CS270 Combinatorial Algorithms & Data Structures **Spring 2004**

Lecture 21: 4.6.04

Lecturer: Prof. Satish Rao *Scribe: Qi Zhu*

21.1 3SAT $\leq_p$ 3DM(3D-Matching)

Definition 21.1 (3DM) *Given a boys-girls-pets compatibility relation (i.e., a set of boys-girls-pets triangles), is there a complete matching which makes everyone happy (i.e., all boys, girls and pets are covered by a set of disjoint triangles)?*

More general way: Given finite disjoint sets X, Y, Z of size n, and a set of triples $\{t_i\} \subseteq X \times Y \times Z$, are there n pairwise disjoint triples?

Theorem 21.2 *3SAT can be reduced to 3DM.*

Reduction from 3SAT

In 3SAT, there are literals, clauses, and consistency constraints. The general technique of 3SAT reduction has three aspects: construct a "flip-flop" for each literals; construct a "constraint" for each clause; and maintain consistency between a literal and its negative form (cannot be satisfied at the same time). A flip-flop is used to represent true-false assignment; a constraint is used to ensure that the clause is satisfied; and the consistency need to be maintained because of the nature of boolean function. To get the reduction from 3SAT, our task is to find the "mapping" relation between 3SAT and the reduced problem.

From 3SAT to 3DM

A 3SAT CNF φ can be reduced to a 3DM $R(\varphi)$, by the following steps:

Flip-Flops: For each literal x in φ, make a gadget with $2k$ triangles, where k is the larger number of occurrence of positive(x) or negative($\overline{x}$) literal form. This gadget is shown in Figure 21.1, on which the tips of the triangles are labeled x or $\overline{x}$, alternatively. Each triangle represents a value assignment (true or false) which x can take.

Constraints: For each clause in φ, add a pair of boy vertex and girl vertex to the graph. If x appears in this clause, connect this boy and this girl to one of the x vertices in x's flip-flop gadget to form a triangle. If $\overline{x}$ is in the clause, then connect them to the $\overline{x}$ vertex from the flip-flop gadget. For other two literals in this clause($y, \overline{z}$ in the figure), we do the same thing. Figure 21.1 shows how this is done. Thus for the new boy and girl to be matched, one of the literals in the clause must be satisfied. And in this process, there are more pets than boys and girls, so some pets will be left homeless. This can be fixed by adding a boy and a girl for each homeless pet and make a triangle to connect them.

Consistency: The topology of the gardet ensures consistency. The construction guarantees that if any x vertex is matched with some vertices outside of the this gadget then all $\overline{x}$ vertex can only be matched by the triangles inside this gadget, and vice versa. Thus the "availability" of a vertex to be matched by an outside vertex corresponds to the truth assignment.

φ is satisfiable if and only if $R(\varphi)$ has a solution. This is evident from the way that G is constructed: $R(\varphi)$ preserves all literals, clauses, and consistency of φ. Thus for a truth assignment that satisfies φ, we can find a complete match in G, which represents a solution to $R(\varphi)$; and vice versa.

