# CS 341 - Algorithms, Winter 2016. University of Waterloo

## Lecture 20: Hard partitioning problems

We will see that the 3-dimensional matching problem and the subset-sum problem are NP-complete.

---

## 3-dimensional matching [kT 8.6]

This is a generalization of the bipartite matching problem.

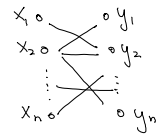
### 3-dimensional matching (3DM)

Input: Disjoint sets $X, Y, Z$ each of size $n$, a set $T \subseteq X \times Y \times Z$ of ordered triples

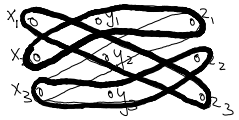
Output: Does there exist a subset of $n$ triples in $T$ so that each element of $X \cup Y \cup Z$ is contained in exactly one of these triples?

The bipartite matching problem is a special case when we only have $X, Y$ and ordeded pairs.

bipartite matching




3DM




Input: $(x_1, y_2, z_3)$, $(x_2, y_1, z_1)$, $(x_3, y_2, z_1)$, $(x_3, y_3, z_2)$

Output: $(x_1, y_2, z_3)$, $(x_2, y_1, z_2)$, $(x_3, y_3, z_2)$ is a perfect matching.

A set of $n$ triples is a perfect 3-dimensional matching if every element is contained in exactly one of these triples.

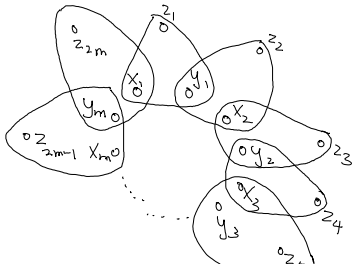
**Theorem** 3DM is NP-complete.

**proof** Clearly 3DM is in NP. We show that it is NP-complete by proving that $3SAT \leq_p 3DM$.

Given a 3SAT formula with $n$ variables $v_1, \ldots, v_n$ and $m$ clauses $C_1, \ldots, C_m$, we would like to construct an instance of 3DM such that the formula is satisfiable iff there is a perfect 3D-matching.

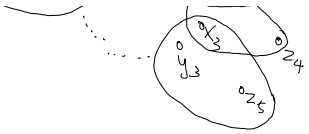
Like in the Hamiltonian cycle problem, we would like to create some variable gadgets to capture the binary decisions of the variables.

For each variable $v_i$, we create the following gadget.




There are $m$ vertices $x_1, \ldots, x_m$, $m$ vertices $y_1, \ldots, y_m$, and $2m$ vertices $z_1, \ldots, z_{2m}$ created for the variable $v_i$.

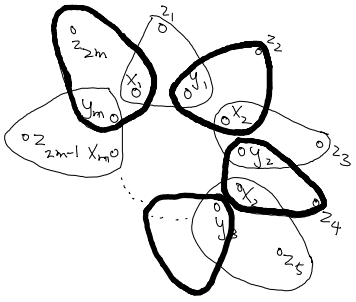
There is a triple $x_i, y_i, z_{2i-1}$ for $1 \leq i \leq m$, and a triple $y_i, x_{i+1}, z_{2i}$ for $1 \leq i \leq m-1$ and a triple $y_m, x_1, z_{2m}$.



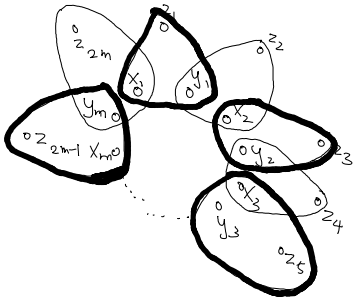

There is a triple $x_i, y_i, z_{2i-1}$ for $1 \le i \le m$, and a triple $y_i, x_{i+1}, z_{2i}$ for $1 \le i \le m-1$ and a triple $y_m, x_1, z_{2m}$.

We will make sure that the elements/vertices $x_i, y_i$ are not contained in any other triples, besides the two triples in the variable gadget.

