Soham Chatterjee
Email: soham.chatterjee@tifr.res.in
Course: Algorithms

Assignment - 5
Dept: STCS
Date: December 12, 2024

Problem 5 (20 marks)

The Subset Sum problem is defined as follows: Given n positive integers $s_1, \ldots, s_n$ and a target value T, does there exist a subset $S \subseteq [n]$ so that the sum $\sum_{i \in [n]} s_i = T$? In class, we showed that this problem is NP-hard.

We now want to show that the problem is hard, *even if we can pick each integer multiple times.* That is, the problem is now defined as: Given n positive integers $s_1, \ldots, s_n$ and a target value T, do there exist nonnegative integers $t_1, \ldots, t_n$ so that $\sum_{i=1}^{n} s_i t_i = T$? Note that some t_is can be zero also, allowing us to not select some of the integers.

Let's call this Subset Sum with Duplicates (or SSD, for short). Show that SSD is NP-complete.

Solution: We will show a reduction from 3SAT to Subset Sum with Duplicates. From a 3SAT formula with n variables and m clauses we will make an instance of Subset Sum with Duplicates with $|S| = 2n + 3m$ where each of the elements of S are $n + 2m$ digit numbers and our target value T will also be a $n + 2m$ digit number. For each number in $k \in S \cup \{T\}$ the first n digits are indexed by variables and then the next m digits the indexed by the clauses and the last m digits are also indexed by the clauses. So we will use k_{x_i} for $i \in [n]$ to denote the digit corresponding to the i^{th} variable of the 3SAT and $k_{C_i^j}$ for $j \in [2]$ and $i \in [m]$ where $k_{C_i^1}$ denotes the the digit corresponding to the i^{th} clause C_i in the first set of digits indexed by the clauses i.e. the digits from $n+1$ to $n+m$ and $k_{C_i^2}$ denotes the the digit corresponding to the i^{th} clause C_i in the second set of digits indexed by the clauses i.e. the digits from $n+m+1$ to $n+2m$.

Now in S we have a number for each literal and we have two numbers for each clause. So we will denote the number for each literal y by n_y and the for each clause C we will denote the two numbers by n_C , n_{C_1} and n_{C_2}. We have $2n$ literals since each variable x gives two literals x and $\neg x$ and from each clause we get 3 numbers. So $|S| = 2n + 3m$. Now we will set how the numbers in $S \cup \{T\}$ will be constructed.

For any literal y in the first n digits n_y has a 1 in the place of the corresponding variable and in rest of the places it have 0's. Then in the next m digits it has a 1 in the corresponding clauses it is in (like we did in the 3SAT to Subset Sum reduction.) and in the last m digits of n_y are 0's.

Now for any clause C in the 3SAT formula for the number n_C and n_{C_1} has a 1 and 2 respectively in the digits corresponding two the clause C in the two set of m digits and the rest of the digits are 0 and for the number n_{C_2} it has a 1 in the digit corresponding clause C in the last m digits and the rest of the digits are 0.

Now the target value T has 1's in the first n digits, then 3's in the next m digits and 2's in the last m digits. Now if the 3SAT instance has a satisfying assignment. Then if any variable x is set 1 then we pick the number n_x and if it is set 0 then we pick the number $n_{\neg x}$. So we add all the numbers we have picked so far. Since all the variables are set to either 1 or 0 the first n digits of the sum of the numbers picked so far matches with the first n digits of T. Now in the next m digits it can be at most 3. If any digit of the sum of numbers already picked up fails to get 3 then if it is 1 behind for any clause C the we pick the number n_C and if it is 2 behind then we pick the number n_{C_1} and otherwise we don't pick anything.

Now the sum of numbers picked up before adding the numbers corresponding to clauses attains at most 3 and at least 1in the next m digits.

- For any digit it attains 3. Let the digit corresponds to the clause C. Then don't pick any of the n_C and n_{C_1}. We pick the number n_{C_2} two times. Then in the last m digits the digit corresponding to C, the sum of already picked numbers attains the target digit corresponding to C in the next and last m digits.
- For any digit it attains 2. Let the digit corresponds to the clause C. We pick the number n_C. Then in the digit corresponding to C in the next m digits attains the target digit 3 corresponding to C. But in the last m digits it is 1 short from the target digit corresponding to C. So we pick the number n_{C_2} once. Then in the last m

digits the digit corresponding to C, the sum of already picked numbers attains the target digit corresponding to C in the next and last m digits.

