



# Advanced algorithms and data structures

MEN Ziyang

## Summary

Cheat Sheet

“Young man, in mathematics you don’t understand things. You just get used to them.”

– John von Neumann

# Indhold

<b>1</b>	<b>Max Flow</b>	<b>4</b>
1.1	Basic concept . . . . .	4
1.2	FF algorithm . . . . .	4
1.3	EK . . . . .	4
<b>2</b>	<b>LP</b>	<b>5</b>
2.1	Basic concept . . . . .	5
2.2	SIMPLEX . . . . .	5
2.3	SIMPLEX Characteristic . . . . .	5
2.4	Auxiliary LP . . . . .	5
<b>3</b>	<b>Hash</b>	<b>7</b>
3.1	Hash function . . . . .	7
3.2	Universal hashing . . . . .	7
3.3	Application . . . . .	7
3.4	Multiply-mod-prime . . . . .	7
3.5	Multiply-shift . . . . .	8
<b>4</b>	<b>Randomized</b>	<b>9</b>
4.1	RandQS . . . . .	9
4.2	RandMinCut(G) . . . . .	9
<b>5</b>	<b>V-EB Tree</b>	<b>10</b>
5.1	Naive approach . . . . .	10
5.2	Twolevel . . . . .	10
5.3	Recursive . . . . .	10
5.4	vEB . . . . .	10
5.5	RS-vEB . . . . .	10
<b>6</b>	<b>NPC</b>	<b>11</b>
6.1	Definition . . . . .	11
6.2	P . . . . .	11
6.3	NP . . . . .	11
6.4	Instances . . . . .	11
6.4.1	$SAT \leq_p 3 - SAT$ . . . . .	11
6.4.2	$3\text{-CNF-SAT} \leq_p \text{SUBSET-SUM}$ . . . . .	12
6.4.3	$3\text{-CNF-SAT} \leq_p \text{CLIQUE}$ . . . . .	12
6.4.4	$\text{CLIQUE} \leq_p \text{VERTEX-COVER}$ . . . . .	12
<b>7</b>	<b>Exact Algorithm</b>	<b>13</b>
7.1	Definition . . . . .	13
7.2	TSP . . . . .	13
7.3	MIS via Branching . . . . .	13
7.4	Bar fight prevention aka k-Vertex Cover . . . . .	14
7.4.1	Problem description . . . . .	14
7.4.2	Kernelization . . . . .	14
7.5	Bounded Search Tree . . . . .	15
7.6	FPT vs XP . . . . .	15
<b>8</b>	<b>Approximately Algorithm (1)</b>	<b>16</b>
8.1	Definition . . . . .	16
8.2	Vertex Cover . . . . .	16
8.3	TSP . . . . .	16
8.4	Set Cover . . . . .	16
8.4.1	Difinition . . . . .	16
8.5	Procedure . . . . .	17
8.6	Approximation . . . . .	17

<b>9</b>	<b>Approximately Algorithm (2)</b>	<b>19</b>
9.1	MAX-3-SAT . . . . .	19
9.2	Weighted Vertex Cover . . . . .	19
9.3	Approximation schemes . . . . .	19
9.3.1	SUBSET-SUM . . . . .	19

# 1 Max Flow

## 1.1 Basic concept

Capacity constraint:  $0 \leq f(u, v) \leq c(u, v)$ .

Flow conservation : for all  $u \in V \setminus \{s, t\}$ , require  $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$ .

The value  $|f|$  of a flow is defined as:  $|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$ , flow in source minus flow out source. **Maximum flow** problem is to make the flow amount maximize.

No **anti-parallel** edge: if  $(u, v) \in E$ , then  $(v, u) \notin E$ .

## 1.2 FF algorithm

Procedure: Start with  $f(u, v) = 0$ , giving an initial flow of value 0, we increase the flow value in G by finding an 'augmenting path' in an associated 'residual network'  $G_f$ . We repeatedly augment the flow until the residual network has no more augmenting paths.

How to build the **residual network**:  $c_f(u, v) = c(u, v) - f(u, v)$ , which is, for those edge does not have full flow, the augmentation path flow is the amount of flow which can be increased, if the flow is full, this edge would not be added to the residual network.

For edge in initial graph, we can add two different edges in  $G_f$ , the same direction edge has amount shown above, and the reverse edge has capacity of  $f(u, v)$ .

The **augmentation flow** is:  $(f \uparrow f')(u, v) = f(u, v) + f'(u, v) - f'(v, u)$ , where  $f$  is the flow in initial graph and  $f'$  is the flow in residual network.

An **augmenting path p** is a simple path from s to t in the residual network  $G_f$ , the residual capacity of p is:  $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$ , this value is the upper bound of  $f'$  we can increase to the  $f$ .

**Net flow** across the cut (S,T) is:  $f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - f(v, u)$ , which is the net flow from S to T minus that from T to S.

**Cut capacity** is sum of the edge capacity, considering only those edges from S to T. Minimum Cut of a network is a cut whose capacity is minimum. The value of any flow f in a flow network G is bounded from above by the capacity of any cut of G.

**Max flow Min cut theorem**: f is a maximum flow in G  $\rightarrow$  The residual network  $G_f$  contains no augmenting paths  $\rightarrow |f| = c(S, T)$  for some cut (S,T) of G.

(2  $\rightarrow$  3) is: all edge from S to T should be full and 0 vice versa.

(3  $\rightarrow$  1) is: flow from S to T is bounded by the capacity of (S,T).

## 1.3 EK

We choose the augmenting path as a shortest path from s to t in the residual network, where each edge has unit distance (weight).

$O(VE^2)$

In the process we find the maximum flow, the same shortest path can remain at most  $O(E)$  time and the maximum shortest length is  $O(V - 1)$ , thus the total complexity is  $O(VE)$ , the complexity of finding the shortest path is  $O(E)$ .

## 2 LP

### 2.1 Basic concept

A particular set of values for variables satisfies all constraints is said to be feasible solution.

General LP form is minimized problem. **Standard Form** is maximization of a linear function, every constraint has less equation, every variable should not smaller than 0. Change max to min only negating the coefficients of the objective function.