This implies that there are only two possibilities to choose the triples in the variable gadget:




Corresponds to setting the variable to be TRUE

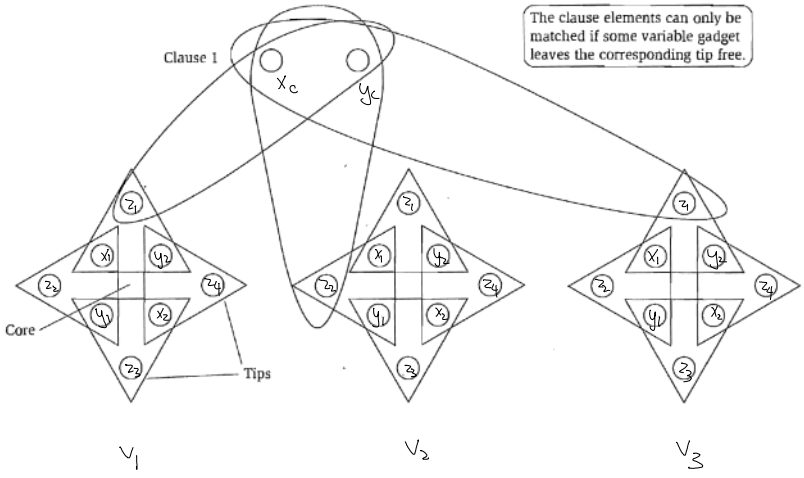



Corresponds to setting the variable to be FALSE

To cover $y_1$, either we choose $x_1, y_1, z_1$ or $x_2, y_1, z_2$. Once we've decided, the remaining is forced.

This captures the binary decision for each variable, as we have one gadget for each variable.

It remains to add some clause structures to the instance so that only satisfying assignments survive.




Say we have a clause $C_j = (v_1 \vee \bar{v}_2 \vee v_3)$.

We create two new vertices $x_c, y_c$ for $C_j$.

If a variable appears as $v$ (but not $\bar{v}$), we add a triple $x_c, y_c, z_{2j-1}$.

Otherwise, if a variable appears as $\bar{v}$, we add a triple $x_c, y_c, z_{2j}$.

We do the same for each clause. Notice that the triples for different clauses are disjoint, because they use different "tips" in the variable gadgets.

Each clause can cover one tip. There will be $2nm - m = (2n-1)m$ uncover tips left over.

We will create $(2n-1)m$ pairs of dummy vertices $x', y'$, and for each pair, we add a triple $(x', y', z)$ for every tip $z$ in every variable gadget.

Totally, we will add $2(2n-1)m$ new dummy vertices and $(2n-1)m \cdot 2m = 2(2n-1)m^2$ dummy triples. (This is a little wasteful but we don't care as long as it is polynomial in $n$ and $m$.)

This is the construction. It is clear that it can be done in polynomial time.

We need to prove that the formula is satisfiable iff there is a perfect 3D-matching.

Suppose there is a satisfying assignment. If $v_i = T$, we cover the variable gadget using the even tips, leaving the odd tips free; otherwise if $v_i = F$, we cover the variable gadget using the triples covering odd tips, leaving the even tips free.

Since this assignment is satisfying all the clauses, for each clause triple, there is a free tip avaiable in the variable gadget, and we will choose any such triple to cover the clause $x, y$.

Finally, for the remaining $(2n-1)m$ uncovered tips, we use the dummy triples to cover them all.

This gives us a perfect 3D-matching.

Suppose there is a perfect 3D-matching.

For each variable gadget, there are two ways to cover the internal $x_i, y_i$. If these triples don't cover the odd tips, we set the variable to be True; otherwise False.

Since we can cover the clause vertices, one of the three tips it connects to is free (i.e. without being used by the triples from the variable gadget).

By our construction, it means that the clause is satisfied by the satisfying assignment. □

---

## Subset-sum [KT 8.8]

Input: $n$ positive integers $a_1, \ldots, a_n$, and an integer $K$.

Output: Does there exist a subset $S \subseteq [n]$ with $\sum_{i \in S} a_i = K$ ?

This is a problem about numbers, and some new ideas are needed to connect to previous combinatorial problems.

Theorem Subset-Sum is NP-complete.

proof It is clear that Subset-sum is in NP. We prove that it is NPc by showing $3DM \leq_p$ subset-sum.