- For any digit it attains 1. Let the digit corresponds to the clause C. We pick the number n_{C_1}. Then in the digit corresponding to C in the next m digits attains the target digit 3 corresponding to C. In the last m digits also attains the target digit 2 corresponding to C.

Thus we get a solution for the instance of the Subset Sum with Duplicates.

Now suppose there is a solution of Subset Sum with Duplicates instance we created. Since the first n digits of T are 1, for each variable x either we pick n_x or $n_{\neg x}$. Since the target value in the last m digits are 2 we can not pick the number n_C 3 times or pick one n_C and one n_{C_1} since that will make the sum in the last m digits cross the target value. Therefore in the middle m digits from $n+1$ to $n+m$ the numbers n_C, n_{C_1}, n_{C_2} contribute 2 at most for each clause C. Since in the middle m digits the target digits are 3 at least we get one 1 for each clause from the numbers picked from $n_x, n_{\neg x}$ for each variable x. Hence for every clause one variable is picked. Hence if we set the variables 1 for which we pick the corresponding literals and set the variables 0 for which we pick the negation of the variable every clause is satisfied. Therefore this leads to a satisfying assignment of the 3SAT instance.

Hence we get an one-one correspondence between satisfying assignment of the 3SAT formula and the solutions of the Subset Sum with Duplicates instance we created from the formula. Since there are $n+2m+1$ numbers and each number is $2n+3m$ digits long the instance for the Subset Sum with Duplicates can be created in polynomial time. Therefore 3SAT is polynomial time reducible to Subset Sum with Duplicates. Hence Subset Sum with Duplicates is NP-hard.

Now for any Subset Sum with Duplicates instance the non-deterministic turing machine can guess the solution for the problem then add the numbers with multiplying the corresponding multiplicities and check if the target value is added. Therefore Subset Sum with Duplicates is in NP. Hence Subset Sum with Duplicates is NP-complete. ■

[Me and Soumyadeep came up with the solution. I discussed with Aakash]

Problem 6 Exercise 12.4-5, CLRS (20 marks)

Consider Randomized-Quicksort operating on a sequence of n distinct input numbers. Prove that for any constant $k > 0$, all but $O(\frac{1}{n^k})$ of the $n!$ input permutations yield a $O(n \log n)$ running time.

Solution: Let X_n denote the random variable indicating the height of the randomly-built binary tree. Then we know

$$\mathbb{E}[2^{X_n}] \leq \binom{n+3}{3} = O(n^3)$$

Therefore expected height of the randomly-built tree is $O(\log n)$. Then by Markov's Inequality we have

$$\mathbb{P}[X_n \geq \log t] = \mathbb{P}[2^{X_n} \geq t] \leq \frac{\mathbb{E}[2^{X_n}]}{t} = O(n^3 t^{-1})$$

Now taking $t = n^{3+k}$ we have

$$\mathbb{P}[X_n \geq (k+3)\log n] \leq O\left(\frac{1}{n^k}\right)$$

Now for each permutation choosing by choosing pivots we create a binary search tree. Therefore in run of the Randomized Quicksort algorithm we create a random binary search tree of the n distinct input numbers. If a binary tree height is t then it took 2^t comparisons to sort the array. Therefore since there can be at most $O\left(\frac{1}{n^k}\right)$ fraction or $n!$ permutations of the n distinct elements which can have yield a binary search tree with height more than $(k+3)\log n$ i.e. the algorithm sorts all but $O\left(\frac{1}{n^k}\right)$ fraction or $n!$ permutations in $O(n \log n)$ time. ■

Problem 7 (15 marks)

Consider the following scheduling problem: Given n jobs where job j requires p_j units of processing time, and m identical machines, find an assignment of jobs to machines so that the total processing time for any machine is minimized (i.e., the objective is to minimize the maximum processing time over machines). Show that a simple greedy algorithm is a 2-approximation algorithm for this problem.

