DESIGN AND ANALYSIS OF ALGORITHMS
Homework 4

Aman Choudhary
MTech Coursework, CSA 2020
Sr No: 17920

November 5, 2020

1 Rank Functions

1.1 Greedy Algorithm for MATROID OPTIMIZATION (Discussed in class)

$//M = (E, I)$ is a matroid.
$//S \in I$ is the optimal independent set produced by the algorithm.

$OPT(M)$

1. Order and index the n elements in E such that $w(1) \geq w(2) \geq ... \geq w(n) \geq 0$
2. Initialize $S = \phi$
3. for $i = 1$ to n
4. if $S \cup i \in I$, the $S = S \cup i$
5. return S

1.2 Proof

When we order the elements of E, such that $w(1) \geq w(2) \geq ... \geq w(n)$, we write $E_i = \{1, 2, ..., i\}$, for $1 \leq i \leq n$, along with $E_0 = \phi$. Let us denote by S_i the set of elements that have been added to the output set after i iterations. It is clear that at the end of iteration i, the algorithm has examined all the elements in E_i and has produced $S_i \subseteq E_i$.

We argue that at the end of every iteration i, $r(E_i) = |S_i|$. Now, rank of a set X captures the size of the largest (cardinality wise) independent subset within X. When we say that $r(E_i) = |S_i|$, we want to convey that S_i is the largest independent subset of E_i. We prove this claim as follows:

Let us now assume that after i iterations, $|S_i| < r(E_i)$ and there exists a larger independent subset T_i such that $r(E_i) = |T_i|$. Since, $|S_i| < |T_i|$, we can say that there exists $e \in T_i - S_i$ such that $S_i \cup e \in I$ **(using Augmentation Property of Matroids)**.

Let us assume that the index of e in the decreasing order of elements is k, that is weight of e is $w(k)$. Now, certainly $k \leq i$. Consider iteration k, when e was examined for inclusion into S_k. Since, $S_i \cup e \in I$, and $S_{k-1} \cup e \subseteq S_i \cup e$, we can say $S_{k-1} \cup e \in I$ **(using Hereditary Property of Matroids)**. Thus, during the kth iteration, we would have included e into S_k and $S_k = S_{k-1} \cup e$. Also, such an e would always be in S_i after i iterations because $S_k \subseteq S_i$. Therefore, we reach a **contradiction** to the fact that claimed existence of e, such that $e \notin S_i$ abd $e \in T_i - S_i$. Hence, $|T_i| = |S_i|$ and $r(E_i) = |S_i|$.

Let us consider the beginning of iteration i. At this point, the condition $r(E_{i-1}) = |S_{i-1}|$ holds true. Now at step 4, we check whether, $S_{i-1} \cup i \in I$. There are two cases:

(i) $S_{i-1} \cup i \in I$. In this case, $S_i = S_{i-1} \cup i$. Hence, we can say that $r(E_i) = |S_i| = |S_{i-1}| + 1 = r(E_{i-1}) + 1$. That is, $r(E_i) - r(E_{i-1}) = 1$. So, the cost of output set increases by $w(i) * 1 = w(i) * \{r(E_i) - r(E_{i-1}\}$.

(ii) $S_{i-1} \cup i \notin I$. In this case, $S_i = S_{i-1}$. Hence, we can say that $r(E_i) = |S_i| = |S_{i-1}| = r(E_{i-1})$. That is, $r(E_i) - r(E_{i-1}) = 0$. So, the cost of output set remains same, or increases by $0 = w(i) * 0 = w(i) * \{r(E_i) - r(E_{i-1}\}$.

The observation from (i) and (ii) is that the cost of output set increases by $w(i) * \{r(E_i) - r(E_{i-1})\}$ during iteration i. So, if we start with an empty output set S_0 with cost 0, the result after n iterations is the output set S_n, whose cost is, $\sum_{i=1}^{n} w(i) * \{r(E_i) - r(E_{i-1})\}$.

Moreover, the output of above algorithm is optimal (Proved in lecture notes). Hence, the weight of an optimal (maximum-weight) independent set, say OPT, can be expressed as,

$$OPT = \sum_{i=1}^{n} w(i) * \{r(E_i) - r(E_{i-1})\}$$

2 Reverse-Delete Algorithm for Matroid Optimization

2.1 Algorithm 2a

The above algorithm does not produce an optimal matroid and hence is incorrect. The proof is as follows:

The output of the **Greedy Algorithm for MATROID OPTIMIZATION** is necessarily a base (Lemma 2, proved in lecture notes). Hence, the output of any algorithm for Matroid Optimization must fulfill two conditions:

(i) It must be a base.