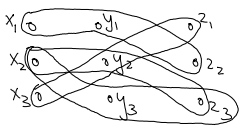
Given an instance of 3DM, we will construct an instance of subset-sum so that there is a perfect 3-dimensional matching iff there is a subset of certain sum $K$ (to be determined later).

The idea is quite natural. First, we map a triple into a bit-vector, and then we translate a bit-vector into a number, and so eventually a triple in 3DM will be mapped to a number.

We first show the reduction using an example and then describe it more formally.

a bit-vector into a number, and so eventually a triple in 3DM will be mapped to a number.

We first show the reduction using an example and then describe it more formally.




3DM instance with 4 triples $(x_1, y_1, z_2)$, $(x_2, y_2, z_3)$, $(x_2, y_3, z_3)$, $(x_3, y_2, z_1)$ $\longrightarrow$

$$\begin{array}{c|cccc} & V_1 & V_2 & V_3 & V_4 \\ x_1 & 1 & 0 & 0 & 0 \\ x_2 & 0 & 1 & 0 & 0 \\ x_3 & 0 & 0 & 0 & 1 \\ y_1 & 0 & 0 & 1 & 0 \\ y_2 & 0 & 0 & 0 & 1 \\ y_3 & 0 & 1 & 0 & 0 \\ z_1 & 0 & 0 & 0 & 1 \\ z_2 & 1 & 0 & 0 & 0 \\ z_3 & 0 & 1 & 1 & 0 \end{array}$$

Let $|X| = |Y| = |Z| = n$ and let $m$ be the number of triples in the 3DM instance.

We will create a vector for each triple. Each vector has $3n$ coordinates, one for each vertex

For each triple $(x_i, y_j, z_k)$, we have a vector with 1 in the $i$-th, $(n+j)$-th, and $(2n+k)$-th coordinate, and all other coordinates zero.

By construction, it is easy to verify that there is a perfect matching in the 3DM instance if and only if there is a subset of vectors that sums to the all-one vector.

In the above example, the vectors $V_1, V_3, V_4$ sum up to the all-one vector, and the corresponding triples form a perfect matching.

We leave the verification of the above claim to the reader.

So, now we have the problem of choosing a subset of 0-1 vectors that sums to 1 is NPc.

We would like to use it to show that the subset-sum problem is NPc.

A very natural idea is to think of the 0-1 vector as the binary representation of a number.

That is, each triple $(x_i, y_j, z_k)$ is mapped to the number $2^i + 2^{n+j} + 2^{2n+k}$.

With this mapping, if there is a subset of triples that form a perfect matching, their corresponding numbers would add up to $\sum_{i=1}^{3n} 2^i$, the number that corresponds to the all-one string.

However, if there is a subset of numbers that add up to $\sum_{i=1}^{3n} 2^i$, it may not correspond to a perfect matching. The problem is that when we add up two numbers both with 1 in the $j$-th coordinate, the resulting number has a one in the $(j+1)$-th coordinate because of "carry", not because that we have chosen a vector with a one in the $(j+1)$-coordinate as we intended to capture.

There is a simple trick to get around this "carry" problem.

There are at most $m$ numbers.

So, if we choose the base $b = m+1$, then there could not be carrying to the next number.

Final construction: Each triple $(x_i, y_j, z_k)$ is mapped to the number $b^i + b^{n+j} + b^{2n+k}$.

Define $K = \sum_{i=1}^{3n} b^i$, the all-one number in base $b$ (except the last position).

It is clear that this can be done in polynomial time.

Claim: There is a perfect 3D-matching iff there is a subset with sum $K = \sum_{i=1}^{3n} b^i$.

$\Rightarrow$) This direction is easy and we have argued before in the bit setting.

$\Leftarrow$) Since there is no carrying, for each position $l$ in the base-$b$ representation, the subset sum records how many times we have covered the $l$-th vertex in the 3DM instance.

As the target number $k$ has one in each position (except the last one), a subset of sum $K$ must correspond to a perfect 3D-matching.

