Chapter 3

NP Completeness III

By Sariel Har-Peled, December 7, 2009[1] Version: 1.0

3.1 Hamiltonian Cycle

Definition 3.1.1 A ***Hamiltonian cycle*** is a cycle in the graph that visits every vertex exactly once.

Definition 3.1.2 An ***Eulerian cycle*** is a cycle in a graph that uses every edge exactly once.

Finding Eulerian cycle can be done in linear time. Surprisingly, finding a Hamiltonian cycle is much harder.

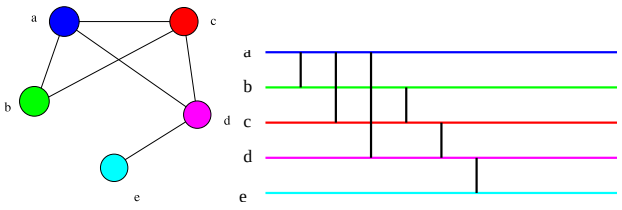
Problem: Hamiltonian Cycle

Instance: A graph G.
Question: Is there a Hamiltonian cycle in G?

Theorem 3.1.3 *Hamiltonian Cycle is* NP-Complete.

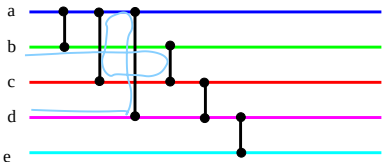
Proof: Hamiltonian Cycle is clearly in NP.

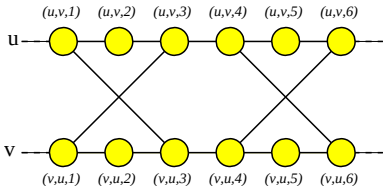
We will show a reduction from Vertex Cover. Given a graph G and integer k we redraw G in the following way: We turn every vertex into a horizontal line segment, all of the same length. Next, we turn an edge in the original graph G into a *gate*, which is a vertical segment connecting the two relevant vertices.



Note, that there is a Vertex Cover in G of size k if and only if there are k horizontal lines that stabs all the gates in the resulting graph H (a line stabs a gate if one of the endpoints of the gate lies on the line).

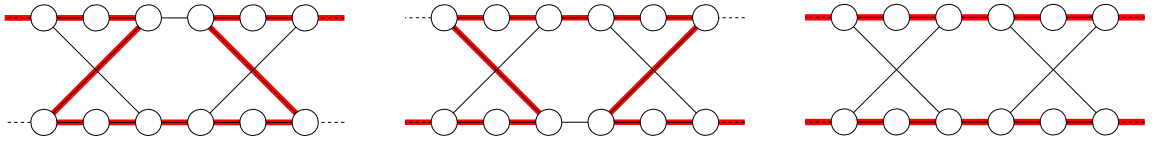
Thus, computing a vertex cover in G is equivalent to computing k disjoints paths through the graph G that visits all the gates. However, there is a technical problem: a path might change venues or even go back. See figure on the right.





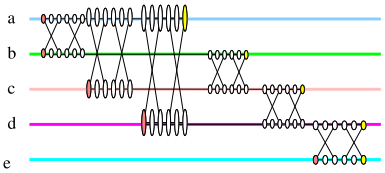
To overcome this problem, we will replace each gate with a component that guarantees, that if you visit all its vertices, you have to go forward and can NOT go back (or change "lanes"). The new component is depicted on the left.

There only three possible ways to visit *all* the vertices of the components by paths that do not start/end inside the component, and they are the following:

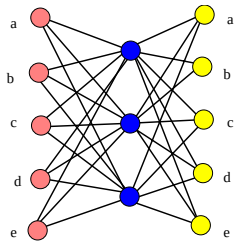


The proof that this is the only three possibilities is by brute force. Depicted on the right is one impossible path, that tries to backtrack by entering on the top and leaving on the bottom. Observe, that there are vertices left unvisited. Which means that not all the vertices in the graph are going to be visited, because we add the constraint, that the paths start/end outside the gate-component (this condition would be enforced naturally by our final construction).

The resulting graph H_1 for the example graph we started with is depicted on the right. There exists a Vertex Cover in the original graph **iff** there exists k paths that start on the left side and end on the right side, in this weird graph. And these k paths visits all the vertices.



The final stroke is to add connection from the left side to the right side, such that once you arrive to the right side, you can go back to the left side. However, we want connection that allow you to travel exactly k times. This is done by adding to the above graph a "routing box" component H_2 depicted on the right, with k new middle vertices. The ith vertex on the left of the routing component is the left most vertex of the ith horizontal line in the graph, and the ith vertex on the right of the component is the right most vertex of the ith horizontal line in the graph.



