Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Felix Kuhr (Übungen)

sowie viele Tutoren

Einschub: Auswertung von Ausdrücken

dict := <(1, “eins”), (5, “fünf”), (42, “zweiundvierzig”)>:Dictionary

x := 2

elem1 := (x, “eins”)

elem2 := (5, “fünf”)

first(elem1) := 1

dict := <elem1, elem2, (42, “zweiundvierzig”)>:Dictionary

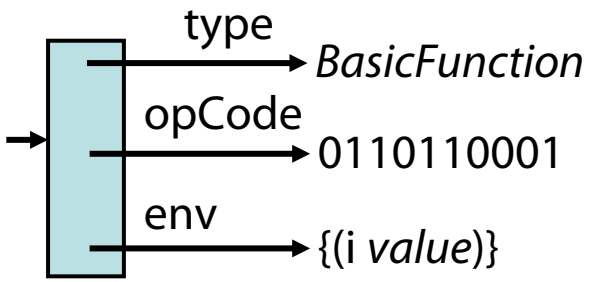
Funktionen

Können wir auch Funktionen als Konstanten “hinschreiben”?

compatible := lambda(x, y) return 4+first(x) = first(y)

// Warum lambda? Historische Gründe – warum auch nicht?

compatible(elem1, elem2) → true



lambda(x, y) 4+first(x) = first(y) (elem1, elem2)

for i from 1 to 10 do

pq := <(i, “i”), (i+1, “j”)>:PQ with key as lambda((a, _)) a + i

…

// return am Ende der Auswertung optional

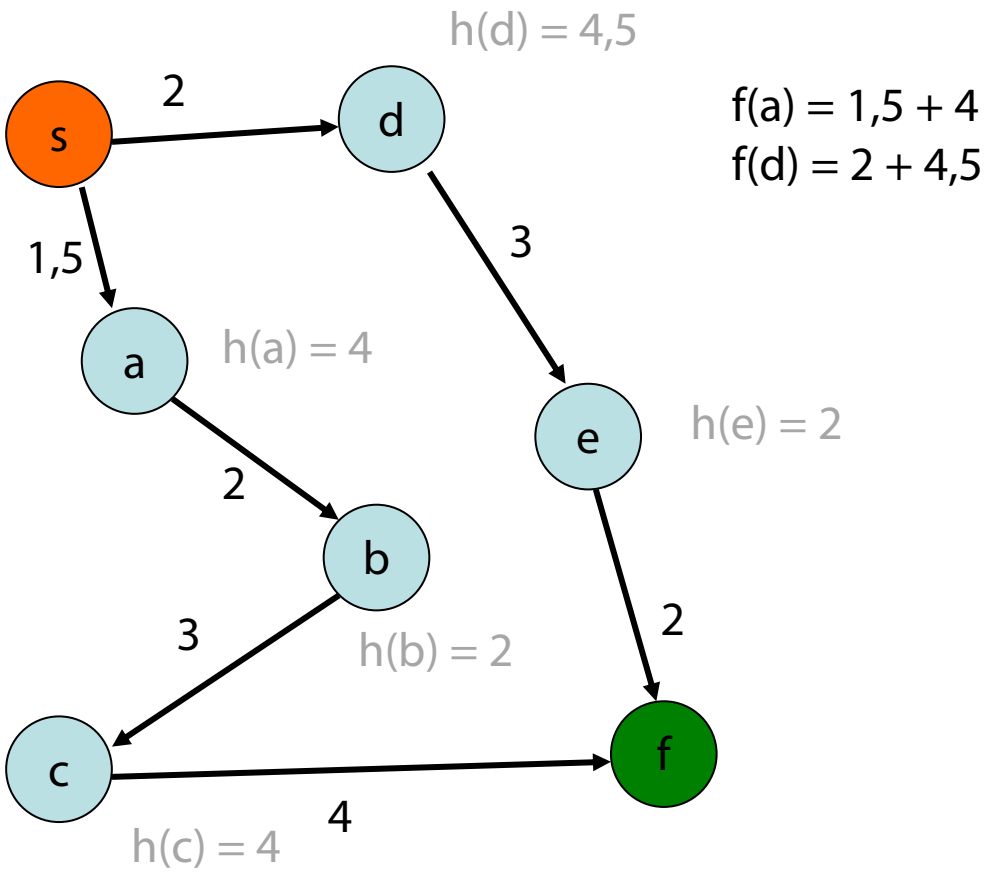
// Lexikalische Bindung

Kürzeste Wege: Heuristische Suche

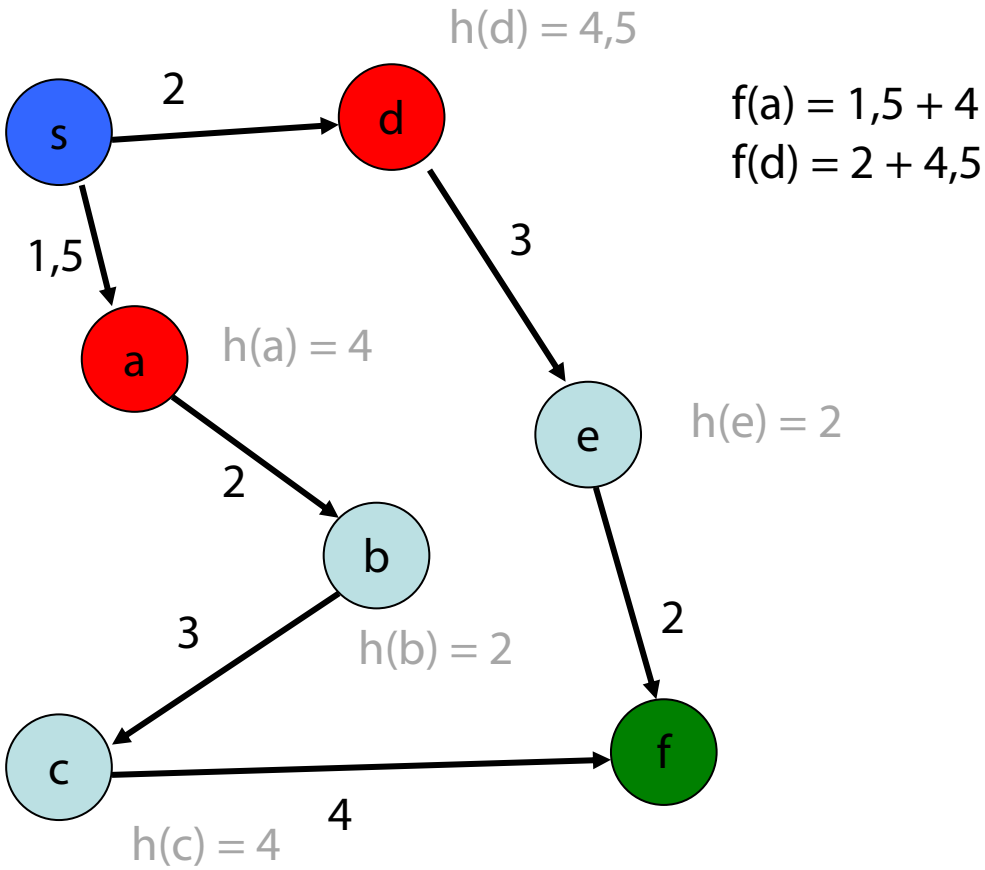
- Was sollten wir machen, wenn nur der kürzeste Weg zu einem gegebenen Knoten gesucht ist?
  - Es werden im Dijkstra-Algorithmus zu viele Knoten betrachtet (d.h. "expandiert")
  - Weiterhin: Keine Abschätzung der Entfernung zum Ziel
- Abhilfe: A*-Algorithmus
  - Informierte Suche mit Zielschätzer (Heuristik)

P. E. Hart, N. J. Nilsson, B. Raphael: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Transactions on Systems Science and Cybernetics SSC4 (2), pp. 100–107, **1968**
P. E. Hart, N. J. Nilsson, B. Raphael: Correction to „A Formal Basis for the Heuristic Determination of Minimum Cost Paths“, SIGART Newsletter, 37, pp. 28–29, **1972**