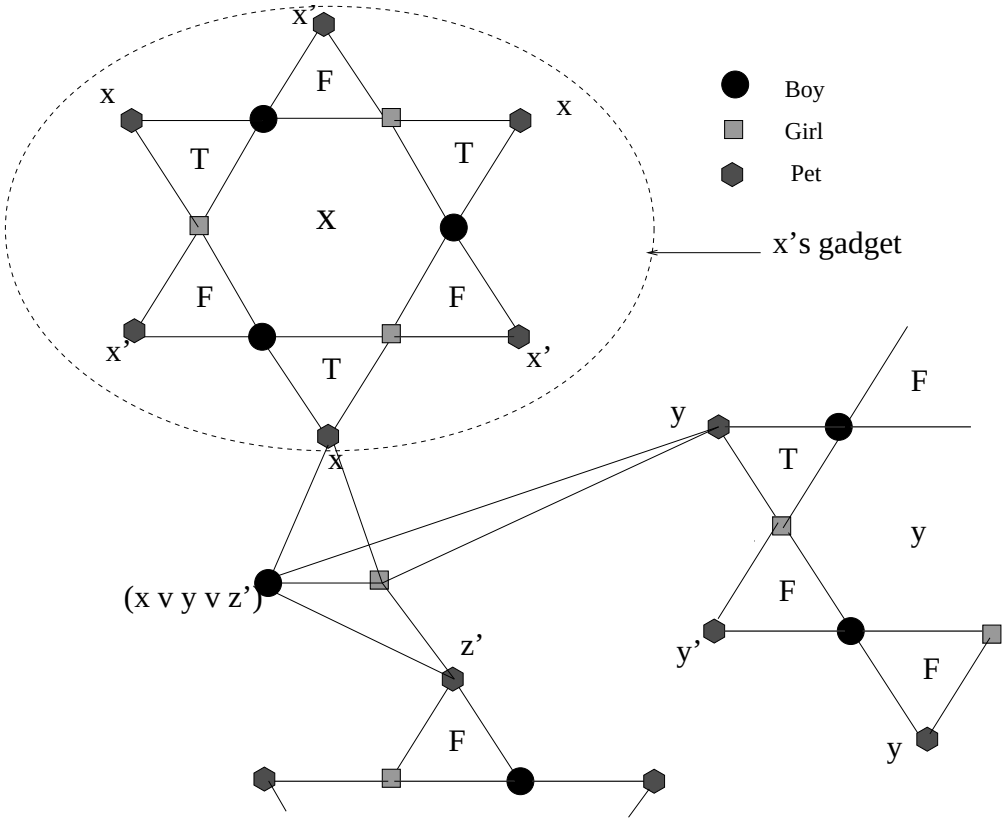


Figure 21.1: 3SAT to 3DM

Obviously 3DM problem is in NP, and it is also NP-hard because 3SAT can be reduced to 3DM. So 3DM is NP-complete.

21.2 Knapsack

Definition 21.3 (Knapsack) *This is also known as the "subset-sum" problem. Given a set of n numbers $S = \{a_1, a_2, \ldots, a_n\}$ and a number K, can we find a subset $S\prime \subseteq S$, such that $\sum_{x \in S\prime} x = K$.*

Theorem 21.4 *3DM can be reduced to Knapsack.*

From 3DM to Knapsack

3DM can be reduced to Knapsack as follows: Suppose we have n boys, n girls, and n pets. For each triangle connecting a boy, a girl and a pet, we create three n-digit numbers. Each number encodes the "index" of corresponding boy, girl or pet. All digits in that number are set to 0 except the one corresponding to the index of that boy (or girl or pet). For instance, suppose $n = 4$, then boy $No.3$ is represented by the number

0010 (start from left). For each triangle of compatible boys, girls and pets, we concatenate their numbers to create a $3n$-digit number. For instance, a triangle which connecting $No.3$ boy, $No.2$ girl and $No.1$ pet is represented by 001001001000, as shown in the first line of following table.

boys	girls	pets	Knapsack
0 0 1 0	0 1 0 0	1 0 0 0	a1
1 0 0 0	0 0 1 0	0 0 1 0	a2
. . . .	. . . .	. . . .	.
. . . .	. . . .	. . . .	.
0 0 0 1	1 0 0 0	0 1 0 0	am
1 1 1 1	1 1 1 1	1 1 1 1	K

We can set the base of all the numbers large enough, in order to eliminate the carry in additions (a base of $\#triangles + 1$ will do).

Then these $3n$-digit numbers (a total of m of them, where m is the number of triangles) will be the set of numbers in Knapsack problem and the target number K is the $3n$-digit number with all digits being 1, as shown in above table.

