CS 630 – Fall 2024 – Lab 4
Oct 2, 2024

1. Approximation function

Suppose you are tasked with solving a graph problem, and two *state-of-the-art* approximation algorithms are available. One has an approximation ratio of $\ln n$, and the other provides a constant $(D+1)$ approximation, where D is the maximum degree of the graph. Can you provide examples of graph families where one algorithm outperforms the other, or where they are asymptotically equivalent? Your answer should focus on asymptotic behavior.

Solution:

1. Any graph type where $O(D) > O(\ln n)$ is a good example for using the $O(\ln n)$-approximation algorithm. For example graphs where there are degrees of size $O(n)$ or size $O(\sqrt{n})$ (this latter is quite common) or $O((\ln n)^k)$, etc. Any graph where $O(D) < O(\ln n)$ would work better with the $(D+1)$-approximation. For example constant degree graphs
2. Here are some notable graphs that you may have heard of. The complete graph $K(n)$ (Kneser graph) has $D = n$, so a $\ln n$ approximation algorithm is preferred in such a graph. In the hypercube graph each vertex has degree $\mathcal{O}(\log n)$ so in that case any of the two approximation algorithms has similar approximation ratios asymptotically.

2. Submodular functions

Let f be a set function on the universe Ω (thus, it takes as input subsets of Ω). Recall the definition of monotone increasing submodular functions.

1. f is monotone increasing, thus for sets $X \subseteq Y \subseteq \Omega$ we have $f(X) \leq f(Y)$.
2. For every $X \subseteq Y \subseteq \Omega$ and every $z \in \Omega \setminus Y$, we have $f(X \cup \{z\}) - f(X) \geq f(Y \cup \{z\}) - f(Y)$. *Intuitively, the benefit of adding z to X results in more gain as adding it to Y.*
3. Equivalent definition to 2.: For every $S, T \subseteq \Omega$ we have $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$

Now apply the properties to the following functions, and decide whether they are monotone submodular functions. Either prove it or give a counter example.

In the following problems the universe $\Omega = \{\omega_1, \omega_2 \ldots \omega_n$ is a set of items. $\Delta = \mathbf{Z}_+$ is the set of positive integers.

Solution:
In general the best approach for such proofs is to try to apply the definition.

1. $X \subseteq \Omega$ and

$$f(X) = 20$$

Solution:
This is submodular since the difference between the value of any two sets is 0.

2. $X \subseteq \Omega$ and

$$f(X) = |X|$$

Solution:
This is submodular. $f(X \cup \{z\}) - f(X) = |X + 1| - |X| = 1$. Same applies to Y.

3. $X \subseteq \Omega$ and

$$f(X) = c \cdot |X|$$

Solution:
This is submodular. $f(X \cup \{z\}) - f(X) = c \cdot |X + 1| - c \cdot |X| = c$. Same applies to Y.

4. $X \subseteq \Delta$ and

$$f(X) = \min(X)$$

Solution:
This is not submodular. Pick $X = \{1\}$, $Y = \{-2, 1\}$, and $x = -1$. Then

$$\begin{aligned} f(X \cup \{x\}) - f(X) &= -1 - 0 = -1 \\ f(Y \cup \{x\}) - f(Y) &= -2 - -2 = 0 \end{aligned}$$

5. $X \subseteq \Delta$ and

$$f(X) = \max(X)$$

Solution:
This is submodular. Since $X \subseteq Y$ we know $\max(X) \leq \max(Y)$. Further we know $\max(X \cup \{z\}) = \max\{\max(X), z\}$. Same applies to Y. Thus if $z > \max Y$ then $f(X \cup \{z\}) - f(X) = f(Y \cup \{z\}) - f(Y)$. Otherwise the right hand side is $= 0$ and hence always

$left \leq 0$. Thus the $\geq$ condition in the definition is met either way.

6. $X \subseteq \Delta$ and
$$f(X) = \begin{cases} 1 & \text{if } X \text{ has odd number of elements} \\ 0 & \text{otherwise} \end{cases}$$

Solution:
This is not submodular. Pick $X = \{1\}$, $Y = \{1, 2\}$, and $x = 3$. Then
$$\begin{aligned} f(X \cup \{x\}) - f(X) &= 0 - 1 = -1 \\ f(Y \cup \{x\}) - f(Y) &= 1 - 0 = 1 \end{aligned}$$

7. $X \subseteq \Omega$, w is a specific element, and $f(X) = \begin{cases} 1 & if\ w \in X \\ 0 & otherwise \end{cases}$

Solution:
This is submodular. For any set X we have $F(X \cup \{w\}) - F(X) = 0$ or 1 depending whether $w \in X$ initially. Now we can compute the two sides of $f(X \cup \{z\}) - f(X) > f(Y \cup \{z\}) - f(Y)$ for any $X \subseteq Y$ and see that the values satisfy the inequality.

3. Job Scheduling

Suppose you are tasked with solving a scheduling problem involving two processors. The model is non-preemptive, meaning that once a job starts processing on a machine, it cannot be paused and resumed later; the job must complete without interruption. An input might look like a sequence of job sizes $s_1, \ldots, s_n$ (all s_i are positive), and you want to minimize the "makespan", which is the time when the last job finishes its execution. This problem is NP-complete, so we do not know a sub-exponential time algorithm for this problem.

1. What is the worst possible makespan? (For this question, you should consider pathologically bad choices.)

Solution:
The worst possible makespan comes from assigning all jobs to one processor and leaving the other processor completely idle. This makes the makespan $\sum_{i=1,\ldots,n} s_i$.

2. What lower bounds can you put on the best possible makespan? To what extent do these depend on the number of processors?

Solution:
Two trivial bounds that apply are $\max_i(s_i)$ and $\frac{1}{2}\sum_{i=1,\ldots,n} s_i$. The former bound comes from the largest job being a lower bound of the makespan, since at least one processor must schedule it. The latter bound comes from spreading out jobs perfectly evenly across the processors; if each processor had less than this bound, their total workloads would be less than the total job size of the problem. This latter bound generalizes to $\frac{1}{m}\sum_{i=1,\ldots,n} s_i$ for m processors.

3. What bounds on approximation factors come from these lower bounds?

Solution:
The former bound gives a upper bound of $\frac{\sum_{i=1,\ldots,n} s_i}{\max_i s_i}$ which is probably loose unless the largest job is a significant fraction of the total. The latter bound gives an upper bound of m on approximation factors for m processors, or two for our specific problem. Note that if you know that the former bound is better than the latter, you can further improve these bounds by thinking about how to place the biggest jobs.

4. A simple greedy algorithm assigns the jobs in order to the processor with the least current workload (total assigned job size) so far. This algorithm is known to have a tight approximation ratio of 3/2 for two processors. A colleague suggests that you can get better performance by sorting the jobs in increasing order of their processing times before assigning jobs to processors.

   Can you provide an instance where this algorithm still has an approximation ratio of 3/2?

Solution:
The instance where the n jobs have length one and the $n+1$-th has a length of n . The greedy algorithm will first schedule the first n jobs, by assigning $n/2$ to each machine, and the last one to any machine. This scheduling has an objective of $n + n/2 = 3/2n$ wile the optimal one is to assign the large one to one machine and the rest to the other. This forms a makespan of n.