It is now easy (but tedious) to verify that the resulting graph $H_1 \cup H_2$ has a Hamiltonian path **iff** H_1has k paths going from left to right, which happens, **iff** the original graph has a Vertex Cover of size k. It is easy to verify that this reduction can be done in polynomial time. ■

3.2 Traveling Salesman Problem

A traveling salesman tour, is a Hamiltonian cycle in a graph, which its price is the price of all the edges it uses.

Problem: TSP

Instance: $G = (V, E)$ a complete graph - n vertices, $c(e)$: Integer cost function over the edges of G, and k an integer. *Question:* Is there a traveling-salesman tour with cost at most k?

Theorem 3.2.1 *TSP is* NP-Complete.

Proof: Reduction from Hamiltonian cycle. Consider a graph $G = (V, E)$, and let H be the complete graph defined over V. Let

$$c(e) = \begin{cases} 1 & e \in E(G) \\ 2 & e \notin E(G). \end{cases}$$

Clearly, the cheapest TSP in H with cost function equal to n iff G is Hamiltonian. Indeed, if G is not Hamiltonian, then the TSP must use one edge that does not belong to G, and then, its price would be at least $n + 1$. ■

3.3 Subset Sum

We would like to prove that the following problem, Subset Sum is NPC.

Problem: Subset Sum

Instance: S - set of positive integers,t: - an integer number (Target) *Question:* Is there a subset $X \subseteq S$ such that $\sum_{x \in X} x = t$?

How does one prove that a problem is NP-Complete? First, one has to choose an appropriate NPC to reduce from. In this case, we will use 3SAT. Namely, we are given a 3CNF formula with n variables and m clauses. The second stage, is to "play" with the problem and understand what kind of constraints can be encoded in an instance of a given problem and understand the general structure of the problem.

The first observation is that we can use very long numbers as input to Subset Sum. The numbers can be of polynomial length in the size of the input 3SAT formula F.

The second observation is that in fact, instead of thinking about Subset Sum as adding numbers, we can think about it as a problem where we are given vectors with k components each, and the sum of the vectors (coordinate by coordinate, must match. For example, the input might be the vectors $(1, 2), (3, 4), (5, 6)$ and the target vector might be $(6, 8)$. Clearly, $(1, 2) + (5, 6)$ give the required target vector. Lets refer to this new problem as Vec Subset Sum.

Problem: Vec Subset Sum

Instance: S - set of n vectors of dimension k, each vector has non-negative numbers for its coordinates, and a target vector $\overrightarrow{t}$. *Question:* Is there a subset $X \subseteq S$ such that $\sum_{\overrightarrow{x} \in X} \overrightarrow{x} = \overrightarrow{t}$?

Given an instance of Vec Subset Sum, we can covert it into an instance of Subset Sum as follows: We compute the largest number in the given instance, multiply it by $n^2 \cdot k \cdot 100$, and compute how many digits are required to write this number down. Let U be this number of digits. Now, we take every vector in the given instance and we write it down using U digits, padding it with zeroes

as necessary. Clearly, each vector is now converted into a huge integer number. The property is now that a sub of numbers in a specific column of the given instance can not spill into digits allocated for a different column since there are enough zeroes separating the digits corresponding to two different columns.

Target	??	??	01	???
a_1	??	??	01	??
a_2	??	??	01	??

Next, let us observe that we can force the solution (if it exists) for Vec Subset Sum to include exactly one vector out of two vectors. To this end, we will introduce a new coordinate (i.e., a new column in the table on the right) for all the vectors. The two vectors a_1 and a_2 will have 1 in this coordinate, and all other vectors will have zero in this coordinate. Finally, we set this coordinate in the target vector to be 1. Clearly, a solution is a subset of vectors that in this coordinate add up to 1. Namely, we have to choose either a_1 or a_2 into our solution.

