4.4 Distanz- und Shortest-Path-Berechung in Straßennetzen

– Von grundlegender Bedeutung für die Anfragebearbeitung auf Straßennetzwerken ist die Berechnung von Distanzen zwischen zwei Knoten
  - Dijkstra: Berechnung der kürzesten Pfade zwischen einem Startknoten und allen verbleibenden Knoten
  - A*-Algorithmus: Berechnung des kürzesten Pfades zwischen einem Startknoten und einem Endknoten

– Die berechneten Distanzen $dist_{net}(v_i, v_j)$ können dann wiederum zur Bearbeitung komplexerer Anfragen auf Straßennetzen verwendet werden
  - Beispiel: Bereichsanfragen und Nächste-Nachbarn-Anfragen
  - $dist_{net}(v_i, v_j)$ im Gegensatz zur euklidischen Distanz nicht unbedingt symmetrisch, d.h. $dist_{net}(v_i, v_j) \neq dist_{net}(v_j, v_i)$
    (z.B. Einbahnstraßen)

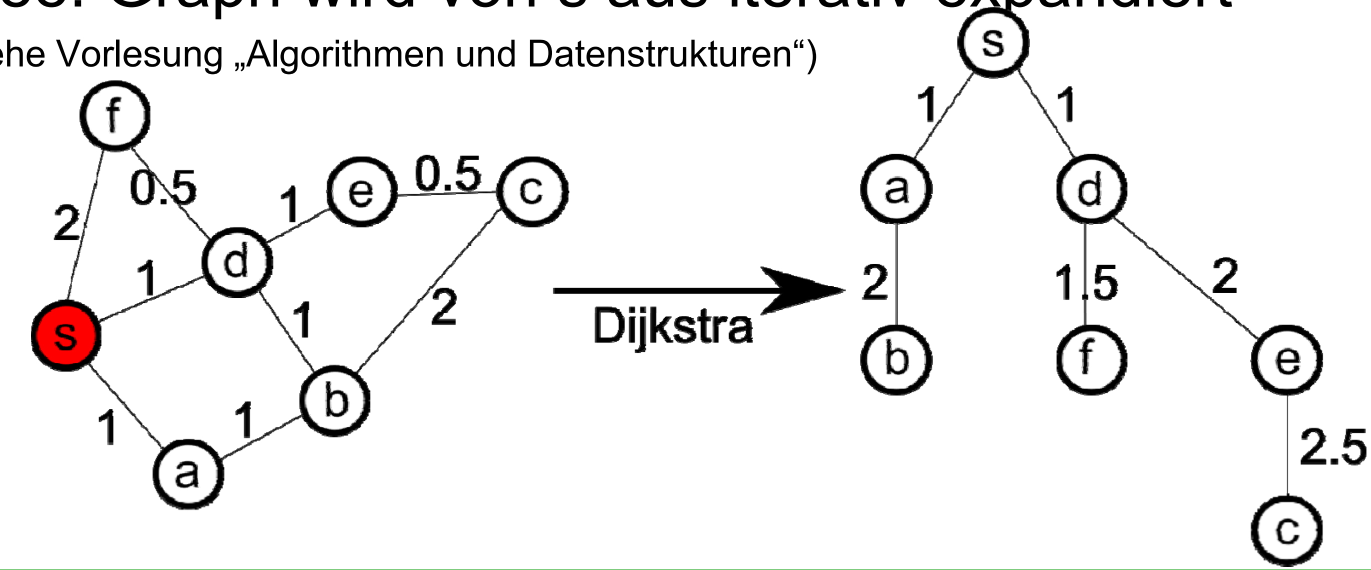
4.3.1 Single-Source Shortest Path – Dijkstra

– Gegeben:
  - (gerichteter) Graph G=(V,E)
  - Kanten haben nicht-negative, reelle Gewichte

– Gesucht:
  - Kürzester Pfad von einem Startknoten s zu allen erreichbaren verbleibenden Knoten

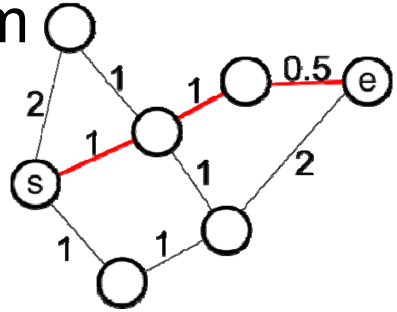
– Idee: Graph wird von s aus iterativ expandiert

(siehe Vorlesung „Algorithmen und Datenstrukturen“)



4.3.2 Der A*-Algorithmus [HNR68]

– Ziel: Bestimme den kürzesten Pfad von einem Startknoten s zu einem Zielknoten e



