Group 2: Assignment 1.3

Niv Adam, David Kaufmann, Casper Kristiansson, Nicole Wijkman

October 19, 2023

We start by mapping the problem to a directed graph $G = (V, E)$. Every object resembles a node $x \in V$. Note that $|V| = n$. We interpret the data from the data analysis department as a set $E = \{(a, b) \in V \times V | a < b\}$ of tuples where the first component is lower than the second component according to the pairwise comparison. Set E are also the directed edges of our graph. Note that given the definition of the pairwise comparison, there exists exactly one edge between each pair of distinct vertices.

In the graph representation of the problem a circle corresponds to contradicting pairwise comparisons. Therefore our algorithm consists of two parts. Part 1 changes the direction of at most k edges. This corresponds to ignoring at most k pairwise comparison. In our analysis we show that it is sufficient to deal with circles of length three. Therefore we test the impact of flipping any of the three edges on such circles. If the graph still contains circles after removing k circles our analysis shows that there exists no solution to the given problem. Therefore the algorithm returns NO. Otherwise, part 2 returns an order of all elements in V that satisfies the problem statement.

Algorithm 1: DFS Circle Remover

```
1  Function RemoveCycles($V$, $E$, $k$, $removedCycles$):
2      if $removedCycles > k$ then
3          return $NO$
4      for $a \in V$ do
5          for $b \in V \setminus \{a\}$ do
6              for $c \in \setminus\{a, b\}$ do
7                  if $a, b, c$ create circle then
8                      $G'$ =RemoveCycles($V, (E \setminus (a, b)) \cup (b, a), k$,removedCycles+1)
9                      if $G' \neq NO$ then
10                         return $G'$
11                     $G'$ =RemoveCycles($V, (E \setminus (b, c)) \cup (c, b), k$,removedCycles+1)
12                     if $G' \neq NO$ then
13                         return $G'$
14                     return $RemoveCycles(V, (E \setminus (c, a)) \cup (a, c), k, removedCycles+1)$
15     return $G=(V,E)$
```

Algorithm 2: Find Longest Path

```
Function FindLongestPath(V, E):
    order = []
    while |V| > 0 do
        Find vertex $x \in V$ with no ingoing edges
        $E = E \setminus \{(x, a) \in E | a \in V, a \neq x\}$
        $order.append(x)$
        $V = V \setminus x$
    return order
```

1 Correctness Analysis

Lemma 1.1. *Let $C = \{(c_{i \bmod n}, c_{i+1 \bmod n}) \in E | i \in \{1..n\}\}$ be a circle of length $n \geq 3$ in G. Then C contains a circle C^* of length 3 that shares at least one edge with C.*

Proof. If $|C| = 3$ then $C^* = C$. Therefore lets assume $n > 3$. Given the construction of the graph we know that either $(c_2, c_n) \in E$ or $(c_n, c_2) \in E$.

Case 1: $(c_2, c_n) \in E$. Since by definition of C, $(c_1, c_2) \in C$ and $(c_n, c_1) \in C$ we get $C^* = \{(c_1, c_2), (c_2, c_n), (c_n, c_1)\}$

Case 2: $(c_n, c_2) \in E$. Since by definition of C a path from c_2 to c_n exists we get a new circle $C_{n-1} = C \setminus \{(c_1, c_2), (c_n, c_1)\} \cup \{(c_n, c_2)\}$ with $|C_{n-1}| = |C| - 1$ that shares $n - 2$ edges with C

We continue with inspecting the edge between c_i and c_n for $i \in \{3..n-2\}$ with the same cases until either Case 1 is true or Case 2 applies to the edge between c_{n-2} and c_n.

If we accept Case 1 in any step, then $C^* = \{(c_x, c_{x+1}), (c_{x+1}, c_n), (c_n, c_x)\}$ shares edge (c_x, c_{x+1}) with C.

Otherwise we know $(c_n, c_{n-2}) \in E$ which gives us $C^* = \{(c_{n-2}, c_{n-1}), (c_{n-1}, c_n), (c_n, c_{n-2})\}$ which shares 2 edges with C. □

From Lemma 1.1 we can follow that if G does not contain circles of length 3, G does not contain any circle. This motivates the following Lemma.

Lemma 1.2. *Flipping an edge on a circle of length three can not increase the number of circles of length three $\#C_3$ in the graph.*

Proof. Let C be a circle of length 3 and (x, y) an edge on that circle. Let $v \in V$ be a vertex not in C. We know that x and y both have edges to v. We can distinguish three cases.

Case 1: x, y, v are on a circle. Flipping (x, y) breaks this circle too and therefore $\#C_3$ is lower than before.

Case 2: x, y, v create a circle after flipping (x, y). Then $\#C_3$ remains the same since flipping (x, y) breaks C and creates a new circle with v.

Case 3: x, y, v don't create a circle before or after flipping (x, y). Then $\#C_3$ is lower afterwards, since C is broken.

Since these cases can be applied to any v the number of circles can not increase by flipping an edge. □

From Lemma 1.2 we get that flipping an edge can only reduce the number of circles in the graph. Since in our algorithm we try all edges on k circles it is clear that there exists no solution if we don't find one.
If G had $<= k$ circles, we can find an order of vertices (=objects) that satisfies the problem leveraging our modified circle-free graph $G' = (V', E')$. We find the longest path by finding and removing vertex $v \in V'$ with $|\{(a, v) \in E' | a \in V'\}| = 0$ and all its outgoing edges $\{(v, a) \in E' | a \in V'\}$ until $|V'| = |E'| = 0$. We will find a vertex to start with, as there are no circles in our graph (anymore) and it must thus exist a vertex with no ingoing edges. The order in which we removed the vertices is an order of objects that satisfies the problem. Algorithm 2 illustrates this procedure.

2 Time Analysis

2.1 Part 1

Each execution of this algorithm tests each possible triplet of vertices on whether they create a circle. This has complexity $O(n^3)$. Whenever such a circle is found the graph is modified by flipping one of the three edges of the circle. Afterwards the same method is called on the modified graphs. This creates a search tree with three children per vertex. Since no branch is longer than $k + 1$ the complexity of this is $O(3^k)$. In total this gives a complexity of $O(3^k \cdot n^3)$.

2.2 Part 2

As exactly one vertex $x \in V$ is removed from V per execution, the while loop is executed $|V|$ times. Assuming the data structure of E and V, finding a vertex $v \in V$ with no ingoing edges required an iteration over each $v \in V$. In the worst case, we need to iterate over all edges $(a, b) \in E$ with $a, b \in V$ to find out if v has any ingoing edges. This has complexity $O(|V| \cdot |E|)$. As all objects end up in the order array eventually, appending to this array lies in $O(|V|)$.

It follow, that the complexity for part 2 is dominated by $O(|V| \cdot |E|)$. Since $|V| = n$ and $|E| = \binom{n}{2} \leq n^2$ the complexity of part 2 is within $O(n^3)$.

2.3 Overall

Taking the time complexity of both parts into account, we are left with an overall complexity of $O(3^k n^3)$ which is $O(f(k)poly(n))$. Our solution thus is an FPT algorithm.