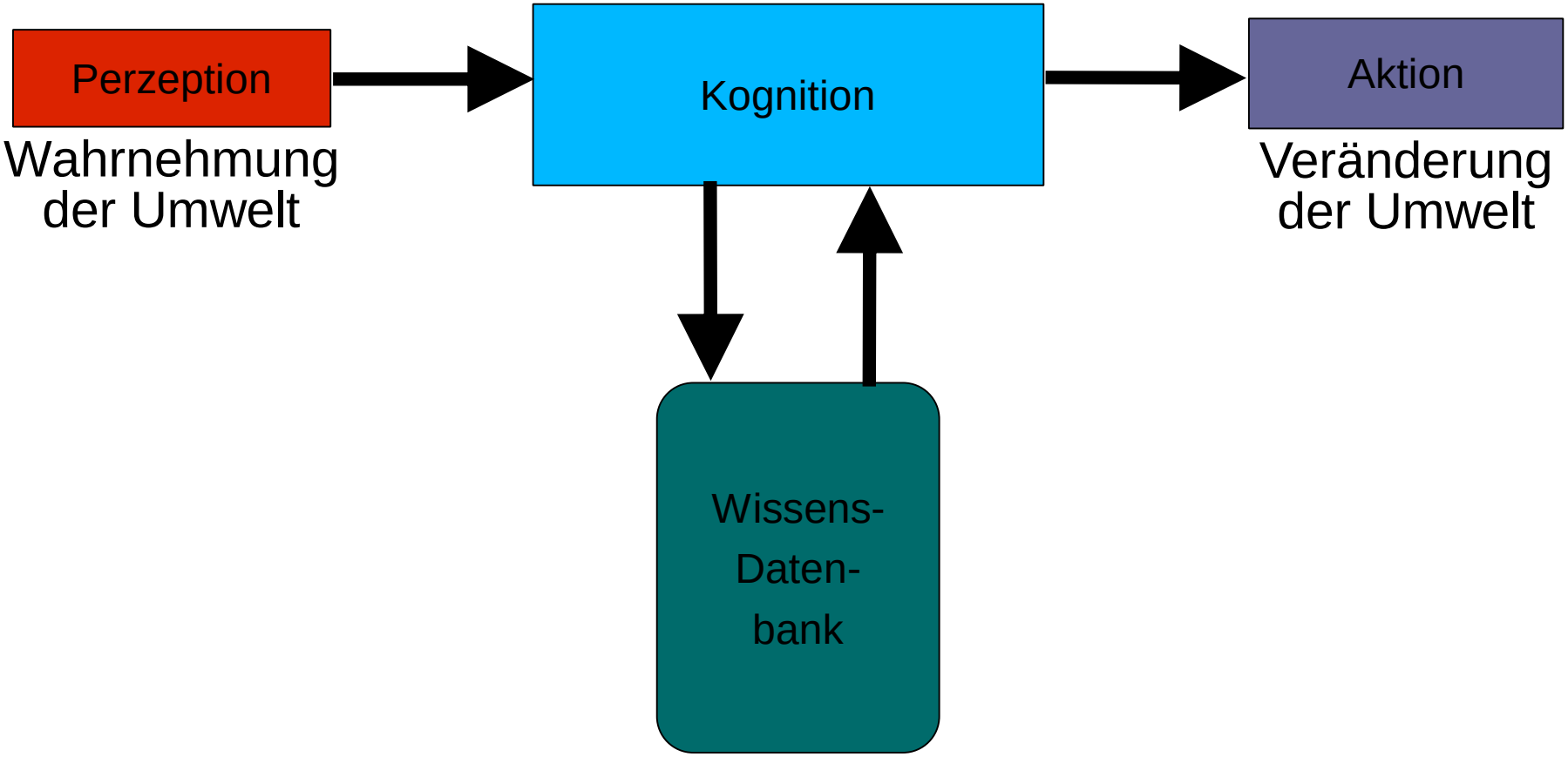
Kognitive Systeme

Wissen und Planung II

Montag, 10. Juli 2016

Elemente eines Kognitiven Systems

Ingenieursmäßiger Ansatz:



Planungssprachen

- STRIPS
- ADL

Repräsentation von Plänen

Wie kann man Probleme so formulieren, dass

- ein Lösungsplan einfach zu erstellen ist?
- die Existenz einer Lösung bewiesen / widerlegt werden kann?

➔ Einschränkungsregeln für formelle Spezifikation von

- Zuständen
- Zielen
- Aktionen

- Beispiele solcher Regelsysteme: STRIPS, ADL

STRIPS

- „**ST**anford **R**esearch **I**nstitute **P**roblem **S**olver“
- Sprache für die Eingaben in das gleichnamige Planungssystem

- „Urvater“ vieler Planungssysteme (Stanford 1971)
- sehr einfach aufgebaut
- teilweise eingeschränkt wegen „closed world assumption“

STRIPS: Zustände

Repräsentation von **Zuständen**:

- Konjunktion positiver aussagenlogischer Literale

$$Blau \wedge Rund$$

- Literale erster Ordnung (L1)

$$IstTasse(T_1) \wedge IstUntertasse(U_1) \wedge StehtAuf(T_1, U_1)$$

- L1 müssen funktions- und variablenfrei sein!

~~$IstTasse(x)$~~ ~~$IstTasse(ObjektAuf(U_1))$~~

- **Geschlossene Welt**:
  - Jeder Zustand ist vollständig bekannt.
  - Nicht im Zustand vorkommende Literale werden implizit als falsch angenommen!

STRIPS: Closed-World Assumption

- Keine Negationen im Zustand und Vorbedingungen, aber in Nachbedingung benötigt
- KEINE Negationen in **Vorbedingung**:

$b \Rightarrow$ *b* **muß** im Weltmodell vorkommen
a **kann** im Weltmodell vorkommen

- Negationen in **Nachbedingung**:

$\neg c \wedge d \Rightarrow$ **entferne** *c* aus dem Weltmodell
füge *d* dem Weltmodell **hinzu**

STRIPS: Ziele

Repräsentation von **Zielen**:

- Ziel: Teilweise spezifizierter Zustand
- Angegeben als Konjunktion von positiven Literalen

$$StehtAuf(T_1, U_1) \wedge StehtAuf(U_1, Tisch)$$

- Ein Zustand *S erfüllt* ein Ziel *Z*, wenn er alle Literale in *Z* enthält

$$StehtAuf(T_1, U_1) \wedge StehtAuf(U_1, Tisch) \wedge StehtAuf(Teller, Tisch)$$

erfüllt

$$StehtAuf(T_1, U_1) \wedge StehtAuf(U_1, Tisch)$$

STRIPS: Aktionen

Aktionen werden angegeben durch

- Aktionsname
- Parameter
- Vorbedingungen
- Effekte

$$A = (N_A, P_A, V_A, E_A)$$

- Beispiel:

$$\begin{aligned} N_A &= StelleAuf, \\ P_A &= (Obj_1, Obj_2), \\ V_A &= IstUntersatz(Obj_2) \wedge Auf(Obj_1, Tisch) \wedge (Obj_2, Tisch), \\ E_A &= \neg Auf(Obj_1, Tisch) \wedge Auf(Obj_1, Obj_2). \end{aligned}$$

STRIPS: Aktionen

- Alternative Schreibweise:

Aktion(*StelleAuf*(Obj_1, Obj_2),
  Vorbed: *IstUntersatz*(Obj_2) $\wedge$
  Auf(Obj_1, *Tisch*) $\wedge$
  Auf(Obj_2, *Tisch*),
  Effekt: $\neg$*Auf*(Obj_1, *Tisch*) $\wedge$
  Auf(Obj_1, Obj_2))

- Manchmal auch Effekt aufgeteilt in
  - *Additionsliste* (positive Literale)
  - *Deletionsliste* (negative Literale)

STRIPS: Semantik

Anwendbarkeit:

- Eine Aktion ist *anwendbar* auf allen Belegungen M, die die Vorbedingung V_A erfüllen

Ergebnis:

- Das *Ergebnis* M' der Ausführung einer Aktion A auf einer Belegung M erhält man durch
  - Entfernen aller negativen Literale des Effekts E_A aus M
  - Hinzufügen aller positiven Literale aus E_A zu M

STRIPS: Einschränkungen

Einschränkungen von STRIPS:

- Literale müssen *funktionsfrei* sein
  - endliche Grammatik
  - jede Aktion kann als endliche aussagenlogische Konjunktion dargestellt werden
  - Lösbarkeitsbeweis einfach
- Nur positive Literale / *„geschlossene Welt“*
  - einfachere Planung
  - aber: Falsch-Sachverhalte nur implizit
- Keine Quantisierung

➔ STRIPS-Aussagen oft lang und unübersichtlich

ADL

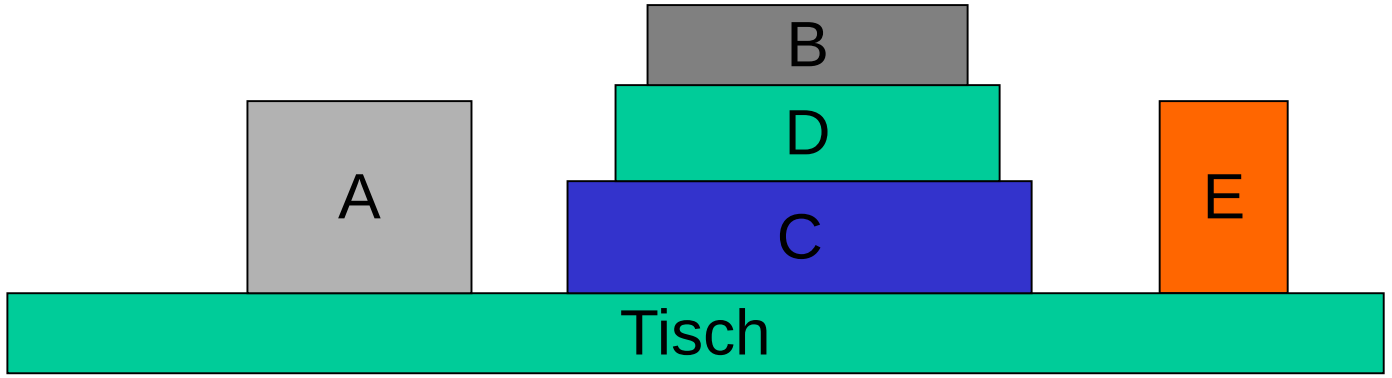
Action **D**escription **L**anguage:

- Weiterentwicklung von STRIPS
- **„offene Welt“**: alle nicht angegebenen Literale gelten als unbekannt
- **negative** Literale und **Disjunktionen** erlaubt
- Effekt $\neg P \wedge Q$:
  - Füge $\neg P$ und Q hinzu
  - lösche P und $\neg Q$
- Gleichheits-Prädikat „=“ eingebaut
  - *StelleAuf*(T_1,T_1) nicht mehr möglich

Typüberprüfung

- Typüberprüfung nötig für Ausführbarkeit bestimmter Aktionen
- in STRIPS nur explizit (als Prädikat):
  IstUntersetzer(x)
  IstTasse(y)
- Häufig weggelassen (Schreibarbeit!), aber eigentlich nötig
- **Aktion**(*StelleAuf(*Obj_1*: Untersatz,* Obj_2*: Manipulierbar),*
  Vorbed: *Auf(*Obj_1*, Tisch)* $\wedge$ *Auf(*Obj_2*, Tisch),*
  Effekt: $\neg$*Auf(*Obj_2*, Tisch)* $\wedge$ *Auf(*Obj_2*,* Obj_1*))*