– Nah verwandt mit dem Dijkstra-Algorithmus

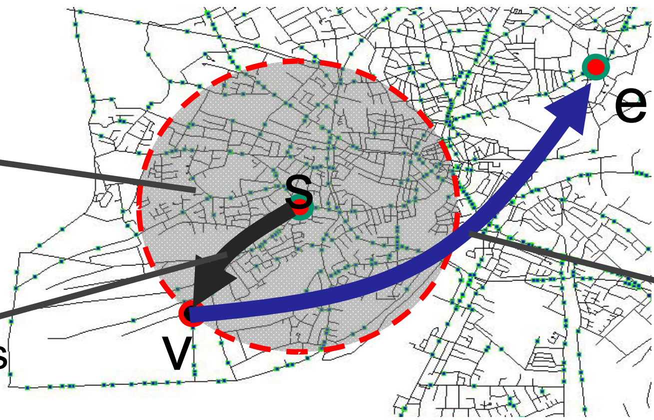
– Idee:

- Verwendung einer Heuristik zur Vorwärtsabschätzung zum Zielknoten e für die Abschätzung der gesamten Pfadkosten während der Expansion:

Netzwerkgraph:

Expandierung des Suchraums

Aktuell erreichte Distanz von Knoten s zu einem Knoten v

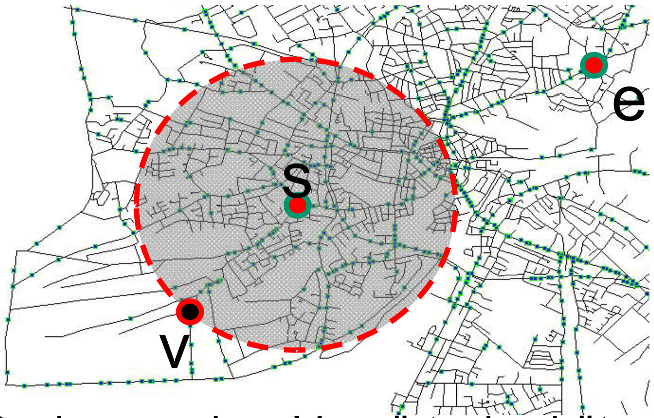


Abgeschätzte Distanz von Knoten v zum Zielknoten e

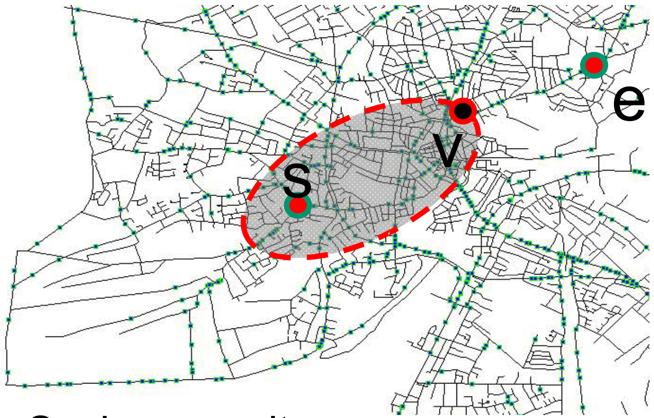
- Best-first-basierte Expansion des Graphen unter Berücksichtigung der Abgeschätzten Gesamtdistanz zwischen s und e, d.h. expandiere nur Knoten v über die der (kürzeste) Pfad von s nach e die Gesamtdistanz optimiert.

=> der Graph wird eher in Richtung des Zielknotens expandiert

⇒ Reduzierung des Suchraums durch vorzeitiges Abschneiden nicht-zielführender Pfade



Suchraum ohne Vorwärtsabschätzung (Dijkstra)



Suchraum mit Vorwärtsabschätzung (A*)

- Korrektheit: Die A*-Suche findet trotz Verwendung einer Heuristik immer den real kürzesten Pfad von s nach e

– A*-Algorithmus:
  - Starte bei Startknoten s
  - Expandiere von s aus den Graphen wie bei Dijkstra iterativ
  - Verwendung eines Heaps zur Verwaltung der **relevantesten** Knoten für die weitere Expansion.
  - Bei der Expansion von s werden die Nachbarknoten von s zunächst in den Heap eingefügt.
  - In dem nächsten Iterationsschritt wird das erste Element aus dem Heap geholt und expandiert, dabei werden nur Pfade verfolgt (zugriff auf Nachbarknoten), die die Gesamtkosten von s nach e minimieren.
  - Die Relevanz f(v) eines Knotens v ist bestimmt durch die geschätzte Länge des Weges von s nach e über v: $f(v) = g(v) + h(v)$
    - f(v) – Kosten von s über v nach e
    - g(v) – Bisher ermittelte minimale Kosten von s nach v
    - h(v) – Schätzung der Kosten von v nach e, z.B. L2-Distanz

```
function A*(s,e)
    closedset = Ø    // Bereits betrachtete Knoten
    g[s] = 0    // Kosten von s nach Knoten x über den besten bekannten Pfad
    h[s] = heuristicCostEstimate(s, e)
    openheap = {(s, g[s] + h[s])}    // Zu betrachtende Knoten
    cameFrom = Ø    // Zur Rekonstruktion des kürzesten Pfades

    while openheap≠Ø
        (x, f) = openheap.poll();
        if x == e
            rekonstruiere Pfad über cameFrom und gib Ergebnispfad zurück
        closedset.add(x);
        foreach y ϵ neighbors(x)
            if y ϵ closedset:
                continue
            gNew := g[x] + dist(x,y)
            if (gNew < g[y])
                newPathIsBetter := true
            else
                newPathIsBetter := false
            if newPathIsBetter
                cameFrom[y] := x
                g[y] := tentativeGScore
                h[y] := heuristicCostEstimate(y, goal)
                lösche ggf. altes y aus openheap und füge (y, g[y] + h[y]) in openheap ein
return null //kein Pfad gefunden
```