In particular, for each variable x appearing in F, we will introduce two rows, denoted by x and $\overline{x}$ and introduce the above mechanism to force choosing either x or $\overline{x}$ to the optimal solution. If x (resp. $\overline{x}$) is chosen into the solution, we will interpret it as the solution to F assigns TRUE (resp. FALSE) to x.

numbers	...	$C \equiv a \vee b \vee \overline{c}$	...
a	...	01	...
$\overline{a}$	...	00	...
b	...	01	...
$\overline{b}$	...	00	...
c	...	00	...
$\overline{c}$	...	01	...
C fix-up 1	000	07	000
C fix-up 2	000	08	000
C fix-up 3	000	09	000
TARGET		10	

Next, consider a clause $C \equiv a \vee b \vee \overline{c}$.appearing in F. This clause requires that we choose at least one row from the rows corresponding to a, b to $\overline{c}$. This can be enforced by introducing a new coordinate for the clauses C, and setting 1 for each row that if it is picked then the clauses is satisfied. The question now is what do we set the target to be? Since a valid solution might have any number between 1 to 3 as a sum of this coordinate. To overcome this, we introduce three new dummy rows, that store in this coordinate, the numbers 7, 8 and 9, and we set this coordinate in the target to be 10. Clearly, if we pick to dummy rows into the optimal solution then sum in this coordinate would exceed 10. Similarly, if we do not pick one of these three dummy rows to the optimal solution, the maximum sum in this coordinate would be $1 + 1 + 1 = 3$, which is smaller than 10. Thus, the only possibility is to pick one dummy row, and some subset of the rows such that the sum is 10. Notice, this "gadget" can accommodate any (non-empty) subset of the three rows chosen for a, b and $\overline{c}$.

We repeat this process for each clause of F. We end up with a set U of $2n + 3m$ vectors with $n + m$ coordinate, and the question if there is a subset of these vectors that add up to the target vector. There is such a subset if and only if the original formula F is satisfiable, as can be easily verified. Furthermore, this reduction can be done in polynomial time.

Finally, we convert these vectors into an instance of Subset Sum. Clearly, this instance of Subset Sum has a solution if and only if the original instance of 3SAT had a solution. Since Subset Sum is in NP as an be easily verified, we conclude that that Subset Sum is NP-Complete.

Theorem 3.3.1 *Subset Sum is* NP-Complete.

For a concrete example of the reduction, see Figure 3.1.

numbers	$a \vee \overline{a}$	$b \vee \overline{b}$	$c \vee \overline{c}$	$d \vee \overline{d}$	$D \equiv \overline{b} \vee c \vee \overline{d}$	$C \equiv a \vee b \vee \overline{c}$
a	1	0	0	0	00	01
$\overline{a}$	1	0	0	0	00	00
b	0	1	0	0	00	01
$\overline{b}$	0	1	0	0	01	00
c	0	0	1	0	01	00
$\overline{c}$	0	0	1	0	00	01
d	0	0	0	1	00	00
$\overline{d}$	0	0	0	1	01	01
C fix-up 1	0	0	0	0	00	07
C fix-up 2	0	0	0	0	00	08
C fix-up 3	0	0	0	0	00	09
D fix-up 1	0	0	0	0	07	00
D fix-up 2	0	0	0	0	08	00
D fix-up 3	0	0	0	0	09	00
TARGET	1	1	1	1	10	10

numbers
010000000001
010000000000
000100000001
000100000100
000001000100
000001000001
000000010000
000000010101
000000000007
000000000008
000000000009
000000000700
000000000800
000000000900
010101011010

Figure 3.1: The Vec Subset Sum instance generated for the 3SAT formula $F = \left(\overline{b} \vee c \vee \overline{d}\right) \wedge (a \vee b \vee \overline{c})$ is shown on the left. On the right side is the resulting instance of Subset Sum.

3.4 3 dimensional Matching (3DM)

Problem: 3DM

Instance: X, Y, Z sets of n elements, and T a set of triples, such that $(a, b, c) \in T \subseteq X \times Y \times Z$.
Question: Is there a subset $S \subseteq T$ of n disjoint triples, s.t. every element of $X \cup Y \cup Z$ is covered exactly once.?

Theorem 3.4.1 *3DM is* NP-Complete.

The proof is long and tedious and is omitted.

BTW, $2DM$ is polynomial (later in the course?).

3.5 Partition

Problem: Partition

Instance: A set S of n numbers.
Question: Is there a subset $T \subseteq S$ s.t. $\sum_{t \in T} t = \sum_{s \in S \setminus T} s$.?

Theorem 3.5.1 *Partition is* NP-Complete.

Proof: Partition is in NP, as we can easily verify that such a partition is valid.

Reduction from Subset Sum. Let the given instance be n numbers $a_1, \ldots, a_n$ and a target number t. Let $S = \sum_{i=}^{n} a_i$, and set $a_{n+1} = 3S - t$ and $a_{n+2} = 3S - (S - t) = 2S + t$. It is easy to verify that there is a solution to the given instance of subset sum, iff there is a solution to the following instance of partition:

$$a_1, \ldots, a_n, a_{n+1}, a_{n+2}.$$

Clearly, Partition is in NP and thus it is NP-Complete. ■