Convert to the standard form:

1. variable  $x_j$  without the non-negative constraints is replaced by  $s_j = x'_j - x''_j$ , which is, we use the difference of two positive number to represent other numbers.
2. reverse the larger inequality to smaller ones.

**Slack variables:** non-negative, add to the constraint to make it become equality.

Make the slack variables all come to the left would change LP to Slack Form, the LHS variable is called basic variables, LHS is base, Setting right-hand side variables of the slack form to 0 yields a basic solution, which does not need to be feasible.

### 2.2 SIMPLEX

For LP in slack form.

Set all nonbasic variables to 0, compute values of basic variables and thus gives the feasible basic solution.

Repeat:

Use the variable  $x_i$  which has maximum positive coefficient, generally increase  $x_i$  to find the first binding (become negative) basic variable  $x_j$ .

Swap the position of  $x_i$  and  $x_j$ , all occurrences of  $x_i$  in LP would be replaced by the RHS of binding constraint. (Pivoting)

Update the new basic solution and objective value by value of  $x_i$  and  $x_j$  when they are binding.

LP is unbounded if no such  $x_j$  exists.

Until all coefficients in the objective function are  $\leq 0$  or LP unbounded.

### 2.3 SIMPLEX Characteristic

The number of basic solutions is finite, for each basic solution, the objective value does not change and the same basic solution appears twice.

Cycling: If SIMPLEX fails to terminate then it cycles

To avoid: a very natural rule:

Select the entering variable with the greatest positive coefficient in the objective function of current slack form.

If the coefficient is same, the candidate with the smallest subscript is made to leave.

To avoid: Always choose the entering and leaving variables with the smallest indices.

### 2.4 Auxiliary LP

Optimal value of auxiliary LP will indicate if the original LP is feasible. If original LP is feasible, then the slack form of the auxiliary LP will yield a feasible basic solution to the original LP

$$\max : \quad -x_0 \quad (1)$$

$$s.t. \quad \sum_{j=1}^n a_{ij}x_j - x_0 \leq b_i \quad \forall i \in [1, m] \quad (2)$$

$$x_j \geq 0 \quad \forall j \in [0, m] \quad (3)$$

**Duality:**

Construct a linear combination of the constraints using non-negative multipliers  $y_1, y_2$ , and  $y_3$ ;

LHS will be an upper bound for the LP if the coefficient at each  $y_i$  is at least as big as the corresponding coefficient in the objective function;

RHS should be greater than the objective function if all constraints above are satisfied;

Good upper bound: minimize right-hand side s.t. constraints.

**Weak Duality:**

$$\sum_{j=1}^n c_j x_j^* \leq \sum_{i=1}^m b_i y_i^*$$

which states  $x^*$  and  $y^*$  are optimal solutions to the primal and to the dual LPs, respectively.

### 3 Hash

#### 3.1 Hash function

Have a large universe  $U$  of keys which randomly to range  $[m] = \{0, \dots, m-1\}$  of hash values.

A **truly random hash function**  $h : U \rightarrow [m]$  is a random variable in the class of all functions  $U \rightarrow [m]$ , that is, it consists of a random variable  $h(x)$  for each  $x \in U$ .

Three things to care:

1. size of random seed;
2. time it takes to calculate  $h(x)$
3. Properties of the random variable.

#### 3.2 Universal hashing

$h$  to be **universal**: for any given distinct keys  $x, y \in U$ , when  $h$  is picked at random (independently of  $x$  and  $y$ ), we have **low collision probability**:  $Pr_h[h(x) = h(y)] \leq 1/m$

$h$  to be **c-approximately universal** if  $Pr_h[h(x) = h(y)] \leq c/m$

**Pairwise events** of the form that for given distinct keys  $x, y \in [u]$  and possibly non-distinct hash values  $q, r \in [m]$ , we have  $h(x) = q$  and  $h(y) = r$

$h : [u] \rightarrow [m]$  is **strong universal** if the probability of every pairwise event is  $1/m^2$ .

Note if  $h$  is strong universal it is also universal since:

$$Pr[h(x) = h(y)] = \sum_{q \in [m]} Pr[h(x) = q \cap h(y) = q] = m/m^2 = 1/m \quad (4)$$

An equivalent definition of strong universality is that each key is hashed uniformly into  $[m]$ , and that every two distinct keys are hashed independently.

We say a random hash function  $h : U \rightarrow [m]$  is **c-approximately strong universal** if:

1. for all  $x \in U$  and for all hash value  $q \in [m]$ , we have  $Pr[h(x) = q] \leq c/m$
2. every pair of distinct keys hash independently.

#### 3.3 Application

**Hash tables with chaining:**

Have a set  $S \subseteq U$  of keys,  $n = |S|$  and  $m \geq n$ , for  $i \in [m]$ ,  $L[i]$  is the list of keys that hash to  $i$ , to find out if a key  $x \in U$  is in  $S$ , we only have to check if  $x$  is in the list  $L[h(x)]$ .

Assume that  $x \notin S$ ,  $I[y]$  be an indicator variable which is 1 if  $h(x) = h(y)$  and 0 otherwise, the expected number of elements in  $L[h(x)]$  is:

$$E_h[|L[h(x)]|] = E_h\left[\sum_{y \in S} I(y)\right] = \sum_{y \in S} E_h[I(y)] = \sum_{y \in S} Pr[h(y) = h(x)] \leq n/m \leq 1 \quad (5)$$

**Assigning a unique signature  $s(x)$  to each key:**

Wish  $s(x) \neq s(y)$  for all distinct keys  $x, y \in S$ ;

Pick universal hash function  $s : U \rightarrow [n^3]$ , the probability of collision is:

$$Pr_s[\exists \{x, y\} \subseteq S : s(x) = s(y)] \leq \sum_{\{x, y\} \in S} Pr[s(x) = s(y)] \leq C_n^2/n^3 \leq 1/(2n) \quad (6)$$

#### 3.4 Multiply-mod-prime

$1 < m < u$  and  $p$  is a prime number  $p \geq u$

Pick uniformly random  $a \in [p]_+ = \{1, \dots, p-1\}$  and  $b \in [p] = \{0, \dots, p-1\}$ ;

Define  $h_{a,b} : [u] \rightarrow [m]$ :

$$h_{a,b}(x) = ((ax + b) \% p) \% m \quad (7)$$

which has collision probability  $< 1/m$ .

