ has a Hamiltonian circuit $(v, w_1, w_2, \ldots, w_{n-1}, v)$. Then $(\alpha, v, w_1, w_2, \ldots, w_{n-1}, \beta, \gamma)$ is a Hamiltonian path in $G'$.

□

8 July

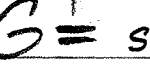
**Corollary.** LONGEST PATH is NP-complete.

[Instance: A graph $G=(V,E)$, edge weights $w_e$ for $e \in E$, vertices $s,t \in V$, and $L \in \mathbb{R}$.
Question: Does there exist a path between $s$ and $t$ having total weight $\geq L$?]

**Proof.** Exercise: LONGEST PATH $\in$ NP.

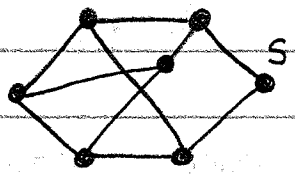
We show HAMILTONIAN CIRCUIT polynomially transforms to LONGEST PATH.

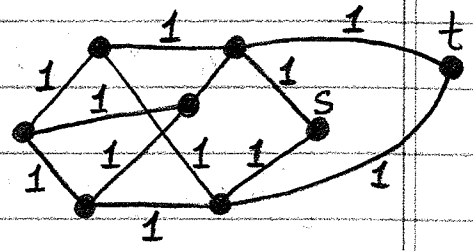
Let $G=(V,E)$ be an instance of HAMILTONIAN CIRCUIT. Our goal is to construct a graph $G'=(V',E')$, edge weights $w_e$ for $e \in E'$, vertices $s,t \in V'$, and $L \in \mathbb{R}$ such that $G'$ contains an $s$-$t$ path of weight $\geq L$ if and only if $G$ contains a Hamiltonian circuit.

Stupid case: If $|V| < 3$, then take $G =$  and $L=2$.


Otherwise $|V| \geq 3$. Let $s \in V$ be any vertex of $G$. Take $V' = V \cup \{t\}$, where $t$ is a new vertex. Take $E' = E \cup \{\{u,t\} : \{u,s\} \in E\}$, that is, join $t$ to all neighbors of $s$. Take $w_e = 1$ for all $e \in E'$, and take $L = |V|$.

Example.

G: 


G′: 


$L = 7$

An s-t path in $G'$ with weight $\geq |V|$ must be a Hamiltonian path, and by reasoning similar to that in the proof of the previous corollary, this gives us a Hamiltonian circuit in $G$.

Conversely, if $G$ has a Hamiltonian circuit $(s, w_1, w_2, \ldots, w_{n-1}, s)$, then $(s, w_1, w_2, \ldots, w_{n-1}, t)$ is an s-t path in $G'$ of weight $|V|$. □

Observe: The shortest path problem is in P — we can solve it efficiently with Dijkstra's algorithm, for example. But the LONGEST PATH problem is NP-complete!

8 July

One more definition: [P&S §16.4.2]

**Defn.** NP-hard.

(a) A decision problem $A$ is <u>NP-hard</u> if every problem in NP polynomially transforms to $A$.

(b) An optimization or evaluation problem is <u>NP-hard</u> if the corresponding decision problem is NP-hard.

**Notes:**

- The condition in part (a) of this definition is exactly the same as the second condition in the definition of NP-complete. But an NP-hard problem is not required to be in NP.
- All NP-complete problems are NP-hard. NP-complete $=$ NP $\cap$ NP-hard.
- Informally, an NP-hard problem is "at least as hard as the hardest problems in NP."

Epilogue: What does NP-hardness mean in practice, and how do we respond to it? [P&S §16.5]

Assuming that $P \neq NP$, if a problem is NP-hard, then there is no algorithm for the problem that
- always produces the correct (or optimal) answer, and
- runs in polynomial time in the worst case.

This may seem like disheartening news. However, showing that a problem is NP-hard is useful because it allows us to refocus our efforts. Instead of searching for a polynomial-time algorithm, we can try one or more of the following approaches.

- Approximation algorithms [P&S Chap. 17]. Efficient algorithms that may not produce the optimal solution but will always produce solutions within a certain percentage of optimality.

- Probabilistic algorithms. It may be possible to design an algorithm whose performance (in terms of running time or output quality or both) is "good" "most" of the time, assuming a certain distribution of instances. (The meanings of "good" and "most" can be made precise.)

8 July

# Epilogue—②

- Superpolynomial algorithms. Despite the fact that an algorithm may not run in polynomial time in the worst case, it still may run fast enough for practical purposes on real-world instances. For example:
  - The simplex algorithm is not known to run in polynomial time, but it is quite efficient in practice and often outperforms polynomial-time algorithms on real-world instances.
  - Some NP-hard problems have <u>pseudo-polynomial algorithms</u> [P&S § 16.2] whose running time is bounded by a polynomial in the <u>numerical values</u> of the numbers in the input rather than the sizes of their representations. Such an algorithm may be good enough for practical purposes. See dynamic programming [P&S § 18.6].
  - The branch-and-bound technique, while not polynomial-time, can be a powerful tool for NP-hard problems.
  - Specialized solvers have been developed for many important NP-hard problems: IP, SAT, TSP, constraint programming, etc. See, for example, "Concorde TSP," which is available for iOS and can solve instances with hundreds of cities in reasonable time.

- Special cases [P&S §16.3]. A problem may be NP-hard only because of a few hard instances (because the running time of an algorithm is measured in the worst case). Perhaps restricting the problem to a subset of instances makes it tractable.
- Local search [P&S Chap. 19]. The idea here is to find a "pretty good" solution, and then repeatedly attempt to improve it by searching for a better solution that is "nearby" in some space. Examples of local search techniques include hill climbing, simulated annealing, evolutionary algorithms, and tabu search.
- Heuristics —— "rules of thumb" that seem to work well in practice. Often heuristics are very difficult to analyze mathematically, and they may be initially based on a "hunch" rather than any theoretical foundation, but heuristics are ubiquitous in methods for solving problems that arise in the real world.