Es hat sehr Spaß gemacht an der Case Study zu knobeln. Ich habe mich besonders darauf konzentriert, einen path finding algorithmus zu finden, der trotz cyclic paths performant bleibt. Dank der cyclic paths gibt es selbst in einem kleinen Graph sehr viele mögliche Pfade. Die Besonderheit meiner Lösung ist, dass zu lange Pfade on-the-fly ausgesiebt werden, was die Performance erhöhen sollte. Meine Lösung sollte besonders gut mit großen Toleranzen umgehen können. Im Umkehrschluss vermute ich, dass meine Lösung bei sehr großen Graphen – speziell bei niedrigen Toleranzen - ggf langsamer zum Ergebnis kommen könnte als andere Algorithmen. Besonders das Detail, dass meine Lösung in extend\_each\_path\_in\_each\_direction() für jeden einzelnen Pfad über alle Nodes des Graphen iteriert, kann bei zu vielen Nodes zum Problem werden.

**Schwächen in der Vorgabe der Case Study:**

1. Vorgebener Code ist nicht zu 100% auf Industrie-Standart.

2. Die Vorgabe, dass die Toleranz bis "Inf" gehen kann, ist nach meinem Verständnis fehlerhaft. In diesem Fall gibt es dank cyclic paths unendlich viele Lösungen.

3. Es fehlen Benchmarks um die Runtime zu evaluieren. Ich weiß nicht ob meine Lösung effizient ist.

**Schwächen in meiner Implementation:**

1. Error handling: Ich habe aus Zeitgründen error handling größtenteils übergangen. In produktiven Code müssten Fälle, wie zB fehlerhaften Graphen, abgefangen werden.

2. Struktur: Ich habe alle Definitionen nur in main.py und utils.py aufgeteilt. Für eine produktive Codebase würde man das vermutlich granularer und deskriptiver aufteilen.

3. Effizienz: Meine Lösung kann vermutlich noch verbessert werden. Besonders das erwähnte Detail in extend\_each\_path\_in\_each\_direction() könnte man klüger lösen. Außerdem könnten andere Algorithmen insgesamt adequater sein als meiner.

4. Falls der Graph fehlerhaft ist und gar kein Weg zum End-Node führt, läuft mein Code für immer.