For any given pair  $(a, b) \in [p]^2$ , define  $(q, r) \in [p]^2$  by:

$$(ax + b) \% p = q \quad (8)$$

$$(ay + b) \% p = r \quad (9)$$

Above equations define a 1-1 correspondence between pairs  $(a, b) \in [p]^2$  and pairs  $(q, r) \in [p]^2$ .

### 3.5 Multiply-shift

Addresses hashing from w-bit integers to l-bit integers.

Pick a uniformly random odd w-bit integer  $a$  and compute  $h_a : [2^w] \rightarrow [2^l]$ :

$$h_a(x) = \lfloor (ax \% 2^w) / 2^{w-l} \rfloor \quad (10)$$

It is a 2-approximately universal function, which is:

$$Pr_{odda \in [2^w]} [h_a(x) = h_a(y)] \leq 2/2^l = 2/m \quad (11)$$



## 4 Randomized

### 4.1 RandQS

Procedure: randomly choose a number  $s_i \in S$  and divided the number into two groups L and R, continue RandQS(L)+[ $s_i$ ]+RandQS(R). Iteration boundary is there is no element.

A sort algorithm with complexity  $O(n \log n)$ , belongs to Las Vegas algorithm.

Complexity analysis:

Luck case:  $s_i$  is always the median,  $T(n) = O(n) + 2T(n/2)$ , totally  $O(\log n)$  terms of size  $O(n)$ , in total  $O(n \log n)$

Unlucky case:  $s_i$  is minimum  $T(n) = \Omega(n) + T(n-1) = \Omega(n^2)$

Average case: X is the number of comparisons.  $E[X] = O(n \log n)$

Let  $X_{ij} :=$  number of comparisons between  $S_i$  and  $S_j$ , they are compared iff they are the first pivot among  $\{S_i, \dots, S_j\}$ :

$$E[X] = E\left[\sum_{i=1}^n \sum_{j>i} X_{ij}\right] = \sum_{i=1}^n \sum_{j>i} E[X_{ij}] = 2nH(n) = O(n \log n) \quad (12)$$

$$E[X_{ij}] = p_{ij} = \frac{2}{j-i+1}.$$

### 4.2 RandMinCut(G)

Procedure: continually contract random edge  $e \in E(G)$  and remove self-loops. The remaining edges are Mincut.

It returns a cut, because if it not, there remains a path from s to t which must pass C, contradiction.

$Pr[\text{Return} - C] = Pr[\epsilon_{<n-1}] \geq \frac{2}{n(n-1)}$ , proof:

$n_i := |V_i| = n - i + 1$ ,  $\epsilon_i := e_i \notin C$  and  $\epsilon_{<j} := \cap_{i=1}^{j-1} \epsilon_i$ ,  $G_1 = G, n_1 = n, |V_{n-1}| = 2, \text{return } E_{n-1}$

$Pr[\epsilon_{n-1}] = \prod_{i=1}^{n-2} Pr[\epsilon_i | \epsilon_{<i}]$

$Pr[\bar{\epsilon}_i | \epsilon_{<i}] = \frac{|C|}{|E_i|} \leq \frac{|C|}{(n_i|C|/2)} = 2/n_i$

## 5 V-EB Tree

### 5.1 Naive approach

A simple List.

empty(S), min(S), max(S), member(S) and delete(S) in worst case  $O(1)$ ;  
 predecessor(x,S), successor(x,S) is  $O(1)$  if  $x \in S$  and worst case  $\theta(|U|)$ ;  
 insert(x,S) is  $\theta(|U|)$ .

### 5.2 Twolevel

$hi_w(x) := \lfloor \frac{x}{2^{\lceil w/2 \rceil}} \rfloor$  and  $lo_w(x) := x \% 2^{\lceil w/2 \rceil}$

Complexity:

empty,min,max,member is worst case  $O(1)$ ;  
 delete =  $O(1) + 2 \times naive = O(1)$ , delete both parts;  
 predecessor and successor: if  $hi(x)$  exists but  $low(x)$  not, should be  $O(1) + 1 \times naive$ ; if  $h(x)$  not  
 $1 \times naive + O(1)$ , overall worst case  $\theta(2^{\lceil w/2 \rceil})$   
 insert:  $O(1) + 2 \times naive =$  worst case  $\theta(\sqrt{|U|})$

### 5.3 Recursive

The recursion depth of this structure, when used on the universe  $U = [2^w]$  is  $\lceil \log_2 w \rceil = O(\log \log |U|)$

Proof:

$d(w)$  is the recursion depth when working with a universe of size  $2^w$  and  $d(w) = \lceil \log_2 w \rceil$ ;  
 when  $w$  is 1,  $d(1) = 0 = \lceil \log_2(1) \rceil$   
 when  $w > 2$ ,  $d(w) = 1 + d(w') = 1 + \lceil \log_2(w') \rceil = \lceil \log_2(2w') \rceil$ , when  $w$  is even, clearly; when  $w$   
 is odd then  $2w' = w + 1$  and thus  $\lceil \log_2(2w') \rceil = \lceil \log_2(2w' - 1) \rceil = \lceil \log_2(w) \rceil$ , because we take the  
 upper bound.

Complexity:

empty, min, max: worst case  $O(1)$ ;  
 member:  $O(1) + 1 \times recursion = O(d(w)) = O(\log \log |U|)$   
 predecessor and successor  $O(1) + 1 \times recursion = O(d(w)) = O(\log \log |U|)$   
 insert and delete  $O(1) + 2 \times recursion = O(2^{d(w)}) = O(\log |U|)$

### 5.4 vEB

Exclude min(S) and/or max(S) from the set of keys stored in summary and clusters.

It makes at most one recursive call on a non-empty substructure.

### 5.5 RS-vEB

Every cluster are hashed.

Why does this change updates from worst case  $O(\log \log |U|)$  to expected  $O(\log \log |U|)$  time? Because  
 updates to a hash table take expected  $O(1)$  time rather than worst case.