Beispiel: Blockwelt



- Klassisches Beispiel für Planungsprobleme
- Besteht aus
  - einem Tisch
  - *n* Quadern
- Je ein Quader kann auf einem anderen oder auf dem Tisch stehen
- Problem: Aktionsabfolge, um C auf E zu stellen

Beispiel: Blockwelt

- Verfügbare Prädikate (in STRIPS):
  - *Auf(b,x)*: Block *b* liegt auf *x* (Block oder Tisch)
  - *Frei(x)*: Nichts liegt auf *x* (Block oder Tisch)

- **Aktion**(*Bewege*(*b*,*x*,*y*),
  Vorbed: $Auf(b,x) \wedge Frei(b) \wedge Frei(y)$,
  Effekt: $Auf(b,y) \wedge Frei(x) \wedge \neg Auf(b,x) \wedge \neg Frei(y)$)

Aktion(*BewegeAufTisch*(*b*,*x*),
Vorbed: $Auf(b,x) \wedge Frei(b)$,
Effekt: $Auf(b,Tisch) \wedge Frei(x) \wedge \neg Auf(b,x)$)

Planungsstrategien

Suche im Zustandsraum - A*-Suche

Partial-Order-Planning

Planungsgraphen

Suche im Zustandsraum

- Intuitiver Ansatz: Von Startzustand ausgehend für alle ausführbaren Aktionen den daraus entstehenden Zustand ermitteln
- In den neuen Zuständen wiederum jeweils alle möglichen Aktionen ausführen
- Man erhält einen Baum, der alle erreichbaren Zustände enthält
- Solange ausführen, bis der gewünschte Zielzustand entsteht

➔ Aktionenfolge, die dorthin geführt hat, tatsächlich ausführen

Suche im Zustandsraum

- Suche im Zustandsraum ist **einfach**:
  - Keine Funktionssymbole
    - ➔ endlicher Zustandsraum
    - ➔ Standard-Algorithmen zur Baumsuche
  - Transformation der Vorbedingungen
    - ➔ Effekte bijektiv
    - ➔Suche auch rückwärts möglich

- **ABER:**
  - Zustandsraum oft *sehr* groß
  - Naive Suche sehr ineffizient
  - Benötigt:
    - gute Heuristik oder
    - Segmentierung in Teilprobleme

Suche beschleunigen

- Suchbaum hat schlimmstenfalls 2^n Knoten (n=Anzahl der Zustandsvariablen)
- Naive Suche nur für sehr kleine Instanzen praktikabel
- Suche muss für realistisch komplexe Situationen effizienter durchgeführt werden
- Lösungsansatz: gezielt suchen
- **Heuristik** verwenden: „welche Suchrichtung scheint am ehesten zum Ziel zu führen?“

Heuristikbasierte Suche

Heuristik:

- Eine Funktion, die für jeden Zustand einen **optimistischen** Schätzwert der „Entfernung“ zum Ziel liefert
- Darf Distanz unter-, aber nicht überschätzen!

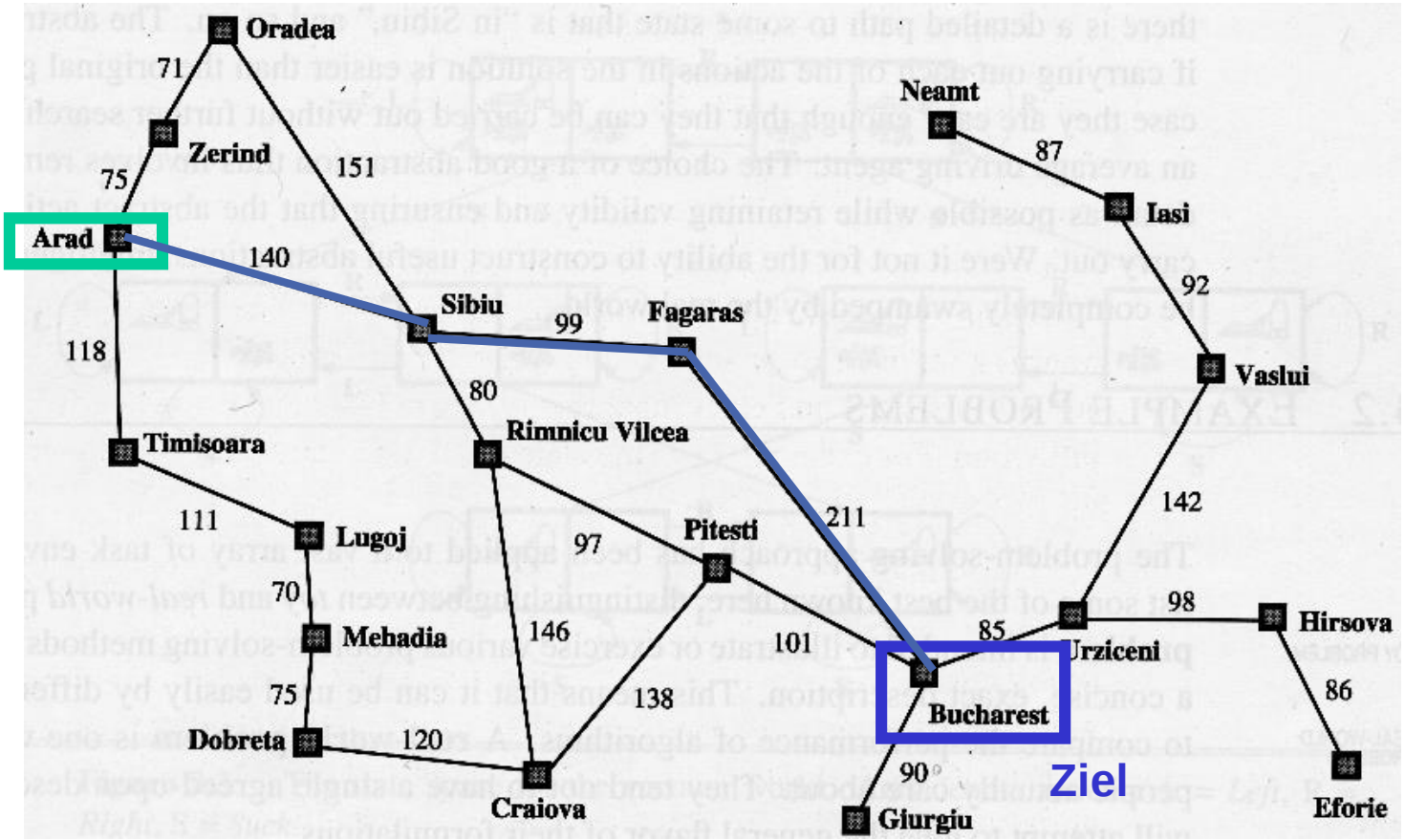
Einfacher Ansatz (**Greedy**-Algorithmus):

- wenn Heuristik zur Verfügung steht: immer von dem Zustand aus weitersuchen, der laut Heuristik dem Ziel am nächsten ist
- Kann Suche deutlich beschleunigen, wenn die Heuristik gut ist, kann aber auch zu großen Umwegen führen
- Liefert i. A. nicht den kürzesten Weg zum Ziel!

Beispiel: Greedy

Straßenkarte von Rumänien

Arad	366	Hirsova	151	Rimnicu Vilcea	193
Bukarest	0	Iasi	226	Sibiu	253
Craiova	160	Lugoj	244	Timisoara	329
Dobreta	242	Mehadia	241	Urziceni	80
Eforie	161	Neamt	234	Vaslui	199
Fagaras	176	Oradea	380	Zerind	374
Giurgiu	77	Pitesti	100		



A* - Suche

A*-Algorithmus

- Allgemeiner Algorithmus für die heuristikgestützte Suche in Graphen
- Liefert kürzesten Weg zum Ziel
- Normalerweise sehr effizient, abhängig von der Güte der Heuristik
- Expandiert den Knoten, für den die Summe

(*geschätzte Entfernung zum Ziel*)

+ (*bisher zurückgelegte Strecke vom Start*)

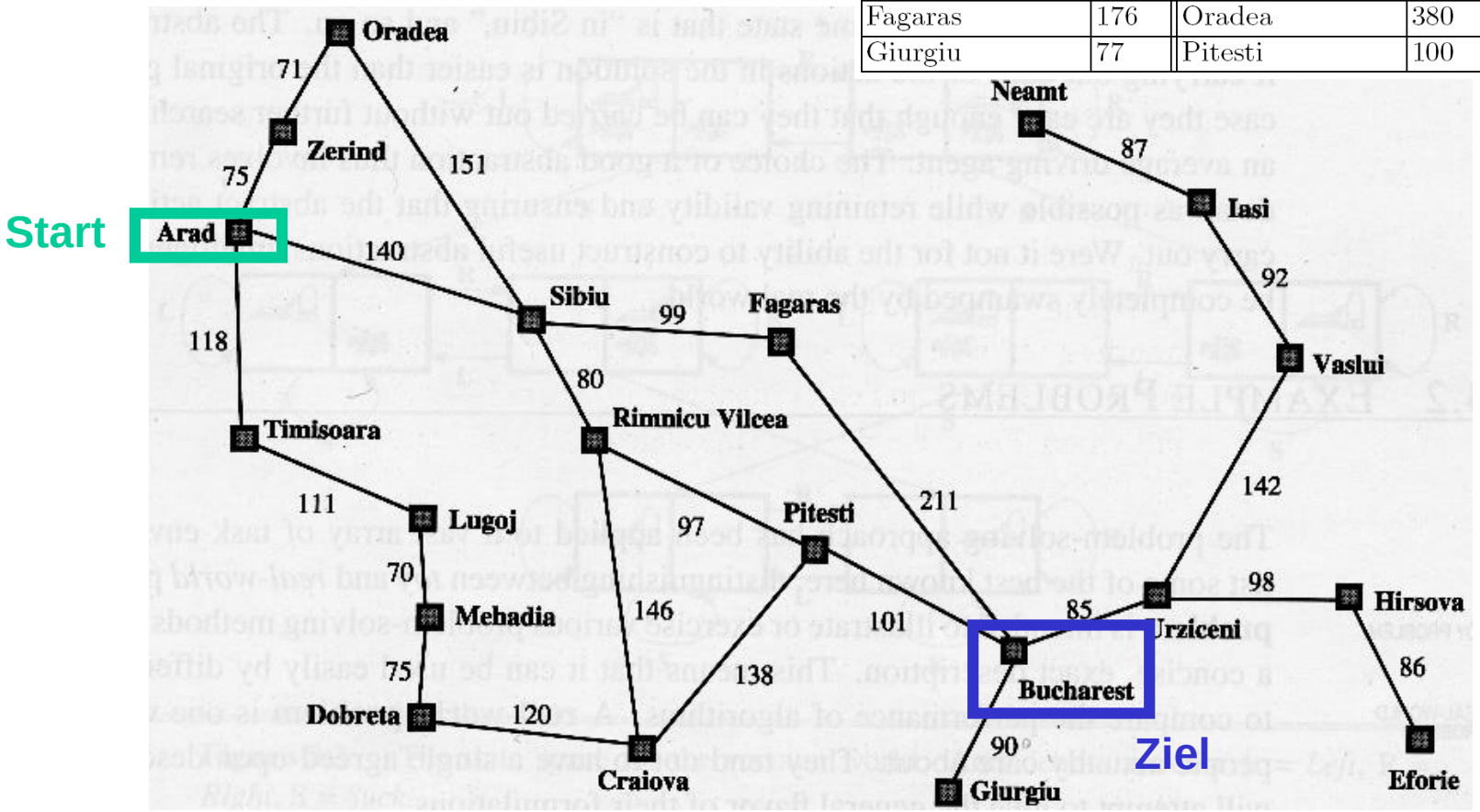
minimal ist.

