Meetings

Vorbereitung für Meetings (Karol)

1. Logo
2. Kommunikationsplatform
3. **Arbeitsweise**
4. Programmiersprache
5. Shared Cloud und Versionsverwaltung
6. Solution Validator
7. Lemma: Maximale und minimale Distanz zweier Ansteuerpunkte in einer Optimalen Lösung
8. Lösungsansätze

1) Logo

Kurz über das Logo abstimmen und besprechen, welche Änderungen wo eingebracht werden sollen.

2) Kommunikationsplatform

WhatsApp ist zwar schnell und einfach, aber sehr unübersichtlich um mehrere Gespräche führen zu können. Gleichzeitig ist es schwerer, einzelne Nachrichten zu bestimmten Themen wiederzufinden. Hierzu bedarf es einer Lösung.

Karols Vorschlag: Slack.

3) **Arbeitsweise**

Es ist sinnvoll, sich zu überlegen, wann, in welchem Abstand und wie man sich treffen möchte. Des Weiteren sollte ein Vorgehen anhand zum Beispiel einer Roadmap erkennbar sein.

4) Programmiersprache

Bestandsaufnahme der bekannten Programmiersprachen. Je nach Lösungsansatz haben verschiedene Programmiersprachen Vorteile. Wahrscheinlich ist es trotzdem besser eine suboptimale Sprache zu wählen, ob für mehr Personen Programmieren zu ermöglichen. Beispiel: C++ ist sehr mächtig aber schwer zu lernen.

Karols Vorschlag: Python - Sehr einfache Sprache; Viele und gute Libaries + Dokumentationen; Selbst recht fitt drinnen.

5) Shared Cloud und Versionsverwaltung

Daten wie zum Beispiel der Programmcode müssen ausgetauscht werden. Wird dafür eine Cloud benötigt? Programmcode in einer regulären Cloud ist mit einem sehr hohen *Merge* Aufwand

verbunden (Code verschiedener Personen zusammenschnipseln). Dies übernehmen in der Regel Versionsverwaltungen wie zum Beispiel GitHub. Lohnt es sich der Aufwand, für alle Git zugänglich zu machen? Wie sollen man *gemeinsam* Programmcode entwerfen.

6) Solution Validator

Es ist möglich, dass der Algorithmus, welchen wir programmieren, Fehler machen kann. Daher ist es notwendig, dessen Lösung mit einem seperaten, einfacheren und viel langsameren Programm zu validieren. Unabhängig vom gewählten Lösungsansatz erhält man die Lösung im gleichen Format: eine Liste von Punkten im Dreidimmensionalen Raum. Daher kann der Entwurf auch zuvor bereits begonnen werden. Dieser Task ist auch nicht zu schwer, da für die Validation einer vorgeschlagenen Lösung viel Zeit genutzt werden kann (wird sehr selten ausgeführt). Daher kann auch die naive Implementierung genutzt werden ($\mathcal{O}(n * m)$ mit n ist die Validierungsauflösung und $m = |solution|$):

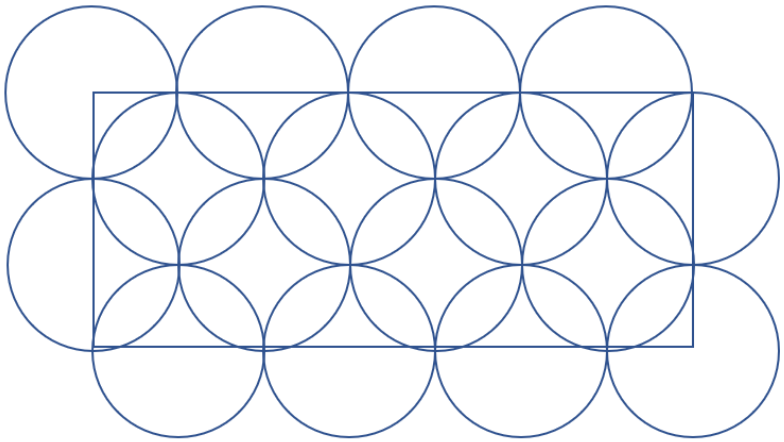
```
def validate(solution, r, n):
    for i in range(1,n):
        p = generate_random_point_on_sphere()

        covered = False

        for v in solution:
            if dist(v,p) < r:
                covered = True
                break
        if covered == False:
            return False

    return True
```

7) Lemma: Maximale und minimale Distanz zweier Ansteuerpunkte in einer Optimalen Lösung