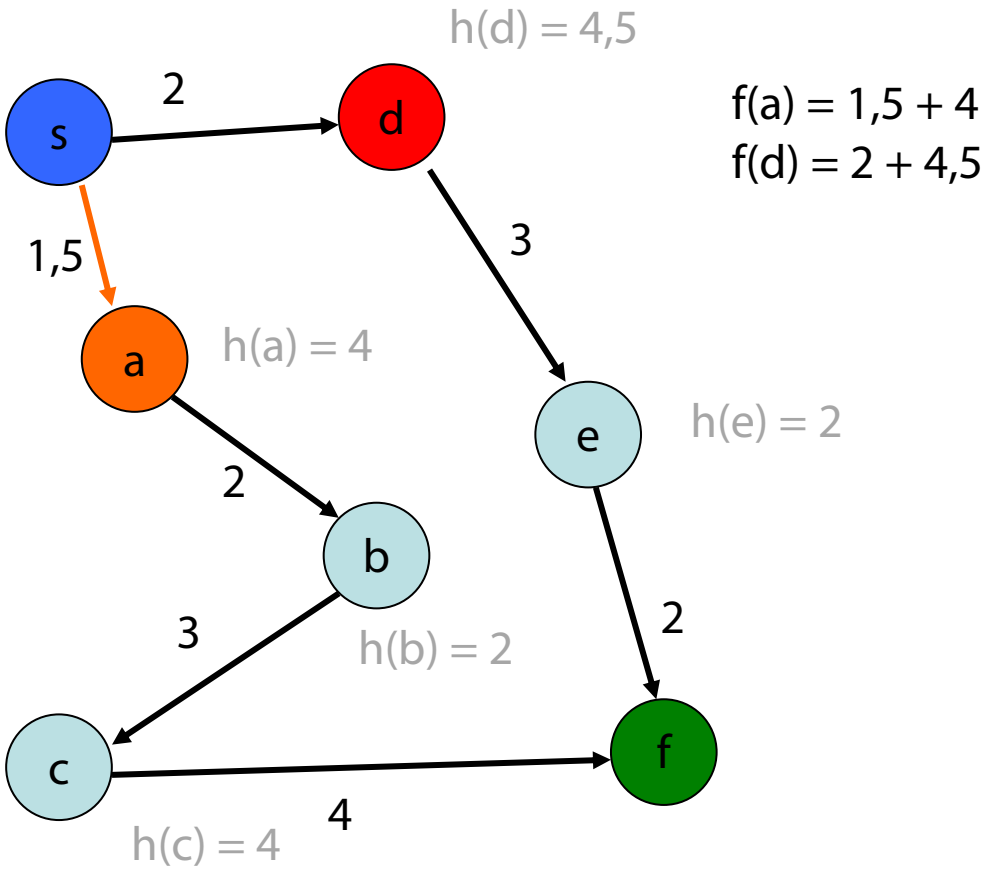
A* – Beispiel



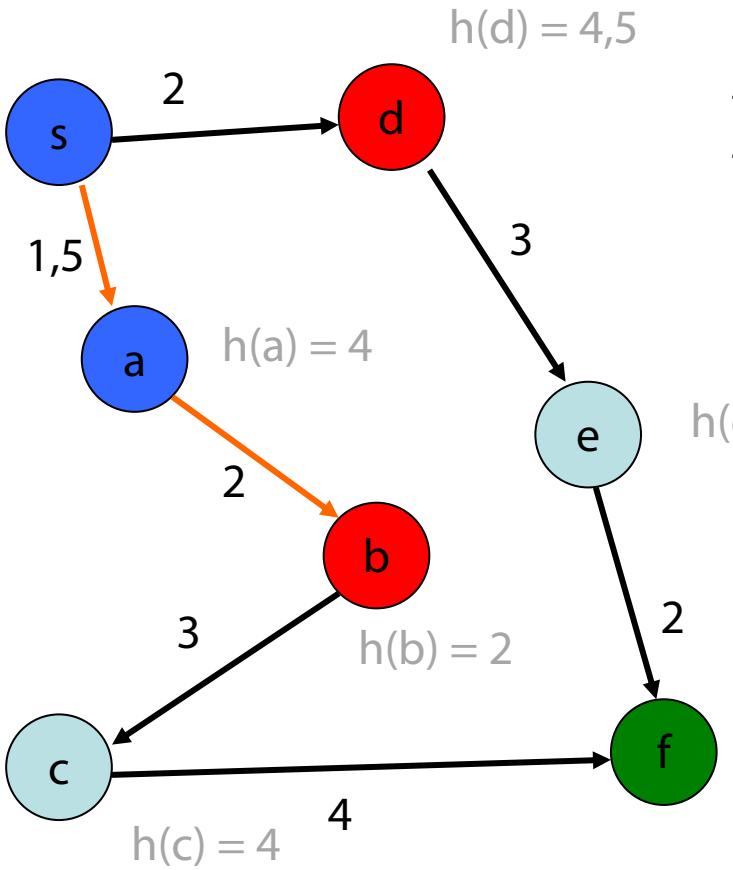
A* – Beispiel



A* – Beispiel

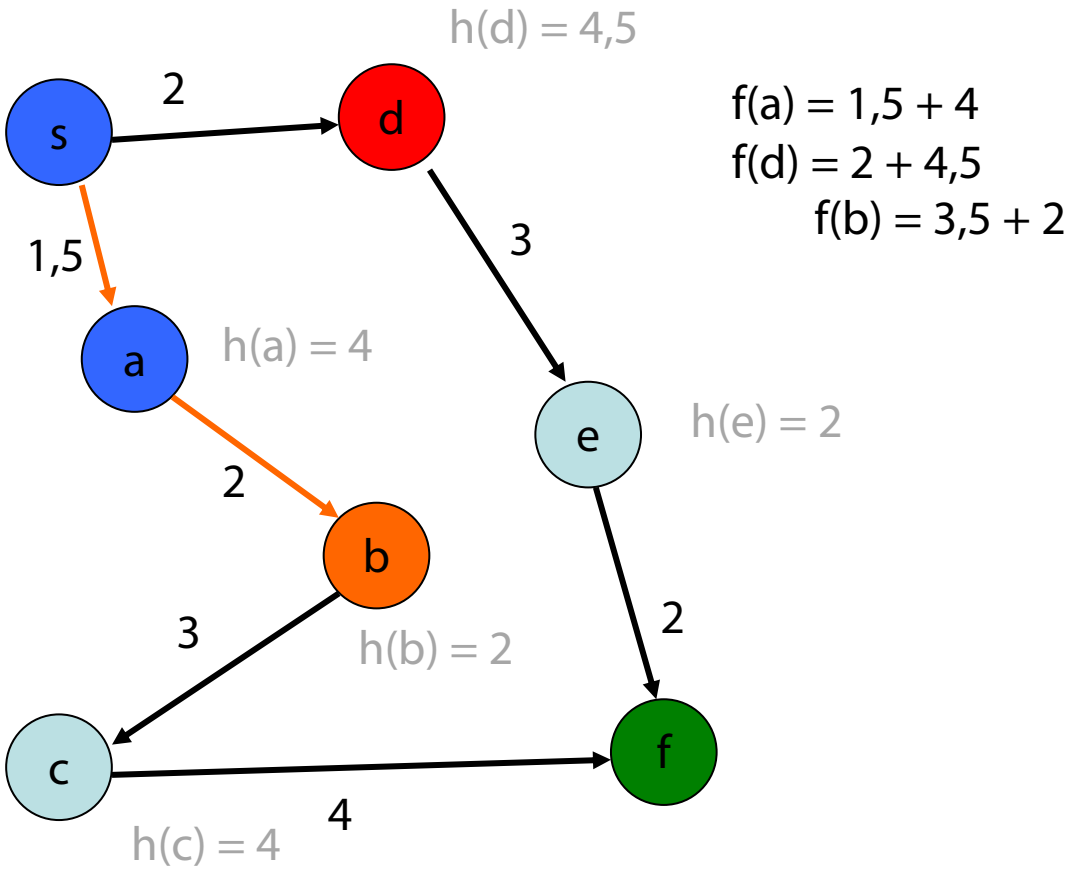


A* – Beispiel

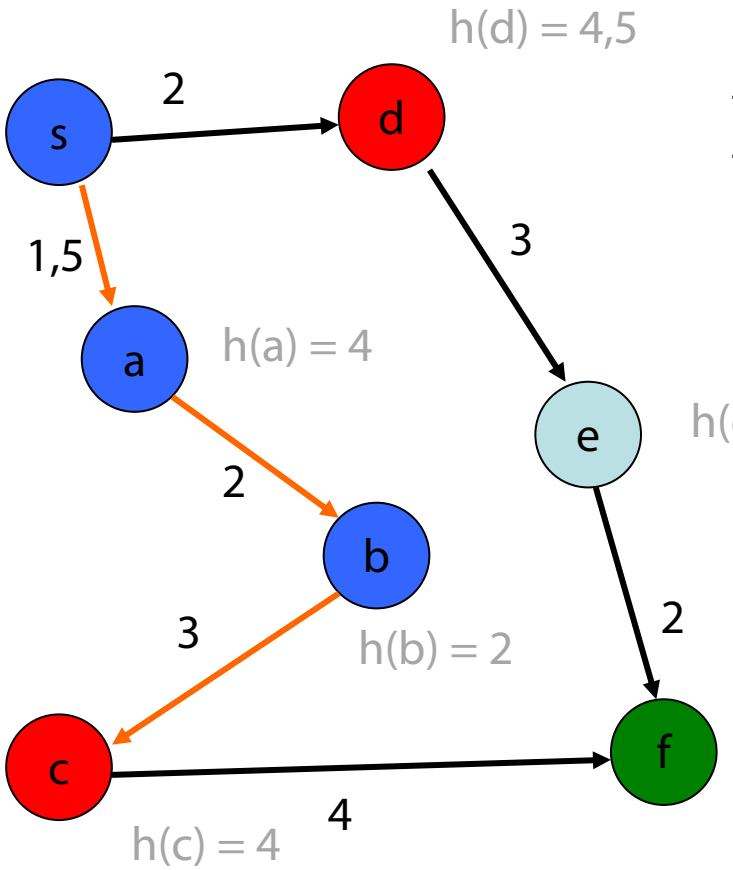


f(a) = 1,5 + 4
f(d) = 2 + 4,5
f(b) = 3,5 + 2

A* – Beispiel