- Es werden immer alle nicht expandierten Knoten

Beispiel A*-Algorithmus

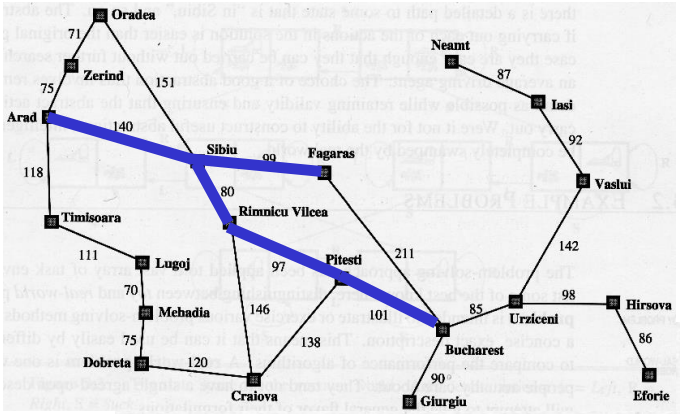
Straßenkarte von Rumänien

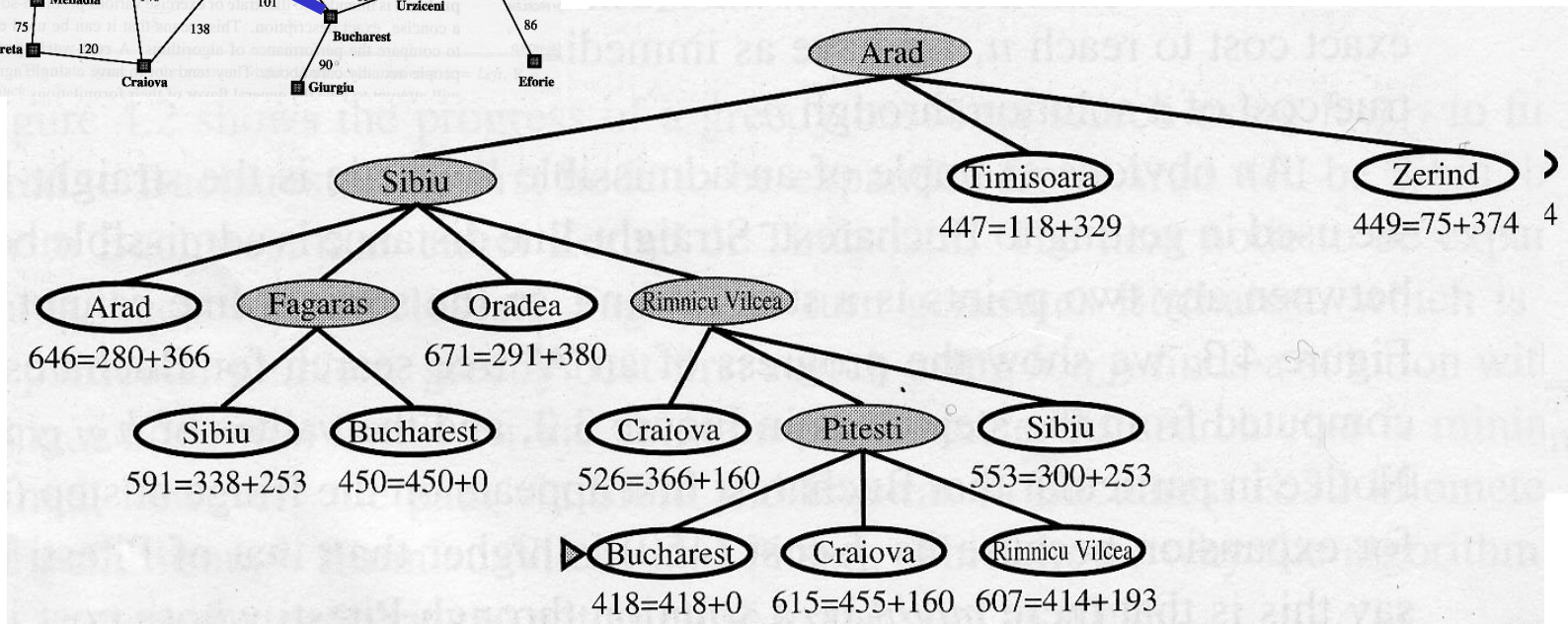
Arad	366	Hirsova	151	Rimnicu Vilcea	193
Bukarest	0	Iasi	226	Sibiu	253
Craiova	160	Lugoj	244	Timisoara	329
Dobreta	242	Mehadia	241	Urziceni	80
Eforie	161	Neamt	234	Vaslui	199
Fagaras	176	Oradea	380	Zerind	374
Giurgiu	77	Pitesti	100		



Beispiel A*-Algorithmus

Arad	366	Hirsova	151	Rimnicu Vilcea	193
Bukarest	0	Iasi	226	Sibiu	253
Craiova	160	Lugoj	244	Timisoara	329
Dobreta	242	Mehadia	241	Urziceni	80
Eforie	161	Neamt	234	Vaslui	199
Fagaras	176	Oradea	380	Zerind	374
Giurgiu	77	Pitesti	100		





Woher kommt die Heuristik?

- **Allgemeine** Heuristik für Planungsprobleme **existiert nicht**
- Für einzelne Aufgaben gibt es spezifische Heuristiken
  - z.B. bei der Routenplanung die euklidische Distanz der Knoten

Suche im Zustandsraum:

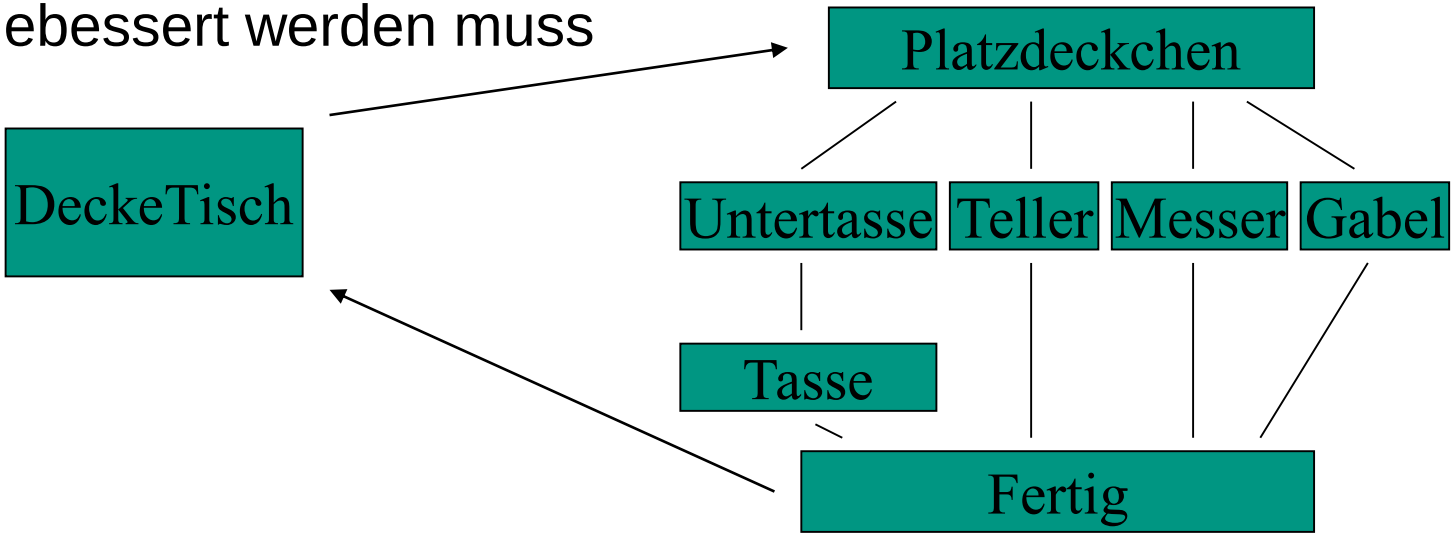
- Spontane Idee: Anzahl noch zu erfüllender Literale für Zielzustand
- Aber: Wenn eine Aktion mehrere Literale erfüllt, überschätzt die Heuristik evtl. den verbleibenden Aufwand

  ➔ ungültige Heuristik (darf unter-, aber nicht überschätzen)
- Der **Planungsgraph** liefert meist gute Heuristik (später mehr dazu)

Partial-Order-Planning

Partial-Order-Planning

- Bisher: Plan entspricht fester Folge von Aktionen
- Oft nicht nötig, Reihenfolge der Aktionen genau festzulegen
- Stattdessen: Anordnung unabhängiger Aktionen freistellen, nur soweit ordnen wie nötig ➔ teilweise geordneter Plan
- Unabhängige Aufgaben können in beliebiger Reihenfolge und evtl. auch parallel bearbeitet werden
- Planung ist flexibler, insbesondere wenn etwas schiefgeht und nachgebessert werden muss



Partial-Order-Planning

- Planung erfolgt jetzt nicht im Zustandsraum, sondern im Raum der gültigen PO-Pläne
- Ein PO-Plan besteht aus
  - Aktionen
  - Ordnungsbedingungen
  - Abhängigkeiten
  - offenen Vorbedingungen
- Zwei Pseudo-Aktionen Start und Ziel,
  - Effekte von Start sind die Literale des Startzustandes
  - Vorbedingungen von Ziel die des Zielzustandes
- PO-Plan zu Beginn enthält Start und Ziel,
  - die Ordnungsbedingung Start < Ziel
  - keine Abhängigkeiten
  - die offene Vorbedingungen von Ziel

Partial-Order-Planning

- Solange noch eine offene Vorbedingung existiert, die nicht durch den Startzustand erfüllt wird, wiederhole:
  - Wähle eine beliebige offene Vorbedingung P einer Aktion B
  - Für alle Aktionen A, deren Effekte P erfüllen, erstelle aus dem alten einen neuen PO-Plan so:
    - Ergänze die Abhängigkeiten um A $\rightarrow_P$ B, die Ordnungsbedingungen um A < B
    - Ist A noch nicht im Plan enthalten, füge es zu den Aktionen hinzu, außerdem die Ordnungsbedingungen Start < A und A < Ziel
- Das wird wiederum mit allen neuen Plänen weitergeführt

Partial-Order-Planning

- Damit ein PO-Plan gültig ist, muss er die Ordnungsbedingungen und Abhängigkeiten berücksichtigen
- Bei den Ordnungsbedingungen darf kein Zyklus A < B < C < A existieren
- Wenn eine Abhängigkeit A $\rightarrow_P$ B besteht, darf keine Aktion C zwischen A und B stattfinden, die P negiert.
  - Aus einem Plan, bei dem diese Möglichkeit durch Hinzufügen einer Aktion oder einer Abhängigkeit entsteht, werden zwei neue Pläne generiert; der eine enthält zusätzlich die Ordnungsbedingung C < A, der andere B < C

Partial-Order-Planning

Beschleunigung

- durch geschicktes Wählen des zu erfüllenden Literals.
- Das Literal wählen, das durch die wenigsten Aktionen erfüllt wird

