1 Introduction

Let \( T \) be a tournament graph, that is, a graph where every pair of vertices is connected by a single directed edge. Our goal is to delete fewer than \( k \) edges to make \( T \) acyclic, or equivalently, to reverse the orientation of fewer than \( k \) edges to make \( T \) acyclic. This is also known as the \( k\)-FAST (\( k\)-Weighted Feedback Arc Set in Tournaments) problem. We will use the algorithm from \cite{1} to solve it.

We can solve this problem in \( 2^{O(\sqrt{k}\log^2 k)} + n^{O(1)} \) time in three steps: In the first step we will use kernelization to reduce the graph \( T \) into a smaller tournament \( T' \) with at most \( O(k^2) \) vertices. In the second step we color-coding to color the vertices of our graph with \( 10\sqrt{k} \) colors, and in the last step we will check whether the color-coding has resulted in a solution of size at most \( k \) using dynamic programming.

2 Step one: Kernelization

Kernelization is a technique which replaces the input for an algorithm with a smaller input, called a "kernel". The result of solving the problem on the kernel should be the same as on the original input.

To demonstrate kernelization, consider the following toy example.

**Toy example: Vertex Cover**

The Vertex Cover problem (given a graph, find a set \( S \) of at most \( k \) vertices such that each edge has at least one of its end points in \( S \)) can be solved in \( 1.618^k n^{O(1)} \), however, we can find a vertex cover in a slightly worse running time of \( 2^{k^2} n^{O(1)} \) using kernelization.

Let \( G \) be the graph we want to compute a size \( k \) vertex cover from. We can create a kernel \( G' \) by picking vertices from \( G \).

If a vertex \( v \) in \( G \) has a degree \( \geq k+1 \), then we must definitely pick it for the vertex cover. Otherwise all of its \( k+1 \) neighbours would have to be in the vertex cover, which results in a solution of size \( > k \).

Let our kernel \( G' \) be the graph \( G \) with all vertices with degree \( \geq k+1 \) removed, and let \( n' \) be the amount of vertices in \( G' \), and \( \Delta \) the maximum degree in \( G' \) (which is less than or equal to \( k \)). We know that each vertex in \( G' \) can cover at most \( \Delta \) edges, so \( G' \) has a vertex cover of size \( \geq \frac{n'}{\Delta} \geq \frac{n'}{k} \).

Since the vertex cover size can be at most \( k \) we have \( n' \leq k^2 \), so we can distinguish the following cases:

- If \( n' \leq k^2 \), we can simply use brute force in \( O(2^{k^2}) \) time.
- If \( n' > k^2 \), there exists no vertex cover of size \( \leq k \).
Now we will apply kernelization to the $k$-FAST problem. First of all, we define a triangle in the tournament as a cycle of length 3. Note that if there is a cycle of any length in a tournament $T$, it must mean there is a triangle in the tournament. Also note that for every triangle in $T$, at least one of its edges must be flipped.

We will transform a tournament $T$ into kernel $T'$ with $\leq k(k + 2)$ vertices. To do this we will use two observations:

1. If an edge $e$ is part of $\geq k + 1$ triangles, we should always flip it.
2. If a vertex $v$ is not part of any triangle, we can remove $v$ from the graph.

We can now say the following about $T'$:

1. All edges in $T'$ are in $\leq k$ triangles.
2. All vertices in $T'$ are in at least one triangle.

Claim 1 If there exists a solution $S$ of size $\leq k$, the number of vertices in $T'$ is $\leq k(k + 2)$

Proof: Let $S$ be a solution of size $\leq k$.

1. Every edge in $S$ can be in at most $k$ triangles, so the total number of triangles involving an edge of $S$ is at most $k^2$.
2. An edge from each triangle in $T'$ must be in $S$.
3. Every vertex of $T'$ is in a triangle.

From this we can bound the number of vertices in $T'$ to $k(k + 2)$.

3 Step two: Color-coding

We color each vertex in $T'$ randomly using $10\sqrt{k}$ colors. What we are aiming for is that for every edge in the solution $S$, the colors of its vertices are different.