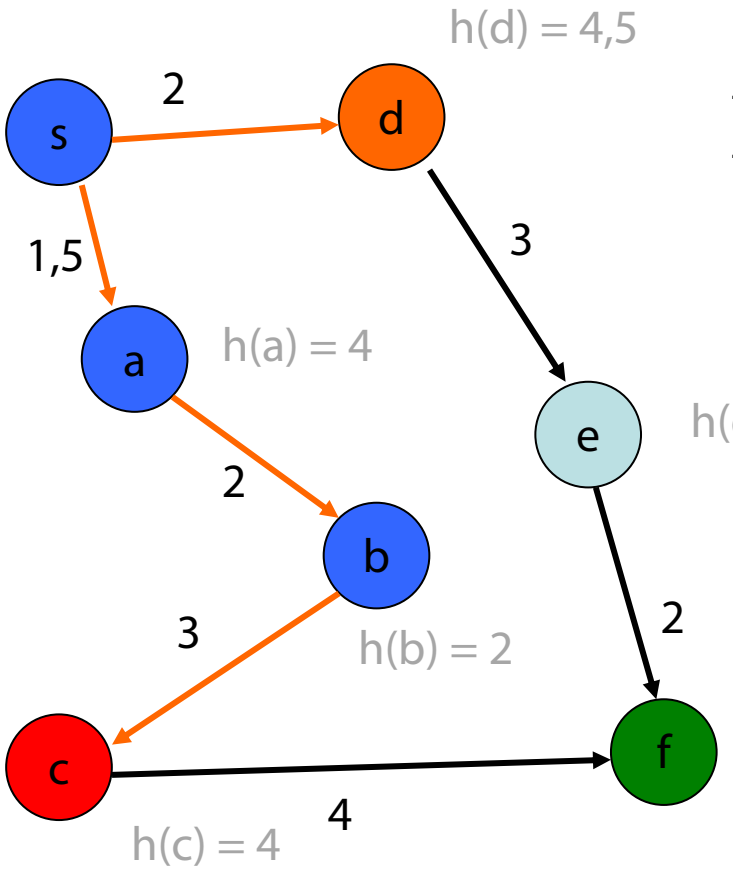


A* – Beispiel



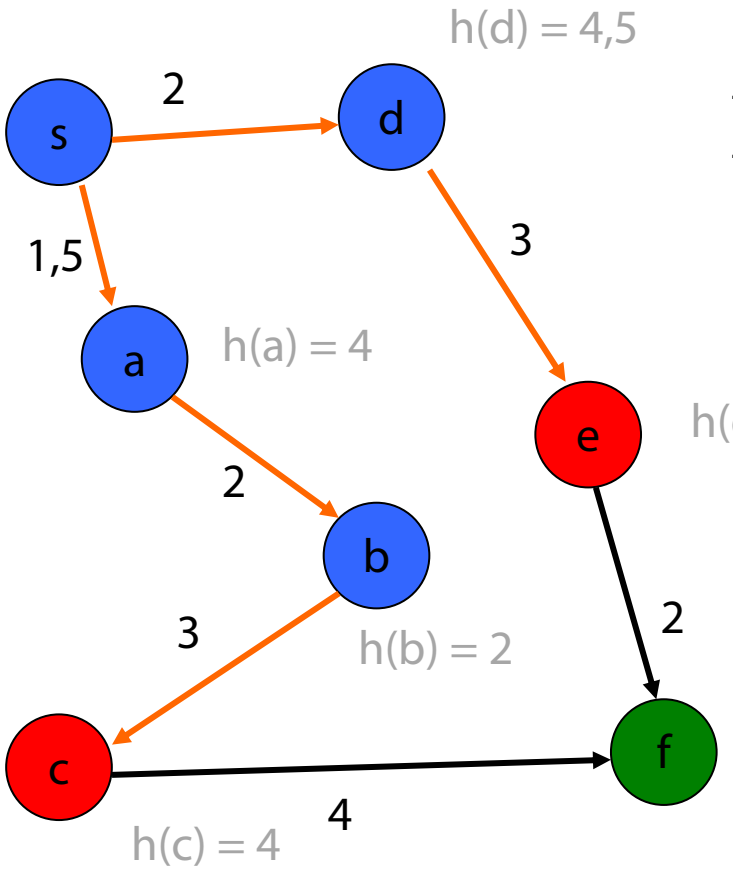
f(a) = 1,5 + 4
f(d) = 2 + 4,5
f(b) = 3,5 + 2
f(c) = 6,5 + 4

A* – Beispiel



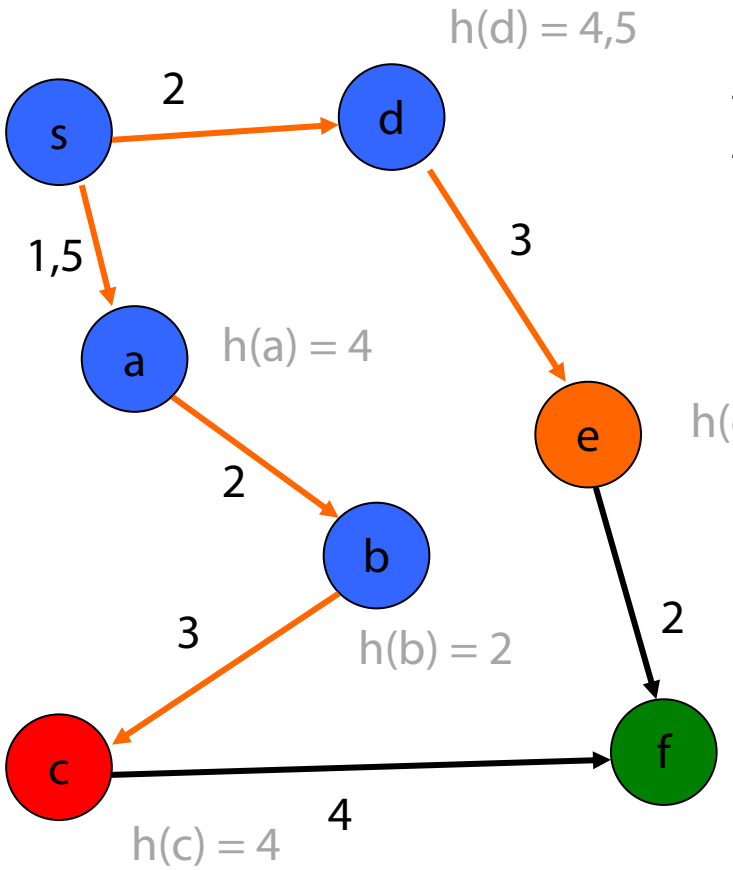
f(a) = 1,5 + 4
f(d) = 2 + 4,5
f(b) = 3,5 + 2
f(c) = 6,5 + 4

