Karlsruher Institut für Technologie
Fakultät für Informatik
ITI Wagner

Algorithmische Geometrie
Sommersemester 2014
Martin Nöllenburg/Benjamin Niedermann

Übungsblatt 13 – Sichtbarkeitsgraph (Lösung)

1 Kürzeste Wege I

Sei S eine Menge von nicht überlappenden Polygonen in der Ebene mit insgesamt n Kanten.

a) Zeigen Sie, dass für jede beliebige Start- und Ziel-Position die Zahl der Segmente auf einem kürzesten Weg durch $O(n)$ nach oben begrenzt ist.

b) Geben Sie ein Beispiel an, bei dem die Anzahl der Segmente $\Theta(n)$ ist.

Lösung:

Zu a): Wiederholung von Lemma 1 aus der Vorlesung zu Sichtbarkeitsgraphen:

Lemma 1. *Für eine Menge S von disjunkten Polygonen in $\mathbb{R}^2$ und zwei Punkte s und t außerhalb S ist jeder kürzeste st-Weg in $\mathbb{R}^2 \setminus \bigcup S$ ein Polygonzug dessen innere Knoten Knoten von S sind.*

Wir wissen, dass $|S| = O(n)$ gilt. Da auf einem kürzesten st-Weg kein Knoten zweimal vorkommen kann und inneren Knoten (s. Lemma 1) nur Knoten aus S sein können folgt die Aussage.

Zu b): Siehe Übungsfolien.

2 Sortieren

Der Algorithmus VisibilityGraph ruft den Algorithmus VisibleVertices genau n mal auf und jeder Aufruf benötigt $O(n \log n)$ Zeit. Damit folgt eine Gesamtlaufzeit von $O(n^2 \log n)$. Zeigen Sie, dass man die Laufzeit dieses Schritts mit Hilfe von *Dualisierung* auf $O(n^2)$ reduzieren kann.

Bonus: Verändert dieser Schritt die Gesamtlaufzeit von VisibilityGraph?

Lösung:

Gute Erläuteurng findet sich in "Lecture Notes CMSC 754 Computational Geometry" von D. Mount. Weitere Illustrationen gibt es auf den Übungsfolien.

Die Laufzeit verringert sich durch diesen Schritt auf $O(n^2)$.

3 Kürzeste Wege II

Der in der Vorlesung vorgestellte Algorithmus zur Berechnung von kürzesten Wegen kann erweitert werden, so dass statt Polygonen auch andere Objekte für die Eingabemenge erlaubt sind. Sei S eine Menge von n nicht überlappenden Scheiben, die nicht notwendigerweise den gleichen Radius haben.

a) Zeigen Sie, dass in diesem Fall der kürzeste Weg zwischen zwei Punkten, die sich nicht direkt sehen, aus (i) Teilen der Grenzen der Scheiben, und/oder (ii) Tangenten zwischen zwei Kreisen, und/oder (iii) Tangenten vom Start/Ziel zu den Scheiben.

b) Adaptieren Sie den Begriff des Sichtbarkeitsgraphens auf diese Situation.

c) Adaptieren Sie den ShortestPath Algorithmus so, dass er in dieser geänderten Situation den kürzesten Weg zwischen zwei Punkten findet.

Zu a): Der Fall (i) folgt ähnlicher Argumentation wie der Beweis von Lemma 1 aus der Vorlesung. Der Fall (iii) folgt direkt aus dem Beweis zu Fall (ii), da ein Punkt nur ein degenerierter Kreis (Kreis mit Radius 0) ist.

Zu zeigen bleibt Fall (ii): Dabei setzen wir voraus, dass bekannt ist, dass der Abstand zwischen zwei Punkten auf dem Rand derselben Scheibe der kürzeste Kreisbogen zwischen beiden Punkten auf diesem Kreisrand ist (Fall (i)).

Zur Vereinfachung nehmen wir an, dass die obersten Punkte beider Kreise die gleiche y-Koordinate haben (Abbildung 1). Sei u ein Punkt auf dem Rabd von A, sodass u nicht B sieht. Analog, sei v ein Punkt auf dem Rand von B, sodass v nicht A sieht.