– Eigenschaften der Schätzfunktion h(v)

- Die Korrektheit des A*-Algorithmus kann nur garantiert werden, wenn die Schätzfunktion h(v) die Kosten für die Pfad von v zum Zielknoten unterschätzt. (WICHTIG !!!)
- Damit gilt: $f(v) \leq dist_{net}(s,v)+dist_{net}(v,e)$.
- Je besser die Schätzung von h(v) (d.h. je größer h(v)), desto weniger Knoten werden besucht.
- Bei Verwendung von $h(v) = 0$, $\forall v \in V$, verhält sich A* wie Dijkstra.

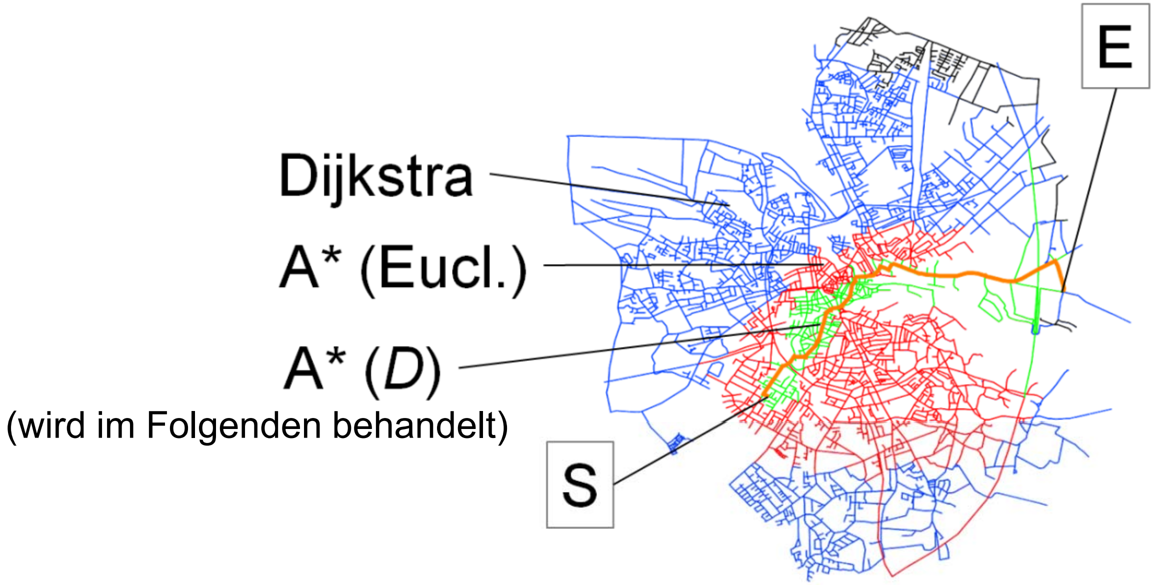
Heuristiken für A: Setze h(v) = 0*

- Die zurückzulegende Distanz wird immer auf 0 gesetzt.
- Dadurch wird der Heap der zu besuchenden Knoten nur anhand der bereits zurückgelegten Strecke sortiert.
- Das entspricht dem Dijkstra-Algorithmus.
- Der Suchraum wird dadurch sehr groß.

Heuristiken für A: Setzte $h(v) = dist_{L2}(v,e)$*

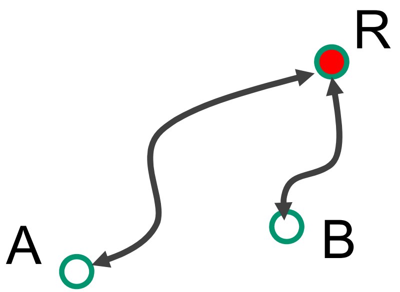
- Verwendet stets die geringste praktisch mögliche Distanz.
- Reale Distanz ist normalerweise weit größer als euklidische Distanz.
- Kleinerer Suchraum als bei Dijkstra, aber immer noch sub-optimale Schätzung.