A* – Beispiel



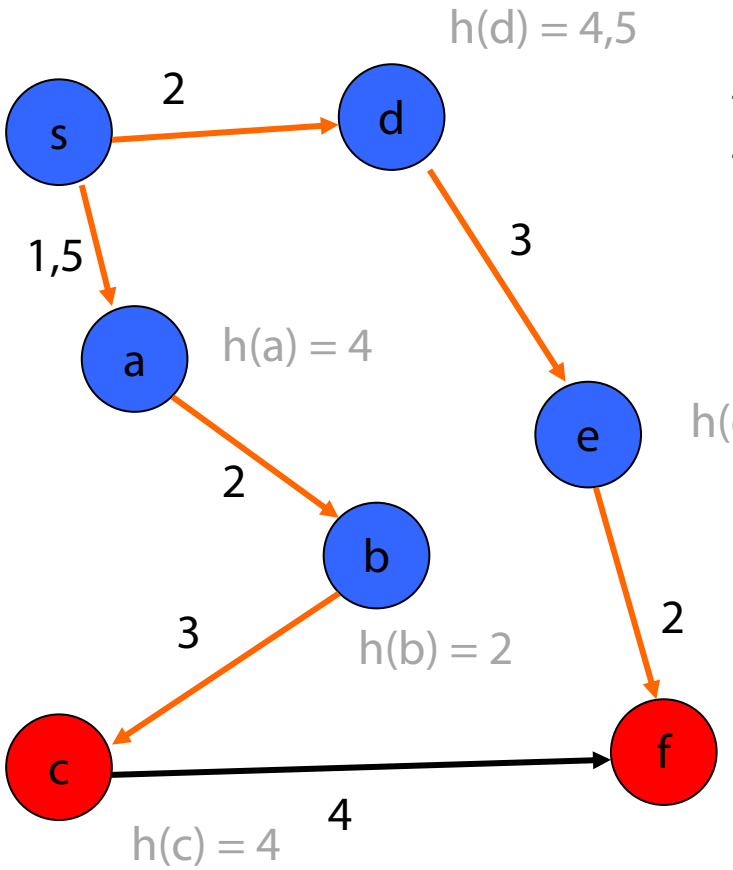
f(a) = 1,5 + 4
f(d) = 2 + 4,5
f(b) = 3,5 + 2
f(c) = 6,5 + 4
f(e) = 5 + 2

A* – Beispiel



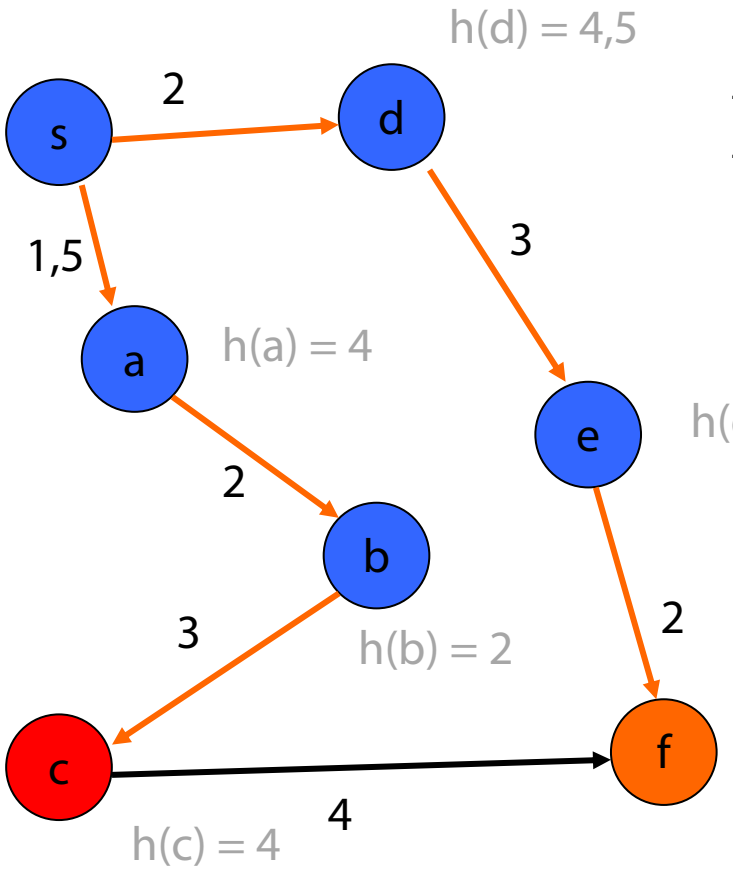
f(a) = 1,5 + 4
f(d) = 2 + 4,5
f(b) = 3,5 + 2
f(c) = 6,5 + 4
f(e) = 5 + 2

A* – Beispiel



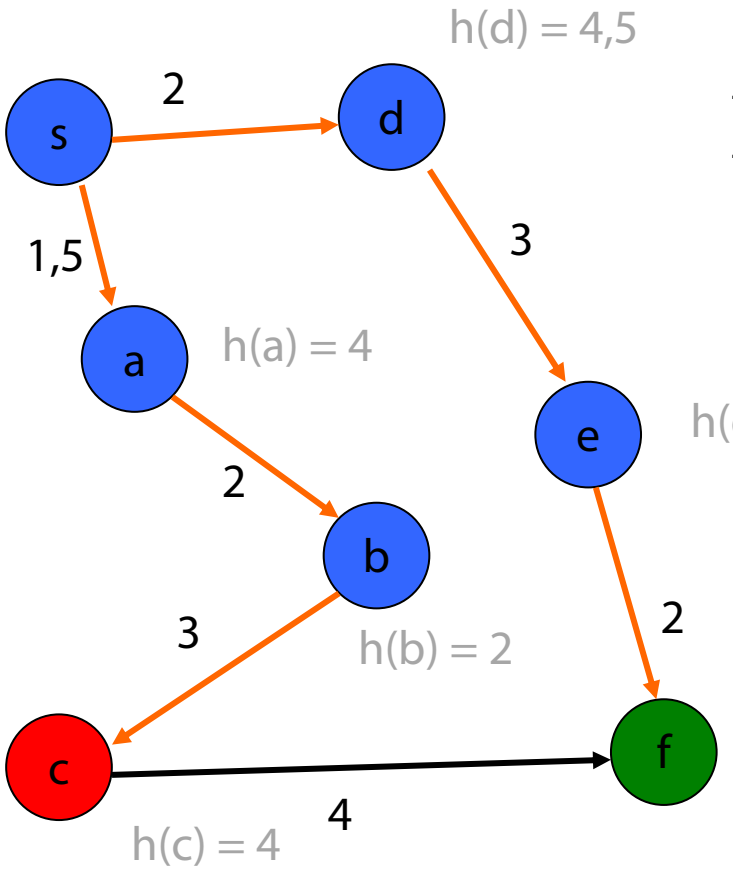
f(a) = 1,5 + 4
f(d) = 2 + 4,5
f(b) = 3,5 + 2
f(c) = 6,5 + 4
f(e) = 5 + 2

