IOE 691: Approximation Algorithms — Date: 01/18/2017

Lecture Notes: Set Cover (Greedy) and Matching (Local Search)

Instructor: Viswanath Nagarajan — Scribe: Qi Luo

1 Set Cover Problem

Definition 1.1 *Given a set of elements $\{e_1, \dots e_n\}$ (called the universe) and a collection $\{S_i\}_{i=1}^m$ of m sets whose union equals the universe, the set cover problem is to identify the smallest sub-collection of sets whose union equals the universe. If each set is assigned a cost, it becomes a weighted set cover problem.*

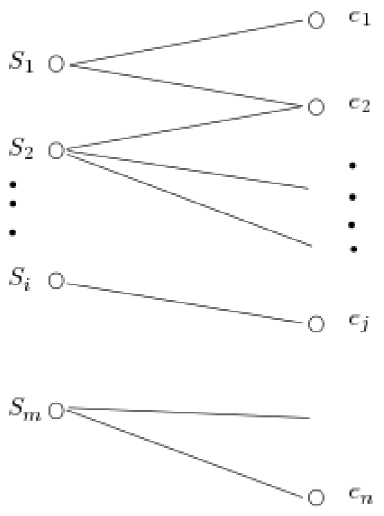

Figure 1: Diagram of a Set Cover problem.

Notice that there is a close relationship between Set Cover and Maximum Coverage. The setting of two problems is the same, but the goals are different: in Maximum Coverage, the total number of sets is given and the goal is to cover as many elements as possible; in Set Cover the goal is to cover all elements with minimum number of sets.

A possible greedy algorithm for Set Cover problem is:

Algorithm 1: Greedy Algorithm for Set Cover Problem

```
Data: A universe $\{e_1, \dots e_n\}$, a family $S = \{S_1, \dots S_m\}$.
/* $U$ is a set of uncovered elements.                                  */
$U = \{e_1, \dots e_n\}$;
while $U \neq \emptyset$, iteration $i$ = 1, 2, ... $l$ do
    pick $A_i = \arg\max_{j=1,\dots m} |S_j \cap U|$
    $U \leftarrow U \backslash A_i$
```

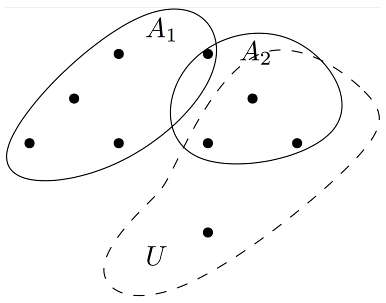

Figure 2: Diagram of first two steps of greedy algorithm for Set Cover problem.

We let l denote the number of iterations taken by the greedy algorithm.

It is clear that the first k iterations of the greedy algorithm for Set Cover are identical to that of Maximum Coverage (with bound k). In the Maximum Coverage problem, we already proved the following lemma 1.1. (Although we only had k iterations in max-coverage, the same analysis works for any number l of iterations.)

Lemma 1.1 *If C_i denotes the set of covered elements at the end of iteration i and C^* denotes the maximum coverage using k sets, then*

$$|C_i| \;\geq\; \frac{C^*}{k}\sum_{j=0}^{i-1}\left(1-\frac{1}{k}\right)^j, \qquad \forall i = 1, \dots \ell$$

Theorem 1.1 *The greedy algorithm is $(1+\ln(n))$-approximation for Set Cover problem.*

Proof: Suppose $k = OPT($ set cover $)$. Since set cover involves covering all elements, we know that the max-coverage with k sets is $C^* = n$. Our goal is to find the approximation ratio α so that $ALG($ set cover $) = \ell \leq \alpha k$. We apply Lemma 1.1 at the second last iteration, i.e. $i = \ell - 1$.

$$\begin{cases} |C_{\ell-1}| \leq n-1 \\ |C_{\ell-1}| \geq \frac{n}{k}\sum_{j=0}^{l-2}(1-\frac{1}{k})^j = \frac{n}{k}\dfrac{1-(1-\frac{1}{k})^{\ell-1}}{\frac{1}{k}} = n(1-(1-\frac{1}{k})^{\ell-1}) \geq n(1-e^{-\frac{\ell-1}{k}}) \end{cases} \tag{1}$$

The first inequality is because the uncovered set must contain at least one element, otherwise the algorithm would have stopped before. The second inequality is from Lemma 1.1 and the fact that $1 + x \leq e^x$ for any $x \in (-\infty, \infty)$.