(ii) The total weight of the base must be maximum.

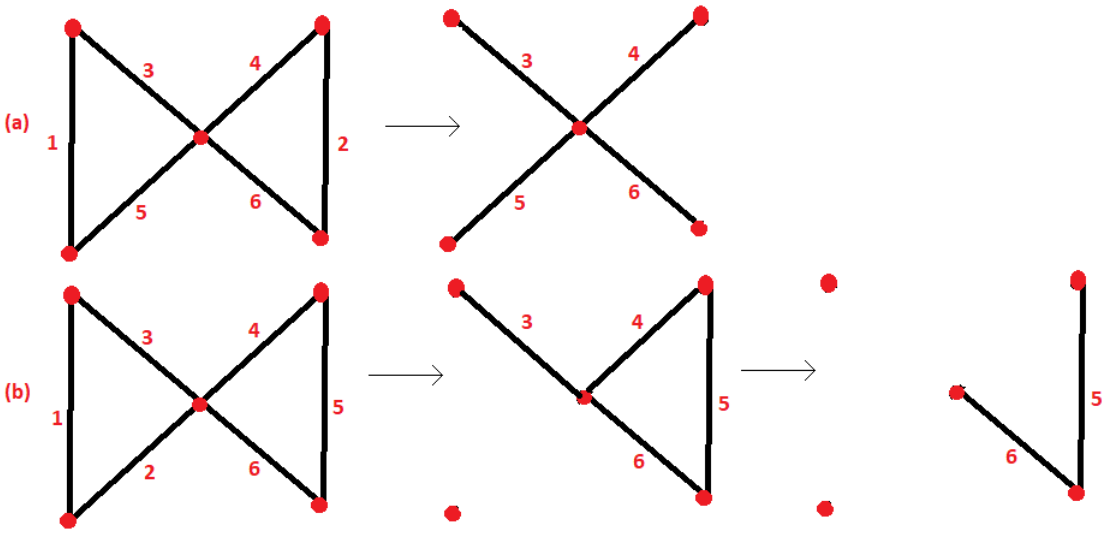
Let us assume that the size of base for our given matroid M is b. Also, the size of set E is n. Hence, any algorithm for Matroid Optimization must exclude $n - b$ elements from the output set.

The Reverse-Delete algorithm keeps deleting elements from S (initialized to E) until it becomes independent. Now, after $n - b$ iterations, the set S has b remaining elements. There are two cases:

(i) The set S is independent. Then, S is a base (which may or may not be optimal).

(ii) The set S is not-independent. Hence, we must go on and remove more elements from S. But, in doing so, the output S ceases to be a base, and hence it cannot be an optimal solution.

From case (ii), we conclude that the algorithm is incorrect. Let us also consider a specific counter example for graphic matroid optimization. We know that the output for the graphic matroid optimization is a Maximum Spanning Tree (Proved in lecture notes). In the figure, we show a graph with 5 vertices, hence the MST will contain 4 edges. In part (a), we show an example where reverse-delete algorithm produces correct results because just after removing 1 and 2 we get an MST. In part (b), we have an example where our algorithm fails, because after removing 1, 2 the graph is still cyclic (not-independent).



2.2 Algorithm to compute RANK OF MATROID

$//M = (E, I)$ is a matroid.
$//S \in I$ is the largest independent set produced by the algorithm.

$RANK(A)$

1. Initialize $S = \phi$
2. for each $e \in A$
3. if $S \cup e \in I$, the $S = S \cup e$
4. return $|S|$

This algorithm is same as the one above (Greedy Algorithm for MATROID OPTIMIZATION). The only difference is that we have eliminated the sorting step, as we are not concerned about the cost of the output. We just want that the output set after n iteration be a base of A. Then we just return the cardinality of the output. We prove that the output of above algorithm is a base:

Let us now assume that the output $S \subseteq A$ is not a base, and T is an actual base of A. Since, $|S| < |T|$, we can say that there exists $e \in T - S$ such that $S \cup e \in I$ **(using Augmentation Property of Matroids)**.

