This week’s lecture is a continuation of the truthful systems discussion from last week. This
time, as last time, we consider a simple truthful system, in which each agent has only one parameter
to report.

1 Example Game

This example takes place on a graph $G$, where each agent wants a path in $G$ from nodes $s \rightarrow t$
because the path has a value $v$. Agents 1 to $k$ have paths from $s_i \rightarrow t_i$, with value $v_i$. The goal
in this game is to find a subset $I \subseteq \{1 \ldots k\}$ where each agent $i \in I$ has a disjoint path $P_i$ in $G$,
while maximizing the social welfare $\sum_{i \in I} v_i$. For simplicity, disjoint paths are assumed, but other
variants of this game exist where a certain number of paths may overlap on an edge. The objective
here is welfare maximization, but other games are possible using other measures, such as income
maximization.

We desire a truthful game for the usual reasons: simplicity and predicability. Note that each
player $i$ currently has three parameters $s_i$, $t_i$, and $v_i$. It is possible that a user might lie about
different combinations of these parameters, but in this example, we assume that the $s_i$ and $t_i$ are
public knowledge.

2 The Path Selection Algorithm

In this system, agents only report $v_i$, and then some public algorithm selects $I$, the paths $P_i$ for
$i \in I$, and the payments $w_i$ for each of the paths. We only collect this payment $w_i$ so that agents
are truthful. The total benefit to agent $i$ receiving a path is $v_i - w_i$. Last week, using the VCG
mechanism, it was shown that if welfare maximization is computable, a truthful system can be
created. Here, unfortunately, welfare maximization on disjoint paths is NP-hard. What else can we do?

This problem can be broken into two major parts:

- Select the agent paths
- Select the agent payments

The path selection algorithm should be good at both maximizing welfare and maintaining truthfulness. We introduce:

Property: If $v_i$ is high enough, with all other $v_j \neq v_i$ fixed, agent $i$ should be able to ensure they
are included in $I$. 


Basically, this property ensures that the values of the agents are not ignored, otherwise a simple truthful algorithm would simply be to ignore the agents’ values entirely. While this property does not ensure welfare is maximized, it does ensure that high values are respected.

**Property:** For a given \( v_i \) where \( v_j \neq v_i \) are fixed, \( \exists v' \) such that if \( v_i > v' \), \( i \in I \), and if \( v_i < v' \), \( \neq I \). Note the strict inequality.

Graphically, in a system where this property does not hold,

![](image1.png)

The separate regions in the \( v_i \) number line rules out truthfulness, because an agent in a higher \( v_i \) region could reduce their value and still get their path. Instead, \( v_i \) needs to look like...

![](image2.png)

This property is known as monotonicity. Does it necessarily lead to truthfulness?

**Proof:** (by contradiction) Assume \( v_i \) is not monotonic, with some \( u > v \) where at \( v_i = u, \neq I \) and \( v_i = v, i \in I \). Then the algorithm cannot be truthful.

![](image3.png)

At \( v_i = v, i \in I \), path \( P_i \) would require a payment \( w_i, w_i \leq v_i \), because otherwise \( v_i - w_i < 0 \).

But then at \( v_i = u, \neq I \), agent \( i \) would want to lie and say their \( v_i = v \), because their benefit at \( u \) would be 0. This would be less than their benefit \( v_i - w_i \geq 0 \) at \( v \).

Therefore, every truthful system has this monotonicity property. It remains to be proven that any monotonic algorithm has payments \( w_i \) that make it truthful.

**Proof:** In a monotonic system, every person has cutoff \( v' \). If we make payments for a path \( P_i \) equal to \( v' \), the system will be truthful, because if an agent’s value is any amount above \( v' \), it won’t matter to them, and if the value is under \( v' \), the agent will only want to increase their price if their true value \( v_i \) is greater than \( v' \).

Can we construct an algorithm with these two properties? Our goal is to maximize social welfare \( \sum_{i \in I} v_i \). (It is easy to see that a function finding the true maximum will be monotone for each \( v_i \)) Since this function will be NP-complete, we want a good approximation algorithm instead that is monotone. The disjoint union of paths has only bad approximation algorithms, but fortunately, these simple algorithms tend to be monotone.

### 3 Approximation Algorithms

**Example:** A greedy path search. Sort \( v_i \) by size, then assign paths for the agents with largest \( v_i \) first.
This simple greedy algorithm satisfies the first property, because an agent can always increase its value enough to ensure its path is chosen. The algorithm is also monotone, because if an agent has a path and increases its value, their path will still always be chosen (assuming no one else changes). Because it supports these properties, the greedy algorithm leads to a truthful system. However, the greedy algorithm has bad worst-case bounds, with an approximation factor of about \( n \times \) the optimal value, where \( n \) is the number of nodes in the graph. This bad case is illustrated below:

\[
\begin{array}{c}
\circ & \circ & \circ & \circ & \circ \\
\end{array}
\]

\[
> x
\]

**Example:** Modified greedy path search. Sort all the unassigned agents (agent \( i \notin I \)) by \( \frac{n}{f(l_i)} \), where \( l_i \) is the length of the current shortest path from \( s_i \rightarrow t_i \). This length is recalculated after each agent path is chosen.

This function satisfies the first property, because an agent that increases its value enough will still always be able to ensure their path (though if their path is long, the value must increase more). The algorithm is also monotone, because if an agent has a path and increases its value, the \( f(l_i) \) remains constant, and so it will always sort the same or higher in the list of agents. As above, the modified greedy algorithm is therefore truthful. This modified algorithm still has bad worst case bounds:

- if \( f(x) = 1 \), then the approximation factor is again \( n \).
- if \( f(x) \geq x \), the approximation factor is \( > n \).

This can be seen in the diagram below:

\[
\begin{array}{c}
\circ & \circ & \circ & \circ & \circ \\
\end{array}
\]

\[
lx
\]

If these are the extremes, then a tradeoff between 1 and \( x \) would yield a better approximation factor. Though not proven here, if \( f(x) = \sqrt{x} \), this gives a \( \sqrt{n} + 1 \) approximation factor. It is possible to do better, if more than one path is allowed on a particular edge of \( G \), but this also requires a more complex approximation function.