From inequalities in (1), we have $ne^{-\frac{\ell-1}{k}} \geq 1$. We can take logarithm on both sides and find the approximation ratio $\alpha \leq 1 + \ln n$ as claimed. ■

While $\alpha = 1 + \ln(n)$ is not a constant factor, it is still a reasonably good approximation ratio because it grows slowly with the input size n (refer to figure 3). Actually, one cannot get *any* better approximation algorithm for the Set Cover problem unless $P = NP$[1].

[1]Moshkovitz, Dana. *The projection games conjecture and the NP-hardness of ln n-approximating set-cover.* In Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pp. 276-287. Springer Berlin Heidelberg, 2012.

A useful property of this greedy algorithm is that its running time only depends on the size of universe n. There are many applications where the number of sets $m \leq 2^n$ is very large, and this algorithm can still be used. In such cases the brute force search for the "best" set to pick in each iteration is inefficient, and one needs an algorithm to find the "best" set using the problem structure.

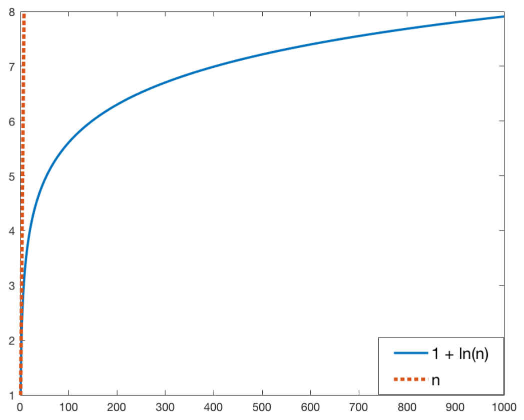

Figure 3: Plot of $y = 1 + log(n)$ solid line, $y = n$ dotted line, $n \in [1, 1000]$.

According to the form of approximation ratio α, we can categorize approximation algorithms within the region of NP-hard problems as in Figure 4. From (a) to (e) the algorithms become more and more loose in obtaining near-optimal solutions. These ratios in (c) to (e) are only examples and other functions of input size n are also possible for the approximation ratio depending on the setting of the problem.

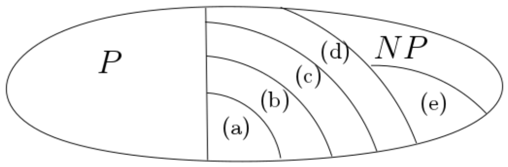

Figure 4: Taxonomy of NP problems (minimization) according to the form of α.

(a) $\alpha = 1 + \epsilon$ with running time of $n^{1/\epsilon}, \epsilon > 0$. e.g. PTAS for the Knapsack problem.

(b) α is constant factor, e.g. k-Center Problem, Maximum Coverage, TSP.

(c) $\alpha = 1 + \log(n)$. e.g. Set Cover problem.

(d) $\alpha = 1 + \log^2(n)$.

(e) α is linear in n.

2 Local Search: Maximum Matching

A second category of approximation algorithms that we are interested is Local Search (LS) algorithms. In a greedy algorithm, we are only allowed to do one type of operation (addition or deletion) at each iteration. In LS algorithm, we are able to do both addition and deletion at each iteration. LS algorithm moves from solution to solution in the space of candidate solutions (the search space) by applying local changes, until a solution deemed locally optimal is found.

The maximum matching problem is the following.

Definition 2.1 *Given a graph $G = (V, E)$. a matching is a subset $K \subseteq E$ where every vertex has degree no greater than 1 in K. The goal of the maximum matching problem is to find a matching K with maximum $|K|$.*

For example, in figure 5, the maximum matching value is 4.

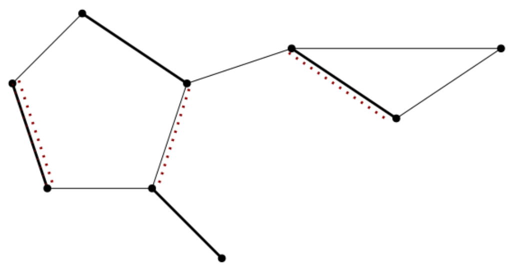

Figure 5: Thick lines are the optimal matching, and red-dotted lines are a feasible matching.