Solution: We will follow the following greedy algorithm:

Algorithm 1: Greedy Algorithm

Input: Jobs $J = \{j_i : i \in [n]\}$, machines $M = \{m_i : i \in [m]\}$ and i^{th} job takes p_j time

Output: Partition J to $J = \bigcup_{k\in[m]} A_k$ to minimize $\max\left\{\sum_{j\in A_k} p_j \mid k \in [m]\right\}$

```
1 begin
2     A_k ⟵ ∅, T_k ⟵ 0 for all k ∈ [m]
3     for i ∈ [n] do
4         l ⟵ min_{k∈[m]} T_k                    //Minimum attained at k^th machine
5         A_l ⟵ A_l ∪ {j_l}
6         T_l ⟵ T_l + p_l
7     return {A_k | k ∈ [m]}
```

Lemma 1. *The optimal solution of the Makespan problem is at least* $\frac{1}{m}\sum_{k=1}^{n} p_k$.

Proof: Let t_k be the time taken by the k^{th} machine for the optimal solution. And T_k be the time taken by the k^{th} machine for the output of algorithm. Now $\sum_{k=1}^{m} t_k = \sum_{i=1}^{n} p_j$. Hence by Pigeon Hole Principle there exists $k \in [m]$ such that $t_k \geq \frac{1}{m}\sum_{i=1}^{n} p_j$. Hence the optimal solution of the Makespan problem is at least $\frac{1}{m}\sum_{k=1}^{n} p_k$. □

Lemma 2. *The solution by the algorithm is at most* $\frac{1}{m}\sum_{k=1}^{n} p_k + \max\{p_i \mid i \in [n]\}$

Proof: Consider the machine k with maximum T_k. Let the j_i be the last job assigned to machine j. When the algorithm scheduled j_i to the m_k, m_k had the smallest load. Therefore before assigning the j_i to m_k, T_k was the smaller than the average load. Therefore $T_k \leq \frac{1}{m}\sum_{k=1}^{n} p_i \leq \frac{1}{m}\sum_{k=1}^{n} p_k + \max\{p_i \mid i \in [n]\}$. □

Let T be the Makespan of the optimal solution for this problem. Then by Lemma 1 $T \geq \frac{1}{m}\sum_{k=1}^{n} p_k$. Also for any $i \in [n]$, $T \geq p_i$. Therefore

$$\frac{1}{m}\sum_{k=1}^{n} p_k + \max\{p_i \mid i \in [n]\} \leq 2T$$

Hence the algorithm outputs a 2-approximation solution for the makespan problem. ■

[Me and Soumyadeep came up with the solution. I discussed with Aakash]

Problem 8 Exercise 26-6, CLRS: The Hopcroft-Karp Bipartite Matching Algorithm (25 marks)

In this problem, we describe a faster algorithm, due to Hopcroft and Karp, for finding a maximum matching in a bipartite graph. The algorithm runs in $O(\sqrt{V}E)$ time. Given an undirected, bipartite graph $G = (V, E)$, where $V = L \cup R$ and all edges have exactly one endpoint in L, let M be a matching in G. We say that a simple path P in G is an **augmenting path** with respect to M if it starts at an unmatched vertex in L, end at an unmatched vertex in R, and its edges belong alternately to M and $E - M$. (This definition of an augmenting path in a flow network.) In this problem, we treat a path as a sequence of edges, rather than as a sequence of vertices. A shortest augmenting path with respect to a matching M is an augmenting path with a a minimum number of edges.

Given two sets A and B the **symmetric difference** $A \oplus B$ is defined as $(A - B) \cup (B - A)$, that is, the elements that are in exactly one of the two sets.

a. Show that if M is matching and P is an augmenting path with respect to M, then the symmetric difference $M \oplus P$ is a matching $|M \oplus P| = |M| + 1$. Show that if $P_1, P_2, \ldots, P_k$ are vertex-disjoint augmenting paths with respect to M, then the symmetric difference $M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)$ is a matching with carnality $|M| + k$.

The general structure of our algorithm is the following:

Algorithm 2: HOPCROFT-KARP(G)

```
1 M = ∅
2 repeat
3 |   let P = {P_1, ..., P_k} be a maximal set of vertex-disjoint shortest augmenting paths with respect
  |     to M
4 |   M = M ⊕ (P_1 ∪ P_2 ∪ ··· ∪ P_k)
5 until P == ∅
6 return M
```

The remainder of this problem asks you to analyze the number of iterations in the algorithm (that is, the number of iterations in the **repeat** loop) and to describe an implementation of line 3.

b. Given two matchings M and M^* in G, show that every vertex in the graph $G' = (V, M \oplus M^*)$ has degree at most 2. Conclude that G' is a disjoint union of simple paths or cycles. Argue that edges in each such simple path or cycle belong alternately to M or M^*. Prove that if $|M| \leq |M^*|$, the $M \oplus M^*$ contains at least $|M^*| - |M|$ vertex-disjoint augmenting paths with respect to M.