  ➔ geringerer Verzweigungsgrad
- Genaue Berechnung zu aufwändig
- In der Praxis bereits sehr hilfreich, nur zwei Spezialfälle zu beachten:
  - Gar keine Möglichkeit, das Literal zu erfüllen: Man kann hier die Suche abbrechen
  - Genau eine Möglichkeit: Muss sowieso gewählt werden, also am besten sofort, dadurch evtl. weitere Einschränkungen bei zukünftigen Entscheidungen

POP-Beispiel: Schuhe anziehen 1

Beispiel

- Startzustand: { }
- Zielzustand: { LinkerSchuhAn $\wedge$ RechterSchuhAn }
- Aktion (LinkenSchuhAnziehen, ( ), Vorr: LinkeSockeAn, Eff: LinkerSchuhAn)
- Aktion (RechtenSchuhAnziehen, ( ), Vorr: RechteSockeAn, Eff: RechterSchuhAn)
- Aktion (LinkeSockeAnziehen, ( ), Vorr: { } , Eff: LinkeSockeAn)
- Aktion (RechteSockeAnziehen, ( ), Vorr: { } , Eff: RechteSockeAn)

POP-Beispiel: Schuhe anziehen 2

- Initialer Plan:
  - Aktionen: Start, Finish
  - Ordnungsbed.: Start < Finish
  - Abhängigkeiten: --
  - Offene Vorbed.: LinkerSchuhAn, RechterSchuhAn

- Wähle bel. offene Vorbedingung: LinkerSchuhAn
- Einzige Aktion, die die Vorbed. erfüllt, ist LinkenSchuhAnziehen

➔ zum Plan hinzufügen

POP-Beispiel: Schuhe anziehen 3

- Neuer Plan:
  - Aktionen: Start, Finish, LinkenSchuhAnziehen
  - Ordnungsbed.:
    - Start < Finish,
    - Start < LinkenSchuhAnziehen,
    - LinkenSchuhAnziehen < Finish
  - Abhängigkeiten: LinkenSchuhAnziehen $\rightarrow_{\text{LinkerSchuhAn}}$ Finish
  - Offene Vorbed.: LinkeSockAn, RechterSchuhAn

- Wähle wieder eine offene Vorbedingung: LinkeSockeAn
- Füge Aktion LinkeSockeAnziehen zum Plan hinzu

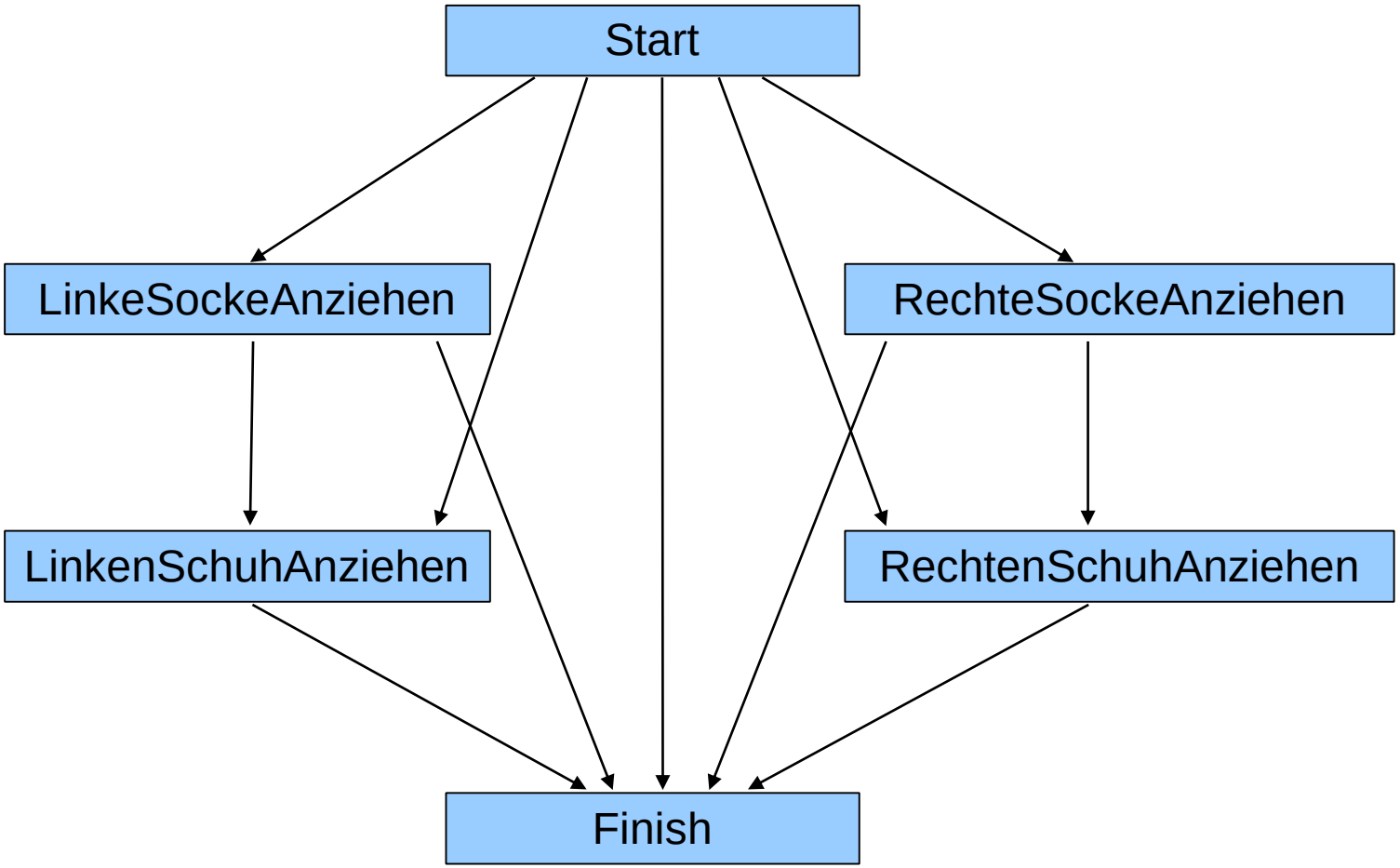
POP-Beispiel: Schuhe anziehen 4

- Neuer Plan:
  - Aktionen: Start, Finish, LinkenSchuhAnziehen, LinkeSockeAnziehen
  - Ordnungsbed.:
    - Start < Finish,
    - Start < LinkenSchuhAnziehen,
    - LinkenSchuhAnziehen < Finish,
    - Start < LinkeSockeAnziehen,
    - LinkeSockeAnziehen <Finish,
    - LinkeSockeAnziehen < LinkenSchuhAnziehen
  - Abhängigkeiten:
    - LinkenSchuhAnziehen $\rightarrow_{\text{LinkerSchuhAn}}$ Finish,
    - LinkeSockeAnziehen $\rightarrow_{\text{LinkeSockeAn}}$ LinkenSchuhAnziehen
  - Offene Vorbed.: RechterSchuhAn
- Für rechte Seite analog

POP-Beispiel: Schuhe anziehen 5

- Fertiger Plan:
  - Aktionen: Start, Finish, LinkenSchuhAnziehen, LinkeSockeAnziehen, RechtenSchuhAnziehen, RechteSockeAnziehen
  - Ordnungsbed.: Start < Finish, Start < LinkenSchuhAnziehen, LinkenSchuhAnziehen < Finish, Start < LinkeSockeAnziehen, LinkeSockeAnziehen < Finish, LinkeSockeAnziehen < LinkenSchuhAnziehen, Start < RechtenSchuhAnziehen, RechtenSchuhAnziehen < Finish, Start < RechteSockeAnziehen, RechteSockeAnziehen < Finish, RechteSockeAnziehen < RechtenSchuhAnziehen
  - Abhängigkeiten:
    - LinkenSchuhAnziehen $\rightarrow_{\text{LinkerSchuhAn}}$ Finish,
    - LinkeSockeAnziehen $\rightarrow_{\text{LinkeSockeAn}}$ LinkenSchuhAnziehen,
    - RechtenSchuhAnziehen $\rightarrow_{\text{RechterSchuhAn}}$ Finish,
    - RechteSockeAnziehen $\rightarrow_{\text{RechteSockeAn}}$ RechtenSchuhAnziehen
  - Offene Vorbed.: --

POP-Beispiel: Schuhe anziehen 6



POP-Beispiel: Schuhe anziehen 7

- Es existieren 6 Möglichkeiten, die Aktionen anzuordnen
- Tatsächliche Reihenfolge der Ausführung bleibt bei POP so weit wie möglich offen
- Planung war in diesem Beispiel einfach, für jede Vorbedingung existierte nur eine mögliche Aktion zur Erfüllung
- Wenn mehrere Aktionen möglich, Verzweigung des Planungsvorganges ➔ es entsteht ein **Baum** von möglichen Plänen

Planungsgraphen

Planungsgraphen

- Löst eine vereinfachte Version der Planungsaufgabe
- Benötigt dafür nur polynomielle Zeit
- Erledigt vereinfachte Aufgabe mit weniger oder gleich vielen Aktionen, als für die tatsächliche Aufgabe nötig
- Liefert also **optimistische** Schätzung für Zahl der noch nötigen Aktionsschritte ➔ **gültige Heuristik**
- Bildet außerdem Grundlage für Graphplan-Algorithmus zur tatsächlichen Planung

Planungsgraphen

- Planungsgraph ist Folge von Abschnitten, die jeweils einem Zeitschritt im Plan entsprechen
- **Abwechselnde** Schritte:
  - Menge der **Literale**, die zu diesem Zeitpunkt wahr sein könnten
  - Menge der **Aktionen**, deren Vorbedingungen zu diesem Zeitpunkt erfüllt sein könnten
- Im darauf folgenden Abschnitt dann wiederum aus den möglichen Aktionen resultierende Literale usw.

Planungsgraphen

Ausserdem:

- Für jedes Literal zusätzlich eine **Pseudoaktion**, die es **unverändert** erhält
- Zu jedem Zeitpunkt die Information, welche Paare von Literalen bzw. Aktionen sich **gegenseitig ausschließen**:

  Gegenseitiger Ausschluss = engl. mutual exclusion (**mutex**)

- Zwischen sich ausschließenden Literalen bzw. Aktionen besteht im Planungsgraph ein sog. **mutex-link**

Planungsgraphen

Mutex-Links:

- Zwei **Literale** schließen sich gegenseitig aus, wenn
  - sie **komplementär** sind (z.B. *P* und ¬*P)*
  - sie nur aus sich gegenseitig **ausschließenden Aktionen resultieren** können

- Zwei **Aktionen** schließen sich aus, wenn:
  - ihre **Effekte** ein **komplementäres Literal** enthalten
  - durch den Effekt der einen eine **Vorbedingung** der anderen **zerstört** wird
  - die **Vorbedingungen** der einen ein Literal enthalten, das ein Literal aus den **Vorbedingungen** der anderen **ausschließt**

Planungsgraphen

Konstruktion des Planungsgraphen:

- Erster Abschnitt:
  - Alle Literale des Startzustandes
- Zweiter Abschnitt:
  - Alle Aktionen, die möglich sind (auch erhaltende Aktionen).
  - Sich gegenseitig **ausschließende Aktionen** durch **mutex-link** verbinden
- Dritter Abschnitt:
  - Alle Literale, die aus den Aktionen resultieren.
  - Sich gegenseitig **ausschließende Literale** durch **mutex-link** verbinden
- Vierter Abschnitt:
  - Alle jetzt wieder möglichen Aktionen usw.
- ...
- Fortsetzen, bis sich bei den **Literalen** keine Veränderung mehr ergibt