A* – Beispiel



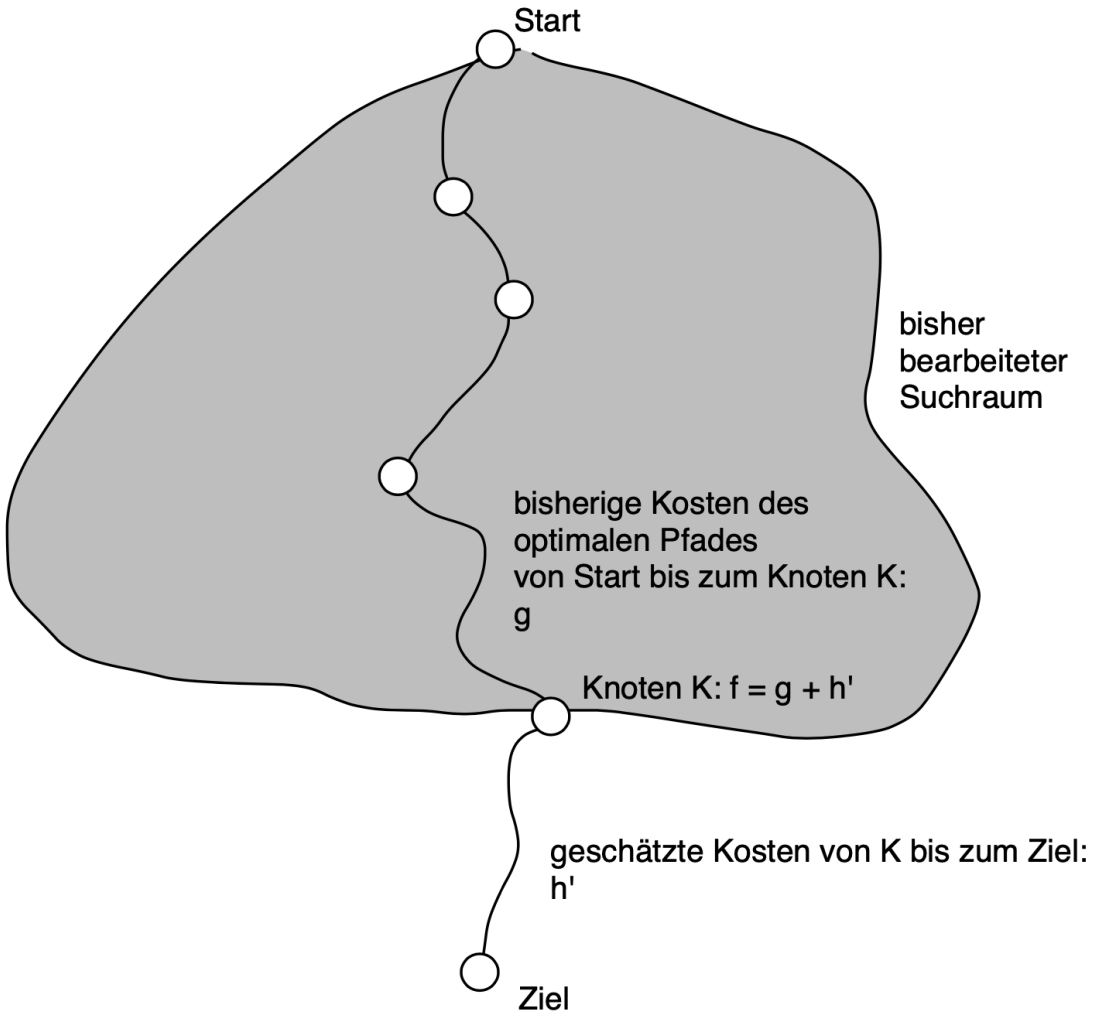
f(a) = 1,5 + 4
f(d) = 2 + 4,5
f(b) = 3,5 + 2
f(c) = 6,5 + 4
f(e) = 5 + 2

A* – Beispiel

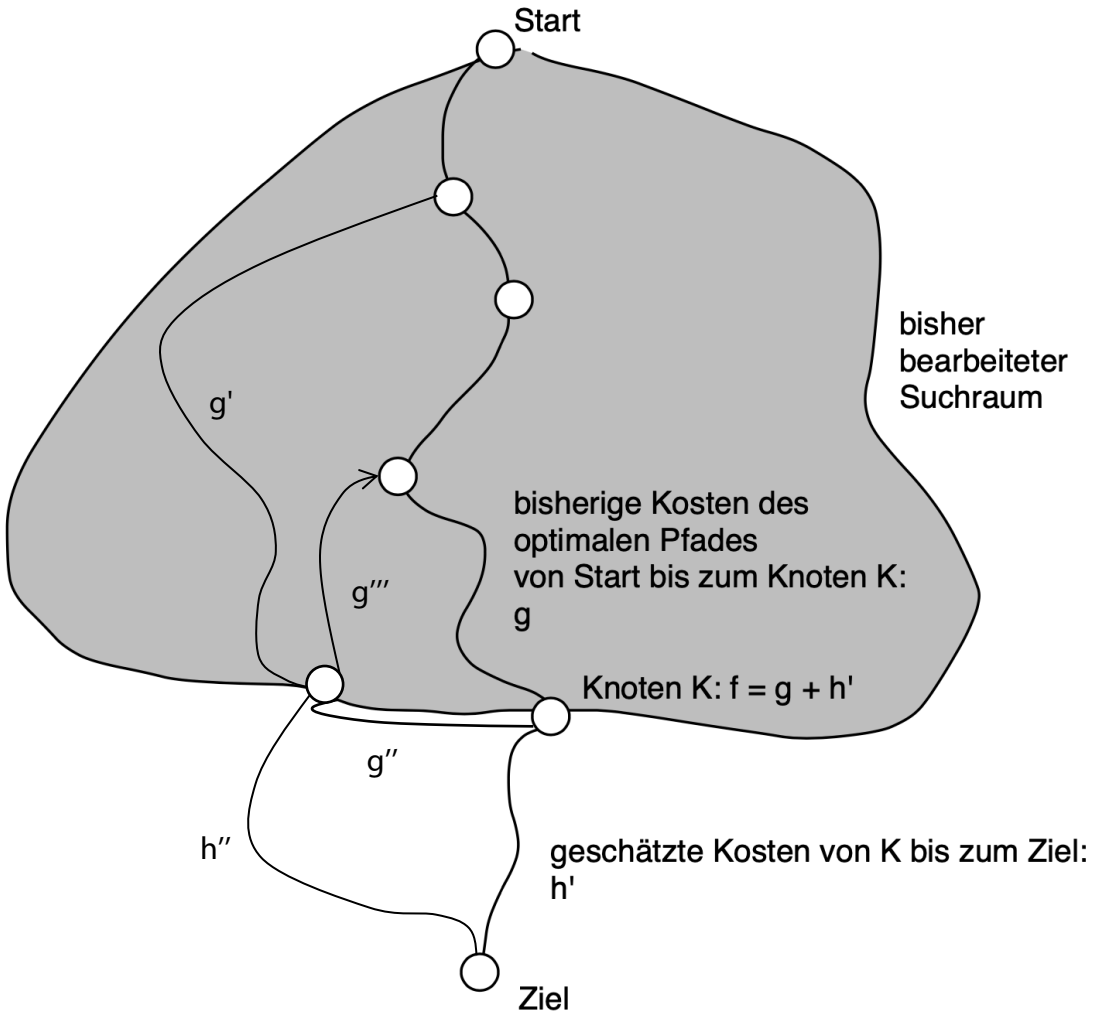


f(a) = 1,5 + 4
f(d) = 2 + 4,5
f(b) = 3,5 + 2
f(c) = 6,5 + 4
f(e) = 5 + 2
f(f) = 7 + 0

Exploration des (implizit definierten) Suchraums



Preis für den Zielschätzereinsatz: Propagierung



Funktion A*

```
function A* (s, ziel?, cost, h, (V, E)):   // Eingabe: Start s ∈ V, Kantenkosten cost, Schätzer h und Graph (V, E)
  pq:=<>:sPQ with key as lambda((_,_, n_f,_)) n_f      // PQ mit Einträgen (Knoten, g-Kosten, f-Kosten, Vorgänger)
  expanded:=<>:Set with key as lambda((v, _, _)) v  // Menge expandierter Knoten (Knoten, g-Kosten, Vorgänger)
  insert((s, 0, 0+h(s), ⊥), pq)
  while not mtQueue?(pq) do
     (u, u_g , _, w) := deleteMin(pq)
     insert((u, u_g, w), expanded)
     if ziel?(u) then return path(u, expanded)  // Ausgabe: Lösungspfad rückwärts zum Start s
     for (u, v)∈ E do
        x := search(v, expanded)                  // search liefert gesuchtes Element oder ⊥ für nicht gefunden
        if x = ⊥then y := search(v, pq)           // Knoten gesehen, nicht expandiert? (pq unterstützt search)
          v_g := u_g + cost(u, v)
          if  y = ⊥ then
              insert( (v, v_g, v_g + h(v), u) , pq)
          else (v, v_g-old, _, _) := y
               if v_g < v_g-old then                    // Günstigerer Weg zu noch nicht expandiertem Knoten?
                  decreaseKey(y, pq, v_g-old - v_g)     // Expandiere früher und trage
                  parent(y) := u; second(y) := v_g      // neuen Vorgänger und neue g-Kosten ein
        else if u_g+ cost(u, v) < second(x) then        // Günstigerer Weg zu schon expandiertem Knoten
                 propagate(u v, u_g+ cost(u, v), expanded, pq, (V, E))  // Propagiere neue Kosten für exp. Knoten
```

