

Grundlagen der Künstlichen Intelligenz (3/6)

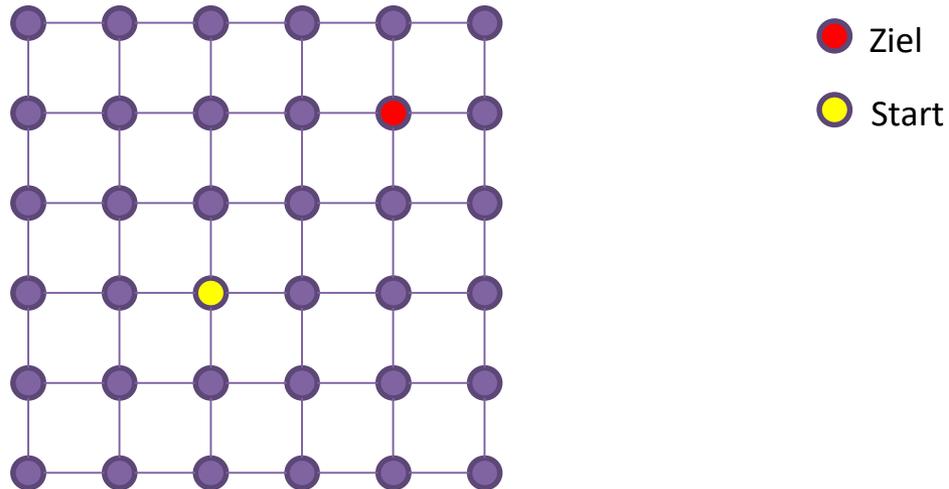
Kostensensitive Suche

Dr. Christian Meilicke, Research Group Data and Web Science



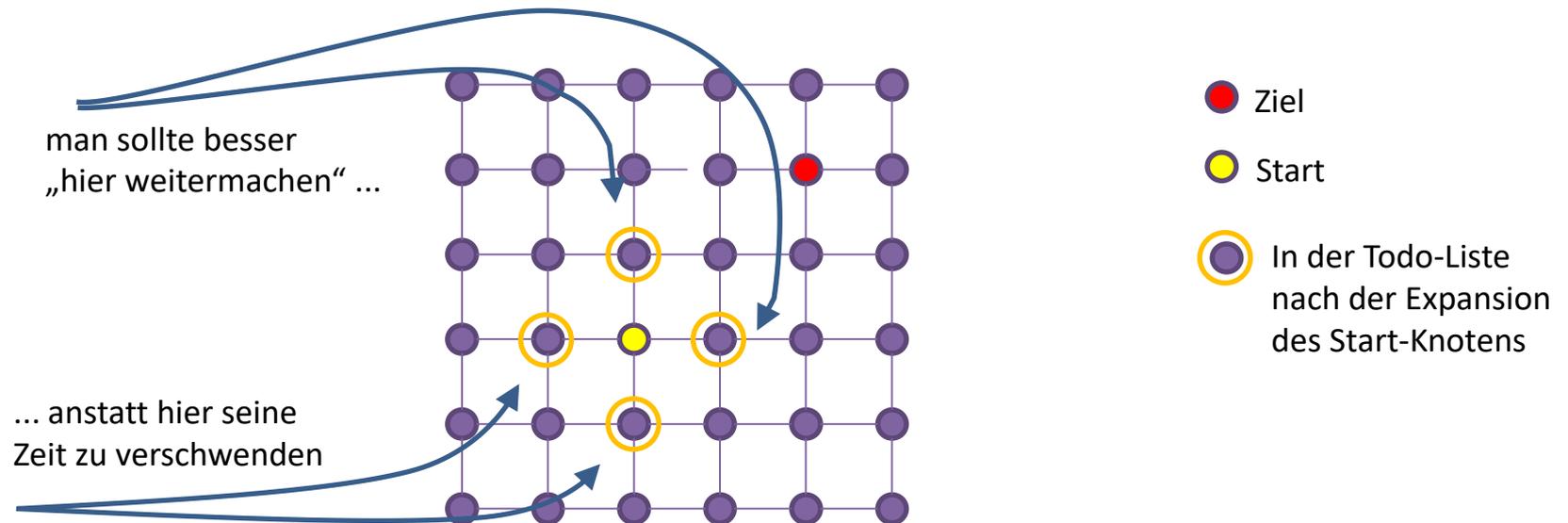
Motivation

- Wieso benötigen wir ein besseres Verfahren als die iterative Tiefensuche?
 - Im Fall nicht-konstanter Kosten ist dies klar => keine optimale Lösung!
 - Auch im Fall von konstanten Kosten können wir bessere Verfahren anwenden! Siehe Beispiel:



Motivation

- Wieso benötigen wir ein besseres Verfahren als die iterative Tiefensuche?
 - Im Fall nicht-konstanter Kosten ist dies klar => keine optimale Lösung!
 - Auch im Fall von konstanten Kosten können wir bessere Verfahren anwenden! Siehe Beispiel:



Informiert/Kostensensitiv

- Informierte Suche
 - Informationen in Bezug auf die Nähe des Ziels werden berücksichtigt
 - Es wird “voraus” gedacht mit Hilfe von Schätzungen
- Kostensensitive Suche
 - Verschiedene Kosten werden berücksichtigt
 - Im Gegensatz zu Verfahren, die nur Ebenen berücksichtigen

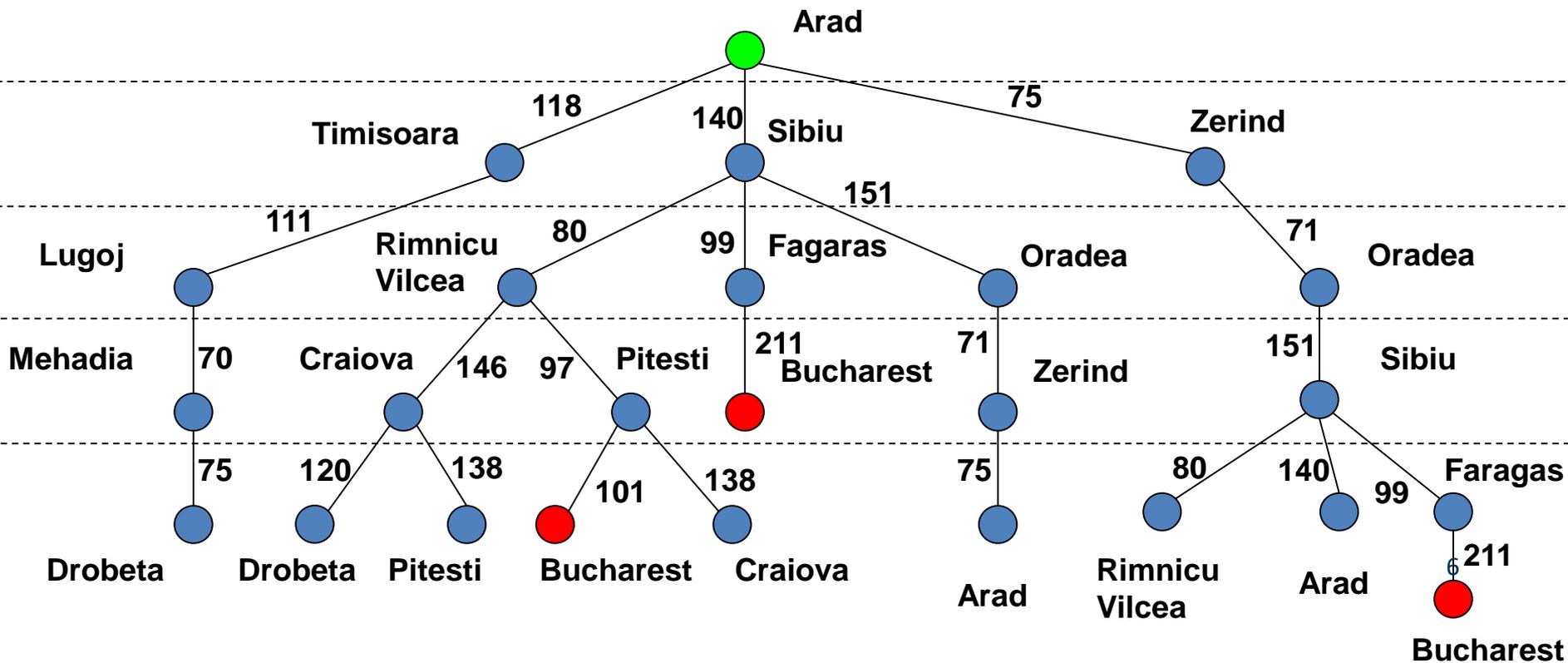
	uninformiert	informiert
nicht kostensensitiv	BS, TS, ...	-
kostensensitiv	Uniforme KS	Greedy, A*