Planungsgraphen - Beispiel

Startzustand:

$$UebungsblattFertig \wedge \neg NameDrauf \wedge \neg BlattGetackert \wedge \neg BlattAbgegeben$$

Zielzustand:

$$NameDrauf \wedge BlattGetackert \wedge BlattAbgegeben$$

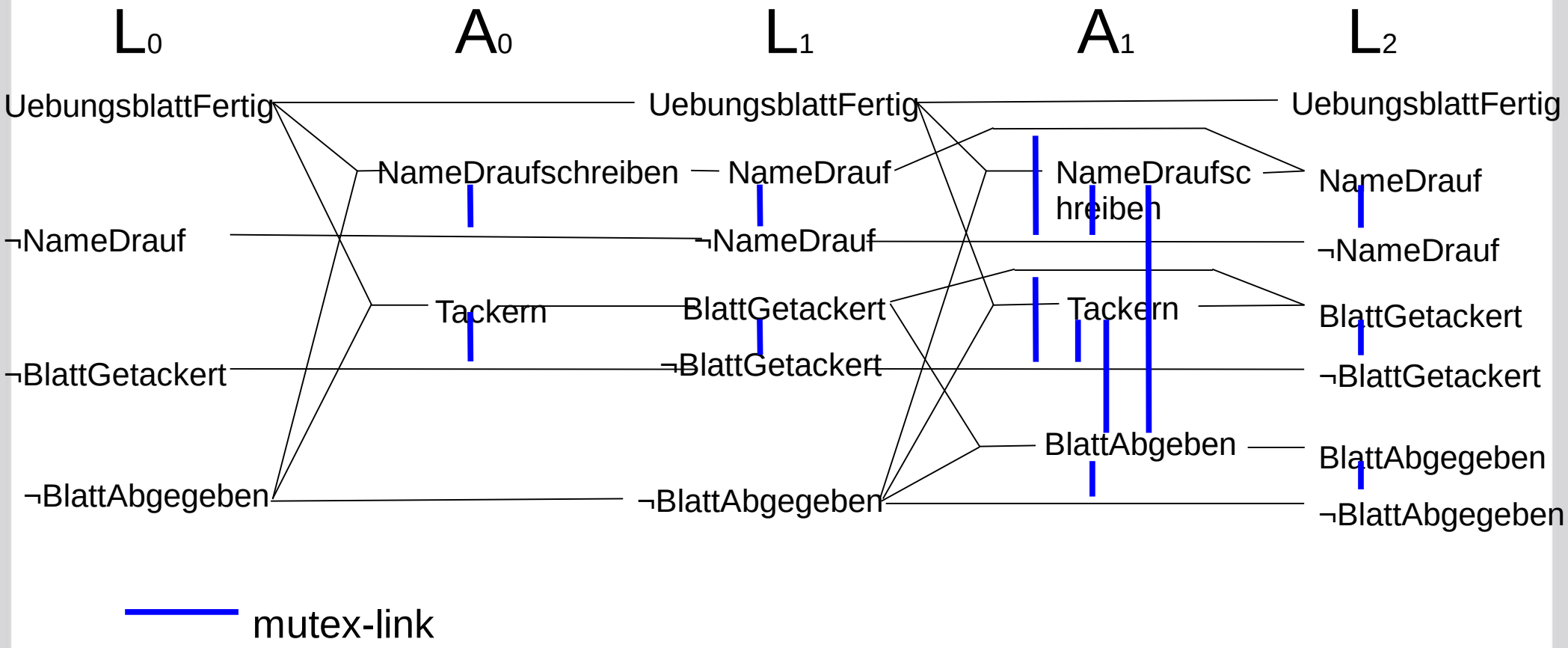
Aktionen:

$$\begin{aligned} &Action(&& Tackern(), \\ &&& Vorbed: && UebungsblattFertig \wedge \neg BlattAbgegeben \\ &&& Effekt: && BlattGetackert \\ &&&) \end{aligned}$$

$$\begin{aligned} &Action(&& NameDraufschreiben(), \\ &&& Vorbed: && UebungsblattFertig \wedge \neg BlattAbgegeben \\ &&& Effekt: && NameDrauf \\ &&&) \end{aligned}$$

$$\begin{aligned} &Action(&& BlattAbgeben(), \\ &&& Vorbed: && BlattGetackert \wedge \neg BlattAbgegeben \\ &&& Effekt: && BlattAbgegeben \\ &&&) \end{aligned}$$

Planungsgraphen - Beispiel



Planungsgraphen

Heuristische Informationen aus dem Planungsgraphen

- Nummer des **ersten Abschnittes**, in dem ein Literal vorkommt, ist (optimistischer) Schätzwert für **Anzahl notwendiger Aktionen**, um es zu erfüllen
- Für eine Menge von Literalen gilt die Nummer des ersten Abschnitts, in denen alle vorkommen, ohne dass mutex-link zwischen zweien bestehen.
- Wird gewünschter Zielzustand im Planungsgraph **nicht erreicht**, ist er **unerreichbar**.
- **Umkehrung gilt nicht!**
  Wenn Ziel im Planungsgraph erreichbar, kann es trotzdem unerfüllbar sein (Widersprüche, die aus drei oder mehr Aktionen resultieren, werden nicht durch mutex-links erfasst)

Geometrisches Wissen und Planung

Umweltmodell

Repräsentation von Hindernissen

Bahnplanungsmethoden

Umweltmodell

- Das Umweltmodell eines kognitiven Systems bildet die reale Umwelt auf eine innere Repräsentation ab
- Logisch-semantische Beziehungen genügen für ein System mit Aktorik (Roboter) nicht
- Zusätzlich:
  - Objektmodellierung:
    - (ungefähre) Form von Objekten in der Umwelt
  - Umgebungskarte:
    - Darstellung der Umgebung, Lokalisierung von Objekten, Einteilung in Freiraum und Hindernisse

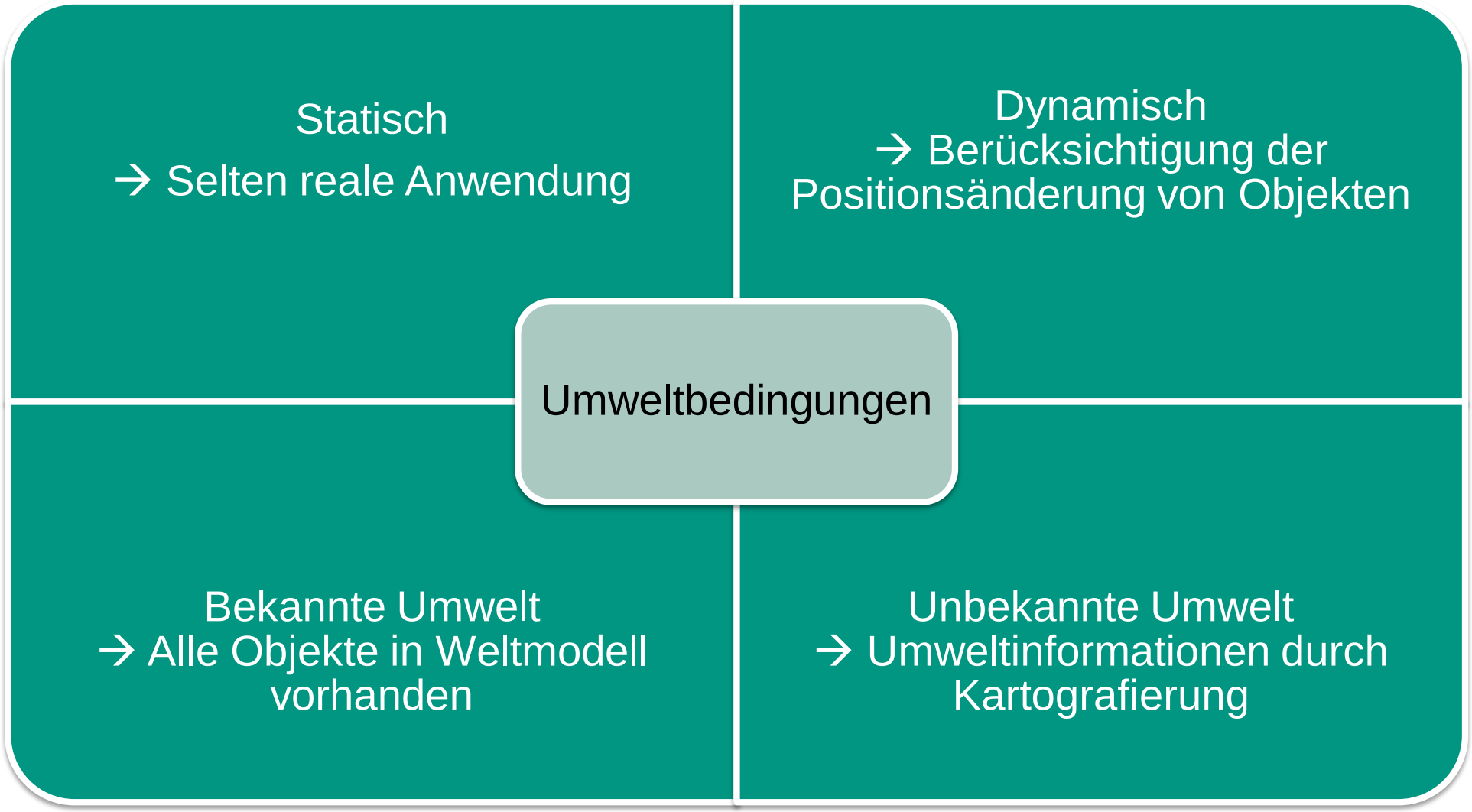
Weltmodelle I

Weltmodelle II

Operationsraum

2-dimensional	2 ½-dimensional	3D-Szene
Alle Objekte auf x-y-Ebene projiziert	Anheben ebener Geometrien senkrecht zur Basisebene „Sweeping“	Objekte der realen Umwelt
Reduzierung auf ebene Geometrien		

Weltmodelle III

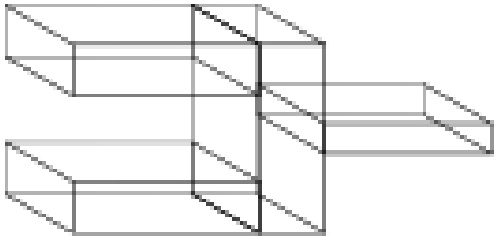


Objektmodellierung

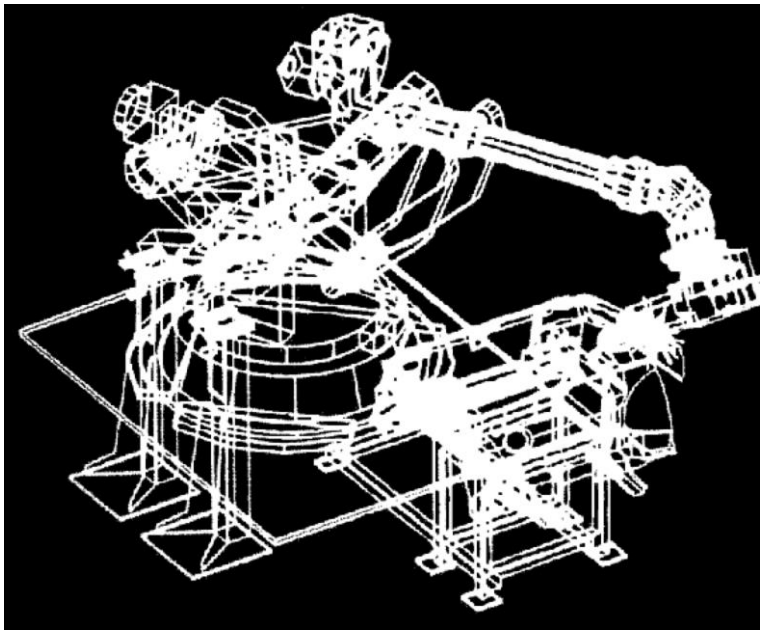
- Darstellung der Objekte der realen Umwelt (Türen, Wände, Hindernisse)
- 3D Darstellung der Umgebung aus Sensor-wahrnehmungen
- Projektion auf x/y-Ebene für Navigation bodengebundener Roboter ausreichend
- Verschiedene Darstellungen
  - Kantenmodelle
  - Oberflächenmodelle
  - Volumenmodelle