Consider iteration k, when e was examined for inclusion into an intermediate set $S' \subseteq S$. Since, $S \cup e \in I$, and $S' \cup e \subseteq S \cup e$, we can say $S' \cup e \in I$ **(using Hereditary Property of Matroids)**. Thus, during the kth iteration, we would have included e into our output. Therefore, we reach a **contradiction** to the fact that claimed existence of e, such that $e \notin S$ and $e \in T - S$. Hence, S is a base.

The algorithm runs for $|A| = n$ iterations, and in each iteration does a query to the oracle which checks for independence for a set. We assume that the oracle access take polynomial time $O(n^k)$. Hence, the overall time complexity of the algorithm is $n * O(n^k)$, which is again **polynomial time**.

2.3 Algorithm 2c

First, we prove that the output of the algorithm is a **base**.

Loop Invariant: At the start of every iteration i, the following condition must be satisfied:

$$r(S) = r(E) = b \qquad ...(1)$$

$r(E)$ captures the size of largest (cardinality wise) independent set of E, which is nothing but the base of E.

Initialization: At the start of the first loop the loop invariant states that condition (1) must be true. In step 2 of algorithm, we initialize $S = E$. Hence, this condition holds true, trivially.

Maintenance: At the start of iteration i, according to the invariant $r(S) = r(E) = b$ is satisfied. Now, we check whether $r(S - i) == r(E)$. Here, we have two cases:

1. $r(S - i) = r(E)$: In this case, we update $S = S - i$. In doing so, we maintain the loop invariant because $r(S) = r(S - i) = r(E)$.
2. $r(S - i) < r(E)$: In this case, we do nothing. Again we maintain the loop invariant because S is unchanged and $r(S) = r(E)$.

Thus, at the beginning of iteration $i + 1$, the condition (1) is satisfied.

Termination: Before the starting iteration $n + 1$, the loop invariant holds true. Moreover the loop terminates when $i = n + 1$. The output set S produced by algorithm has rank equal to $r(E)$. Hence, S is necessarily a base.

Now, we prove that the output of the algorithm is **optimal**.

Let us assume that the output of our algorithm is $S = \{s_1, s_2, ..., s_b\}$. We want to show that this is an optimal independent set. Let us assume that this is not the case, and there exists another set $T = \{t_1, t_2, ..., t_b\}$ which has greater weight than S. Here, both the sets have been written in **decreasing order** of weights, i.e. $w(s_1) \geq w(s_2) \geq ... \geq w(s_b)$ and $w(t_1) \geq w(t_2) \geq ... \geq w(t_b)$.

Now, let p be the smallest index such that $w(t_p) > w(s_p)$. We consider A and B such that,

$$A = \{t_1, t_2, ..., t_{p-1}, t_p\}$$
$$B = \{s_1, s_2, ..., s_{p-1}\}$$

Here, $|B| < |A|$ and both are independent, hence there exists $e \in A - B$ such that $B \cup e \in I$. (using Augmentation Property of Matroids). Now, lets $C = B \cup e$.

$$C = \{s_1, s_2, ..., s_{p-1}\} \cup e$$
$$S - C = \{s_p, s_{p+1}, ..., s_b\}$$

Here, $|C| = p < |S| = b$ and both are independent. Note that $|S - C| = b - p + 1$. Now, we are going to augment C, with $b - p$ elements from $S - C$ to achieve a set S' such that $|S'| = b$. So, essentially S' will have all elements from S, expect for one s_j $(p \leq j \leq b)$ which is **displaced** by e.

$$S' = \{s_1, s_2, ..., s_{p-1}, s_p, s_{p+1}, ..., s_b\} - \{s_j\} \cup e$$

Now, $e \in A$ and A was chosen in a way, such that all elements in A had weights strictly greater than s_p. Thus, for all $j, w(e) > w(s_j)$. As have displaced exactly one element s_j $(p \leq j \leq b)$ from S during construction of S' and S, we thus claim $w(S') > w(S)$. Now, we have S' which is an independent set of b elements (as we constructed it via augmentation), and has $w(S') > w(S)$. Had there existed such an S', our algorithm would have produced S' instead of S, which **contradicts** that S was produced as result. Hence, our initial assumption that S is an optimal set holds true.
