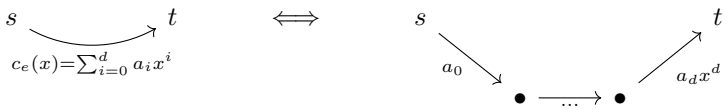
**CS 6840 Algorithmic Game Theory** 10/04, 2024

# Lecture 17: Engineering Interpretations of PoA

*Instructor: Eva Tardos* *Scribe: Tolkien Bagchi*

## 1 Classes of Functions for which we can Compute Price of Anarchy

In the first class we worked out PoA bounds for functions of the form $f(x) = x^n$. We claim that if $\mathfrak{C} = \{$polynomials of degree d with positive coefficients$\}$ then $\alpha(\mathfrak{C}) =$PoA on 2-node networks. We can see this by noting that for a cost function $c(x) = \sum_{i=0}^{d} a_i x^i \in \mathfrak{C}$ we can partition the $2-$node network in the last class by dividing the variable cost edge into $d+1$ directed edges with costs $c_i(x) = a_i x^i$.




Recall now that PoA depends on the worst edge. See notes from lecture 1 for the exact value.

## 2 Engineering Insights from PoA for Network Routing Games:

There is often a dilemma for engineers tasked with maintaining a routing network between optimizing flow vs. adding capacity. We claim that adding capacity is no worse than optimizing flow. To show this we consider an equilibrium flow $f$ for a routing game defined in the last lecture with the following notation:

(i) A fixed network $N = (V, E)$.

(ii) Sources and targets $(s_i, t_i) \in V^2$ with rate $r_i \geq 0$.

(iii) Each edge $e \in E$ has a cost $c_e(x)$ where $x$ is the flow on $e$ which is continuous and monotone increasing.

(iv) For all paths $P$ from $s_i$ to $t_i$, a flow $f_P \geq 0$ such that

$$\sum_{\text{paths } P \; s_i \text{ to } t_i} f_P = r_i$$

(v) For an edge $e \in E$ let $f(e) = \sum_{P: e \in P} f_P$

(vi) The cost of a path $P$ given a flow $f$ is given by $C_P(f) = \sum_{e \in P} f(e) c_e(f(e))$.

(vii) Because $f$ is an equilibrium flow, for any path $P$ $s_i$ to $t_i$, $f_P > 0$ implies that $P$ is a min-cost path $s_i$ to $t_i$.

(viii) $\text{cost}(f) = \sum_P C_P(f)$

Now consider a $f^*$ be the min cost flow on the same network except all source target pairs $(s_i, t_i)$ have rate $2r_i$.

**Theorem 1.** $\text{cost}(f) \leq \text{cost}(f^*)$ *when cost is increasing, continuous, and non-negative.*

**In English:** Doubling capacity is just as good as optimizing flow.

**Proof.** Following a similar argument as with the theorem from lecture 16.

$$\begin{aligned} \text{cost}(f) &= \sum_e f(e)c_e(f(e)) \\ &= \sum_P c_P(f)f_P \end{aligned}$$

Now, let $\bar{c}_e = c_e(f(e))$ be fixed. We want to bound $\text{cost}(f^*)$ below using $\bar{c}$ costs.
**Claim 1:** $\text{cost}(f^* \text{ with } \bar{c} \text{ costs}) \geq 2\,\text{cost}(f)$.
This follows since $f$ carries all flow with min cost, $f^*$ has double the rate, and all costs are fixed.
**Claim 2:** $f^*(e)c_e(f^*(e)) \geq f^*(e)c_e(f(e)) - f(e)c_e(f(e))$.
We have two cases:

(1) $f^*(e) \geq f(e)$. Since $c_e$ is monotone, the inequality holds as $f^*(e)c_e(f^*(e)) \geq f^*(e)c_e(f(e))$

(2) $f^*(e) < f(e)$. The inequality holds because the RHS is negative while LHS is non-negative.

We can now use the inequality from **claim 2** to obtain

$$\sum_e f^*(e)c_e(f^*(e)) \geq \sum_e f^*(e)c_e(f(e)) - \sum_e f(e)c_e(f(e))$$

by summing over all $e \in E$. From lecture 16 we have the following identities

$$\sum_e f^*(e)c_e(f^*(e)) = \text{cost}(f^*) \tag{1}$$

$$\sum_e f^*(e)c_e(f(e)) = \text{cost}(f^* \text{ with } \bar{c} \text{ costs}) \tag{2}$$

$$\sum_e f(e)c_e(f(e)) = \text{cost}(f) \tag{3}$$

Hence, we obtain

$$\text{cost}(f^*) \geq \text{cost}(f^* \text{ with } \bar{c}) - \text{cost}(f).$$

Now we can combine this result with our first claim to obtain $\text{cost}(f^*) \geq 2\,\text{cost}(f) - \text{cost}(f) = \text{cost}(f)$. ■

It should be noted that different classes of cost functions (for example linear functions) can lead to better bounds. A favorite form of cost function in queuing theory is

$$c_e(x) = \frac{1}{u - x}\ ;\ x < u$$

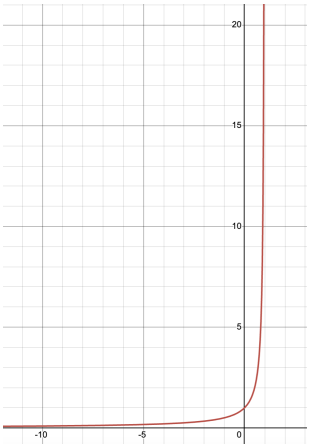
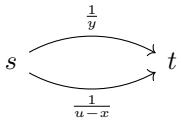



Figure 1: Graph of $c_e(x) = 1/(1-x)$

It is used for package routing where the cost functions represent capacity constraints as they increase to infinity as $x$ approaches $u$. PoA in this case is infinite. For a sketch of the proof let $r = u - y$ and reduce the problem to the 2-node case.




Then PoA works out to be $\sim \frac{1}{2\sqrt{y/u}}$.

# 3 Next Week

We will switch to a discrete version of the theorem next time. In particular we will be looking towards applying learning since it doesn't work in continuous contexts.