Let l be the length of a shortest augmenting path with respect to a matching M, and let $P_1, P_2, \ldots, P_k$ be a maximal set of vertex-disjoint augmenting paths of l with respect to M. Let $M' = M \oplus (P_1 \cup \cdots \cup P_k)$, and suppose that P is a shortest augmenting path with respect to M'.

c. Show that if P vertex-disjoint from $P_1, P_2, \ldots, P_k$, then P has more than l edges.

d. Now suppose that P is not vertex-disjoint from $P_1, P_2, \ldots, P_k$. Let A be the set of edges $(M \oplus M') \oplus P$. Show that $A = (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P$ and $|A| \geq (k+1)l$. Conclude that P has more than l edges.

e. Prove that if a shortest augmenting path with respect to M has l edges, the size of the maximum matching is at most $|M| + \frac{|V|}{l+1}$.

f. Show that the number of **repeat** loop iterations in the algorithm is at most $2\sqrt{|V|}$. (*Hint:* By how much can M grow after iteration number $\sqrt{|V|}$?)

g. Give an algorithm that runs in $O(E)$ time to find a maximal set of vertex-disjoint shortest augmenting paths $P_2, P_2, \ldots, P_k$ for a given matching M. Conclude that total running time of HOPCROFT-KARP is $O(\sqrt{V}E)$.

Solution:

a. P is an augmenting path. Therefore the even edges of P are matched edges and odd edges of P are unmatched edges. Since P starts and ends with an unmatched edge P is of odd length. Hence the number of unmatched edges in P is one more than the number of the matched edges. Therefore in $M \oplus P$ we remove the matched edges from P and include the unmatched edges of P into $M \oplus P$. This gives a new matching between the vertices of P but we get a larger matching since two new vertices are also matched. Hence $|M \oplus P| = |M| + 1$.

$P_1, \ldots, P_k$ are vertex-disjoint augmenting paths with respect to M. Therefore using the previous result we get that $M \oplus P_1$ is also a matching and $|M \oplus P_1| = |M| + 1$. Since P_2 is vertex-disjoint to P_1, P_2 is a augmenting path with respect to $M \oplus P_1$ too. Hence again using the previous result $(M \oplus P_1) \oplus P_2 = M \oplus (P_1 \oplus P_2)$ is a matching and $|M \oplus (P_1 \oplus P_2)| = |M \oplus P_1| + 1 = |M| + 2$. Continuing like this let $M \oplus \left(\bigoplus_{j=1}^{i} P_j \right)$ is a matching and $\left| M \oplus \left(\bigoplus_{j=1}^{i} P_j \right) \right| = |M| + i$. Since P_{i+1} is vertex-disjoint to $P_1, \ldots, P_i$, P_{i+1} is also a augmenting path with respect to $M \oplus \left(\bigoplus_{j=1}^{i} P_j \right)$. Therefore using the previous result we get that $M \oplus \left(\bigoplus_{j=1}^{i+1} P_j \right)$ is also a matching and $\left| M \oplus \left(\bigoplus_{j=1}^{i+1} P_j \right) \right| = |M| + i + 1$. Therefore in the end we get for $i + 1 = k$ that $\left| M \oplus \left(\bigoplus_{j=1}^{k} P_j \right) \right| = |M| + k$.

b.

- M and M^* are matchings in G. For every vertex $v \in V$ either v is matched in both M and M^* or it is matched in one of the matchings or v is not matched. For all the cases there can be at most 2 edges in $M \oplus M^*$ coming out of v. Hence degree of v can be at most 2. Since v is an arbitrary vertex of V this is true for all the vertices in V. Hence the graph $G' = (V, M \oplus M^*)$ has degree at most 2.
- G' has degree at most 2. Now in a connected graph if every vertex has degree at most 2 then either the graph contains one single vertex or a path or a cycle. Therefore each connected component of G' is either a singleton vertex or a path or a cycle. Each of the singleton vertices can be considered as a 0 length paths. Hence G' is a disjoint union of simple paths or cycles.
- Let C be a cycle in G'. Now let (u, v) and (v, w) are consecutive edges in C. Since the edges of G' are from M and M^*, $(u, v), (v, w) \in M \oplus M^*$. Now both edges can not appear in the same matching. Therefore in C the edges alternate between M and M^*. Since the edges alternate C must have same number of edges from both matching.