Suchstrategien

- Grundlegende Methoden
 - Breitensuche
 - Tiefensuche
- Varianten der Tiefensuche
 - Backtracking-Suche
 - Tiefenbegrenzte Suche
 - Iterative Tiefensuche
- **Kostensensitive Suche (auch Bestensuche)**
 - **Uniforme Kostensuche**
 - **Greedy Suche**
 - **A* Suche**

Kostensensitive Suche

Bei der "Bestensuche" gibt es eine Funktion f , die jedem Zustand einen Wert zuordnet. Basierend auf diesem Wert wird der jeweils aktuell "beste Zustand" expandiert. => Implementierung als Priority Queue



Wieder derselbe Algorithmus

Kostensensitive Suche

```
PriorityQueue todo = [startState]
DO LOOP
  IF todo = []
    RETURN "Fail, no solution found"
  ELSE
    State s = selectState(todo)
    IF isSolution(s)
      RETURN "Solution found"
    ELSE
      List expandedStates = expand(s)
      add(expandedStates, todo)
```

Wieder derselbe Algorithmus

```
List todo = [startState]
DO LOOP
  IF todo = []
    RETURN "Fail, no solution found"
  ELSE
    State s = selectState(todo)
    IF isSolution(s)
      RETURN "Solution found"
    ELSE
      List expandedStates = expand(s)
      add(expandedStates, todo)
```



FIFO \Rightarrow Breitensuche

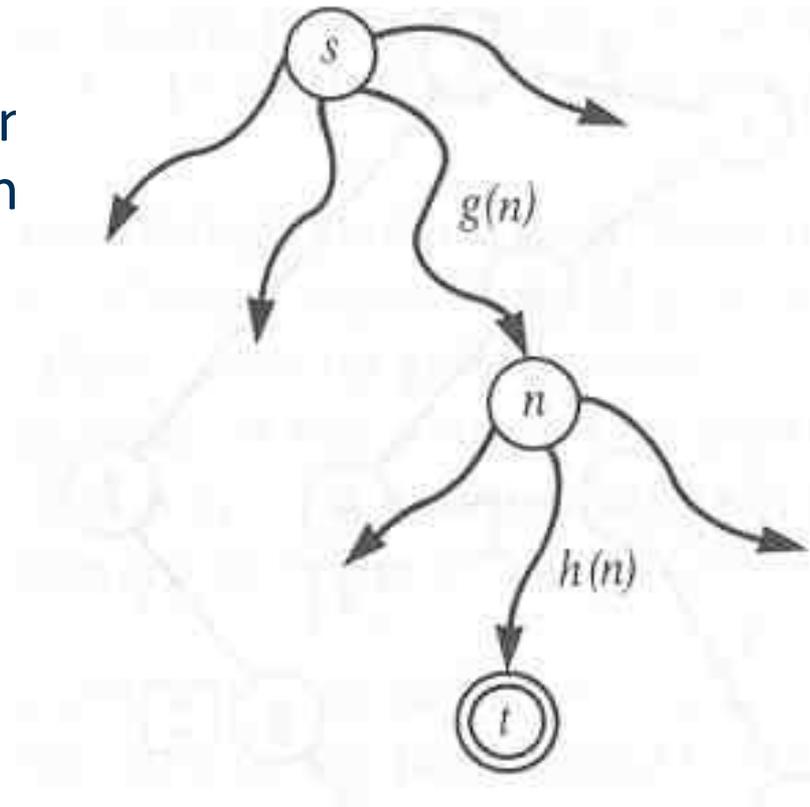
LIFO \Rightarrow Tiefensuche

PriorityQueue \Rightarrow Bestensuche

Wie die Priorität berechnet wird,
bestimmt die Art der Bestensuche

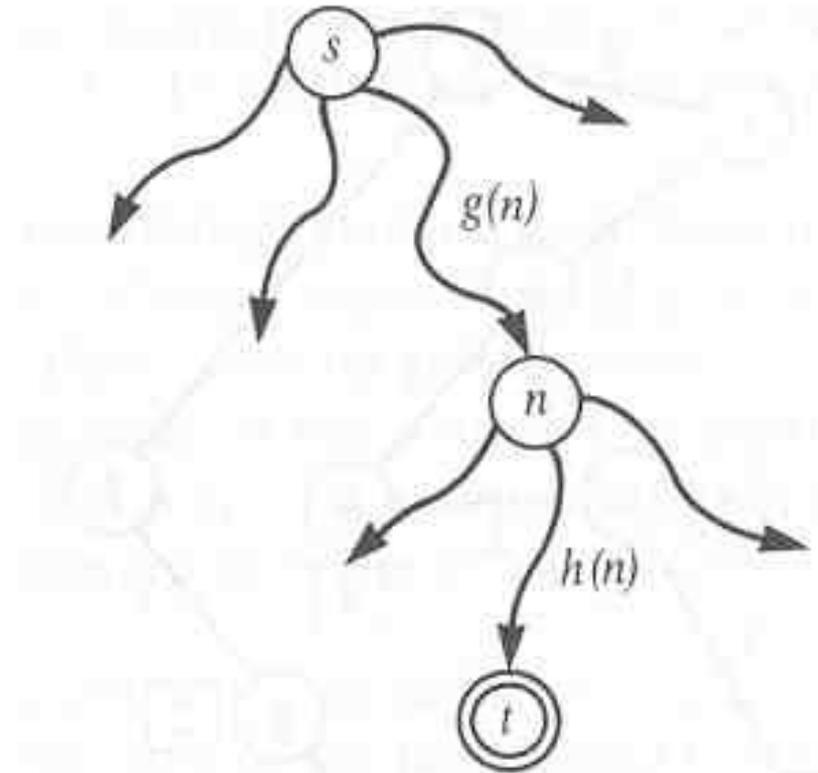
Bestimmung der Priorität

- s = start, t = target (ziel)
- Es wird immer der Knoten n mit der kleinsten Wert für eine Funktion $f(n)$ als nächstes bearbeitet
- Relevante Kosten eines Knotens n
 - bisherige Kosten $g(n)$
 - noch erwartete Kosten $h(n)$