Hilfsfunktionen

```
function path(v, expanded)
  // Konstruiere Pfad p aus Menge von expandierten Knoten
  // Einträge in expanded haben die Form (Knoten, g-Kosten, Vorgänger)
  p= <>:Stack
  while true do
     x := search(v, expanded)
     if x = ⊥ then
        return p
     else (u, _, v) := x
           push(u, p)
```

Hilfsfunktionen

```
function propagate(u, v, new-g, expanded, pq, (V, E)):
  // Propagiere neue Kosten g für Knoten v in die Nachfolger von v
   x := search(v, expanded)
   if x = ⊥ then
      x := search(v, pq)
      (v, old-g, _, _) := x                          // Noch nicht expandiert!
      if  new-g < old-g then                         // Besserer Weg und damit
         decreaseKey(v, pq, old-g – new-g) // neue Priorisierung
         second(x) := new-g; fourth(x) := u  // neues g und neuer Vorgänger
  else (v, old-g, _) := x                            // Schon expandiert!
       if new-g < old-g then                         // Besserer Weg und damit
         second(x) := new-g; fourth(x) := u  // neues g und neuer Vorgänger
         for (v, w) ∈ E do                           // weiter propagieren
              propagate(v, w, new-g + cost(v, w), expanded, pq, (V, E))
```

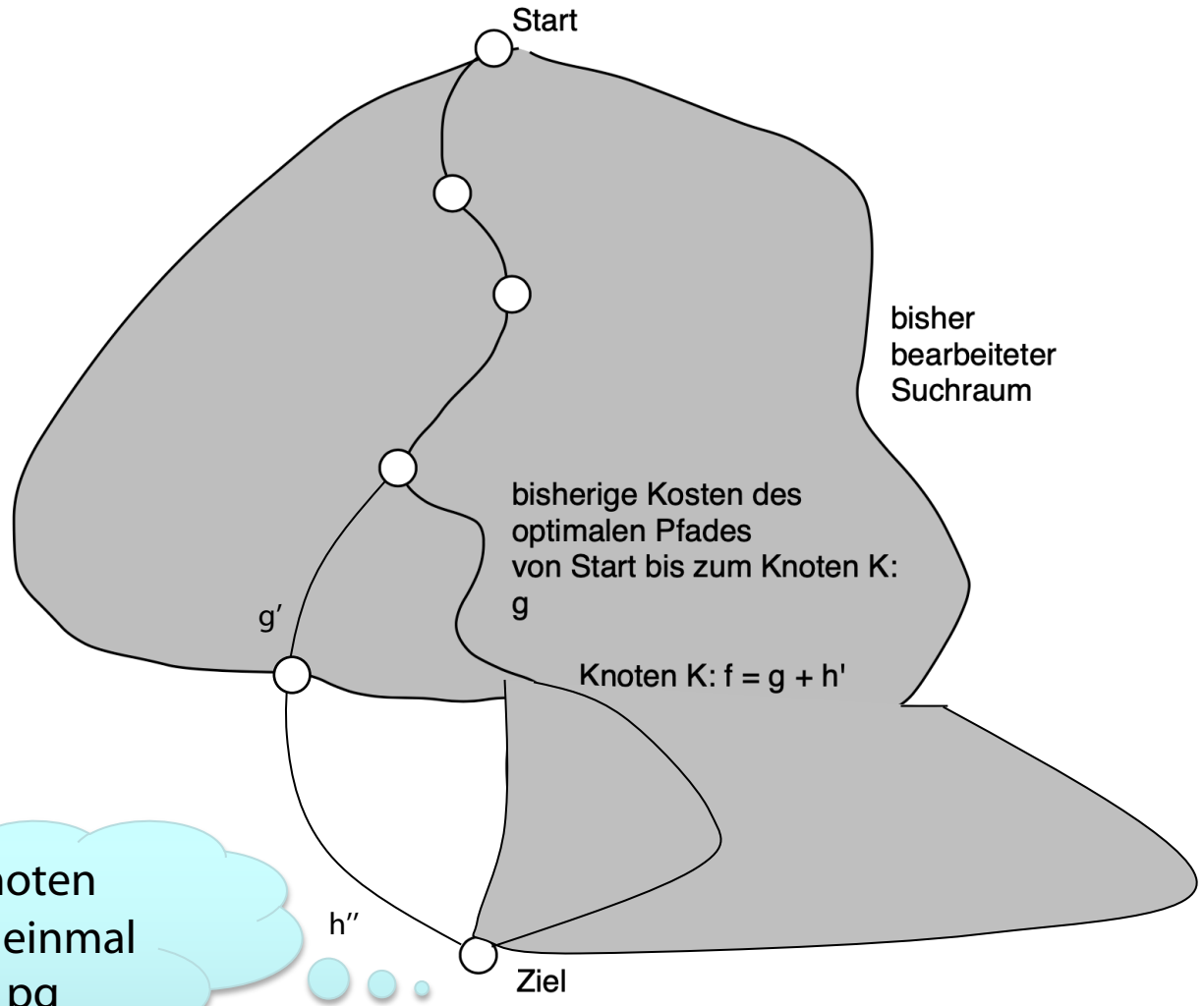
Analyse

Sei $G=(V, E)$ ein Graph mit positiven Kantenkosten und h^* eine Funktion, die die tatsächlichen Kosten von jedem Knoten $u \in V$ zum Ziel $v \in V$ bestimmt (also: $ziel?(v) = true$).

Definition: Ein Schätzer h heißt zulässig, wenn für alle $v \in V$ gilt, dass $h(v) \leq h^*(v)$, wobei $h^*(v)$ die optimale Schätzfunktion darstellt, die die tatsächlichen Kosten genau einschätzt.

Behauptung: A* findet die optimale Lösung, wenn h zulässig ist

Zulässigkeit des Schätzers und optimale Lösung



Auch Zielknoten kommen erst einmal nur in die pq

Analyse von A*

- Kantenkosten müssen positiv sein
- Schlimmster Fall:
  - $h(n) = 0$ für alle Knoten n
  - Dann Verhalten wie beim Dijkstra-Algorithmus
- Aber: Je besser der Schätzer, desto besser das Verhalten
  - Bei optimalem Schätzer h^* Verhalten linear zur Länge des Lösungspfades (durch h^* ist der Name A* motiviert)
- Schätzer h ist nicht immer einfach zu bestimmen
  - Anwendungsspezifisches Wissen
  - Funktion h geht als Parameter in A* ein