The size of this table is $3nm$, and clearly the reduction is polynomial time in size of 3DM. The equivalence of the two problems is evident from the one-to-one mapping between boys, girls or pets to n-digit numbers and between the triangles of compatible boys, girls and pets to the $3n$-digit numbers. Because no carry would happen when the addition is done, the all-1 target number K corresponds to a complete 3D-matching among the n boys, n girls, and n pets.

Since we can reduce 3DM to Knapsack, then we know that Knapsack is NP-hard because 3DM is NP-complete. Furthermore, Knapsack is obviously in NP, so Knapsack is NP-complete.

21.3 3SAT $\leq_p$ IMF(Integer Multicommodity Flow)

Definition 21.5 (IMF) *Given a directed network graph $G = (V, E)$, each edge has a capacity $c(e)$. N commodities need to be shipped across the network, each of them is described by its source-sink pair $s_i - t_i$ and a demand $d_i \in Z^+$. Let $f_i(e)$ denote the amount of commodity i flowing on arc e. Can we find a set of $f_i(e) \in Z^+$ which satisfies the capacity constraints, flow conservation and demand:*

1.capacity constraints: for all e, $\sum_{i=1}^{N} f_i(e) \leq c(e)$.

2.flow conservation: for $\forall$ node $v \notin s_i, t_i$, $\sum_{e \in \delta_{in}(v)} f_i(e) = \sum_{e \in \delta_{out}(v)} f_i(e)$.

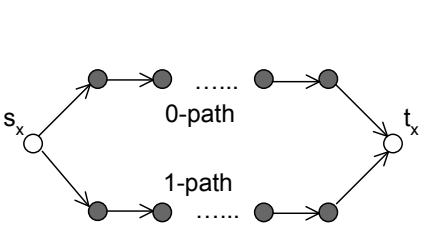
3.demand: $\sum_{e \in \delta_{out}(s_i)} f_i(e) \geq d_i$.

From 3SAT to IMF

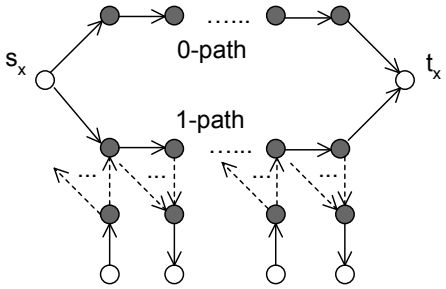
As we mentioned in the reduction from 3SAT to 3DM, the general 3SAT reduction includes three aspects: flip-flop for literals, constraints for clauses and consistency maintenance. We will use the similar technique in following reduction from 3SAT to IMF. And first of all, we set the demand for any $s_i - t_i$ pair to 1 and also set the capacity on each edge to 1. Therefore, no two flows can share one edge. This will make our transformation easier.

Flip-Flops: Each literal is represented by a flip-flop subnetwork, as shown in Figure21.2(a). For instance, the corresponding flip-flop of literal x contains a source s_x and a sink t_x. There are two possible paths from s_x to t_x. To meet the demand d_x of pair $s_x - t_x$, a flow must (and only need to) travel through one of the two paths. If the literal x is assigned 1, we choose the $1 - path$(bottom path in the figure) to satisfy d_x; otherwise we use $0 - path$(top path in the figure). Since we set the capacity of each edge to 1 and also the demand to 1, one path can be used only for one flow, i.e., two flows cannot share any edge. This is the cornerstone to implement constraints.