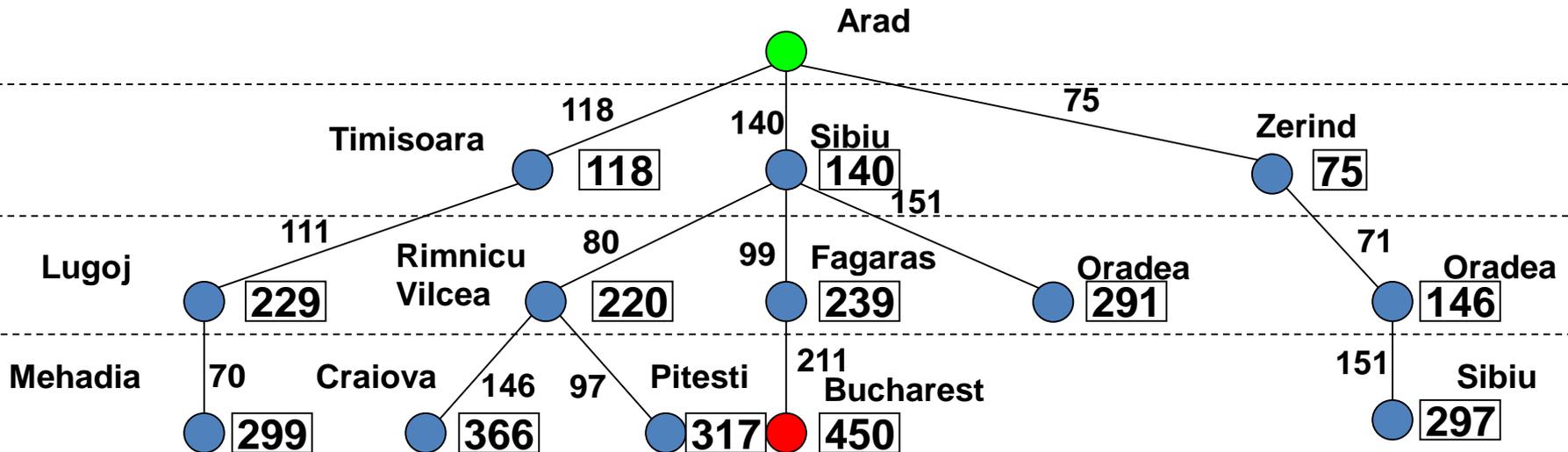
Definition von f

- Es sei
 - s der Startzustand,
 - n der aktuelle Zustand
 - t ein Zielzustand
- Uniforme Kostensuche
$$f(n) = g(n)$$
- Greedy Suche
$$f(n) = h(n)$$
- A* Suche
$$f(n) = g(n) + h(n)$$



Uniforme Kostensuche

Es wird immer der Knoten mit den bisher geringsten Gesamtkosten $g(n)$ expandiert, d.h. $f(n) = g(n)$



Eigenschaften: Uniforme Kostensuche

d = Tiefe der Lösung
m = Max. Tiefe des Suchbaums
b = Branching Faktor
c = Kosten der Lösung
e = Minimale Kosten pro Schritt

- Vollständigkeit:

JA/NEIN: bei Schritten mit Kosten 0 können Kreise entstehen. Sind alle Kosten grösser als ein Wert $e > 0$, ist das Verfahren vollständig

- Optimalität:

JA: Der erste Lösungsknoten, der expandiert werden soll ist die optimale Lösung, da er die geringsten Kosten aufweist

- Zeitkomplexität:

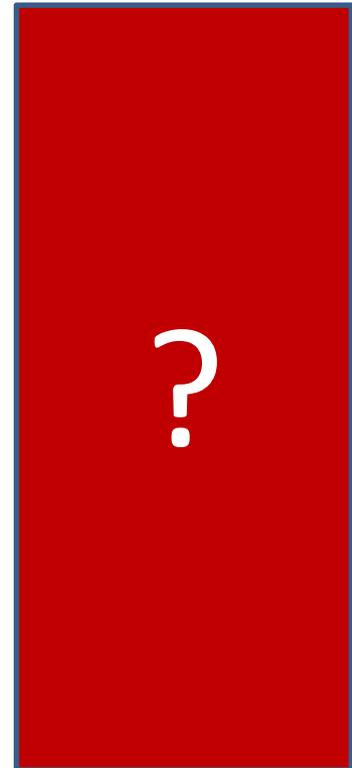
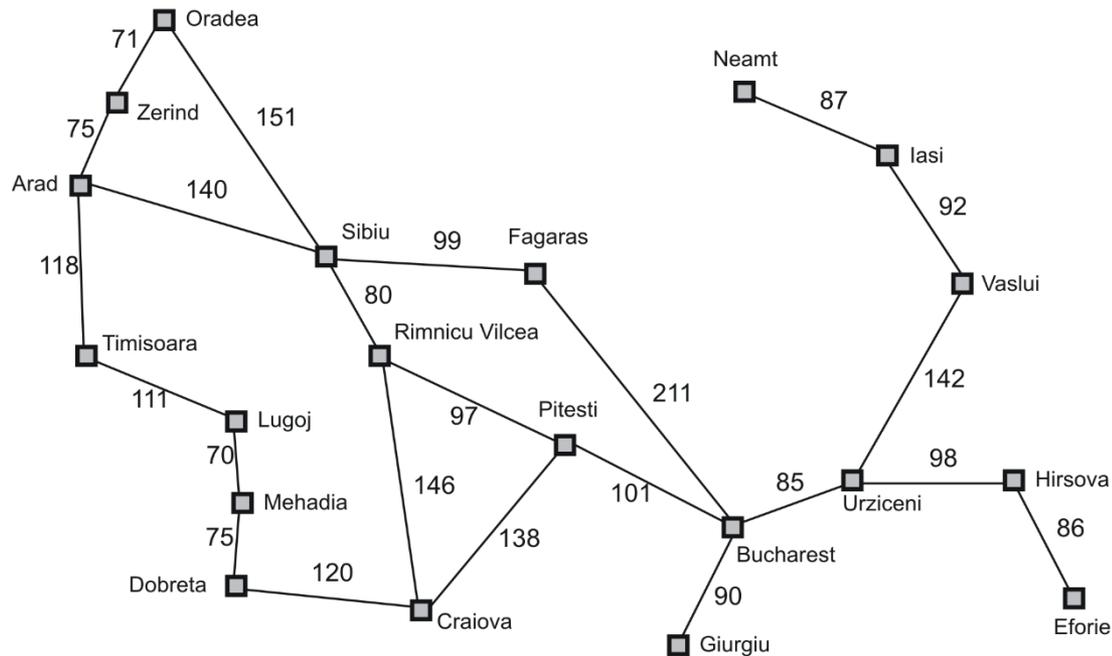
$O(b^{c/e})$: Anzahl der Schritte hängt von den Kosten der Lösung ab. Um auf diese Kosten zu kommen sind maximal c/e Schritte nötig

- Speicherkomplexität:

$O(b^{1+c/e})$: Wie bei der Breitensuche werden alle expandierten Knoten gespeichert.

Zu erwartende Kosten

- Wie kann man eine Abschätzung der Restkosten erhalten?



8-Puzzle

hier ist es nicht so einfach die Restkosten abzuschätzen

Online Demo

z.B. <https://murhafsousli.github.io/8puzzle/#/>

oder in Google nach `8 puzzle online` suchen

Shuffle => Show numbers => Selbst probieren => Solve

Heuristische Funktionen

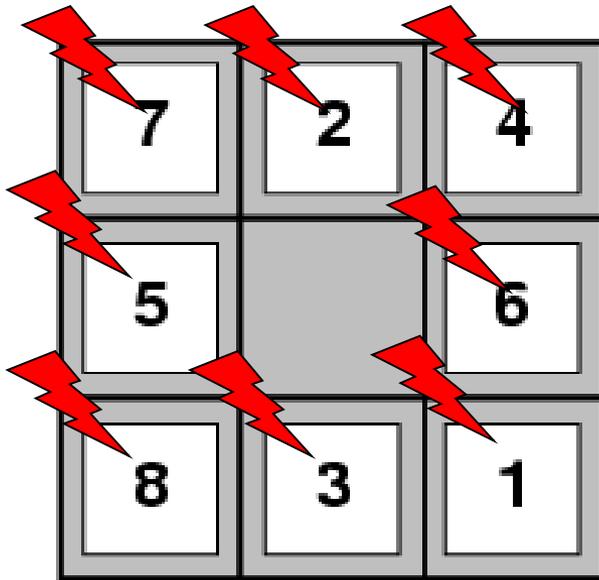
7	2	4
5		6
8	3	1

Start State

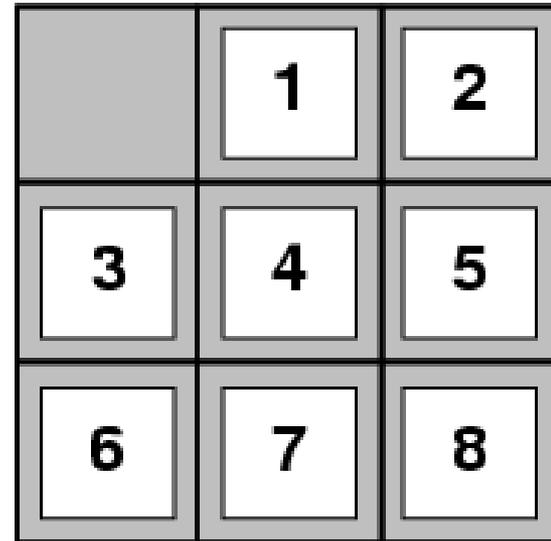
	1	2
3	4	5
6	7	8

Goal State

Misplaced Tiles



Start State



Goal State

Alle 8 Steine sind auf dem falschen Feld: Kosten = 8

Gaschnig's Heuristik

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Lösung für den Fall, dass Steine direkt auf das freie Feld gelegt werden dürfen.