Here we discuss a local search approximation algorithm for maximum matching. (Note that there is even an efficient exact algorithm for matching- using different techniques.)

The LS algorithm is parameterized by a constant $t \geq 1$ which determines the running time and the approximation ratio. Roughly speaking, t is the number of local changes allowed to a solution.

Algorithm 2: Local Search Algorithm for Matching Problem

```
Data: Graph $G = (V, E)$, with $|V| = n, |E| = m$.
Start with any matching $M \neq \emptyset$
while there are edges $R$ to remove and $A$ to add such that:
/* local move                                                    */
$A \subseteq E$, $R \subseteq M$, $|R| < |A| \leq t$ and $(M \backslash R) \cup A$ is a matching
do
    $M \leftarrow (M \backslash R) \cup A$
```

Lemma 2.1 *The running time for the LS algorithm for matching is at most $n^{O(t)}$.*

Proof: Note that the total number of potential local moves is at most $(m^t \times n^t)$; the first term denotes the choices for A and the second term denotes the choices for R. Moreover, the total number of iterations is at most $\frac{n}{2}$: this is because $|M|$ increases by at least 1 in each iteration. So the runtime is at most $O(n^{t+1}m^t)$. Since $m = |E| \leq \binom{n}{2}$, the lemma follows. ■

Theorem 2.1 *The LS algorithm is a $(1 - \frac{1}{t})$-approximation algorithm for matching.*

Proof: Let N denote the optimal matching for G and M the locally optimal solution output by the algorithm. We consider $M \bigoplus N$ where the exclusive operator $\bigoplus$ denotes the union of edges in M but not N, and edges in N but not M. See Figure 6 for an example.

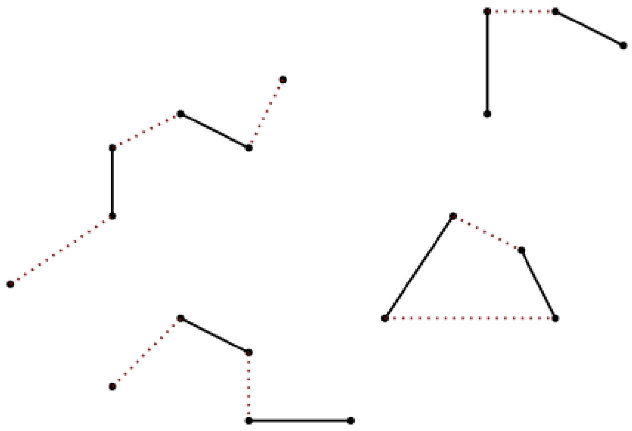

Figure 6: Graph $M \bigoplus N$, where solid lines are M and red-dotted lines are N.

Note that each vertex in $M \bigoplus N$ has degree at most two. So each connected component $M_i \cup N_i$ in $M \bigoplus N$ is of one of the following types:

1. Cycle: then there must be an even number edges and $|M_i| = |N_i|$, otherwise it is contradictory to the definition of matching.
2. Path of even number edges: then $|M_i| = |N_i|$, because edges in either set are disjoint.
3. Path of odd number edges and $|M_i| > |N_i|$: then $|M_i| = |N_i| + 1$ and the approximation output is better than optimal in this subset.
4. Path of odd number edges and $|M_i| < |N_i|$: then $|M_i| = |N_i| - 1$. We observe that we must have $|N_i| \geq t$. Suppose that this is not the case: then we can do a swap $R = M_i, A = N_i$ so that $(M \setminus R) \cup A$ is a valid matching (of larger size), which is a contradiction to the local optimality of M. Now,

$$\begin{cases} |M_i| \geq t \\ |N_i| \geq t \end{cases} \Rightarrow |M_i| = |N_i| - 1 \geq |N_i| - \frac{|N_i|}{t} = (1 - \frac{1}{t})|N_i|$$

Adding over all components $M_i \cup N_i$ we can derive that $|M| \geq (1 - \frac{1}{t})|N|$. ■

Note that the running time also depends on t: so there is a trade-off in choosing t for the runtime and approximation ratio.