  Let P be any path in G'. Now let (u, v) and (v, w) are consecutive edges in P. Since the edges of G' are from M and M^*, $(u, v), (v, w) \in M \oplus M^*$. Now both edges can not appear in the same matching. Therefore in P the edges alternate between M and M^*. Now for P both the starting and ending edge can belong to same matching or they can be from different matching. In first case P has one more edge from a matching than the other one and in the second case P has equal number of edges from both matching. Hence any path can have at most one more edge from a matching than the other.

  Since $|M^*| \geq |M|$. The only connected components which leads to difference between the $|M|$ and $|M^*|$ are paths. Each path which starts and ends with edges from M^* has one more edge from M^* than M and these paths are vertex-disjoint augmenting paths with respect to M. Therefore there are at least $|M^*| - |M|$ vertex-disjoint augmenting paths.

c. l is the length of shortest augmenting path with respect to the matching M and $P_1, \ldots, P_k$ are the maximal set of vertex-disjoint augmenting paths of length l with respect to M. P is augmenting path with respect to M'. Since P is vertex-disjoint from $P_1, \ldots, P_k$, P is also a augmenting path with respect to M. If P has at most l edges then $P_1, \ldots, P_k, P$ becomes a larger set of vertex-disjoint augmenting paths of length l with respect to M which contradicts that $P_1, \ldots, P_k$ is maximal set. Hence P has more than l edges.

d. For any two sets A and B, any element in $A \oplus B$ is either in $A \setminus B$ or $B \setminus A$. Therefore any element in $(A \oplus B) \oplus B$ is either in $(A \oplus B) \setminus B = A \setminus B$ or $B \setminus (A \oplus B) = B \setminus (B \setminus A) = A \cap B$. Hence $(A \oplus B) \oplus B = A$. In our case therefore we have

$$M \oplus M' = M \oplus (M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)) = P_1 \cup P_2 \cup \cdots \cup P_k \implies (M \oplus M') \oplus P = (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P$$

Now P is not vertex-disjoint from $P_1, \ldots, P_k$. We will show that P is edge disjoint from P_i for all $i \in [k]$. Assume the contrary. Let there is an edge $e \in P$ such that $e \in P_i$. The edges before and after e are from $E \setminus M'$ hence they are from $E \setminus M$. Since P is also an augmenting path with respect to M, we have $e \in M$. But then e can not be in P_i. Hence contradiction. Therefore P is edge disjoint from $P_1 \ldots, P_k$. Since P is the shortest augmenting path with respect to M' and the length of shortest augmenting path with respect to M we have $|P| \geq l$. Therefore $|A| \geq (k+1)l$.

e. Let M^* is a matching with more than $|M| + \frac{|V|}{l+1}$ edges. Hence by part (b) there are more than $\frac{|V|}{l+1}$ vertex-disjoint augmenting paths with respect to M. Since the shortest augmenting path with respect to M has l each of the vertex-disjoint augmenting paths with respect to M has at least l edges. Therefore they are incident on at least $l+1$ vertices. The paths are vertex disjoint. Hence in total the number of vertices in all those vertex-disjoint augmenting paths with respect to M has more than $\frac{|V|}{l+1}(l+1) = |V|$ vertices. But there can not be more vertices than V. Hence contradiction. Therefore the size of the maximum matching is at most $|M| + \frac{|V|}{l+1}$.

f. Let M^* be a maximal matching. If $|M^*| \geq |M|$ then by part (b) there are at least $|M^*| - |M|$ vertex-disjoint augmenting paths in $M \oplus M^*$ with respect to M. Now after iteration $\sqrt{|V|}$ let the resulting matching is called M. Now all the vertex-disjoint augmenting paths with respect M has length at least $\sqrt{|V|}$. Therefore there can be at most $\sqrt{|V|}$ such vertex-disjoint augmenting paths. Therefore $|M^*| - |M| \leq \sqrt{|V|}$. In each loop we pick the maximal set of shortest vertex-disjoint augmenting paths with respect to M. Therefore at most $\sqrt{|V|}$ can happen. Therefore in total at most $2\sqrt{|V|}$ iterations can occur.

g. We will use a modified version of the algorithm for finding augmenting paths. For any vertex the algorithm returns the shortest augmenting path starting with that vertex. Here we will start from all the vertices L which are unmatched and at each step we will increase the length of the alternating tree with respect to the matching M and we stop at the first encounter of the augmenting path and then we check for which unmatched vertices we obtained an augmenting path. Then we only return those who are vertex disjoint. This successfully returns the maximal set of shortest augmenting paths with respect to the matching.