Gaschnig's Heuristik

7	2	4
5		6
8	3	1

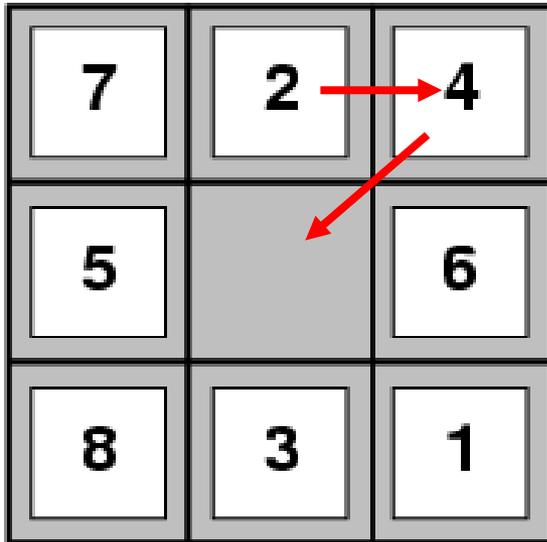
Start State

	1	2
3	4	5
6	7	8

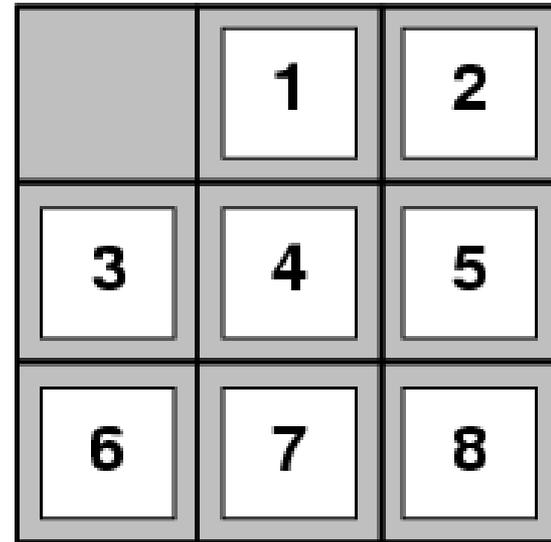
Goal State

Lösung für den Fall, dass Steine direkt auf das freie Feld gelegt werden dürfen.

Gaschnig's Heuristik



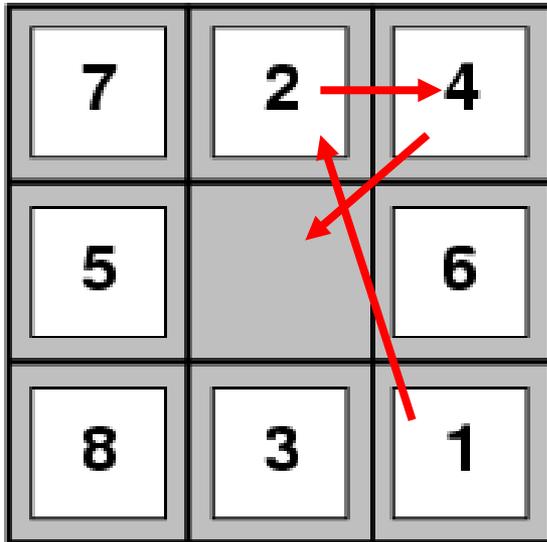
Start State



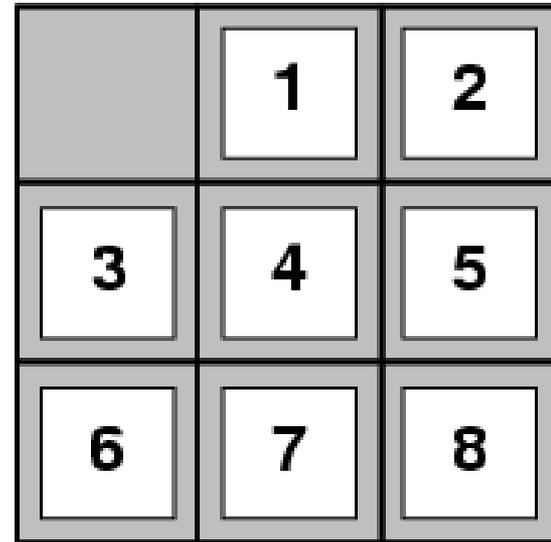
Goal State

Lösung für den Fall, dass Steine direkt auf das freie Feld gelegt werden dürfen.

Gaschnig's Heuristik



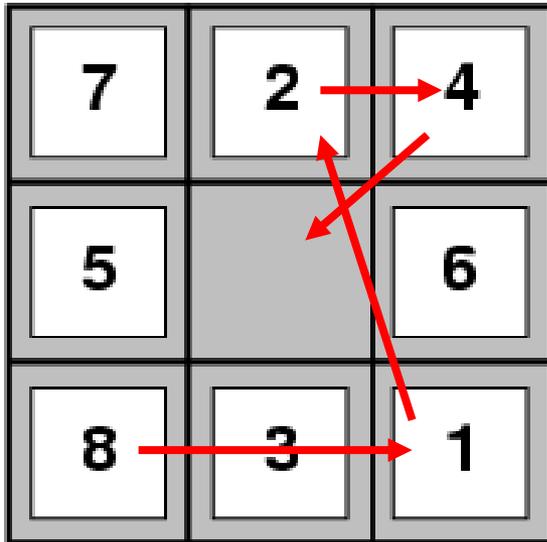
Start State



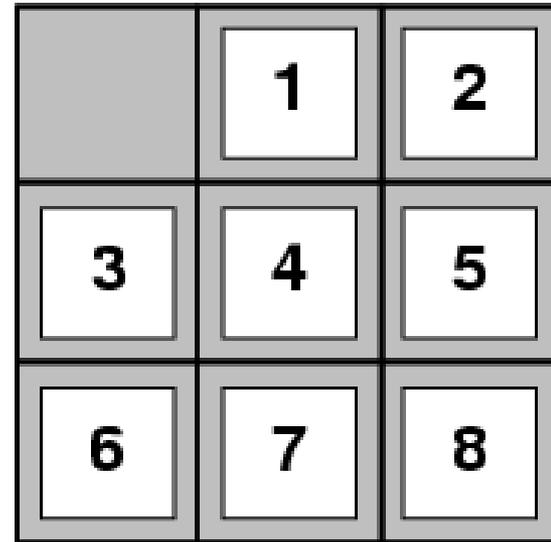
Goal State

Lösung für den Fall, dass Steine direkt auf das freie Feld gelegt werden dürfen.

