Minimale Aufspannende Bäume - Minimum Spanning Trees (MST)

Definition

Für einen zusammenhängenden ungerichteten Graphen G mit positiven Gewichten ist eine Teilmenge E' von E, die alle Knoten miteinander verbindet, ein aufspannender Graph. Der aufspanndende Graph, der die Summe aller Gewichte der Kanten in E' minimiert, ist der minimale aufspannende Baum.

MST

Grundbegriffe

- Schnitt:

  Teilmenge E' von E, so dass $G \backslash E'$ nicht verbunden ist. Zwei Teilmengen von nicht verbundenen Knoten: $S \subset V$ und $V \backslash S$.

- Kreuzen:

  Liegt für $(u, v) \in E$ ein Endpunkt in S und der andere in $V \backslash S$, so kreuzt (u, v) den Schnitt E'.

- Respektieren:

  Eine Kante, die einen Schnitt nicht kreuzt, respektiert ihn.

- Leichte Kante:

  Eine Kante, die den Schnitt kreuzt und minimales Gewicht hat.

Eigenschaften

Eigenschaft des Schnitts:

Sei S eine Menge von Knoten von G mit $|S| > 1$ und $|S| < n$. Jeder minimale aufspannende Baum von G enthält eine leichte Kante (u, v) mit u in S und v in $G \backslash S$.

Keine Zyklen:

Sei c ein einfacher Zyklus in G, und sei (u, v) die Kante in c, die das höchste Gewicht hat. Dann ist (u, v) nicht Teil eines minimalen aufspannenden Baums.

Grundidee der folgenden Algorithmen:

Wiederholtes Einfügen von leichten Kanten

Kruskal Algorithmus

Algorithm 1: Kruskal Algorithmus zum Finden eines MST:

```
1  A = ∅
2  for alle Knoten v von G do
3  |   Speichere Knoten v als separaten Baum
4  end
5  Sortiere die Kanten von G in aufsteigender Reihenfolge (Gewicht)
6  for alle Kanten (u, v) von G in aufsteigender Reihenfolge do
7  |   if u und v sind in in zwei verschiedenen Bäumen then
8  |   |   A = A ∪ (u, v)
9  |   |   Verbinde den Baum, der u enthält mit dem Baum, der v
   |   |   enthält mit Kante (u, v)
10 |   end
11 end
12 Ergebnis: A
```

Kruskal Algorithmus

Darstellung eines Baums:

- Jeder Knoten von G bekommt eine eindeutige id zwischen 1 und n
- Merke für jeden Baum den Knoten mit der kleinsten id
- Damit: Laufzeit $O(mn + m \log m)$
- Mit besserer Implementierung des Verbindens von Bäumen geht es auch schneller: $O(m \log m)$

Prim Algorithmus

- Algorithmus und Laufzeitanalyse ähnlich zu Dijkstra Algorithmus
- Min-Priority Queue Q enthält Knoten v mit Priorität $v.key$

Prim Algorithmus

Algorithm 2: Prim Algorithmus zum Finden eines MST:

```
1  for alle Knoten u von G do
2  |   u.key = ∞
3  |   u.predecessor = NULL
4  end
5  s.key = 0
6  Füge alle Knoten von G in Q ein
7  while Q ≠ ∅ do
8  |   u = EXTRACT-MIN(Q)
9  |   for alle Knoten v mit (u, v) in G do
10 |   |   if v ∈ Q und ω(u,v) < v.key then
11 |   |   |   v.key = ω(u,v)
12 |   |   |   v.predecessor = u
13 |   |   end
14 |   end
15 end
16 Ergebnis: Kanten {(v, v.predecessor)}
```

Anwendung: finde Strukturen in Daten

- n Datenpunkte und Distanz zwischen allen Paaren von Punkten
- Kompletter ungerichteter Graph G mit positiven Gewichten (=Distanzen)
- **Cluster** von Datenpunkten: gewünschte Eigenschaften
  - Maximale Distanz zwischen zwei Punkten desselben Clusters $d_{inner} \rightarrow \min$
  - Minimale Distanz zwischen zwei Punkten in verschiedenen Clustern $d_{inter} \rightarrow \max$
  - Beide Zielen können im Allgemeinen nicht gleichzeitig erreicht werden

Finde k Cluster

Algorithmus

- Finde einen minimalen aufspannenden Baum T von G
- Entferne die $k-1$ längsten Kanten von T

Eigenschaften:

- d_{inter} wird maximiert
- Leider keine Garantie für d_{inner}