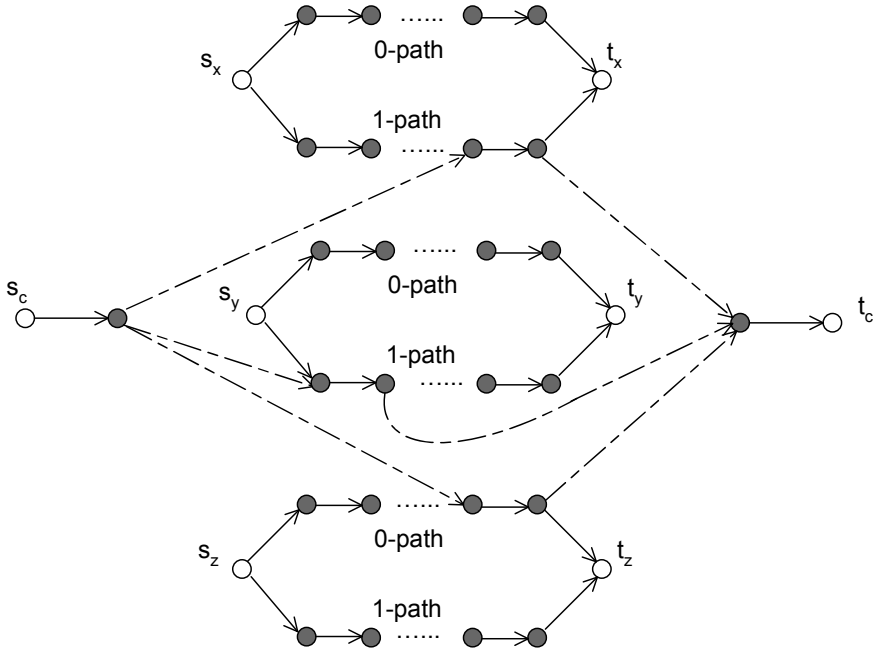
Constraints: A clause c is also represented by a subnetwork as shown in Figure21.2(b). The subnetwork contains three paths, which go through the paths of the flip-flop networks of corresponding literal. The constraints are implemented by saying that a flow from s_c to t_c must be satisfied by going through the $0 - path$ or $1 - path$ of at least one flip-flop network.



(a) Flip-flop subnetwork for literal



(c) Shared items



(b) Clause subnetwork

Figure 21.2: From 3SAT to IMF

For instance, in Figure21.2(b), the subnetwork of clause $c = \overline{x} \vee \overline{y} \vee z$ contains three flip-flops for three

literals. If the term of one literal is positive, the clause subnetwork will go across the $0-path$ of that literal's flip-flop network, otherwise the clause subnetwork will go across the $1-path$ of that flip-flop network, shown by the dotted lines. To satisfy the demand from s_c to t_c, at least one of the three paths must be "available", which is decided by the assignment of the literals. For the example in the figure, if x is assigned to 0, $s_x - t_x$ will be satisfied by the $0-path$ of x, then $s_c - t_c$ can be satisfied through the $1-path$ of x. If x is assigned to 1, $1-path$ of x must be used to satisfied the demand of $s_x - t_x$, so $s_c - t_c$ must try to use other two paths to satisfy the demand.

And the flip-flop subnetwork of one literal can be reused for different clauses, which is shown in Figure21.2(c). Two clauses c_1 and c_2 both contain literal z, so their subnetworks both go across the subnetwork of z. Furthermore, in the figure, two clauses both go across the $1-path$ of z, which indicates that they both contain the term $\overline{z}$.

Consistency: As we mentioned in the construction of flip-flops, exactly one path of the flip-flop subnetwork must be chosen to meet the demand. And one path cannot have two flows on it. This constraint is consistent to the assignment constraint for boolean function.

So we can see that IMF is NP-hard, and it can be check in polynomial time. So IMF is NP-complete.

21.4 3SAT $\leq_p$ Clustering

Definition 21.6 (Clustering) *Given a set of n points $P = \{p_1, p_2, ..., p_n\} \subseteq Z^m$ in m dimension space with distance definition d, an integer K and a limit R, is there a partition of P into sets $\{S_1, S_2, ..., S_K\}$ such that the following constraint holds:*

$$\max_{1 \leq i \leq K} \{ \min_{c_i \in S_i} \{ \max_{p_j \in S_i} d(p_j, c_i) \} \} \leq R$$

where c_i is the "center" of subset S_i.