Gaschnig's Heuristik



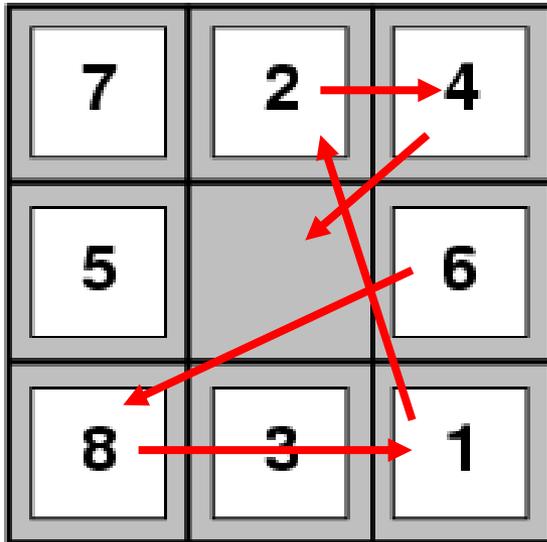
Start State



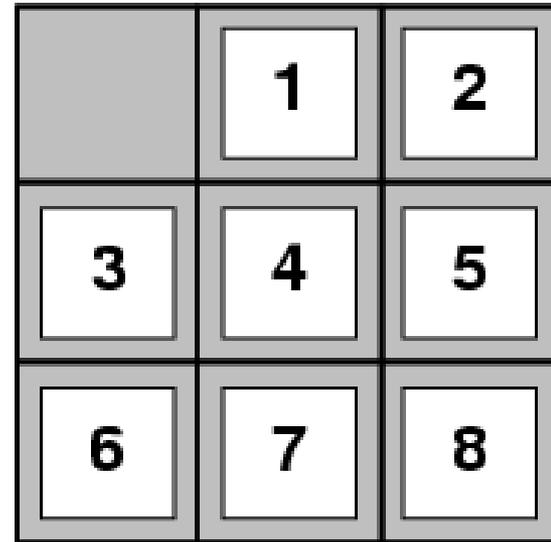
Goal State

Lösung für den Fall, dass Steine direkt auf das freie Feld gelegt werden dürfen.

Gaschnig's Heuristik



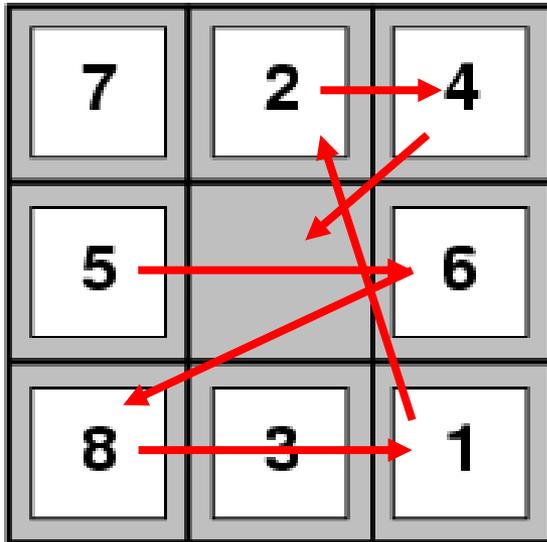
Start State



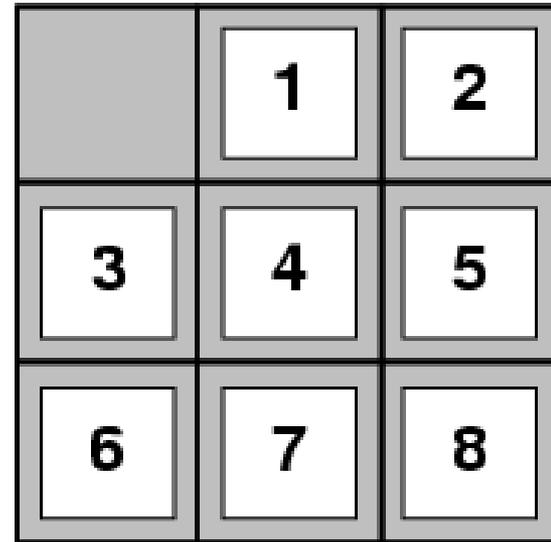
Goal State

Lösung für den Fall, dass Steine direkt auf das freie Feld gelegt werden dürfen.

Gaschnig's Heuristik



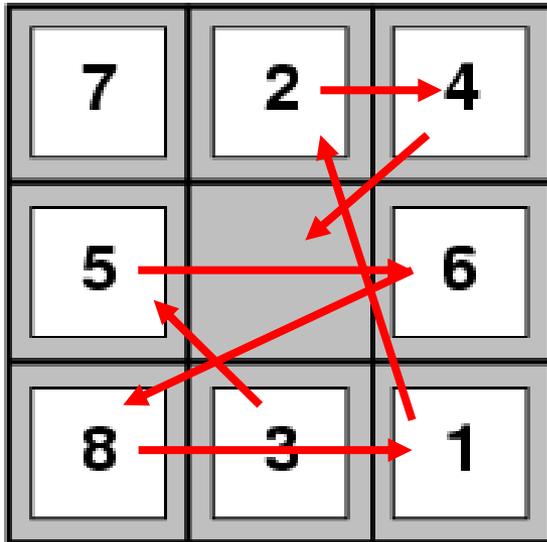
Start State



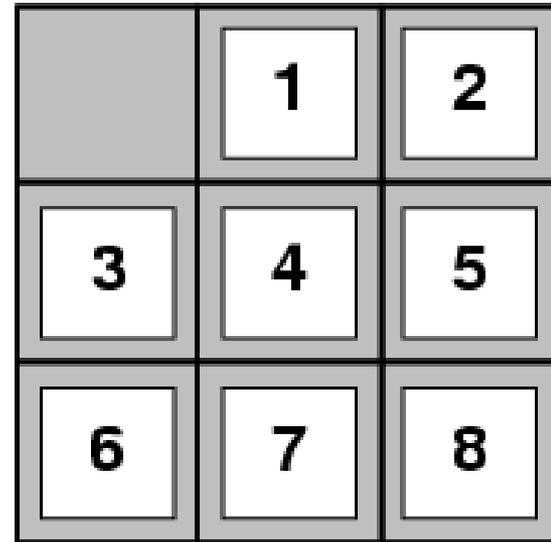
Goal State

Lösung für den Fall, dass Steine direkt auf das freie Feld gelegt werden dürfen.

Gaschnig's Heuristik



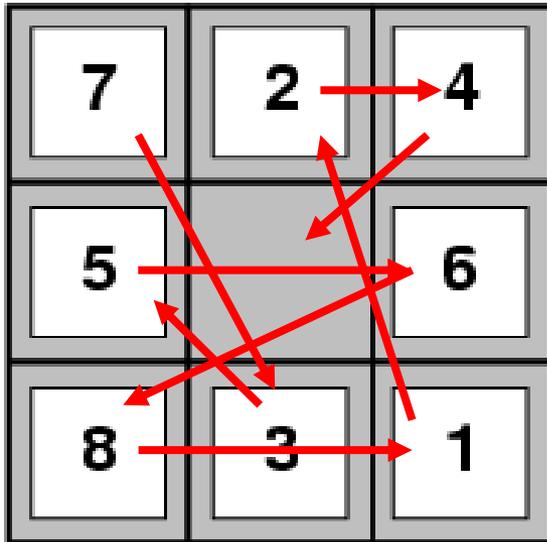
Start State



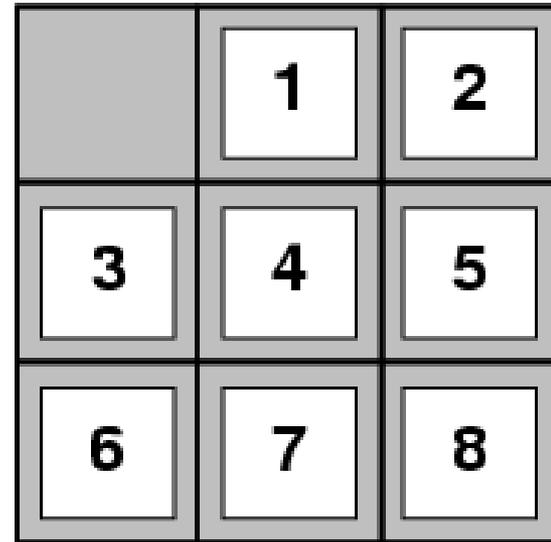
Goal State

Lösung für den Fall, dass Steine direkt auf das freie Feld gelegt werden dürfen.