Kantenmodelle

- Ermittlung von markanten Punkten
- Verbinden durch geeignete Kanten auf Oberfläche des Objekts



Kantenmodell



Kantenmodell zur Simulation von Schweißaufgaben

Oberflächenmodelle

- Nachbildung der Objektoberflächen
- Darstellung ebener Flächenelemente mit Polygonen
- Gekrümmte Flächenelemente:
  - mathematische Grundflächen (Zylinder, Kegel, Torusflächen)
  - Bezier-Flächen
  - näherungsweise durch Freiformflächen (Patches)

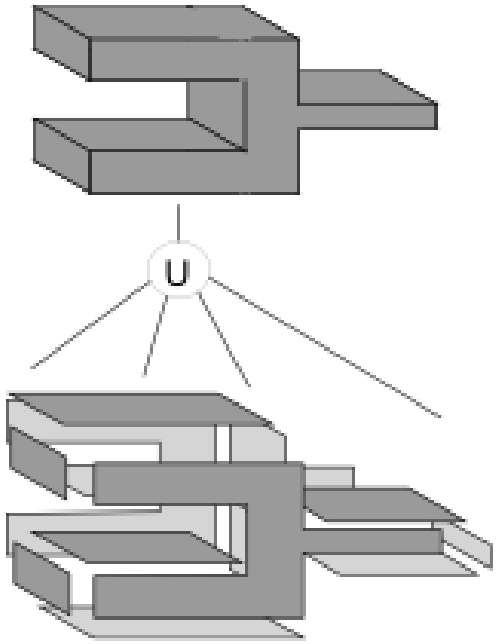
Volumenmodelle

- Unterscheidung von Raumpunkten hinsichtlich ihrer Lage zum Objekt (innen-/außenliegend)
- Repräsentationsmöglichkeiten:
  - Begrenzungsflächenmethode
  - Grundkörperdarstellung
  - Zellzerlegung (Oct-tree)
  - Volumenapproximation
  - Einhüllende Quader
  - Geradensegmente

Begrenzungsflächen

- Beschreibung des Körpers durch umgebende geometrische Elemente (Flächen, Kanten, Punkte)
- festgelegter Richtungssinn

➔Volumen beschrieben durch Flächennormale



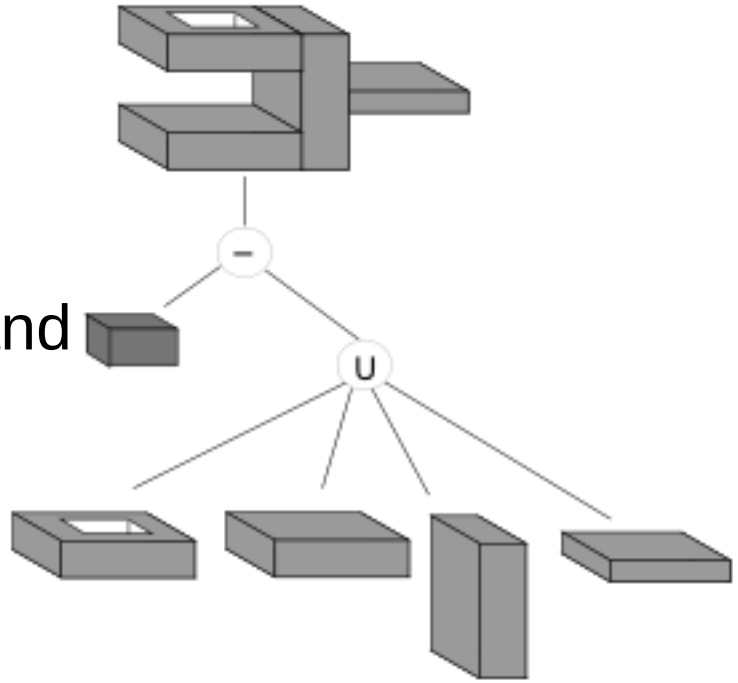
Zerlegung eines Objekts in seine Flächensegmente

Constructive Solid Geometry (CSG)

- Zusammensetzung von Körpern aus parametrierbaren Grundelementen
- relevante Parameter müssen bekannt oder gespeichert sein
- Transformationen zur Positionierung im Koordinatensystem, übliche boolesche Operatoren

$$(\cup, \cap, -)$$

- Nachteil: hoher algorithmischer Aufwand



Umgebungskarten

- Repräsentation der Umgebung, wichtigste Unterscheidung: Freiraum und Hindernisraum
- Grundlage für Bewegungsplanung
- Vorher bekannt oder mit Hilfe gerade gewonnener Sensorinformationen generiert / aktualisiert

Dimensionalität

- Dimensionalität hängt von Anwendung ab
- 3D für direkte Repräsentation der Umgebung,
- Reduktion auf 2D-Bodenplan z.B. für mobile Plattformen
  - eigentlich nicht hinreichend, aber leichtere Bahnplanung
- Höhere Dimension, um direkt im Gelenkwinkelraum zu planen, z.B. für Greifarme, Beine usw.
- Dimension entspricht dann der Anzahl der Freiheitsgrade
- Evtl. zu hochdimensional für Planung ➔ Reduktion auf 3D und Rückberechnung der Gelenkwinkel

Repräsentation von Hindernissen

Bahnplanungsmethoden

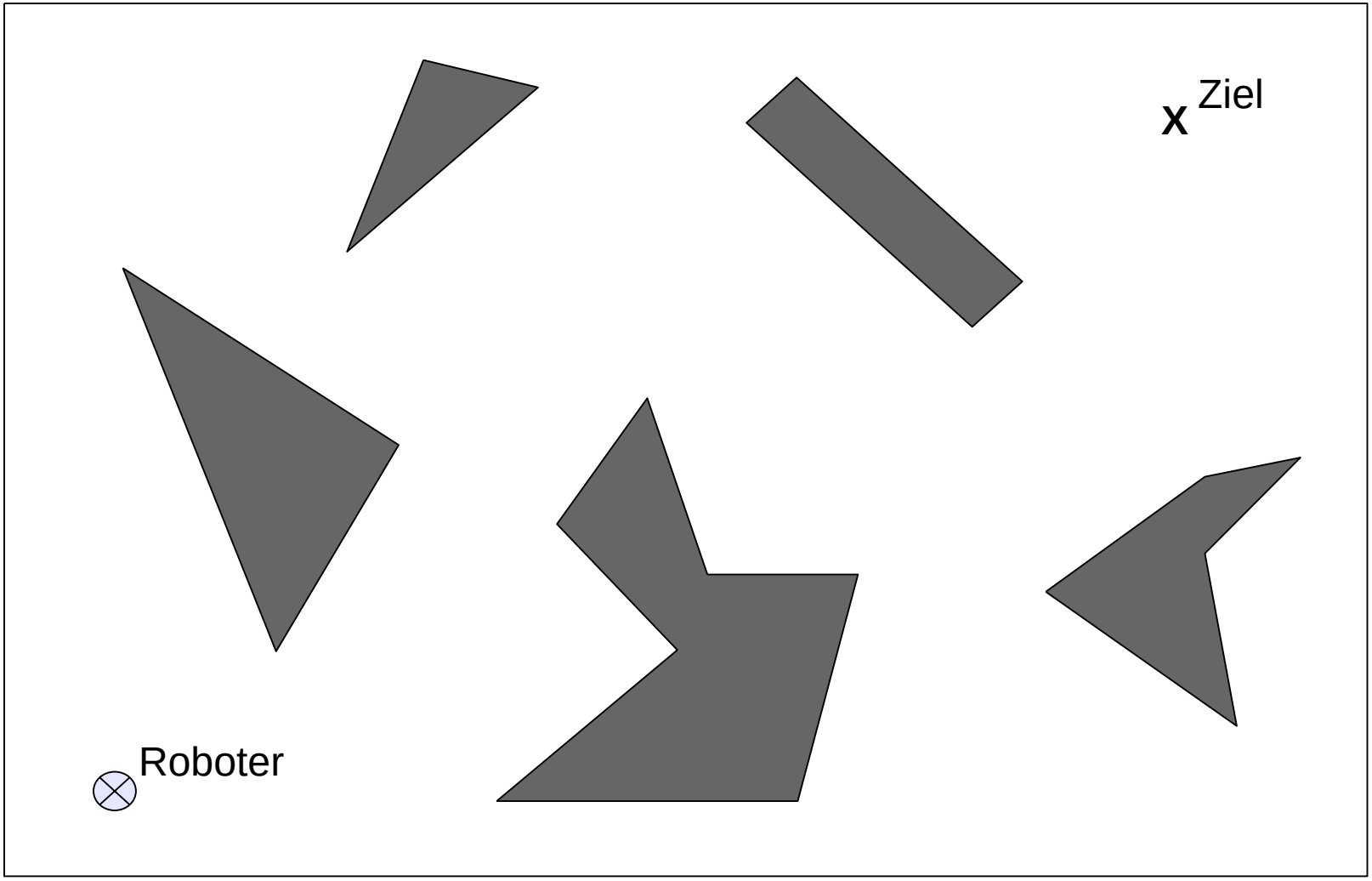
Übersicht

- Konfigurationsraum
- Polygonzerlegung
- Quadtrees
- Sichtgraph
- Voronoi-Diagramme
- Potentialfeldmethode

Freiraum und Hindernisraum

- Wichtigste Unterscheidung, da Bewegung nur im Freiraum möglich
- Normalerweise werden Hindernisse modelliert
- Freiraum ist dann gesamter Raum abzüglich Hindernissen
- Bahnplanung dann im Freiraum