– Vergleich der Heuristiken:

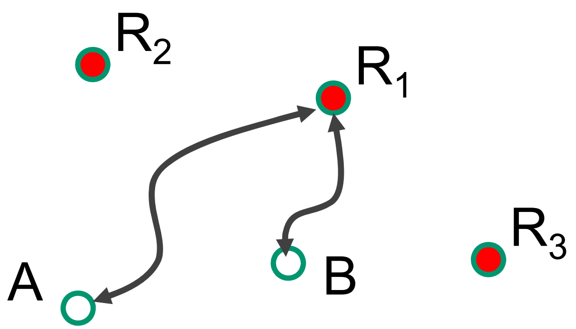


Heuristiken für A: Graph Embedding (D-Distanz)* [KKKRS08]

- Idee:
  - Berechne Distanzen zu einem/oder mehreren Referenzknoten vor.
  - Verwendung dieser Distanzen mit Referenzknoten zur Abschätzung der Distanz zweier beliebiger Knoten



$dist_{net}(A,B) \geq |dist_{net}(A,R) - dist_{net}(B,R)|$

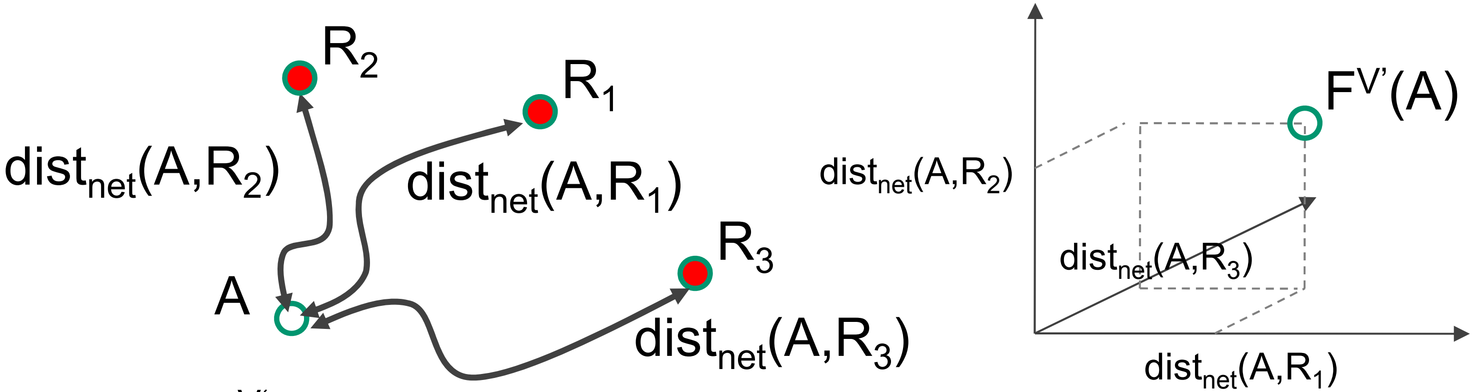


Bei k Refernzknoten $R_{1,..,k}$:
$dist_{net}(A,B) \geq max_{i=0..k}(|dist_{net}(A,R_i) - dist_{net}(B,R_i)|)$

Führt zur besseren Abschätzung als mit nur einem Refernzknoten.

- Wähle eine Teilmenge von Knoten $V' \subseteq V$, $|V'| = k \geq 1$
- Definiere eine Funktion $F^{V'} : V \rightarrow \mathbb{R}^k$
- Für das Reference Node Embedding definiere $F^{V'}$ folgendermaßen:

$$F^{V'}(v) = (F_1^{V'}(v), \ldots, F_k^{V'}(v))$$
$$F_i^{V'}(v) = dist_{net}(v, v_i) \text{ (Preprocessing!)}$$



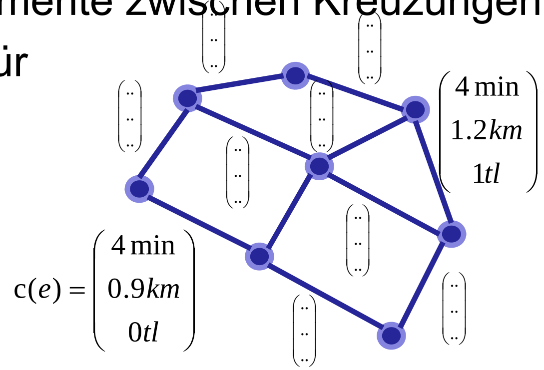
- $F^{V'}(v)$ kann direkt zur Berechnung von $h(v)$ verwendet werden, denn es gilt $\forall\, v_i, v_j \in V$: $dist_{L_\infty}(F^{V'}(v_i), F^{V'}(v_j)) \leq dist_{net}(v_i, v_j)$.
- Geringerer Suchraum als bei Verwendung der euklidischen Distanz.