From 3SAT to Clustering

We will reduce 3SAT to a Clustering problem on euclidean space. As the reductions of 3SAT in previous sections, we will consider three aspects.

Flip-Flops: The flip-flop gadget for one literal is shown in Figure21.3(a). For x, we set two points labeled x and $\overline{x}$. When we assign x to 1 or 0, we will correspondingly choose point x or $\overline{x}$ as a "center". And based on the topology, we know that if x is the center, then every point in the gadget belongs to the cluster centered at x except the point c_2 connecting to $\overline{x}$. If $\overline{x}$ is the center, only c_1 connecting to x does not belong to the cluster centered at $\overline{x}$. c_1, c_2 are connection points which are used to connect other gadgets.

Constraints: A clause also corresponds to a gadget(an triangle of side $2R$), as shown in Figure21.3(b). We assign each term in the clause to be the midpoint of one side of the clause gadget. For clause $(x \vee \overline{y} \vee z)$ in the figure, midpoints are x, $\overline{y}$, z. And the connection points farthest from them are labeled x', $\overline{y}'$, z' respectively. If one of the midpoints are chosen as the "center" of a cluster, all the other points in the gadget belongs to this cluster except that farthest connection point.

We connect each clause to three corresponding literal's flip-flop gadgets through its connection points. Figure21.4 is an example for formula $(\overline{x} \vee y \vee \overline{z}) \wedge (x \vee \overline{y} \vee z)$. When a clause gadget is connected to a flip-flop gadget, we use an empty dummy gadget. Then the gadget for one literal can be reused for different clauses.

From the example, we notice that x' is connected to x through a dummy gadget. Therefore, if x is chosen, i.e. x is assigned 1, b_2 must be chosen to cover the dummy gadget. Then x' is covered by the cluster entered

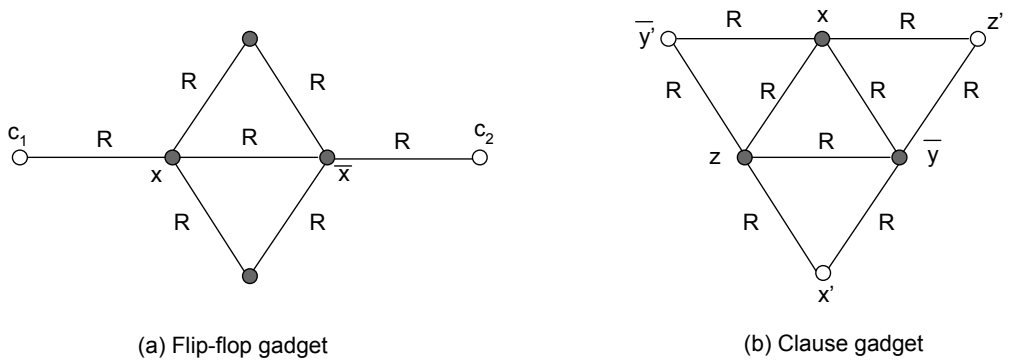


Figure 21.3: Gadget used in reduction of 3SAT to CLUSTERING

on b_2, so the clause gadget can be covered by cluster centered on x. If $\overline{x}$ is chosen, i.e. x is assigned to 0, then b_1 must be chosen to cover the dummy gadget. So to cover x', we must chose one of $\overline{y}$ and z. Here we can say "must choose" some gadgets because we make a constraint on the number K of sets for Clustering problem. We set $K = \#literals + 4 \times (\#clauses) = \#gadgets$, then each gadget only corresponds to one set.