Why does this only use  $O(n \Delta d(w)) = O(n \log \log |U|)$  space? Because the empty structure uses  $O(1)$   
 space and insertw only creates or updates  $O(d(w))$  substructures in the worst case. Each of these  
 costs at most an additional  $O(1)$  space

## 6 NPC

### 6.1 Definition

An instance is a tripe  $\langle G, s, t \rangle$ , a solution is a sequence of vertices forming a shortest s-to-t path.

A decision problem is a mapping from instances to  $S = \{0, 1\}$

Use the notation  $\langle x \rangle$  to refer to a chosen encoding of an instance  $x$  of a problem.

Any language  $L$  over  $\Sigma$  is a subset of  $\Sigma^*$

We can view a decision problem as a mapping  $Q(x)$  from instances  $x$  to  $\Sigma = \{0, 1\}$ , we can view  $Q$  as a language  $L$ :

$$L = \{x \in \Sigma^* \mid Q(x) = 1\} \quad (13)$$

Let  $A$  be an algorithm for a decision problem  $A(x) \in \{0, 1\}$

$A$  accepts a string  $x$  if  $A(x) = 1$

$A$  rejects a string  $x$  if  $A(x) = 0$

A language accepted by  $A$  is:  $L = \{x \in \{0, 1\}^* \mid A(x) = 1\}$ . In addition that all strings not in  $L$  are rejected by  $A$ , then we say that  $L$  is decided by  $A$ .

### 6.2 P

$P = \{L \subseteq \{0, 1\}^* \mid \text{there exists an algorithm } A \text{ that decides } L \text{ in polynomial time}\}$

If  $L$  is accepted by a polynomial-time algorithm  $A$ , it is decided by a polynomial-time algorithm  $A'$ .

### 6.3 NP

$A$  verifies a string  $x$  if there is a certificate  $y$  such that  $A(x, y) = 1$

The language verified by  $A$  is:

$$L = \{x \in \{0, 1\}^* \mid \text{there is a } y \in \{0, 1\}^* \text{ such that } A(x, y) = 1\} \quad (14)$$

$x$  is the problem and  $y$  is the solution to be verified.

$L \in NP$  iff there is a polynomial-time verification algorithm  $A$  and a constant  $c$  such that:

$$L = \{x \in \{0, 1\}^* \mid \text{there is a } y \in \{0, 1\}^* \text{ with } |y| = O(|x|^c) \text{ such that } A(x, y) = 1\} \quad (15)$$

Language  $L_1$  is polynomial-time reducible to language  $L_2$  if there is a polynomial-time computable function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  such that for all  $x \in \{0, 1\}^*$ ,  $x \in L_1 \iff f(x) \in L_2$

In this case, we write  $L_1 \leq_p L_2$ .

$L_1$  is in a sense no harder to solve than  $L_2$

Language  $L$  is **NP-complete** if:

$L \in NP$  and

$L' \leq_p L$  for every  $L' \in NP$

$L$  is **NP-hard** if property 2 holds.

### 6.4 Instances

#### 6.4.1 $SAT \leq_p 3-SAT$

A literal in  $\phi$  is an occurrence of a variable or its negation.  $\phi$  is the AND of clauses and each clause is the OR of exactly 3 distinct literals.

Procedure:

We construct a parse tree for  $\phi$  where leaves are literals and internal nodes (gate) are connectives. Construct formulas  $\phi'_k$  for each of these gates. Let  $\phi' = y_1 \cap \phi'_1 \cdots \cap \phi'_k$ .  $\phi''_i = \phi'_i$ , rewrite  $\phi''_i$  as a sum of multiply from truth table. Negative it again to obtain the form of 3-SAT.

Introducing dummy variables  $p$  and  $q$  to extend each  $\phi'$  into three variables.

#### 6.4.2 3-CNF-SAT $\leq_p$ SUBSET-SUM

Definition: SUBSET-SUM =  $\{ \langle S, t \rangle \mid \exists S' \subseteq S \text{ so that } t = \sum_{s \in S'} s \}$

Proof:

Digit  $C_j$  of  $v_i$  is 1 if  $x_i \in C_j$ .  
Digit  $C_j$  of  $v'_i$  is 1 if  $\neg x_i \in C_j$ .  
Digit  $C_i$  of  $s_i$  is 1 and digit  $C_i$  of  $s'_i$  is 2.

	$x_1$	$x_2$	$x_3$	$C_1$	$C_2$	$C_3$	$C_4$
$v_1$	1	0	0	1	0	0	0
$v'_1$	→ 1	0	0	0	1	0	1
$v_2$	0	1	0	0	1	1	0
$v'_2$	→ 0	1	0	0	0	0	1
$v_3$	0	0	1	0	0	1	1
$v'_3$	0	0	1	1	0	0	0
$s_1$	1	0	0	1	0	0	0
$s'_1$	1	0	0	0	1	0	1
$s_2$	0	1	0	0	1	1	0
$s'_2$	0	1	0	0	0	0	1
$s_3$	0	0	1	0	0	1	1
$s'_3$	0	0	1	1	0	0	0
$s_4$	1	0	0	1	0	0	0
$s'_4$	1	0	0	0	1	0	1
$t$	1	1	1	4	4	4	4

$\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$   
 $C_1 = (x_1 \vee x_2 \vee x_3)$   
 $C_2 = (x_1 \vee x_2 \vee x_3)$   
 $C_3 = (x_1 \vee x_2 \vee x_3)$   
 $C_4 = (x_1 \vee x_2 \vee x_3)$   
Satisfying assignment  
 $x_1 = 0$   
 $x_2 = 0$   
 $x_3 = 1$

Assign values to  $x_1, \dots, x_n$  that satisfy  $\phi$ .  
For  $i = 1, \dots, n$ , if  $x_i = 1$  then include  $v_i$  in  $S'$ ; otherwise include  $v'_i$ .  
Include additional numbers from  $\{s_1, s'_1, s_2, s'_2, \dots, s_k, s'_k\}$  to reach target  $t$ .