Gaschnig's Heuristik



Start State



Goal State

Lösung für den Fall, dass Steine direkt auf das freie Feld gelegt werden dürfen. Kosten = 8.

Misplaced Tiles vs. Gaschnig

- Sind die beiden Heuristiken gleich ?
 - Gegenbeispiel: zwei Steine sind vertauscht
- Welche ist genauer ?
 - Gaschnig ist genauer
- Genereller Ansatz zur Entwicklung von Heuristiken:
 - Spielregeln vereinfachen, d.h. schummeln!

Manhattan-Distanz

7	2	4
5		6
8	3	1

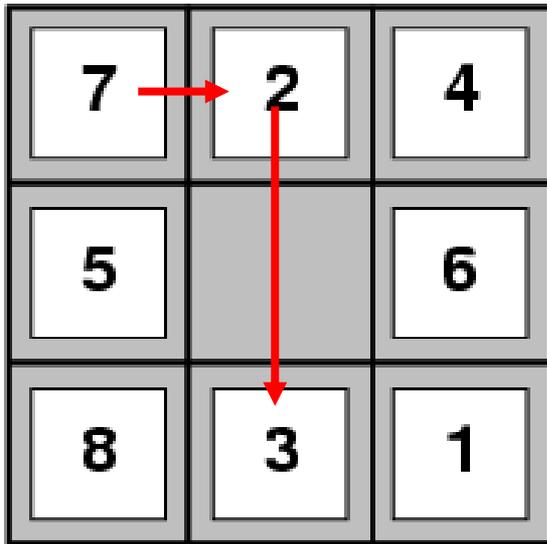
Start State

	1	2
3	4	5
6	7	8

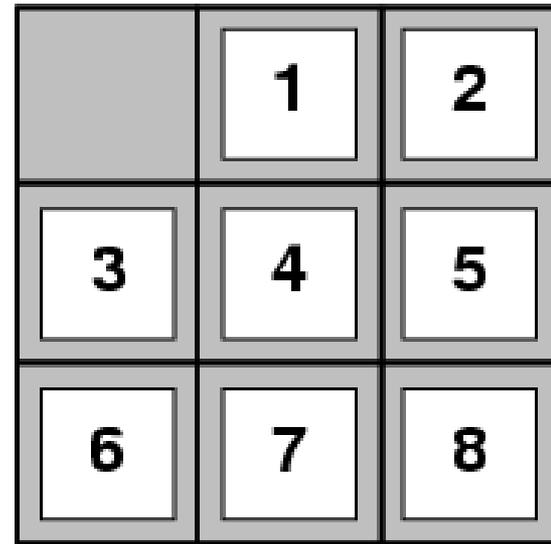
Goal State

Steine dürfen über andere geschoben werden.