4.5 Routen-Suche in Multiattributs-Verkehrsnetzwerken

- Gegeben:
  - Verkehrsnetzwerk
  - Straßensegmente haben mehrere Kosten-Attribute, wie z.B.:

    Weglänge, Fahrzeit, Anzahl der Ampeln, Benzinverbrauch (abhängig vom Fahrzeugtyp), …
  - Graph G(V,E,c):
    - V: Menge von Knoten <-> Kreuzungen, Sackgassen
    - $E \in V \times V$: Menge von Kanten <-> Strassensegmente zwischen Kreuzungen
    - c: $E \rightarrow IR^d$: Kostenfunktion <-> Kostenvektor für Kanten mit d Kostenattributen
  - Annahmen:
    - Keine negativen Kosten (Warum?)
    - Alle Kostenattribute sind additiv

      (Gilt für Kosten eines Pfades (Route))



- Route:
  - p = <$v_1,\ldots,v_n$> mit $(v_i,v_{i+1}) \in E$, $\forall 1 \le i < n$
  - ohne Zyklus, d.h. $\forall 1 \le i < n, \forall 1 \le j < n$, $i \ne j$, $v_i \ne v_j$
  - Kosten einer Route: $\mathrm{cost}(\mathrm{p}) = \sum_{e \in \mathrm{p}} c(e) = \begin{pmatrix} \mathrm{cost}(p)_1 \\ \vdots \\ \mathrm{cost}(p)_d \end{pmatrix}$

– Gesucht:

- “Bester” Pfad von S nach Z

Zielknoten Z

Startknoten S

:= Pfad mit minimalen Kosten $\sum_{i=1}^{d} w_i \cdot \mathrm{cost}(p)_i$?

– Problem

- Mehrere (Pfad-)Attribute (Kostenattribute) zu berücksichtigen (i.d.R. eine Auswahl an Kostenattribute)

Zielknoten Z

Startknoten S

Route A
7 km, 5 Minuten

Route B
6 km, 7 Minuten

Welcher Pfad ist optimal A oder B?

- Kostenattribute schwierig zu vergleichen:
  #Ampeln ←→Strecke?
- Präferenzen sind abhängig vom Benutzer und Anwendung:
  Geschäftlich oder privat unterwegs?
- Geeignete Gewichtung der Attribute schwer zu ermitteln

– Ansatz:

  Ausgabe aller pareto-optimaler Pfade → Route Skyline

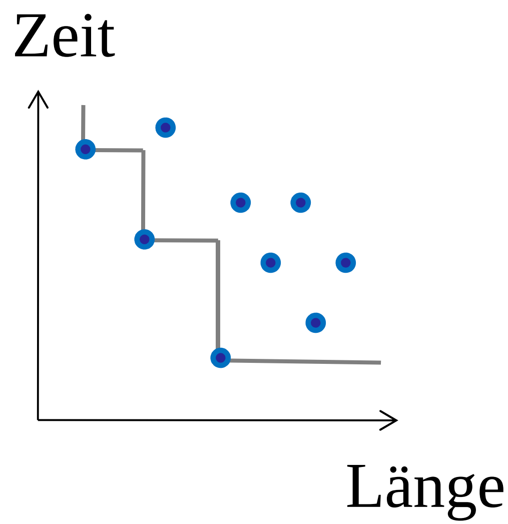
– Anfrage:

  - Finde die Menge RS alle Pfade zwischen S und Z, sodaß für jeden Pfad p∈RS kein Pfad p' zwischen S und Z existiert mit der folgenden Eigenschaft:

    $$\forall 1 \leq i \leq d : \text{cost}(p')_i \geq \text{cost}(p)_i$$

    $$\exists 1 \leq i \leq d : \text{cost}(p')_i > \text{cost}(p)_i$$

  - Man spricht: p wird nicht von p' dominiert

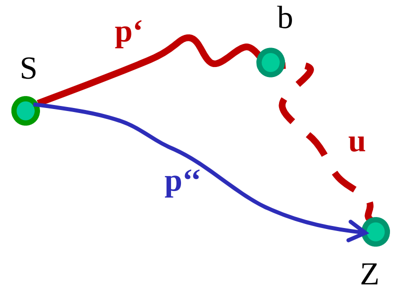


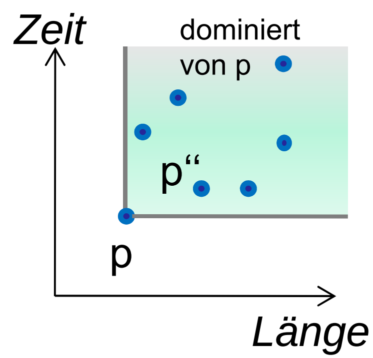
– Probleme:

  - Anzahl potentieller Pfade extrem hoch
  - Materialisierung der Kosten aller Pfade extrem teuer