Target  $t$  is defined to be:

$$t = \overbrace{11 \dots 1}^n \overbrace{44 \dots 4}^k.$$

#### 6.4.3 3-CNF-SAT $\leq_p$ CLIQUE

Definition: CLIQUE =  $\{ \langle G, k \rangle \mid G \text{ is a graph containing a clique of size } k \}$

Proof:

For each  $C_r = l_1^r \cup l_2^r \cup l_3^r, r \in k$ , include three vertices  $v_1^r, v_2^r, v_3^r$  to  $G$

There is an edge  $v_i^r, v_j^s$  in  $G$  iff  $r \neq s$  (not in one cluster) and  $l_i^r$  is not the negation of  $l_j^s$  (they are not the negation literal to each other: no self loop)

Pick one true literal in cluster, containing the corresponding node.  $k$  nodes selected in total, other unchosen legal literal can be another nodes in graph.

#### 6.4.4 CLIQUE $\leq_p$ VERTEX-COVER

Definition: A vertex cover of  $G(V, E)$  is a subset  $V' \subseteq V$  such that every edge of  $E$  has at least one endpoint in  $V'$ .

$$VERTEX - COVER = \{ \langle G, k \rangle \mid G \text{ has a vertex cover of size } k \} \quad (16)$$

$n$  is number of node in a graph, then  $\langle G, k \rangle \in \text{CLIQUE} \iff \langle \bar{G}, n - k \rangle \in \text{VERTEX} - \text{COVER}$

## 7 Exact Algorithm

### 7.1 Definition

For any  $a > 1, \epsilon > 0$ , and any  $c \in R$ , we have  $O(n^c \cdot a^n) \subset O((a + \epsilon)^n)$

Define:  $f(n) \in O^*(g(n)) \leftrightarrow \exists c \in R : f(n) \in O(n^c \cdot g(n))$ .

The running time for the simple brute-force algorithm is:  $O(2^{m(x)} \text{poly}|X|) = O^*(2^{m(x)})$

### 7.2 TSP

Definition: Given cities  $c_1 \cdots c_n$  and distances  $d_{ij} = d(c_i, c_j)$ , find tour of minimal length, visiting all cities exactly once.

For all  $S \subseteq \{c_2 \cdots c_n\}$  and  $c_i \in S$ , define  $OPT[S, c_i] :=$  the length of any shortest path in  $S \cup c_1$  that starts in  $c_1$ , visit all of  $S$  once and ends in  $c_i$ .

Then  $\min\{OPT[\{c_2 \cdots c_n\}, c_i] + d(c_i, c_1) \mid c_i \in \{c_2 \cdots c_n\}\}$  is the length of the minimal tour.

$$OPT[S, c_i] = \begin{cases} d(c_1, c_i) & \text{if } S = \{c_i\} \\ \min\{OPT[S \setminus \{c_i\}, c_k] + d(c_k, c_i) \mid c_k \in S \setminus \{c_i\}\} & \text{if } \{c_i\} \subset S \end{cases} \quad (17)$$

Complexity:  $O(n^2 \cdot 2^n)$

Thus, we can easily compute  $OPT[S, c_i]$  in order of increasing size of  $S$ .

```

1: function TSP( $\{c_1, \dots, c_n\}, d$ )
2:   for  $j \leftarrow 2 \dots n$  do
3:      $OPT[\{c_j\}, c_j] \leftarrow d(c_1, c_j)$ 
4:   for  $j \leftarrow 2 \dots n-1$  do
5:     for  $S \subseteq \{c_2, \dots, c_n\}$  with  $|S| = j$  do
6:       for  $c_i \in S$  do
7:          $OPT[S, c_i] \leftarrow \min\{OPT[S \setminus \{c_i\}, c_k] + d(c_k, c_i) \mid c_k \in S \setminus \{c_i\}\}$ 
8:   return  $\min\{OPT[\{c_2, \dots, c_n\}, c_i] + d(c_i, c_1) \mid c_i \in \{c_2, \dots, c_n\}\}$ 

```

#### Lemma

The above procedure solves TSP by computing  $O(n^2 \cdot 2^n)$  shortest paths.

#### Proof.

The number of path lengths computed in line 7 is

$$\sum_{j=2}^{n-1} \binom{n-1}{j} \sum_{i=1}^j (j-1) \leq n^2 \sum_{j=1}^n \binom{n}{j} = n^2 \cdot 2^n$$

$n^2$  is the cycle number and  $2^n$  is the subset enumeration amount.

### 7.3 MIS via Branching

MIS: Maximum Independent Set (MIS): for undirected graph, find the maximum cardinality of  $I \subseteq V$  so each edge has at most one endpoint in  $I$ ,  $I$  is the MIS.

closed neighborhood of  $v$ :

For  $v \in V$  define  $v$  and  $v$ 's neighbor as  $N[v] := \{v\} \cup \{w \in V \mid (v, w) \in E\}$ .

Since we want to find MIS, so  $N[v] \cap I \neq \emptyset$  for all  $v \in V$ .

Procedure:

```

if  $V = \emptyset$  then return 0;
 $v \leftarrow$  vertex in  $V$  of minimum degree;
return  $1 + \max(MIS(G \setminus N[v]) \mid w \in N[v])$ 

```

Choose the node  $v$  in  $V$  which has least degree and iteration on graph without every element in  $N[v]$  sequentially.

Complexity:

$$T(0) = 1 \quad (18)$$

$$T(n) \leq \sum_{w \in N[v]} T(n - (d(w) + 1)) \quad (19)$$

$$\leq 1 + (d(v) + 1) \cdot T(n - (d(v) + 1)) \quad (20)$$

$$= 1 + s \cdot T(n - s) \quad (21)$$

$$\in O(3^{n/3}) \subset (1.44225^n) \quad (22)$$

## 7.4 Bar fight prevention aka k-Vertex Cover

### 7.4.1 Problem description

You knows everyone and knows which pair of people would fight. Allow you to block k of the n people who wants in, whether this enough to prevent fight.

Equivalent: whether there is a Independent Set of size  $n - k$ ;

or: whether there is a subset  $C \subset V$  of size  $|C| \leq k$  such that every edge has at least one endpoint in C, namely the k-Vertex Cover.

### 7.4.2 Kernelization

Target: Every edge has at least one endpoint in the node set.