For a single edge $e$ we have:

$$Pr\{\text{both endpoints } e \text{ have different colors} \} = 1 - \frac{10\sqrt{k}}{(10\sqrt{k})^2} = 1 - \frac{1}{10\sqrt{k}}.$$

In fact, for $k$ edges we have:

$$Pr\{\text{both endpoints of all } k \text{ edges have different colors} \} = \left(1 - \frac{1}{10\sqrt{k}}\right)^k \geq e^{-c\sqrt{k}},$$

for some constant $c$. We will prove this using the Inductive coloring argument:
Inductive coloring argument

Lemma 2 Given any graph $G$ with $n$ vertices and at most $k$ edges:

$$\Pr\{\text{No edge in } G \text{ is monochromatic}\} \geq \frac{1}{2O(\sqrt{k})}$$

Let's order the vertices as follows: We start with the vertex with the smallest degree $d_1$, remove it from $G$, then remove the next vertex with the (current) smallest degree $d_2$, remove it, and so on, until we’ve removed all vertices.

Claim 3 The degree $d_i$ of any vertex is at most $2\sqrt{k}$.

Proof: Note that $d_1 \leq 2k/n$, since the sum of all degrees in the graph is at most $2k$. We have $n \geq \sqrt{k}$. This means:

$$d_i \leq \frac{2k}{n-i+1} \leq \frac{2k}{\sqrt{k}} \leq 2\sqrt{k}.$$  \hfill \Box

Now imagine coloring the vertices in order of descending degree, so starting with vertex $n$. If vertex $i$ is being colored, all vertices $j > i$ have already been colored. Vertex $i$ has at most $2\sqrt{k}$ colored neighbours, so:

$$\Pr\{\text{Vertex } i \text{ gets the same color as one of its colored neighbours}\} \leq \frac{2\sqrt{k}}{10\sqrt{k}},$$

so:

$$\Pr\{\text{No edge in } G \text{ is monochromatic}\} \geq \frac{1}{2^{c\sqrt{k}}}.$$  

To be more precise, we have:

$$\Pr\{\text{Vertex } i \text{ gets the same color as one of its colored neighbours}\} = \frac{d_i}{10\sqrt{k}},$$

so:

$$\Pr\{\text{No edge in } G \text{ is monochromatic}\} = \prod_i \left(1 - \frac{d_i}{10\sqrt{k}}\right)$$

$$= e^{\left(\sum_i - \frac{d_i}{10\sqrt{k}}\right)}$$

$$\approx e^{-c\sqrt{k}},$$

since $d_i \leq 2\sqrt{k}$.

This means that the probability that no edges in our solution $S$ in our color-coded $T'$ are monochromatic—which means our color-coding is valid—is at least $e^{-c\sqrt{k}}$.  

3
4 Step three: Dynamic programming

Let $T' = (V_1 \cup V_2 \cup \ldots \cup V_{10\sqrt{k}}, E')$ be a colored tournament, split into one piece $V_i$ for each color $i$. We now have a graph $T$ with $O(k^2)$ vertices, colored using $10\sqrt{k}$ colors. Assuming we got a valid coloring in step two, we can say that the subgraph $V_i$ consisting of vertices of color $i$ is acyclic (after all, if there is a cycle in $V_i$ at least one of its edges would be in $S$, but this would mean its endpoints have different colors). This means we have a big transitive piece for each color.

Using dynamic programming, we can now solve the original problem in $n^{10\sqrt{k}}$ time. This involves finding a global ordering on the vertices such that the number of back edges is minimized.

We will show the algorithm for only two colors, say red and blue. Since both the red and blue groups are transitive, we can use the following recurrence to order the vertices:

$$v(r, b) = \min(v(r - 1, b) + \text{number of back edges from } r\text{'th red to } \{1, \ldots, r - 1, 1, \ldots, b\},$$
$$v(r, b - 1) + \text{number of back edges from } b\text{'th blue to } \{1, \ldots, r, 1, \ldots, b - 1\})$$

This has a running time of $O(2nn^2)$. For $10\sqrt{k}$ colors, our dynamic programming algorithm consists of filling in a $10\sqrt{k}$ dimensional table with values $v(c_1, c_2, \ldots, c_{10\sqrt{k}})$, with a running time of $O(10\sqrt{k}nn^{10\sqrt{k}})$.

This means for $n := k^2$ we have a running time of $O(10\sqrt{k}k^2(k^2)^{10\sqrt{k}}) = O(k^{20\sqrt{k}})poly(k)$, which leads to a total running time of $2^{O(\sqrt{k}\log k)}n^{O(1)}$.

References