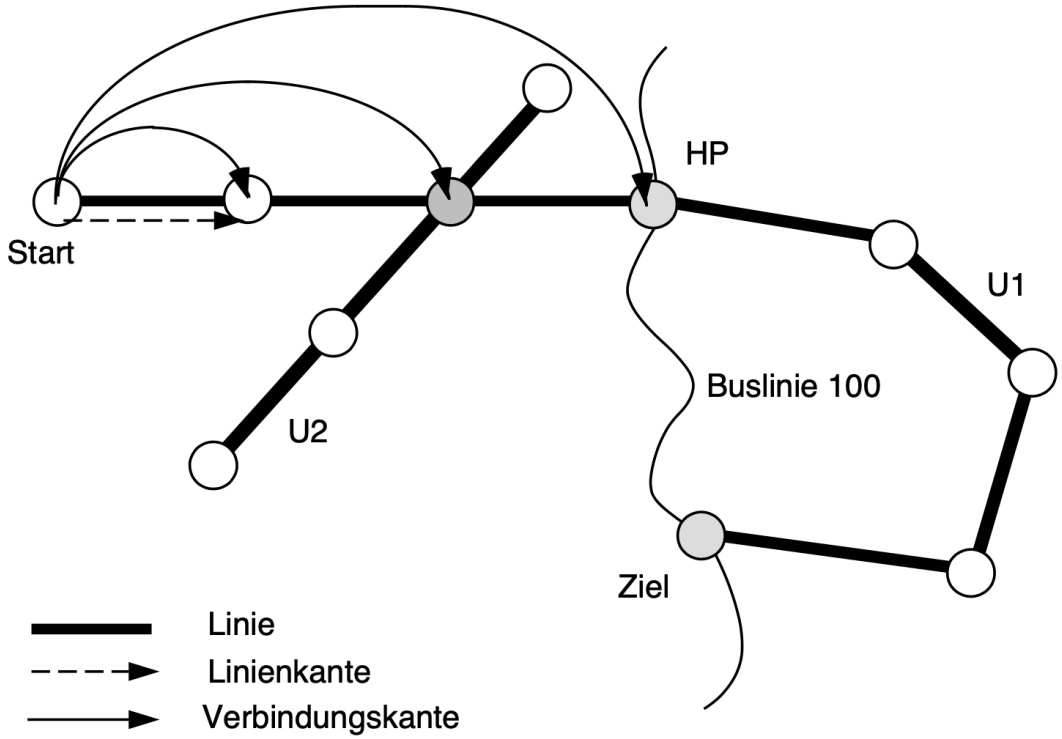
Anwendung: Fahrplaninformationssystem

- Vorwärtssuche
  - Gegeben: Start- und Zielhaltepunkt und frühestmögliche Abfahrtzeit am Start
- Rückswärtssuche
  - Gegeben: Start- und Zielhaltepunkt und spätestmögliche Ankunftszeit am Ziel
- Zu berücksichtigen
  - Umsteigezeiten zwischen Linien an einem Haltepunkt
    - Fußwegezeiten (vgl. z.B. Jungfernstieg)
  - Umsteigemodalitäten
    - Behindertengerechter Umstieg (1991 nicht selbstvertändlich)
  - Erweiterung: Start- und Zielort (Fußwegezeiten zu den umliegenden Haltepunkte zu bestimmen)

R. Möller, L. Hotz. Suchalgorithmen und Interaktionstechniken für Fahrplan-Informationssysteme. Technical Report Bericht Nr. LKI-M-91/3, Labor für Künstliche Intelligenz, **1991**

Anwendung: Fahrplaninformationssystem

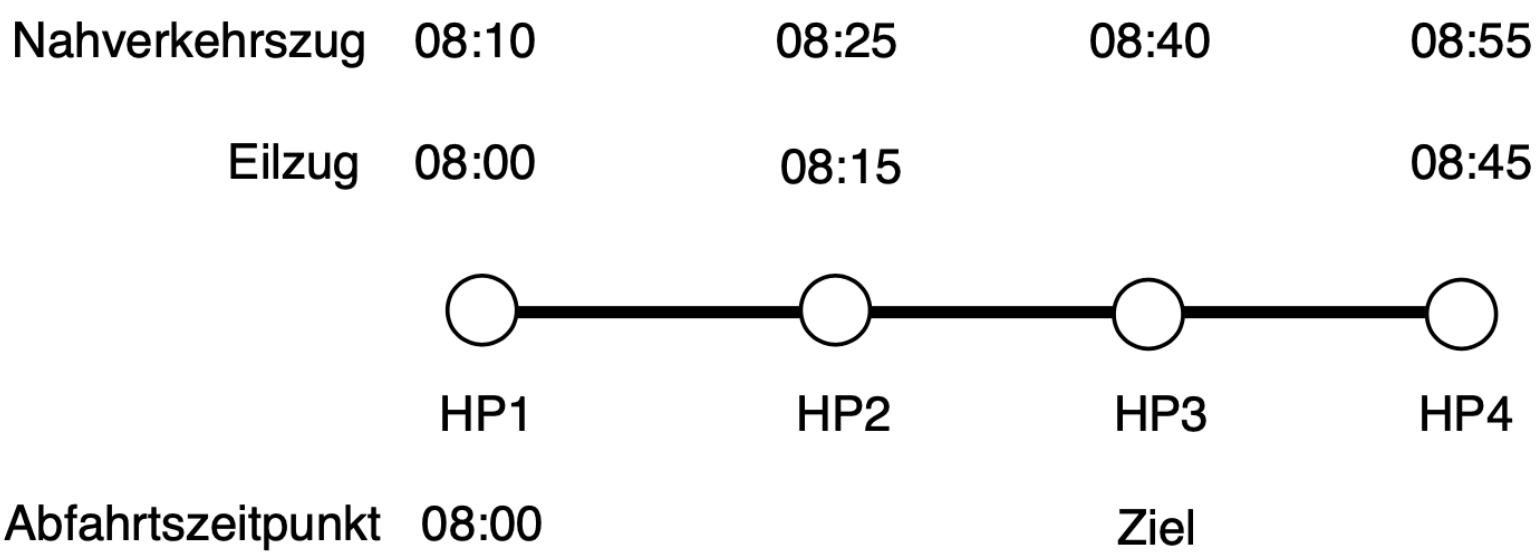
- Nachverkehrssystem repräsentiert als Graph



- Realisierung mit A*: Wie ist der Suchgraph definiert?

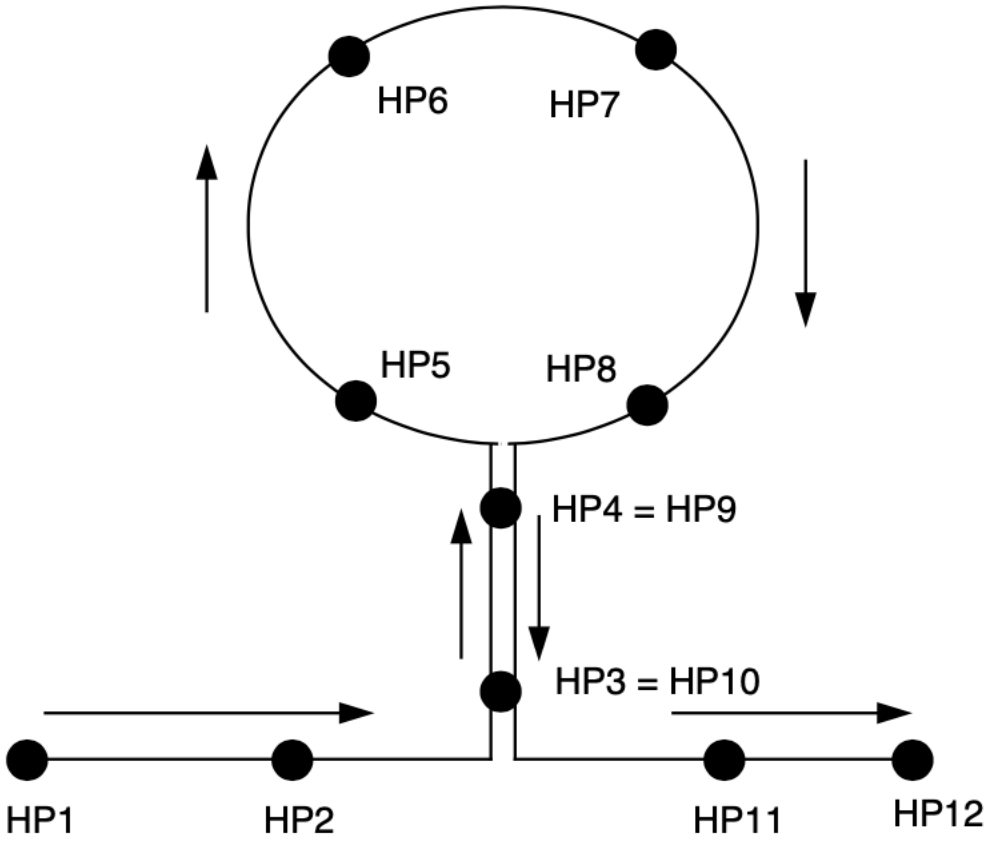
Planung von Haltepunkt zu Haltepunkt?