Now this algorithm behaves like a Breadth First Search from the unmatched vertices. Hence the algorithm takes at most $O(|E| + |V|) = O(|E|)$ time. Therefore in $O(|E|)$ time we can find a maximal set of vertex-disjoint shortest augmenting paths for a given matching M. Therefore at each iteration of the algorithm it takes $O(E)$ time. There are $O(\sqrt{V})$ iterations. Hence total running time of the Hopcroft-Karp algorithm is $O(\sqrt{V}E)$

■

Problem 9 (20 marks)

Given an undirected graph $G = (V, E)$ with positive weight w_e on each edge, we want to find a cut $(X, V \setminus X)$ to maximize the weight of edges across the cut. We denote by $w(X) := \sum_{e \in \delta(X)} w(e)$ the weight of edges across the cut. The problem is known to be NP-hard. Our objective in this problem is to come up with a deterministic 2-approximation for this.

(a) (10 points) Give a randomized algorithm that in expectation is 2-approximate (i.e., if $(X, V \setminus X)$ is the cut obtained by the algorithm, and $(X^*, V \setminus X^*)$ is the optimal cut, then $\frac{w(X^*)}{\mathbb{E}[w(X)]} \leq 2$.

(b) (10 points) Use the method of conditional expectations to derandomize this algorithm, to obtain a deterministic 2-approximate algorithm.

Solution:

- We have the following algorithm:

Algorithm 3: Randomized Algorithm

Input: $G = (V, E)$ and $w(e)$ for all $e \in E$.
Output: Find max weight cut $(X, V \setminus X)$.

```
begin
    X ⟵ ∅
    for v ∈ V do
        u ⟵ generate a uniformly random coin from {0, 1}
        if u == 1 then
            X ⟵ X ∪ {v}
    return (X, V \ X)
```

Let I_e be the indicator random variable for $e \in E$ such that $I_e = 1$ if e is a cut edge between X and $V \setminus X$ and otherwise $I_e = 0$. Let $e = (u, v)$ then $\mathbb{P}[I_e = 1] = \frac{1}{2}$. Hence

$$\mathbb{E}[w(X)] = \sum_{e \in E} w(e)\mathbb{P}[I_e = 1] = \frac{1}{2}\sum_{e \in E} w(e) \geq \frac{1}{2}w(X^*)$$

Hence this algorithm gives a cut $(X, V \setminus X)$ such that $\frac{w(X^*)}{\mathbb{E}[w(X)]} \leq 2$.

- Let S is the variable denote the set of cut edges by picking each vertex with probability $\frac{1}{2}$. Let R_i denote the random variable for i^{th} vertex. Also denote

$$C(r_1, \ldots, r_i) = \mathbb{E}[w(S) \mid R_1 = r_1, \ldots, R_i = r_i]$$

where $r_j \in \{0, 1\}$ for all $j \in [n]$. And $C(\emptyset) = \mathbb{E}[w(S)] = \frac{1}{2}\sum\limits_{e \in E} w(e)$. Let us consider the state of the conditional expectation after i choice is made. Let $S_i = \{v_j \mid j \leq i, R_j = 1\}$ and $T_i = \{v_j \mid j \in i, R_j = 0\}$ and $U_{i+1} = \{vj \mid j > i\}$. Let $w(S, T) = \sum\limits_{e \in E \cap S \times T} w(e)$. Then we have

$$C(r_1, \ldots, r_i) = w(S_i, T_i) + \frac{1}{2}\sum_{e \in E \cap U_i \times (V \setminus U_i)} w(e)$$

Suppose we have calculated already $C(r_1, \ldots, r_i)$. Then we can calculate $C(r_1, \ldots, r_{i+1})$ by calculating $w(S_{i+1}, T_{i+1})$ and $\sum\limits_{e \in E \cap U_i \times (V \setminus U_{i+1})} w(e)$. So we calculate $C(r_1, \ldots, r_i, 0)$ and $C(r_1, \ldots, r_i, 1)$ and we pick the larger of the two possibilities. So after calculating the conditional expectations we can derandomize the algorithm.

■