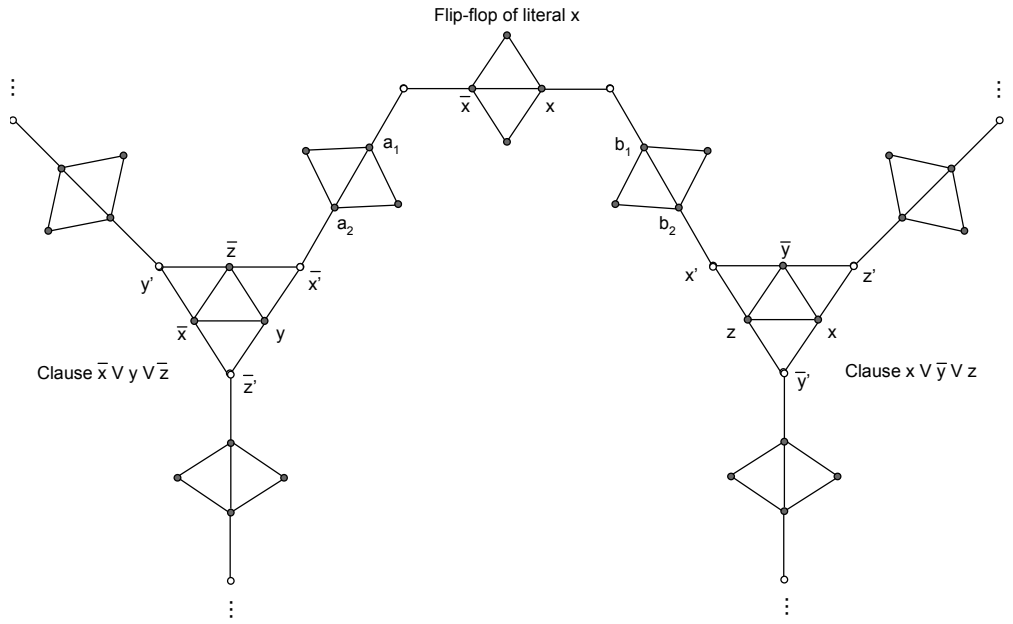


Figure 21.4: Formula reduction in 3SAT to CLUSTERING

Consistency: As we mentioned, for one flip-flop gadget, only one cluster can be chosen, centered on the point labeled either positive form or negative form. So it is consistent to the literal assignment in CNF.

So we conclude that Clustering problem is also NP-hard, and it can be checked in polynomial time, so it is NP-complete.

21.5 Reduction by Generalization

Reduction by Generalization is a reasonably systematic reduction method which can be used to prove NP-hard. If problem A is generalized by another problem B, and A can be reduced to B, then the NP-completeness of A implies the NP-hard of B, i.e., the general case is at least not "easier" than special case.

$$A \rightarrow B \Rightarrow NPC(A) \rightarrow NP - hard(B)$$

We will show some examples.

21.5.1 From HAM to TSP

We can prove the NP-hard of famous TSP(Traveling Salesman Problem) by the generalization(reduction) of HAM(Hamiltonian Problem). We assume HAM is NP-complete.

Definition 21.7 (HAM) *Given a graph $G = (V, E)$, find an ordering $< v_{\pi_1}, v_{\pi_2}, ..., v_{\pi_n}, v_{\pi_1} >$ of vertices such that every vertex is visited exactly once and all the edges on path $(v_{\pi_i}, \pi_{i+1}) \in E$.*

Definition 21.8 (TSP) *Given a graph $G = (V, E)$ where each edge has a cost, find a Hamiltonian cycle with minimal cost, i.e., find a sequence $s =< v_{\pi_1}, v_{\pi_2}, ..., v_{\pi_n}, v_{\pi_1} >$ such that $s.cost = \min_{c \in Hamiltonian\ cycles} c.cost$.*

Generalization: Given a graph for TSP, we can set the cost of each edge to 1. Then all Hamiltonian cycles(if exist) will have a cost n(n is the number of vertices), and the algorithm to TSP will find those cycles, i.e., any algorithm solves TSP can solve HAM.

By using reduction by generalization, we can prove that TSP is NP-hard. Furthermore, it is in NP, so TSP is NP-complete.

To be continued on reduction by generalization...

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein: *Introduction to Algorithms*, 2001

[2] Michael Sipser: *Introduction to the Theory of Computation*, 1997