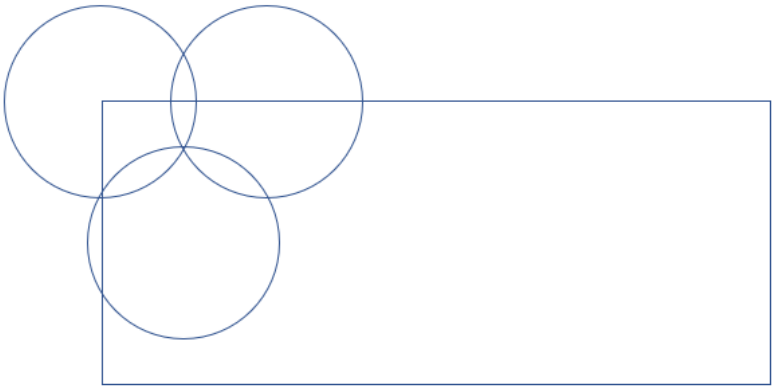
Karol: Ich habe die Vermutung, dass man nicht immer die ganze Sphere betrachten möchte. Es könnte reichen ausgehend von einem aufgenommenen Foto im Punkt p, Kandidaten in einer gewissen Distanz θ für das nächste Foto zu betrachten (Distanz meint hier nicht die euklidische Norm, sondern $d(x, y) = \arccos(x, y)$).

Diese Vermutung basiert auf der Überdeckung eines Rechtecks mit Kreisen. Betrachtet man die Grafik, dann wird einem schnell klar, dass in jeder überdeckenden Lösung Kreise entweder sich schneiden oder zumindest sich berühren müssen. Andernfalls gäbe es Punkte, welche nicht

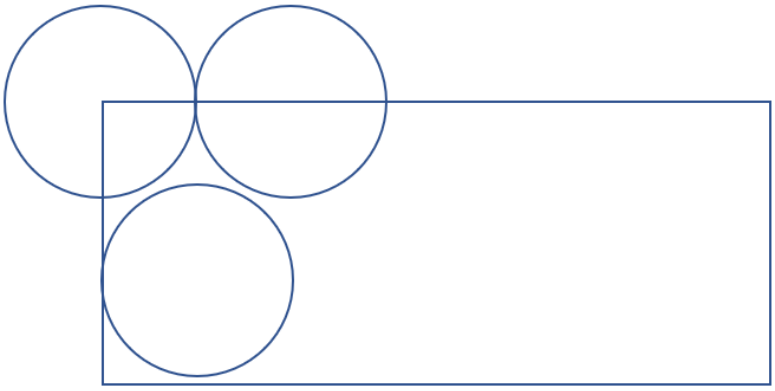
Überdeckung eines Rechtecks mit Kreisen



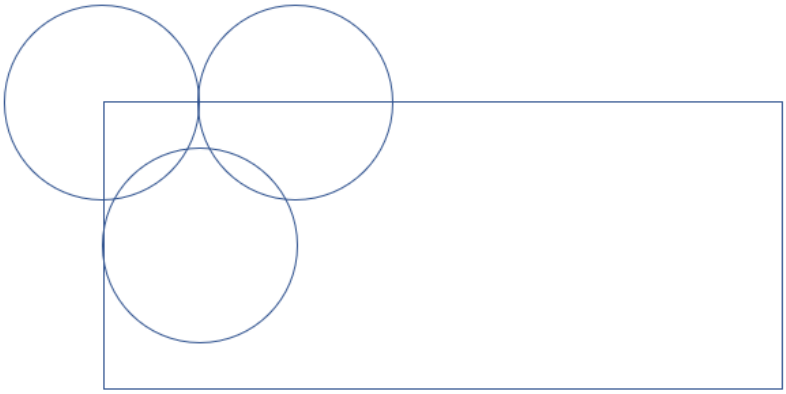
Überdeckung möglich, da Überschneidungen vorhanden



Überdeckung nicht möglich, da keine Überschneidungen vorhanden



Überdeckung nicht möglich, obwohl Überschneidungen vorhanden



Versteht man die Kreise als Konten von Graphen, so würde der Begriff **zusammenhängend** Sinn machen.

Es gilt also das folgende Lemma zu beweisen:

Lemma

Für alle $B = B(x, r)$ einer Überdeckung I(X,r) von X mit jenen Kugeln gilt folgendes:

$$\exists B' = B(x', r) \in I(X, r) : d(x, x') \leq 2r$$

8) Lösungsansätze

- Minimal Dominating Set (MDS) + Genetic Algorithms: Geniere einen Graphen, dessen Knoten auf der Sphere liegen. Bestimme mit dem **Greedy-Approch** und einer Heuristik die einzelnen Lösungen. Wende dann auf die Parameter der Heuristik einen Genetic Algorithms an (vgl. Evolution).
  - Greedy-Approch: Auswählen von Knoten für Dominating Set.
  - Greedy-Approch: Auswählen von Knoten, welche nicht zum Dominating Set gehören.
  - ggf. mit 7) auf MCDS erweiterbar.