Manhattan-Distanz



Start State



Goal State

Kosten = 3

Manhattan-Distanz

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

$$\text{Kosten} = 3 + 1$$

Manhattan-Distanz

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

$$\text{Kosten} = 3 + 1 + 2$$

Manhattan-Distanz

7	2	4
5		6
8	3	1

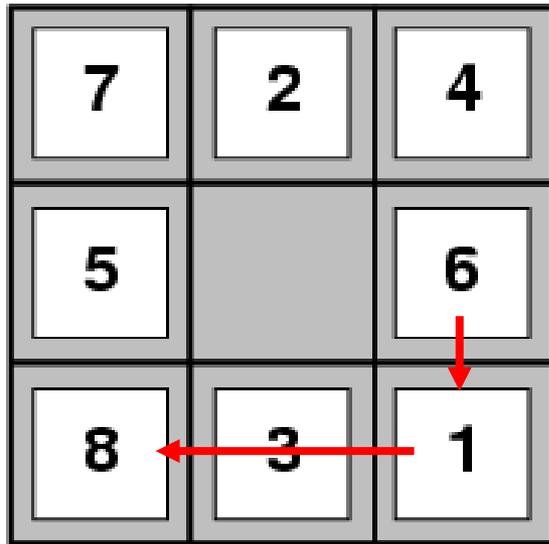
Start State

	1	2
3	4	5
6	7	8

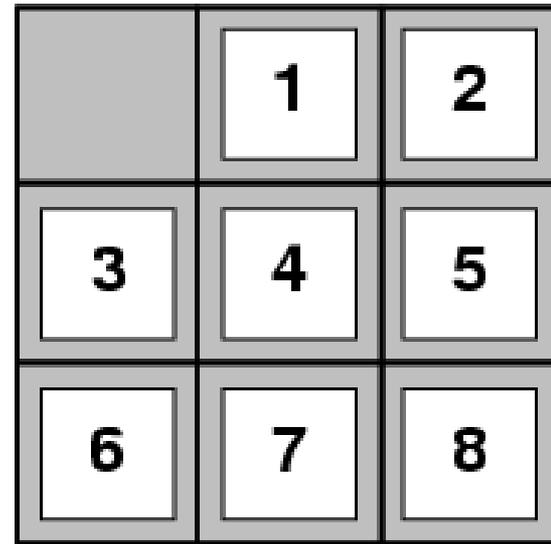
Goal State

$$\text{Kosten} = 3 + 1 + 2 + 2$$

Manhattan-Distanz



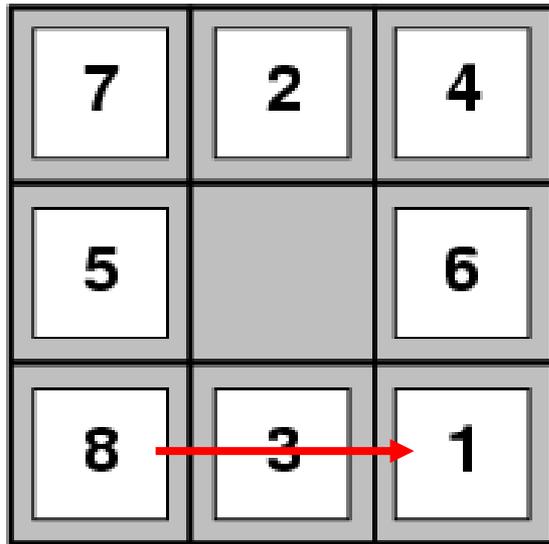
Start State



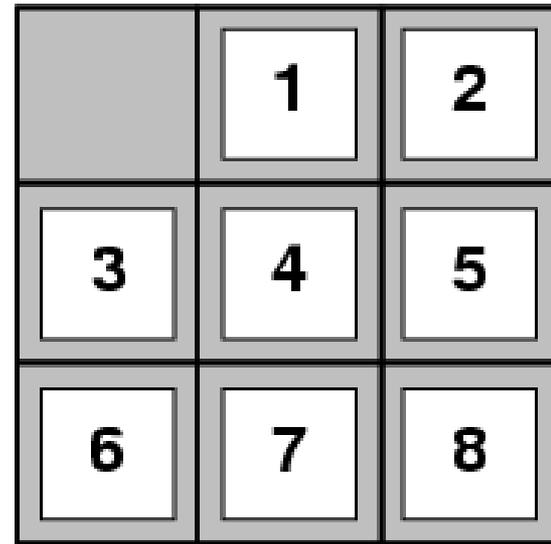
Goal State

$$\text{Kosten} = 3 + 1 + 2 + 2 + 3$$

Manhattan-Distanz



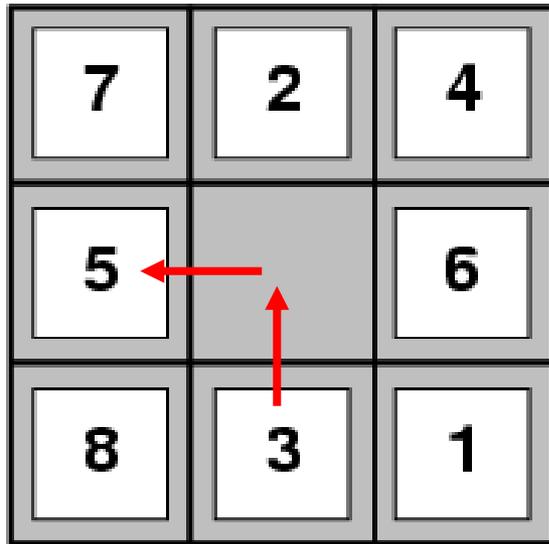
Start State



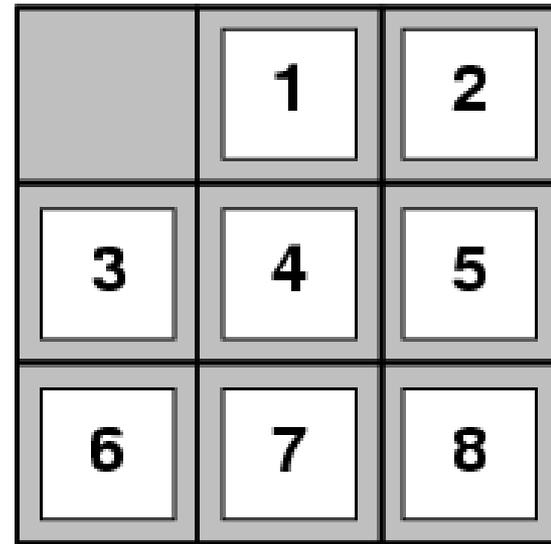
Goal State

$$\text{Kosten} = 3 + 1 + 2 + 2 + 3 + 2$$

Manhattan-Distanz



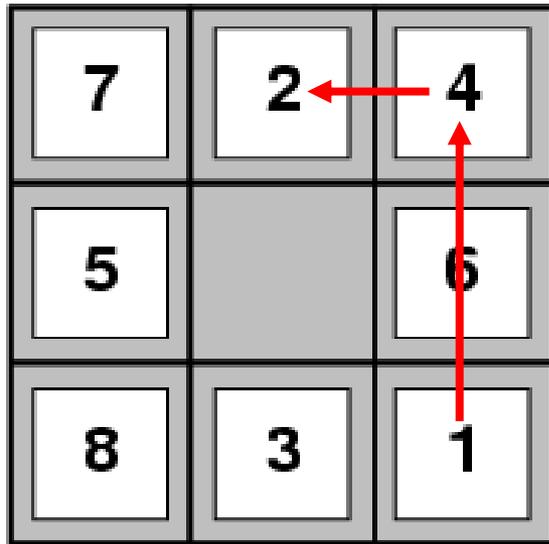
Start State



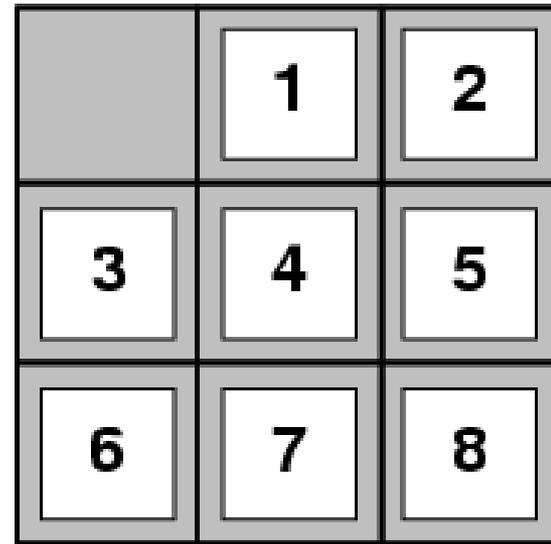
Goal State

$$\text{Kosten} = 3 + 1 + 2 + 2 + 3 + 2 + 2$$

Manhattan-Distanz



Start State



Goal State

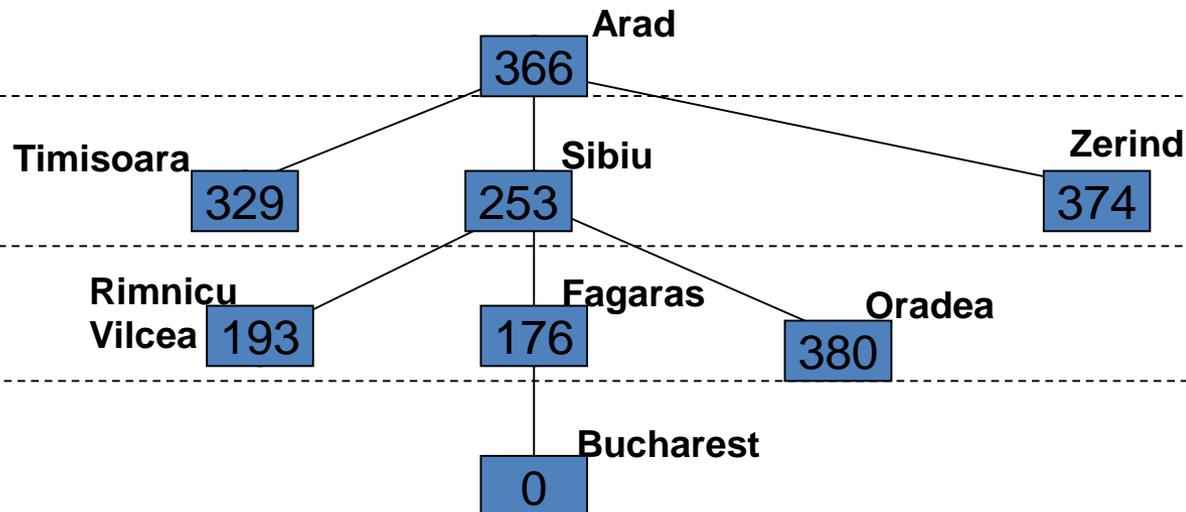
$$\text{Kosten} = 3 + 1 + 2 + 2 + 3 + 2 + 2 + 3 = 18$$

Restkosten Abschätzung

- Definition von $h(n)$ ist in der Regel problemspezifisch
- Möglich durch zusätzliche externe Informationen
 - Luftlinie
- Als Heuristikfunktion $h(n)$, die selbst erst berechnet werden muss
 - Vereinfachte Spielregeln & Schummeln
 - Kann selbst ein zu lösendes nicht triviales Suchproblem sein
- Überschätzen / Unterschätzen (später wichtig) ?

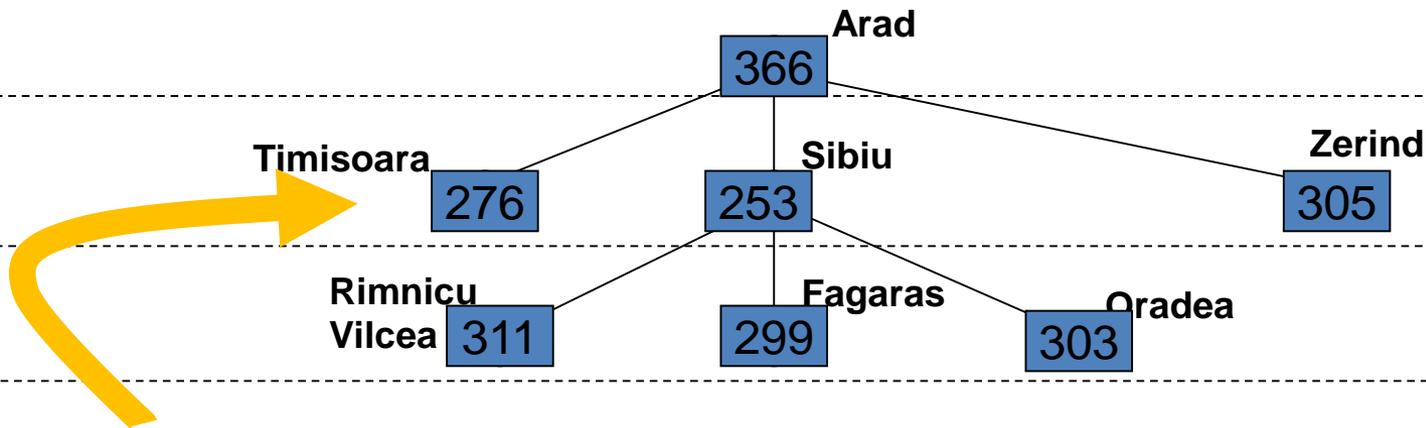
Greedy-Suche

Die Greedy Suche expandiert immer den Knoten mit den geringsten zu erwartenden Restkosten ($f(n) = h(n)$)



Greedy-Suche

Greedy Suche wird oft falsch verstanden
siehe folgendes modifizierte Beispiel



Welcher Knoten wird als nächstes expandiert?