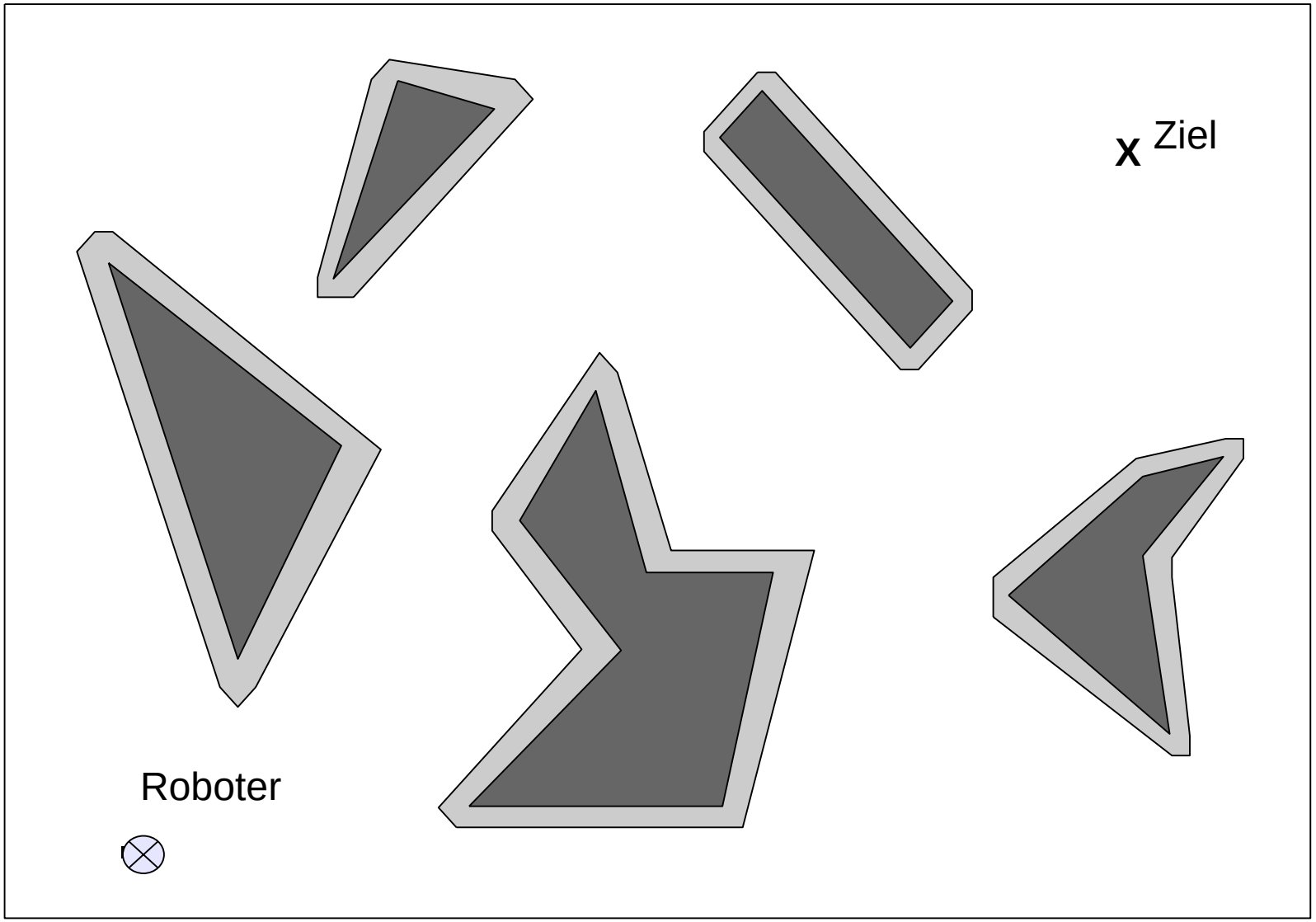
Freiraum und Hindernisraum



Konfigurationsraum

- Konfigurationsraum im Gegensatz zu „realem“ Raum
- Raum, dessen Dimensionen die für die Planung relevanten Parameter sind, z.B. Gelenkwinkelraum
- Kann angepasst werden, um Planung zu unterstützen
- Beispielsweise Hindernisse vergrößern, um Roboter als Punkt behandeln zu können: Wesentliche Vereinfachung der Bahnplanung

Konfigurationsraum

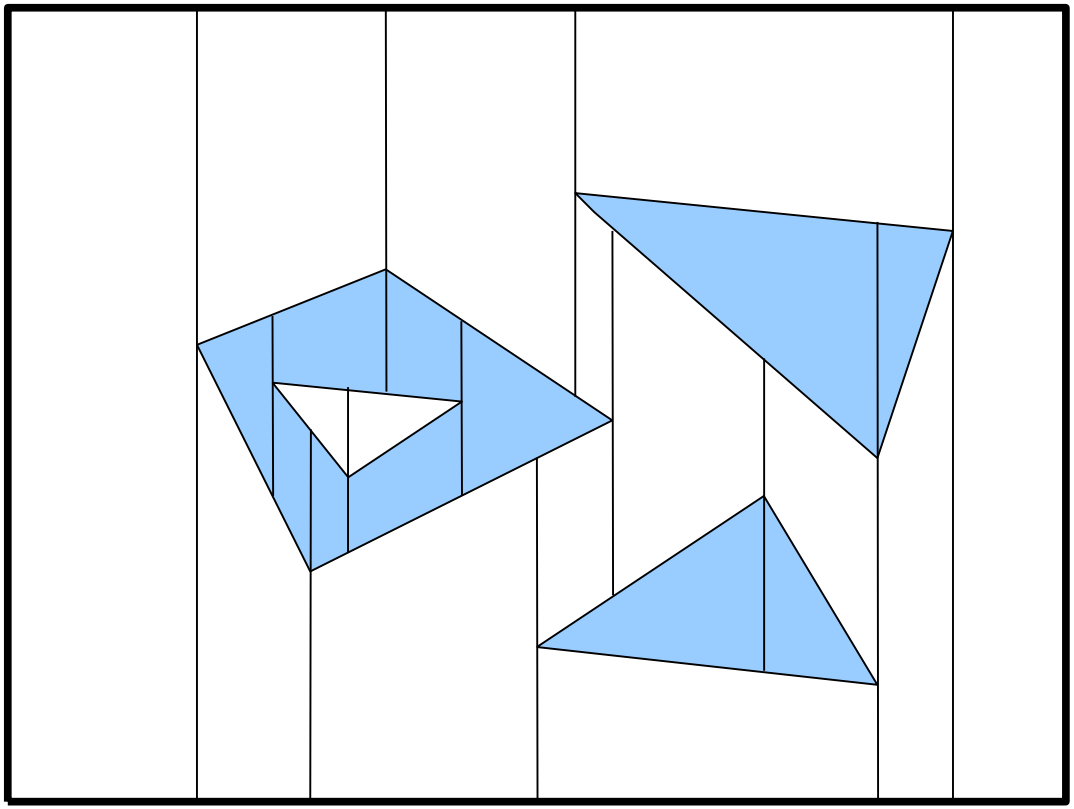


Bahnplanung

- Gegeben: Karte des Raumes, in dem geplant werden soll
- Ansatz: Wegenetz im Raum anlegen, als Graph repräsentieren, Wegsuche im Graphen
  - Polygonzerlegung
  - Quadtrees
  - Voronoi-Diagramme
  - Sichtgraphen
- Alternativer Ansatz: Potentialfeldmethode

Polygonzerlegung

- Sehr effizientes Verfahren für 2D-Karten
- Raum in Polygone zerlegen, indem von jeder Hindernisecke aus eine Trennlinie nach oben und unten bis zur nächsten Hinderniskante gezogen wird
- Polygone sind entweder komplett frei oder komplett unpassierbar
- Wegenetz durch die Freiraumpolygone anlegen

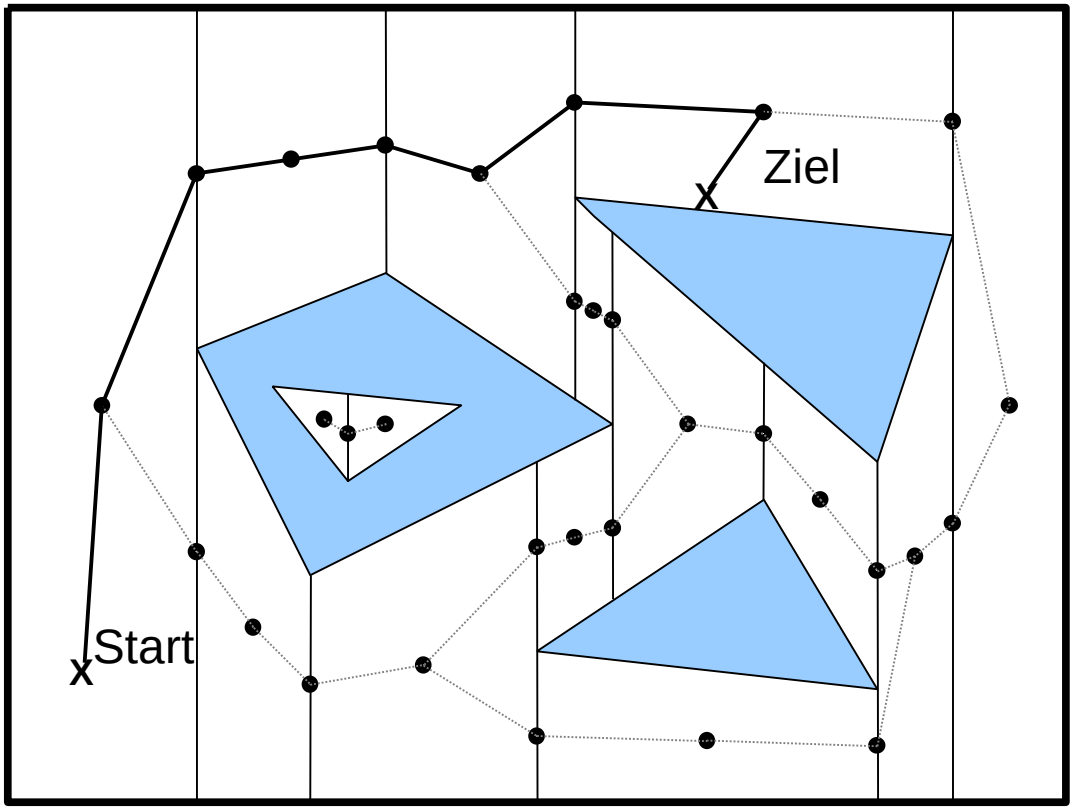


Polygonzerlegung

- Mittelpunkte jedes Freiraumpolygons sind Knoten des Suchgraphen
- Weg zwischen benachbarten Polygonen führt durch Mittelpunkt der gemeinsamen Trennlinie