- Lokale Sicht erzeugt unnötiges Umsteigen



- Was ist eine Linie?
  - Nahverkehrszug und Eilzug unterschiedliche Linien
- Suchgraph über Linienverbindungskanten

Besondere Linienformen in der Praxis

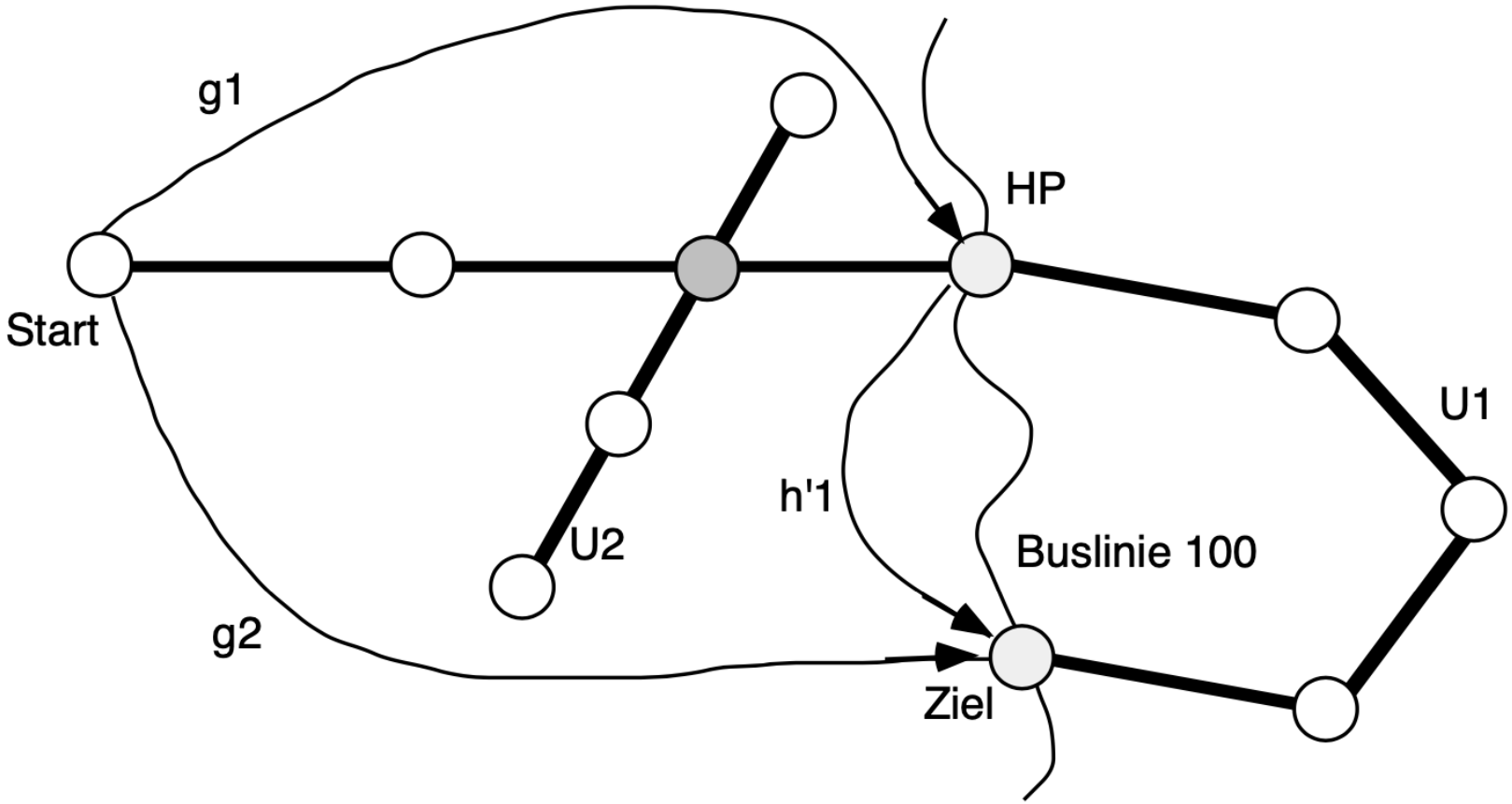


Kosten

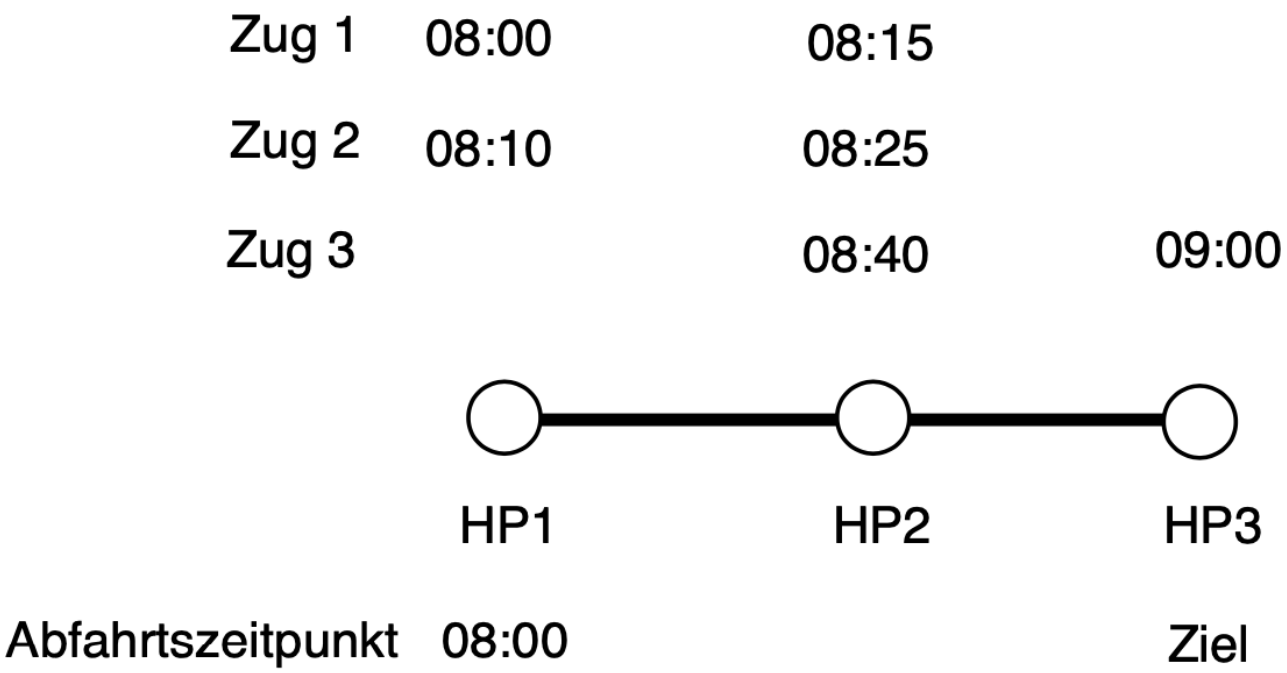
- Fahrtzeit
- Anzahl der Umsteigevorgänge
- Fußwegezeit
- Behindertengerechtheit
- Verkehrsmitteltyp
- …

- Pro Kante: (reine Fahrzeit, abstrakte Fahrtzeit)
  - Reine Fahrtzeit: Bestimmung frühestmögliche Abfahrtzeit/Ankunftszeit am Ziel
  - Abstrakte Fahrtzeit: Kosten

Zeit, Kosten, Zielschätzung



Vorwärts- und Rückwärtssuche



- Auf Vorwärtssuche (frühestmögliche Ankunftszeit) folgt Rückwärtssuche (spätestmögliche Abfahrtszeit)

Schätzergenerierung: Offline-Berechnung von h

- Schätzergenerierung aus geographischen Daten?
  - Haben wir verworfen (Elbe-dazwischen-Problem)
- Schätzergenerierung aus Fahrplan
  - Schnellstmögliche direkte Verbindung zwischen jedem Paar von Haltepunkten unabhängig vom Zeitpunkt (ohne Umsteigen)
  - Es entsteht ein Graph mit direkten Verbindungen als Kanten, deren Kostenbeschriftung in Fahrtzeit angegeben wird
- Zu lösen: All-Pairs Shortest Paths Problem zur Bestimmung der schnellsten Verbindung mit Umsteigen
  - Wartezeiten auf Anschluss ignoriert
  - Fußwegezeiten ignoriert
- Schätzer unterschätzt Kosten und ist daher zulässig

Zusammenfassung

- Suche mit Zielfokussierung
- A*-Algorithmus: Informierte Suche
- Anwendung: Fahrplaninformationssystem

- In praktischen Anwendungen
  sind die Graphen meist implizit gegeben
  - Knoten und Kanten werden dynamisch generiert