  => Skylineanfragemethoden für Vektorobjekte nicht (effizient) anwendbar

- Effiziente Route-Skyline Berechnung: [KRS 10]
- Idee:
  - Graphdurchlauf bei Startknoten S starten analog zu Dijkstra/A*-Suche
  - Erweitere eine Route p’ falls p’ Teilroute einer Skylineroute p sein kann, d.h. erweitere p’ nicht wenn abgeschätzt werden kann dass p von einer anderen Route p’’ dominiert wird
  - Stoppe falls keine Route mehr erweitert werden kann
- Gegeben eine Teilroute p’=<S,…,b>,

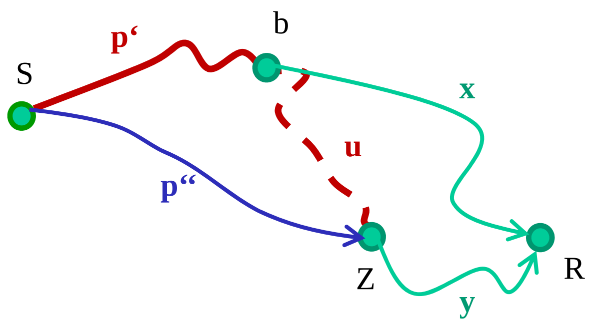




- Wie kann man abschätzen ob p’ Teil einer Skylineroute ist? => Pruningkriterien

– Pruningkriterium I: Pruning mittels Vorwärtsabschätzung

- Basiert auf Kostenabschätzung mittels Referenzknoten (siehe Kap. zu Graph Embedding)
- Schätze Gesamtkosten (Kostenvektor) cost(p) einer potentiellen Route (über einen Referenzknoten R) ab
- Falls bereits ein Pfad p'' existiert, der den Vektor cost(p) dominiert, dann kann p' nicht zu einer Skylineroute erweitert werden



$\text{cost}(u)_i \geq |\text{cost}(x)_i - \text{cost}(y)_i|$

- Funktioniert solange die Kostenabschätzung die untere Schrankeneigenschaft erfüllt

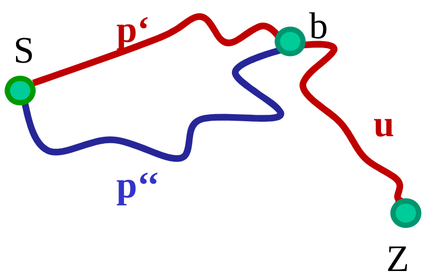
– Pruningkriterium II: Pruning unter Verwendung der Monotonieeigenschaft der Dominanzbeziehung

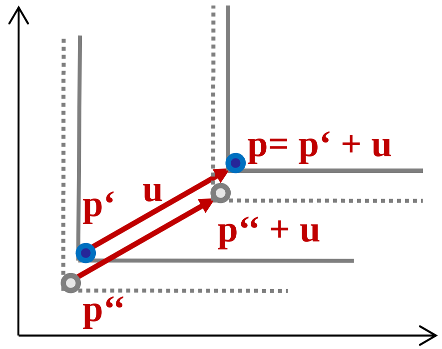
Theorem:

Gegeben sei eine Skyline Route p=<S,…,b,…,Z>, d.h. p wird von keiner anderen Route zwischen S umd Z dominiert. Dann wird jede Teilroute p'=<S,…,b> von p von keiner anderen Teilroute p''=<S,…,b> dominiert.

Beweis:

Annahme es gibt eine Route p''=<S,…,b>, die die Teilroute p'=<S,…,b>≠p'' von p dominiert. Dann würde die Erweiterung von p'' über die Teilroute u=<b,…,Z> von p zu einer Route führen, die die Route p dominiert => Wiederspruch zur Annahme daß p Skylineroute ist.





– Skyline-Route-Algorithmus:

- Idee: wie A*-Suche, aber verwalte bei jedem Knoten n eine Skyline (Skyline bzgl. aller Pfade p’=<S,…,n>)
- Algorithmus:

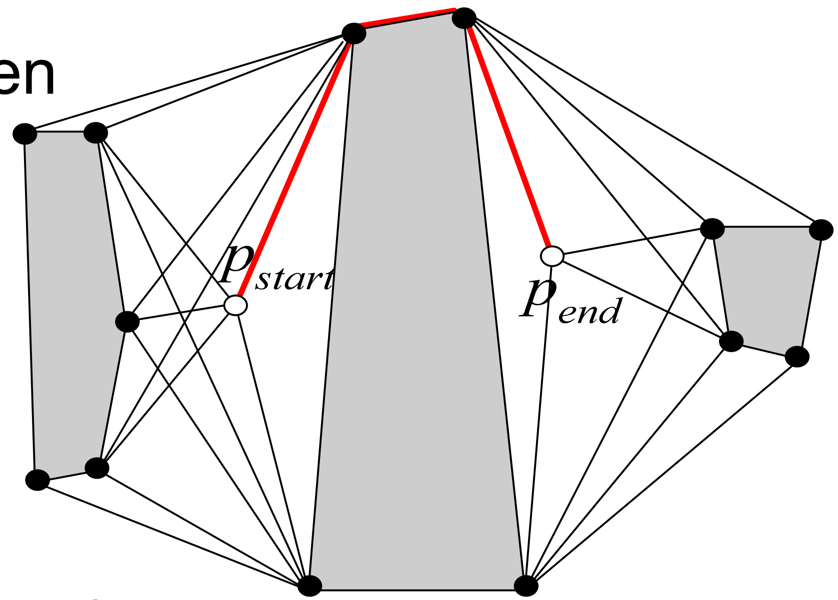
Input: Start S, Ziel Z, Graph(V,E,L) (mit Embedding)