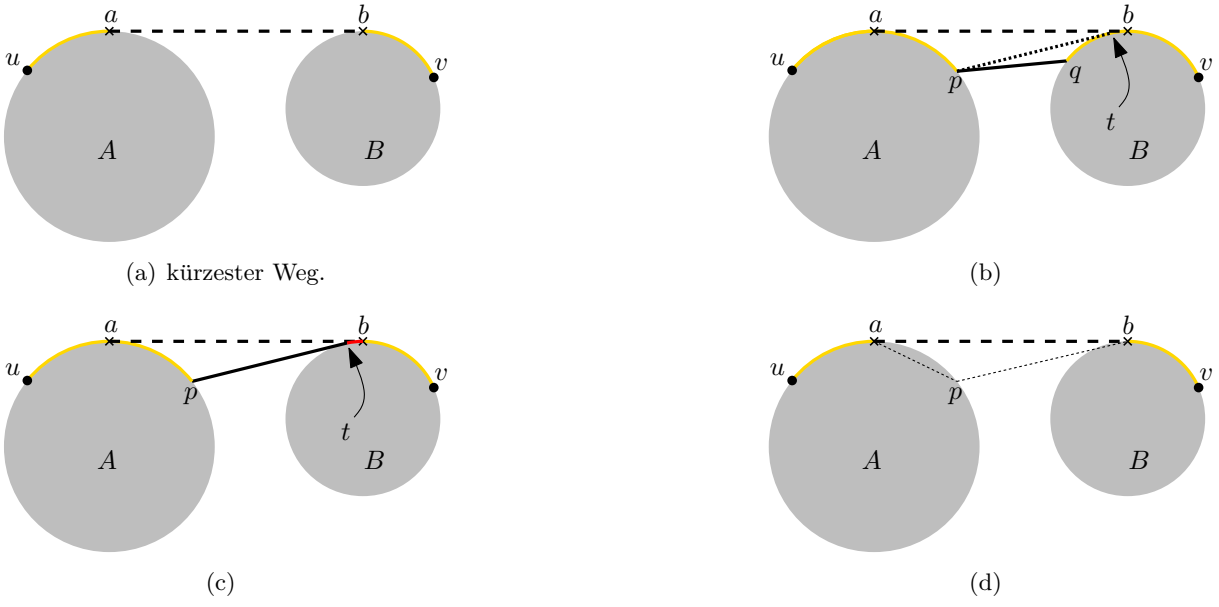

Abbildung 1: Illustration zur Beweisskizze von Aufgabe 3

1. Sei p ein Punkt auf dem Rand von A, sodass p die Scheibe B sieht. Wir zeigen, dass der kürzeste Weg von p nach v über eine Tangente an p und B führt (siehe Abbildung 1(b)). Beachte,

dass der kürzeste Weg, sobald er die Scheibe B erreicht hat vollständig auf dem Rand der Scheibe verläuft (Fall(i)). Dabei passiert der kürzeste Weg de Punkt t, welcher der Schnittpunkt der Tangente von p aus an B ist. Der kürzeste Weg von p zu t ist aber die direkte Verbindung (also entlang der Tangente). Damit folgt diese Teilbehauptung.

2. Es bleibt zu zeigen, dass wir nicht von beliebigen Punkten aus Tangenten an Scheiben legen, sondern nur von solchen Punkten die Schnittpunkte der Geraden mit den Scheiben ist, wobei die Geraden die Tangenten an beide Scheiben sind. Dabei muss man diverse Unterfälle betrachten. Hier zeigen wir nur einen Beispielfall. Mit ähnlicher Argumentation folgt dann der Rest. Wir zeigen, dass der Weg (u, p, t, v) der in Abbildung 1(c) dargestellt ist nicht der kürzeste ist. Diese Aussage folgt aus folgender Beobachtung: Betrachte das Dreieck a, p, b (siehe Abbildung 1(d)). Die Länge der beiden gepunkteten Linien ist kürzer als der tatsächliche Weg von a nach p und von p nach b, da der tatsächliche Weg nicht direkt sondern entlang der Scheiben verläuft. Aber selbst der gepunktete Weg von a nach p und von p nach b ist länger als die direkte Verbindung von a nach b. Damit folgt die Behauptung.

Eine genaue Fallanalyse zeigt dann alle Unterfälle.

Zu b): Definiere die folgenden Mengen:

- Für $a, b \in S$ sei $X(a, b)$ die Schnittpunkte der Geraden, die Tangenten an a und b sind.
- $V(S) = \bigcup_{a,b \in S} X(a, b)$
- $E_{\text{vis}}(S) = \{uv \mid u, v \in V(S) \text{ und } u \textit{ sieht } v\}$
- $E_{\text{arc}}(S) = \{\text{Kreisbogen } uv \mid u, v \in V(S) \text{ und } u, v \text{ auf gleicher Scheibe 'hintereinander' }\}$

Für die Kanten/Kreisbögen definieren wir das Gewicht der Kanten in $E_{\text{vis}}(S)$ und $E_{\text{arc}}(S)$ als ihre euklidische Länge.

Dann ist $G_{\text{vis}}(S) = (V(S), E_{\text{vis}}(S) \cup E_{\text{arc}})$ der **Sichtbarkeitsgraph** von S.

Definiere $S^\star = S \cup \{s, t\}$ und $G_{\text{vis}}(S^\star)$ analog. Bemerke dazu, dass ein Punkt ein degenerierter Kreis ist (Kreis mit Radius 0).

Zu c): Anpassungen bei ShortestPath offensichtlich. Im Prinzip funktioniert alles wie bisher, aber der Sichtbarkeitsgraph muss noch erzeugt werden. Dazu muss die Funktion angepasst werden.

Algorithmus 1 : ShortestPath

Input : Hindernismenge S, Punkte $s, t \in \mathbb{R}^2 \setminus \bigcup S$
Output : kürzester kollisionsfreier st-Weg in S

```
1 G_vis ← VisibilityGraph(S ∪ {s, t})
2 foreach uv ∈ E_vis(S) ∪ E_arc(S) do w(uv) ← |uv|
3
4 return Dijkstra(G_vis, w, s, t)
```

Die Funktion VisibilityGraph(S):

Algorithmus 2 : VisibilityGraph

Input : Menge disjunkter Scheiben S
Output : Sichtbarkeitsgraph $G_{\text{vis}}(S)$

```
1 E ← ∅
2 Berechne Menge V(S) //Siehe Aufgabenteil b)
3 foreach v ∈ V(S) do
4 |   W ← VisibleVertices(v, V(S))
5 |   E ← E ∪ {vw | w ∈ W}
6 |   E ← E ∪ { arc zum nächsten Nachbarn auf der Scheibe }
7 return E
```

Die Funktion VisibleVertices funktioniert nach genau dem gleichen Prinzip wie für Polygon-Hindernisse. Wir führen einen 'rotational sweep' durch und verwalten die nächste Scheibe in einem balanciertem binären Suchbaum $\mathcal{T}$. Das klappt, weil von jedem Punkt aus genau zwei Tangenten an einen Kreis gelegt werden können. Da die Kreise sich nicht überlappen kann man einfach die Verbindungslinie zwischen den zwei Schnittpunkten der Tangenten am Kreis als eine Kante eines Polygons auffassen. Dann funktioniert der Algorithmus ohne weitere Modifikation.