Strategy:

"Bar fight prevention" via Kernelization

Consider the conflict graph  $G = (V, E)$ .

Idea: If  $d(v) = 0$ : let  $v$  in and drop  $v$  from  $G$ .

Why? Safe because no conflicts.

Idea: If  $d(v) \geq k + 1$ : reject  $v$ , drop  $v$  from  $G$ , and decrease  $k$ .

Why? Not rejecting  $v$  means rejecting  $d(v) > k$  people.

Note: If  $d(v) \leq k$  for all  $v$  and  $|E| > k^2$ , there is no solution.

Why? Each rejection resolves at most  $k$  conflicts.

Better 2: The above ideas reduce to a graph  $H$  with  $|V| \leq 2k^2$  vertices.

Why?  $|V| = \sum_{v \in V} 1 \leq \sum_{v \in V} d(v) = 2|E| \leq 2k^2$

Now try all  $\binom{2k^2}{k}$  subsets of  $k$  people.  $((\frac{2 \cdot 10^2}{10}) \approx 2.24 \cdot 10^{16})$ .

Idea: If  $N[v] = \{v, w\}$ : reject  $w$ , drop  $N[v]$  from  $G$ , and decrease  $k$ .

Why? In any solution that lets  $w$  in, we can let  $v$  in instead. Never worse.

Better 3: The above ideas reduce to a graph  $H$  with  $|V| \leq k^2$  vertices.

Why?  $|V| = \sum_{v \in V} 1 = \frac{1}{2} \sum_{v \in V} 2 \leq \frac{1}{2} \sum_{v \in V} d(v) = |E| \leq k^2$

Now try all  $\binom{k^2}{k}$  subsets of  $k$  people.  $((\frac{10^2}{10}) \approx 1.73 \cdot 10^{13})$ .

Figure 2: Kernal principle

"Bar fight prevention" via Kernelization

```

1: function BarFightPrevention(k, G)
2:   k', H, C ← BFP-Kernel(k, G)
3:   if H has ≤ (k')2 edges and BFP-Brute-Force(k', H) returns a solution C' then
4:     return C ∪ C'
5:   return "No solution"
6: function BFP-Kernel(k, G)
7:   k' ← k, H ← G, C ← ∅
8:   loop
9:     if Some v has d(v) = 0 then
10:      H ← H \ {v}
11:     elseif Some v has d(v) ≥ k' + 1 then
12:      H ← H \ {v}, C ← C ∪ {v}, k' ← k' - 1
13:     elseif Some v has N[v] = {v, w} for some w then
14:      H ← H \ N[v], C ← C ∪ {w}, k' ← k' - 1
15:     else
16:      return k', H, C
17: function BFP-Brute-Force(k, G = (V, E))
18:   for every subset C ⊆ V of size k do
19:     if C is a vertex cover of G then
20:       return C
21:   return "No solution"

```

Figure 3: Algorithm

All node rejected, in total  $k'$ , in above are included in the solution, then the final enumeration should just enumerate  $(k - k')$  via  $k^2$  nodes.

Complexity analysis: For any fixed k:

$$\mathcal{O} \left( m + n + \binom{k^2}{k} k^2 \right) \subseteq \mathcal{O} (m + n + (ke)^{2k+2}) = \mathcal{O}_k(m + n) \quad (23)$$

m is to check whether H has  $\leq (k')^2$  edges and n is to check the node degree constraint.

checking a solution need to check the solution is a vertex cover, maximum  $k^2$  edges in kernel needs to traverse.

The subgraph H we reduced to before brute-forcing is called a Kernel for the Bar Fight Prevention problem, and the process of finding such a kernel is called Kernelization.

The general idea is to use the parameter k to quickly reduce to a smaller subproblem of the same type, whose size ideally depends only on k and not on n.

## 7.5 Bounded Search Tree

Note: For each edge  $(u, v) \in E$ , at least one of  $u, v$  must be rejected.

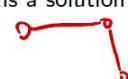
Idea: Pick arbitrary edge  $(u, v)$ , and recursively try with  $u$  rejected and with  $v$  rejected.

```

1: function BFP-Bounded-Search( $k, G$ )
2:   if  $G$  has no edges then
3:     return  $\emptyset$ 
4:   if  $k > 0$  then
5:     Let  $(u, v)$  be an arbitrary edge of  $G$ 
6:     for  $w \in \{u, v\}$  do
7:       if BFP-Bounded-Search( $k - 1, G \setminus \{w\}$ ) returns a solution  $C$  then
8:         return  $C \cup \{w\}$ 
9:   return "No solution"

```

$|E| = \frac{1}{2} \sum_{v \in V} d(v)$   
 hand-shaking!



This recursive procedure has depth at most  $k$  (just need to reject  $k$  people), thus total subproblems needs to be considered at most  $2^k$ ;

If we start by rejecting all vertices of degree  $d(v) \geq k + 1$ , the resulting graph has at most  $|E| = \frac{1}{2} \sum_{v \in V} d(v) \leq \frac{1}{2}nk$  edges, so constructing (checking) each subproblem can be done in  $O(nk)$  time.

$O(m + nk \cdot 2^k)$ ,  $m$  is the selection of arbitrary edge,  $nk$  is to check whether it is a vertex cover.

## 7.6 FPT vs XP

FPT Definition: A parameterized problem is Fixed Parameter Tractable (FPT) if it has an algorithm with running time  $f(k) \cdot n^c$  for some function  $f$  and some constant  $c \in \mathbb{R}$ .

XP Definition: A parameterized problem is Slice-wise Polynomial (XP) if it has an algorithm with running time  $f(k) \cdot n^{g(k)}$  for some functions  $f, g$ .

## 8 Approximately Algorithm (1)

### 8.1 Definition

An algorithm for an optimization problem has approximation ratio  $\phi(n)$  if for every input of size  $n$ :

$$\max\left\{\frac{C}{C^*}, \frac{C^*}{C}\right\} \leq \phi(n) \quad (24)$$

minimization problem is left and maximization is right.