Output: alle Skyline-Routen zwischen S und D

```
initialisiere Routen-Heap (queue);
queue.insert(S);
while (queue is not empty)
    aktRoute = queue.top();
     // erweitere alle Skyline Routen von Knoten aktNode
    cand = extend(aktRoute.last_node.SkR);
    for all c∈cand
      if ∃p∈Z.SkR, p dominiert cost_est(c), then lösche c aus cand;
       else
                    update(c.last_node.SkylineRoutes,c);
                    queue.insert(c);
report(Z.SkR);
```

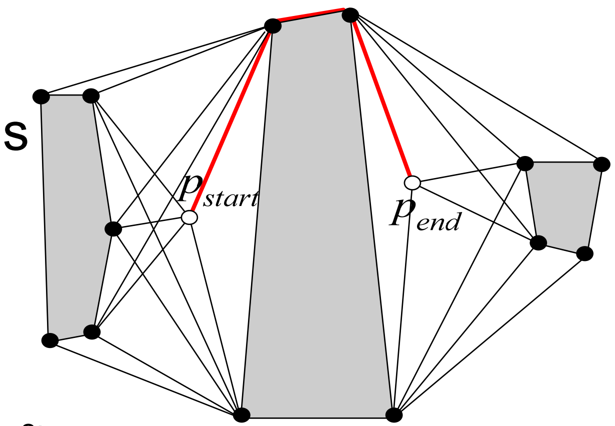
4.6 Anfragen im Euklidischen Raum mit Hindernissen [ZPMZ04]

- Ziel: Anfragebearbeitung (z.B. NN-Anfragen) im euklidischen Raum bei Anwesenheit von Hindernissen (Häuser, Seen, …)
- Mögliche Pfade (mit kürzesten Distanzen) können mittels Sichtbarkeitsgraphen modelliert werden.
- Für die Ermittlung der kürzesten Distanz zwischen zwei Punkten können Graphalgorithmen (z.B. Dijkstra) verwendet werden.



4.6.1 Sichtbarkeitsgraph

– Dient der Modellierung der kürzesten Pfade zwischen zwei beliebigen Punkten im Raum mit Hindernissen
– Hindernisse werden als Polygone modelliert
– Knoten: Eckpunkte von Hindernissen, Objekte (p_i)
  Gilt nur im 2-dimensionalen Raum !!!
– Kanten: Verbinden Knoten die gegenseitig „sichtbar“ sind (d.h. dürfen kein Hindernis kreuzen)
– Ermittlung des Sichtbarkeitsgraphen:
  - Suche für jeden Eckpunkt eines Polygons die Eckpunkte anderer Polygone die direkt „sichtbar“ sind.
    (siehe Computational Geometry [BKOS97])
    – Naive Suche: $O(n^3)$
    – Rotational plane-sweep: $O(n^2 \log n)$ [SS84], $O(n^2)$ [W85, AGHI86]

