Online Maximum Directed Cut

(A bird in the hand is worth $\sqrt{3}$ in the bush)

Amotz Bar-Noy and Michael Lampis

City University of New York
Outline

- Problem definition and motivation
- $\frac{3\sqrt{3}}{2}$-competitive algorithm for DAGs
- Lower bound
- 3-competitive algorithm for general graphs.

Note: only deterministic algorithms are considered.
Max (Di) Cut

- Input: a (di)graph $G$ with weights on the edges
- Goal: divide the vertices into two sets $V_0, V_1$ so as to maximize the total weight of edges going from $V_0$ to $V_1$.
- In directed version arcs going from $V_1$ to $V_0$ are not counted in the cut.
Max (Di) Cut

- Input: a (di)graph $G$ with weights on the edges
- Goal: divide the vertices into two sets $V_0, V_1$ so as to maximize the total weight of edges going from $V_0$ to $V_1$.
- In directed version arcs going from $V_1$ to $V_0$ are not counted in the cut.
Max (Di) Cut

Input: a (di)graph $G$ with weights on the edges

Goal: divide the vertices into two sets $V_0, V_1$ so as to maximize the total weight of edges going from $V_0$ to $V_1$.

In directed version arcs going from $V_1$ to $V_0$ are not counted in the cut.
Max (Di) Cut

Input: a (di)graph $G$ with weights on the edges

Goal: divide the vertices into two sets $V_0, V_1$ so as to maximize the total weight of edges going from $V_0$ to $V_1$.

In directed version arcs going from $V_1$ to $V_0$ are not counted in the cut.
Max (Di) Cut

- Input: a (di)graph $G$ with weights on the edges
- Goal: divide the vertices into two sets $V_0, V_1$ so as to maximize the total weight of edges going from $V_0$ to $V_1$.
- In directed version arcs going from $V_1$ to $V_0$ are not counted in the cut.
Known results

- Max Cut was one of Karp’s 21 original NP-complete problems.
- Very good SDP-based approximations (1.383 (Goemans and Williamson) and 1.165 (Feige and Goemans))
- Trivial 2 and 4-approximate combinatorial algorithms.
- For Max Di Cut combinatorial 2-approximation (Halperin and Zwick)
- Max Di Cut is NP-hard even if restricted to DAGs (last year’s ISAAC!)
Why online?

- Disclaimer: No real application is known!
- For Max Di Cut it is natural to consider a class of greedy heuristics
  - A vertex of high out-degree should be more likely to be placed in $V_0$.
  - Specifically, in DAGs, sources should always be placed in $V_0$.
  - For subsequent vertices we have a choice between a certain profit and a potential profit.
- This won’t work (the problem is NP-hard). But how bad is it?
Online model: algorithms make local choices based on the past and vertex degree.

The adversary reveals with each vertex its connections to previous vertices and its total in and out-degree.

The algorithm then places the vertex in $V_0$ or $V_1$.

For DAGs the adversary must reveal vertices respecting the topological ordering of the DAG.
Example – Naive Greedy
Example – Naive Greedy
Example – Naive Greedy
Example – Naive Greedy

Diagram:

1 -> 2
Example – Naive Greedy
Example – Naive Greedy
Example – Naive Greedy

1 → 2 → 3 → 4 → ...
Naive greedy is bad because it’s too optimistic.

Better to weigh the risks we take. Place a vertex in $V_0$ only if the promised potential profit is at least twice as much as the certain profit of $V_1$.

- Now harder to fool the algorithm to assign a long string of 0s. The edge weights must increase exponentially.
- Easy to see algorithm is no better that 2-competitive.
Worst case example
Worst case example

1 → 2
Worst case example
Worst case example
Worst case example
Worst case example
Worst case example
Worst case example
Worst case example
Worst case example

1 → 2 → 4 → 8 → ... → 2^{n-1} → 2^n - \varepsilon
Worst case example

1 → 2 → 4 → 8 → ⋮ → 2^{n-1} → 2^n - \varepsilon
Worst case example
Worst case example
Worst case example
A bird in the hand is worth $\lambda$ in the bush

- Previous algorithm is $\frac{8}{3}$-competitive.
- Natural to optimize the weighing of risk versus payoff.
- Optimal value is $\lambda = \sqrt{3}$ which gives a $\frac{3\sqrt{3}}{2} \approx 2.6$-competitive algorithm.
Main result: no deterministic algorithm can do better.

Now we play as the adversary and try to force any algorithm to be $\lambda$-competitive.

Strategy: construct a directed path. Weights are calculated in such a way that if the algorithm places a 1, we immediately win ($OPT \geq \lambda SOL$).

If we can maintain this invariant and the weight decreases at some point, the algorithm must assign 1 and we win!
Example: Lower bound of 2
Example: Lower bound of 2
Example: Lower bound of 2
Example: Lower bound of 2

1 2
Example: Lower bound of 2
Example: Lower bound of 2
Example: Lower bound of 2
Example: Lower bound of 2
Example: Lower bound of 2
Example: Lower bound of 2
Example: Lower bound of 2
Example: Lower bound of 2
For which $\lambda$ can the adversary succeed with this strategy?

We must have $w_k + w_{k-2} + \ldots \geq \lambda w_{k-1}$ for all $k$, and $w_k$ is not always increasing.

Cleaner form $w_k = \lambda (w_{k-1} - w_{k-3})$. This is not always increasing for $\lambda < \frac{3\sqrt{3}}{2}$.

Upper and lower bounds match!
Again, weigh certain payoffs twice as much as potential ones.

This algorithm is a greedy derandomization of the trivial randomized algorithm.

⇒ 4-competitive

Is this the best we can say?
Actually, we know that $\text{SOL} \geq \frac{|E|}{4} \geq \frac{\text{OPT}}{4}$.

These inequalities cannot be tight at the same time.

Using first inequality and modified arguments from the case of DAGs we have $\text{SOL} \geq \frac{\text{OPT}}{3}$.

There exists a tight example for this algorithm, but best lower bound is the one for DAGs.
Open problems

- How to optimize $\lambda$ for general graphs?
- How to close upper-lower bound for general graphs?
- Randomized algorithms?
  - Vertices considered in random order?
  - Decisions involving randomization based on vertex degree?
THANK YOU!