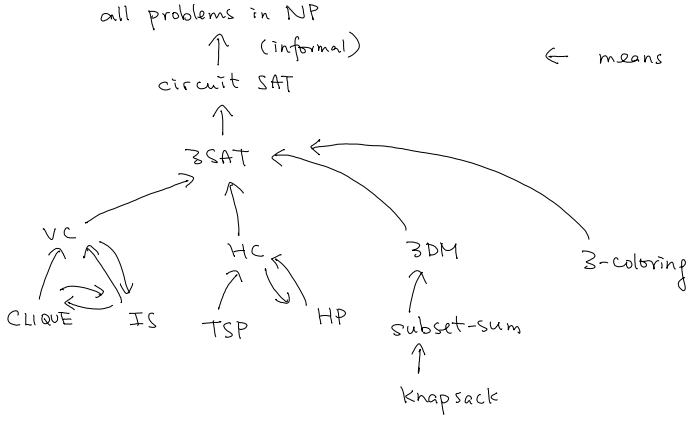
The claim finishes the proof of the theorem. □

As we have mentioned in L11, the subset-sum problem is a special case of the knapsack problem. Clearly, Knapsack $\in$ NP and so we have Knapsack is NP-complete.

---

## Concluding remarks

First, we summarize what we have done so far.




Unlike what we have done in earlier topics, where we showed you the basic examples

and asked you to do more advanced examples in homework, in NP-completeness, we have showed you the difficult reductions and ask you to do simple reductions in HW.

## Techniques of doing reductions

As we mentioned before, doing reduction requires a different way of thinking: we need to find a hard problem $Y$ and show that $Y \leq_p X$ for our problem $X$.

It requires practices to search for the right problem $Y$.

Once you know more NP-complete problems, it is easier to find a similar problem to do the reduction, e.g. covering type uses VC, partitioning type uses 3DM, etc.

There are three major techniques in proving NP-Completeness

### Specialization

Observe that a hard problem is a (very) special case of your problem.

This is the easiest and most useful technique, as more practical problems are often more complicated with different parameters, and often you realize that restricting to a simple setting already captures an NP-complete problem.

Some examples that we have seen include TSP and knapsack.

You will see many more in homework 5 and supplementary exercises.

### Local replacement

This is not as simple as specialization, but not as difficult as gadget design.

We replace each simple structure in problem $Y$ by a simple structure in problem $X$.

Some examples that we have seen include circuit-SAT $\leq_p$ 3SAT where we replace each gate by some simple clauses with the same functionality, DHC $\leq_p$ HC where we replace each directed edge by an undirected path of length three and connect the paths accordingly, 3DM $\leq_p$ subset-sum which is a more non-trivial example of this kind where we replace a triple by a number.

We will see more examples in HW5 and supplementary exercises.

### Gadget design

This requires creativity, imagination and good understanding to design gadgets and the plan for the reduction.

We have seen 3SAT $\leq_p$ VC, 3SAT $\leq_p$ DHC, 3SAT $\leq_p$ 3DM, 3SAT $\leq_p$ 3-coloring are all of this kind.

We won't ask this type of questions in HW and of course not in the final exam.

## 2 vs 3

It is an interesting phenomenon to observe that 2 is easy but 3 is hard.

For example, 2SAT is easy (see supplementary exercise list 2) but 3SAT is hard,

2-coloring is easy (bipartiteness, BFS) but 3-coloring is hard,

2D-matching is easy (bipartite matching) but 3-DM is hard.

Usually, 2 is easy because once you make a decision then everything else is forced, while 3 is diffrcult because after you make a choice you are still faced with binary decisions.

## Decision problem vs search problem

You may wonder whether restricting to decision problems will limit the scope of our problems, because we are usually interested in finding an optimal solution.

Many search problems can be easily reduced to decision problems.

For example, to find a Hamiltonian cycle, we can delete an edge and ask again whether the graph still has a Hamiltonian cycle, to determine if an edge is in the solution or not.

You are encouraged to try this for other problems, e.g. VC, subset-sum, 3SAT, etc.

Actually, one can prove formally that any NP-complete search problems can be reduced to polynomially many NP-complete decision problems, by reconstructing a solution bit by bit.

So, in terms of polynomial-time solvability, this is without loss of generality.

Finally, to guess the optimal value (say for min-VC, max-IS), we can simply do binary search to guess the optimal value in logarithmic number of steps.

---