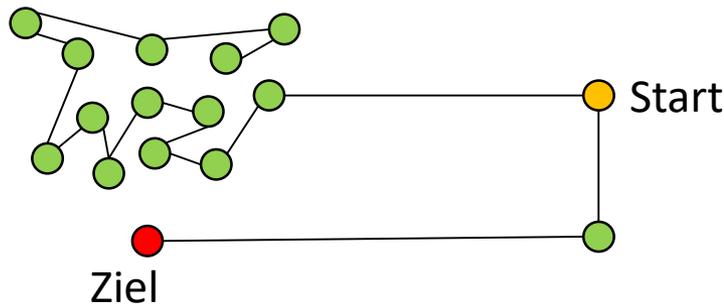
Eigenschaften: Greedy Suche

d = Tiefe der Lösung
m = Max. Tiefe des Suchbaums
b = Branching Faktor
c = Kosten der Lösung
e = Minimale Kosten pro Schritt

- **Vollständigkeit:**
NEIN: Der Suchraum kann unendlich sein. Beginnt die Suche in einem unendlichen Zweig mit niedrigen Werten für $h(n)$, werden Lösungen in anderen nicht gefunden
- **Optimalität:**
NEIN: Der zuerst expandierte Zweig kann eine Lösung enthalten, deren Gesamtkosten höher sind als die der optimale Lösung (Wenn viele kleine Schritte nötig sind)
- **Zeitkomplexität:**
 $O(b^m)$: Es müssen (fast) alle möglichen Knoten eines Zweigs expandiert werden, auch wenn keine Lösung enthalten ist.
- **Speicherkomplexität:**
 $O(b^m)$: Es müssen (fast) alle möglichen Knoten gespeichert werden, da jeder zur endgültigen Lösung gehören kann.

Greedy-Suche

- Alle “theoretischen Eigenschaften” sind schlecht
 - Weil sich worst case Szenarien konstruieren lassen, in denen das Suchverfahren in die Irre läuft

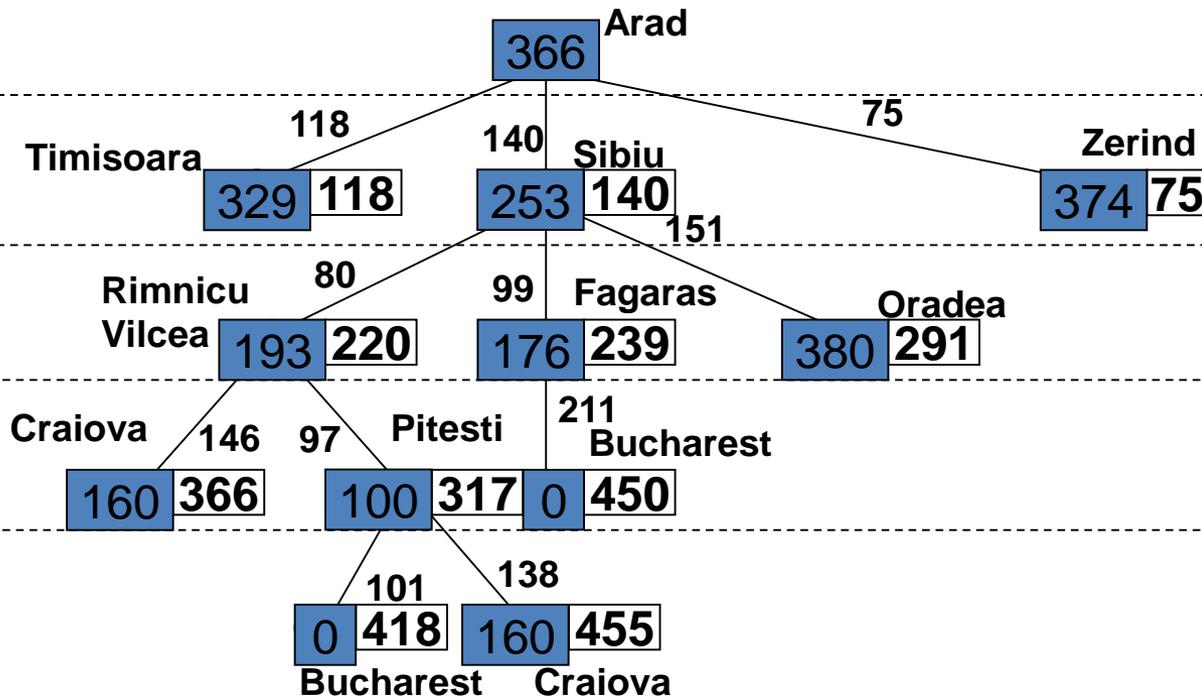


Heuristik = Luftlinie

- Aber: In vielen konkreten Anwendungsfällen tauchen solche Spezialfälle nicht auf
 - Greedy Suche funktioniert oft gut!

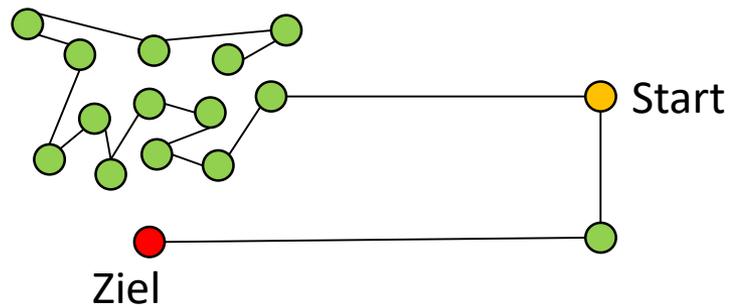
A* Suche

Die A* Suche expandiert immer den Knoten mit den geringsten zu erwartenden Gesamtkosten, d.h. $f(n) = g(n) + h(n)$.



Greedy-Suche vs A* Suche

- Vergleich anhand des zuvor betrachteten Beispiels:



Heuristik = Luftlinie

Zusammenfassung

- **Uniforme Kostensuche**
 - Eine Alternative zur Breitensuche, die Kosten berücksichtigt
 - Leidet unter ähnlichen Komplexitätsproblemen
- **Greedy-Suche**
 - Verwendet eine Heuristik zur Abschätzung der Restkosten
 - Ist nicht optimal, bei realen Probleme jedoch oft sehr effizient
- **A* Suche**
 - Die Methode der Wahl, wenn “gute” Heuristiken vorhanden sind
 - Sehr attraktive theoretische Eigenschaften, da es bewiesenermaßen keine effizienteres optimales Verfahren geben kann
 - Im vielen Fällen allerdings immer noch exponentiell
- **Heuristiken über zu erwartenden Kosten können helfen**
- **Qualität der verwendeten Heuristiken ist entscheidend**

Vorausschau

- Spielbaumsuche = Suchverfahren als Grundlage für eine Spiele KI
- Bisher:
 - Vollständig beobachtbar vs. teilweise beobachtbar
 - Deterministisch vs. stochastisch
 - Episodisch vs. sequenziell
 - Statisch vs. dynamisch
 - Diskret vs. stetig
 - Einzelagent vs. Multiagent



Vorausschau

- Spielbaumsuche = Suchverfahren als Grundlage für eine Spiele KI
- Im folgenden wird es etwas schwerer:
 - Vollständig beobachtbar vs. teilweise beobachtbar
 - Deterministisch vs. stochastisch
 - Episodisch vs. sequenziell
 - Statisch vs. dynamisch
 - Diskret vs. stetig
 - Einzelagent vs. Multiagent



Fragen?