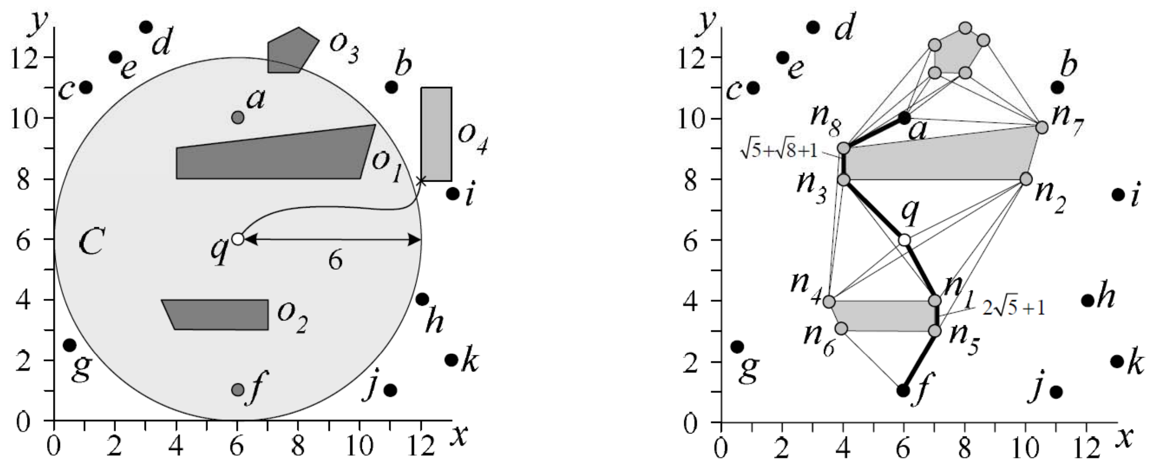


4.6.2 Idee bei Nachbarschaftsanfragen in Räumen mit Hindernis

- Berechnung des gesamten Sichtbarkeitsgraphen zu aufwändig.
- Deshalb Reduktion auf relevante Teilgraphen, bei denen nur relevante Hindernisse betrachtet werden.
- Welche Hindernisse sind für die Anfragebearbeitung relevant?
- Euklidische Distanz zwischen zwei Punkten bildet eine untere Schranke für die Distanz in Räumen mit Hindernissen.
- Ermittlung der relevanten Hindernisse über die Verwendung der Euklidischen Distanz (Filterschritt)

4.6.2 Bereichsanfrage mit Hindernissen

1. Euklidische Bereichsanfrage auf Ergebniskandidaten P von Anfragepunkt q mit Distanz e (filter)
2. Suche von für die Anfrage relevanten Hindernissen O. Diese schneiden die Fläche der Bereichsanfrage.
3. Lokalen Sichtbarkeitsgraph über P und O aufbauen
4. False Hits aus P über Sichtbarkeitsgraph entfernen



(a) Obstacle range query

(b) Local visibility graph

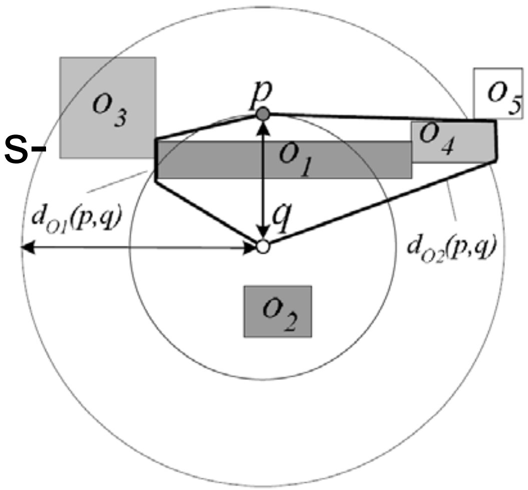
4.6.2 NN-Anfrage mit Hindernissen

– Entspricht dem IER-Algorithmus, wobei jede Runde der entsprechende Sichtbarkeitsgraph erzeugt werden muss, um $dist_{NET}(p,q)$ zu berechnen.

– Initialisiere NN-Rankinganfrage über die Euklid. Dist.

– Berechnung der Netzwerkdistanz $dist_{NET}(p,q)$ für jeden NN-Kandidaten p über den Sichtbarkeitsgraphen:

1. Berechne Hindernisse O, die [p,q] schneiden
2. Berechne den Sichtbarkeitsgraph über O ∪ {p,q} und daraus $dist'_{NET}(p,q)$
3. Frage alle Hindernisse O an, die die Bereichs-anfrage von q bzgl. $dist'_{NET}(p,q)$ schneiden
4. Führe 3. und 4. so lange durch, bis O oder $dist'_{NET}(p,q)$ sich nicht mehr verändert. Es gilt: $dist_{NET}(p,q) = dist'_{NET}(p,q)$



[HNR68]: Hart, Nilsson, Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In IEEE Transactions of Systems Science and Cybernetics, 1968.

[KRS10]: Kriegel, Renz, Schubert: Route Skyline Queries: A Multi-Preference Path Planning Approach. In Proc. of ICDE, 2010.

[KKKRS08]: Kriegel, Kröger, Kunath, Renz, Schmidt: Efficient Query Processing in Large Traffic Networks. In Proc. of ICDE, 2008.

[PZMT03]: Papadias, Zhang, Mamoulis, Tao. Query Processing in Spatial Network Databases. In Proc. of VLDB, 2003.

[BKOS97] de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O. Computational Geometry. pp. 305-315, Springer, 1997.

[SS84] Sharir, M., Schorr, A. On Shortest Paths in Polyhedral Spaces. STOC, 1984.

[W85] Welzl, E. Constructing the Visibility Graph for n Line Segments in O(n2) Time, Information Processing Letters 20, 167-171, 1985.

[AGHI86] Asano, T., Guibas, L., Hershberger, J., Imai, H. Visibility of Disjoint Polygons. Algorithmica 1, 49-63, 1986.

[ZPMZ04]: Zhang, Papadias, Mouratidis, Zhou. Spatial Queries in the Presence of Obstacles. In Proc. of EDBT, 2004.