### 8.2 Vertex Cover

Procedure:

while  $E(G)$  not empty, arbitrary choose one edge  $(u, v)$  in  $E$  and add endpoints into  $C$ ;  
remove all edges incident on  $u$  or  $v$  from  $E(G)$

Complexity:

$O(|V| + |E|)$ ;

$|V|$  is maximum number of endpoints can be removed and  $|E|$  is the maximum number of edges can be chosen.

Thm.: APPROX-VERTEX-COVER is a 2-approximation algorithm

Proof:

Since one endpoint of each in  $A$  must be in  $C^*$ , because  $C^*$  needs to cover every edge;

$$|C^*| \geq |A| = |C|/2 \implies \frac{|C|}{|C^*|} \leq 2$$

### 8.3 TSP

Graph satisfy the Triangle inequality:  $c(uw) \leq c(uv) + c(vw)$ , find minimum weight cycle through all vertices.

Procedure and proof:

#### Theorem

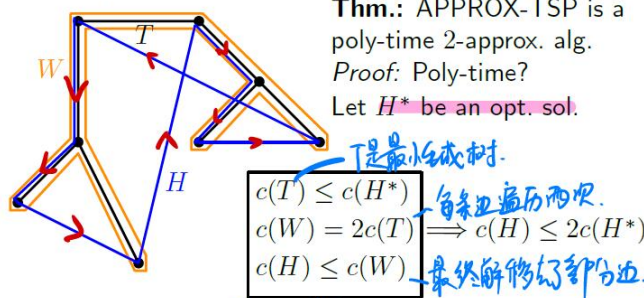
APPROX-TSP( $G, c$ )

Find MST  $T$  每个顶点至少访问一次.

Make Euler tour  $W$  using each edge of  $T$  twice

Shortcut  $W$  to  $H$  by skipping duplicates

Return  $H$



Thm.: APPROX-TSP is a poly-time 2-approx. alg.

Proof: Poly-time?

Let  $H^*$  be an opt. sol.

Correct: Euler tour is to visit every edge once and return to the start point.

MST is smaller than  $H^*$  because it does not have to visit every node twice.

Third constraint is based on Triangle inequality.

### 8.4 Set Cover

#### 8.4.1 Definition

**Input:** Pair  $(X, F)$ , where  $X$  is a finite set and  $F \subseteq P(X)$  is a family of subsets of  $X$

**Goal:** Find  $C \subseteq F$  covering  $X$ , i.e.,  $\bigcup_{S \in C} S = X$ , with  $|C|$  minimum.



Select minimum number of subset to cover S.

## 8.5 Procedure

Greedy Algorithm

GREEDY-SET-COVER( $X, \mathcal{F}$ )

$i := 0$  当前还有点可选.

while  $X \setminus S_{<i+1} \neq \emptyset$

$i := i + 1$

Pick  $S_i \in \mathcal{F}$  with  $\max |S_i \setminus S_{<i}|$  选交集

Return  $\mathcal{C} := \{S_1, \dots, S_i\}$  选剩下集合中包含最多未选点的集合

Here,  $S_{<i} := \bigcup_{j=1}^{i-1} S_j$ .

第i步前已经选到的点,

直到全部 cover

## 8.6 Approximation

Thm.:

$|C| \leq H_{|X|} \cdot |C^*|$ , where  $H_n := \sum_{i=1}^n 1/i \leq \ln n + 1$ , so it is a  $O(\log n)$ -approx. alg.

Proof:

**Thm.:**  $|C| \leq H_{|X|} \cdot |C^*|$ .

For  $x \in S_i \setminus S_{<i}$ , define

$$c_x := \frac{1}{|S_i \setminus S_{<i}|}.$$

↓ 构造点的数量

这一次

$$\sum_{x \in X} c_x = \sum_{i=1}^{|C|} \sum_{x \in S_i \setminus S_{<i}} c_x = \sum_{i=1}^{|C|} 1 = |C|.$$

↓  $|S_i \setminus S_{<i}|$

C 为近似解

↓  $|S_i \setminus S_{<i}| \cdot c_x = 1$

GREEDY-SET-COVER( $X, \mathcal{F}$ )

$i := 0$

while  $X \setminus S_{<i+1} \neq \emptyset$

$i := i + 1$

Pick  $S_i \in \mathcal{F}$  with  $\max |S_i \setminus S_{<i}|$

Return  $\mathcal{C} := \{S_1, \dots, S_i\}$

$|S_i \setminus S_{<i}|$  elements.

It has shown that  $|C|$  can be indicated as the exact sum of elements in  $X$ .

Then we have the Lemma:

$$\sum_{x \in S} c_x \leq \sum_{i=1}^{|S|} \frac{1}{i} = H_{|S|}, \quad \forall S \in \mathcal{F} \quad (25)$$

Proof of Thm.:

$$|C| = \sum_{x \in X} c_x \quad (26)$$

$$\leq \sum_{S \in C^*} \sum_{x \in S} c_x \quad (27)$$

$$\leq \sum_{S \in C^*} H_{|S|} \quad (28)$$

$$\leq \sum_{S \in C^*} H_{|X|} = |C^*| \cdot H_{|X|} \quad (29)$$

Eq. 27 is that every solution always contain more or equal elements than  $|X|$   
Eq. 28 is applying Eq. 25

Proof of Lemma:

Let  $u_i := |S \setminus S_{<i+1}|$   
then  $u_{i-1} = |S \setminus S_{<i}| \leq |S_i \setminus S_{<i}|$ , since we choose maximum  $|S_i \setminus S_{<i}|$  every time.

$$\sum_{x \in S} c_x = \sum_{i=1}^{|C|} \sum_{x \in S \cap (S_i \setminus S_{<i})} \frac{1}{|S_i \setminus S_{<i}|} \quad (30)$$

$$= \sum_{i=1}^{|C|} \frac{|S \cap (S_i \setminus S_{<i})|}{|S_i \setminus S_{<i}|} \quad (31)$$

$$= \sum_{i=1}^{|C|} \frac{u_{i-1} - u_i}{|S_i \setminus S_{<i}|} \leq \sum_{i=1}^{|C|} \frac{u_{i-1} - u_i}{u_{i-1}} \quad (32)$$

## 9 Approximately Algorithm (2)

### 9.1 MAX-3-SAT

Definition:

Find assignment that maximizes the number of true clauses.

Procedure:

For each variable  $x_i \in \phi$ , choose  $x_i \in \{0, 1\}$  by flipping fair coin.

Thm.:

RANDOM-ASSIGNMENT is a  $8/7$ -approximately alg.

Proof:

$$\Pr[\text{Clause } C_i \text{ not satisfied}] = \frac{1}{8}$$

$$X := \sum_{i=1}^n [C_i] = \# \text{ satisfied clauses.}$$

$$E[X] = \sum_{i=1}^n E[C_i] = \frac{7n}{8}$$

$$\text{Approximation ratio: } \frac{C^*}{C} = \frac{C^*}{7n/8} \leq \frac{n}{7n/8} = 8/7$$

### 9.2 Weighted Vertex Cover

Definition:

Find vertex cover  $C$  with minimum  $w(C) = \sum_{v \in C} w(v)$  Solution:

$$x_v \in \{0, 1\}, \quad \forall v \in V \quad (33)$$

$$x_u + x_v \geq 1, \quad \forall uv \in E \quad (34)$$

$$\text{minimize } \sum_{v \in V} w(v)x_v \quad (35)$$

Algorithm:

compute opt. sol.  $\bar{x}$  to LP, return  $C := \{v \in V | \bar{x}_v \geq 1/2\}$

Analysis:

At least one node of a edge would be selected which make the solution become a vertex cover with minimum weight.

Prove  $\frac{w(C)}{w(C^*)} \leq 2$ :

let  $z^* := \sum_{x \in V} \bar{x}_v w(v)$ , where  $z^* \leq w(C^*)$ ;

$$w(C) = \sum_{v \in C} w(v) = \sum_{v \in V} w(v)[v \in C] = \sum_{v \in V} w(v) [\bar{x}_v \geq \frac{1}{2}],$$

Here  $X = [\bar{x}_v \geq \frac{1}{2}]$  is 0-1 variable take the corresponding value by the inside condition. X should  $\leq 2\bar{x}_v$ . (considering  $X = 0$  and  $X = 1$  separately)

$$w(C) \leq 2 \sum_{v \in V} w(v)\bar{x}_v = 2z^* \leq 2w(C^*) \implies \frac{w(C)}{w(C^*)} \leq 2$$

### 9.3 Approximation schemes

Polynomial-time approximately scheme (PTAS):

Approximation algorithm that takes instance  $I$  of an optimization problem  $P$  and  $\epsilon > 0$  as input.

For any fixed  $\epsilon$  works as  $(1 + \epsilon)$ -approximation algorithm for  $P$ ;

The running time of a PTAS is required to be polynomial in  $n$  for every fixed  $\epsilon$  but can be different for different  $\epsilon$

Fully polynomial-time approximation scheme (FPTAS):

PTAS with runtime polynomial in  $1/\epsilon$  and the size of  $I$ .

#### 9.3.1 SUBSET-SUM

Input: Set  $S = \{x_1, \dots, x_n\}$  and  $t$

Find:  $U \subset S$  s.t.  $\sum_{x \in U} x \leq t$  with maximum  $\sum_{x \in U} x$

Alg.:

EXACT-SUBSET-SUM  $(S, t)$ :

$L_0 = [0]$

for  $k = 1, \dots, n$

$L_k = \text{MERGE-LISTS}(L_{k-1}, L_{k-1} + x_k)$

remove from  $L_k$  duplicates and elements  $> t$

return  $\text{last}(L_n)$

Cumulative subset sum greedy.

Running time:

Computing  $L_k : O(|L_{k-1}|)$

Total:  $O(\sum_{k=1}^n |L_k|) = O(nt) = O(n2^{\log t})$ ,  $|L_k| \leq t$

$\text{last}(L_n)$  is optimal but is exponential.

To improve: Use Trim

$\text{TRIM}(L = [s_1, \dots, s_m], \delta)$ $L' = [s_1]$ for $i = 2, \dots, m$ if $s_i > \text{last}(L') \cdot (1 + \delta)$ $L' = L' \cup [s_i]$ return $L'$	$\text{APPROX-SUBSET-SUM}(S, t, \varepsilon)$ $L'_0 = [0]$ for $k = 1, \dots, n$ $L'_k = \text{MERGE-LISTS}(L'_{k-1}, L'_{k-1} + x_k)$ $L'_k = \text{TRIM}(L'_k, \varepsilon/2n)$ remove duplicates and elm.s $> t$ return $\text{last}(L'_n)$
---	--

That is: Trim list  $L \rightarrow \{0, 1, \dots, t\}$  with parameter  $\delta > 0$ : if we keep  $s \in L$ , then remove  $(s, (1 + \delta)s]$ .

Thm.: The alg. is an FPTAS

$(1 + \epsilon)$ -approx:

Lemma 1:  $\forall s \in L_k, \exists s' \in L'_k : s' \leq s \leq (1 + \delta)^k s' \implies \frac{s}{s'} \leq (1 + \delta)^k$

Lemma 2: there is  $s' \in L'_n$  such that  $\frac{s_{max}}{s'} \leq (1 + \delta)^n$

Claim:  $(1 + \delta)^n \leq 1 + 2n\delta$ , where  $\delta := \epsilon/2n \implies 1 + 2n\delta \leq 1 + \epsilon$

$\implies$  Approximation ratio:  $\frac{s_{max}}{\text{last}(L'_n)} \leq \frac{s_{max}}{s'} \leq (1 + \delta)^n$

poly. in  $\frac{1}{\epsilon}$  and  $n$ :

$O(\sum_{k=1}^n |L'_k|)$ , need to prove  $|L'_k| = O(\frac{n \log t}{\epsilon})$ ;

Let  $L'_k = [0, s_0, s_1, \dots, s_m]$ , then  $t \geq s_m > (1 + \delta)s_{m-1} > \dots > (1 + \delta)^m s_0 \geq (1 + \delta)^m$

$\implies m < \log_{1+\delta} t = \frac{\ln t}{\ln(1+\delta)} \leq \frac{\ln t}{\delta/(1+\delta)} \leq \frac{4n \ln t}{\epsilon}$

Finish.