➔ Kanten des Graphen

- Finde mit A* Weg vom Start- zum Zielpolygon



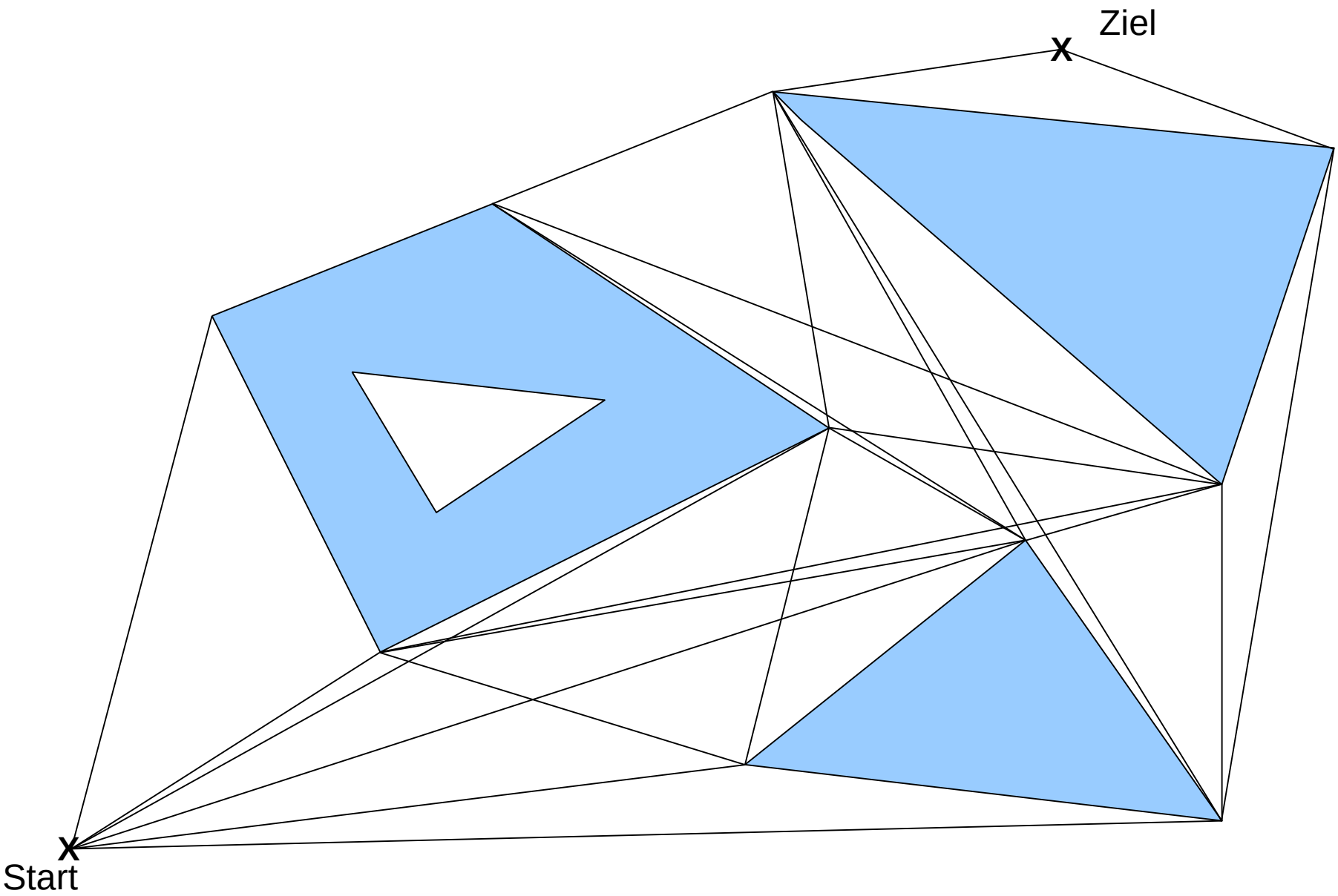
Polygonzerlegung

- Tatsächlicher Weg: Von Startposition geradeaus zum Mittelpunkt des enthaltenden Polygons, von dort dem im Graphen gefundenen Weg folgen, vom Mittelpunkt des Zielpolygons direkt zum Ziel
- Funktioniert, da Polygone komplett hindernisfrei
- Einfacher Algorithmus, der garantiert einen kollisionsfreien Weg findet, wenn einer existiert
- Nachteil: bei großen Freiräumen große Polygone, evtl. große Umwege

Sichtgraphen

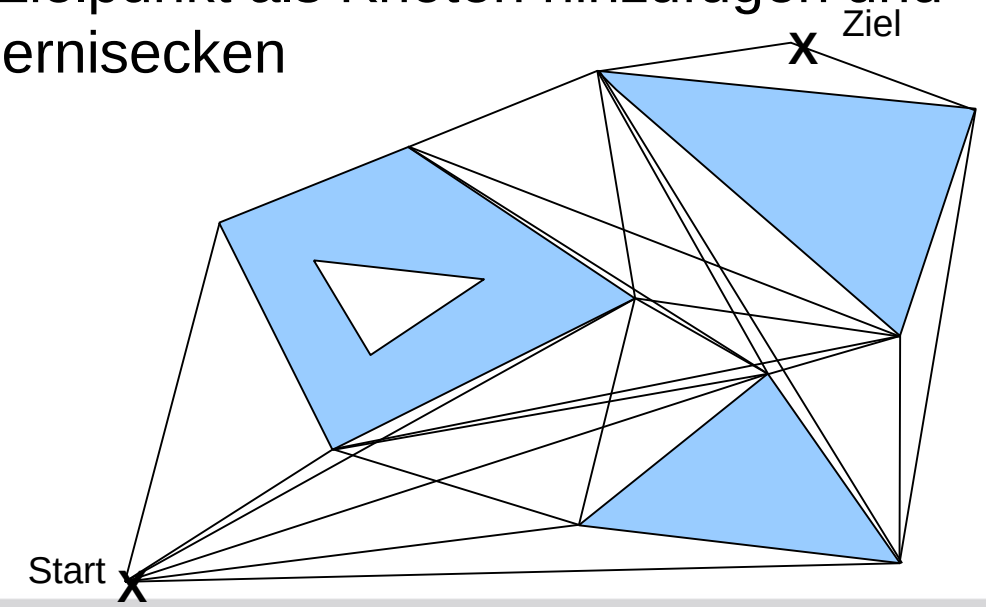
- Bisherige Ziele waren beliebige Wege oder Wege mit **maximaler Hindernisfreiheit**
- Jetzt soll der **kürzeste Weg** gesucht sein
- Vorüberlegungen:
  - Kürzeste Verbindung zwischen zwei Punkten ist eine Gerade
  - Wenn Hindernis zu umfahren ist, dann direkt an dessen Rand vorbei
- Resultat: Sichtgraphen

Sichtgraphen



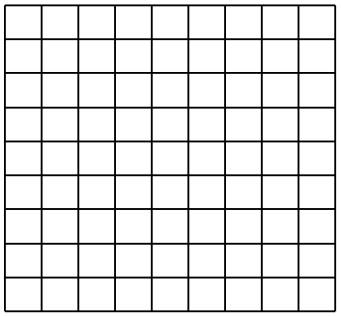
Sichtgraphen

- Wegstücke sind die Sichtverbindungen **zwischen den Ecken** der Hindernisse
- Ecken sind **Knoten des Graphen**, Kante zu jedem Knoten der sichtbar ist
- **Auch Ränder der Hindernisse** sind Verbindung zwischen den Ecken an ihren Enden
- Für die Wegplanung Start- und Zielpunkt als Knoten hinzufügen und Kanten zu allen sichtbaren Hindernisecken

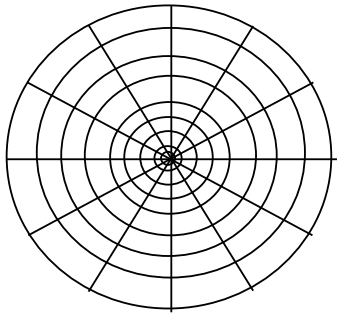


Gitter

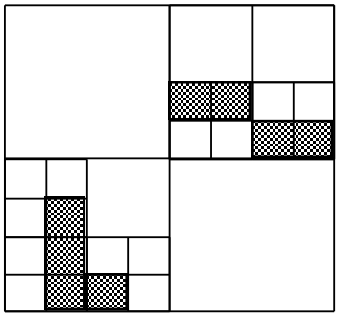
- Zwei bzw. dreidimensionale Rasterstruktur
- Annäherung der Form des Hindernisses durch Anzahl zugehöriger Zellen



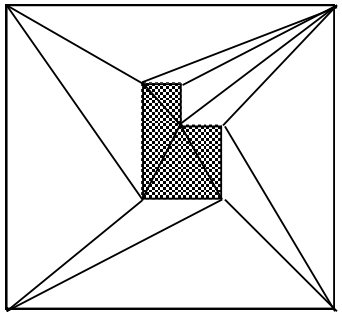
Quadratisches Gitter



Sphärisches Gitter



Hierarchisches Gitter



Beliebige Gitterstruktur

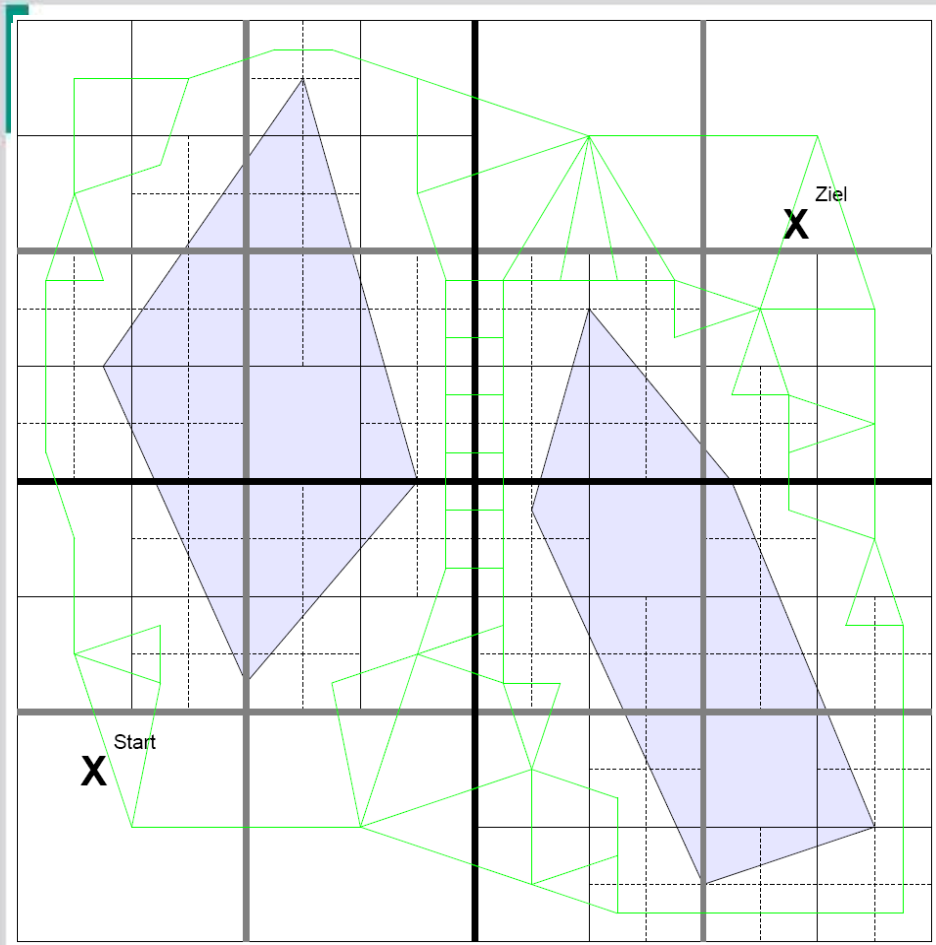
Sichtgraphen

- Resultat der Wegsuche im Graphen ist der **kürzeste Weg**, wenn die Kanten mit Länge der Strecken gewichtet sind
- Aber der Weg führt direkt an **Hindernissen** vorbei ➔ zur Kollisionsvermeidung müssen Hindernisse vorher **hinreichend vergrößert** werden
- Methode ist exakt, wenn ein Roboter nur **zwei translatorische Freiheitsgrade** hat und sowohl Roboter als auch Hindernisse durch Polygone dargestellt werden können
- Methode auch im R^3 anwendbar, jedoch sind die gefundenen Wege i.A. keine kürzesten Wege mehr

Quadtrees

Idee:

- Aufteilung des Raumes in Quadrate (2D) bzw. Kuben (3D) adaptiver Größe
- Hier für 2D erläutert, in 3D analog (Oct-trees), im Prinzip für beliebig viele Dimensionen anwendbar
- Gesamter Raum als ein Quadrat, dieses wiederum in vier Unterquadrate eingeteilt
- Quadrat ist entweder komplett frei, komplett unpassierbar oder gemischt
- Gemischte Quadrate werden wiederum in vier Unterquadrate eingeteilt, die frei, belegt oder gemischt sein können, letztere werden dann wieder unterteilt usw.



Freiraum

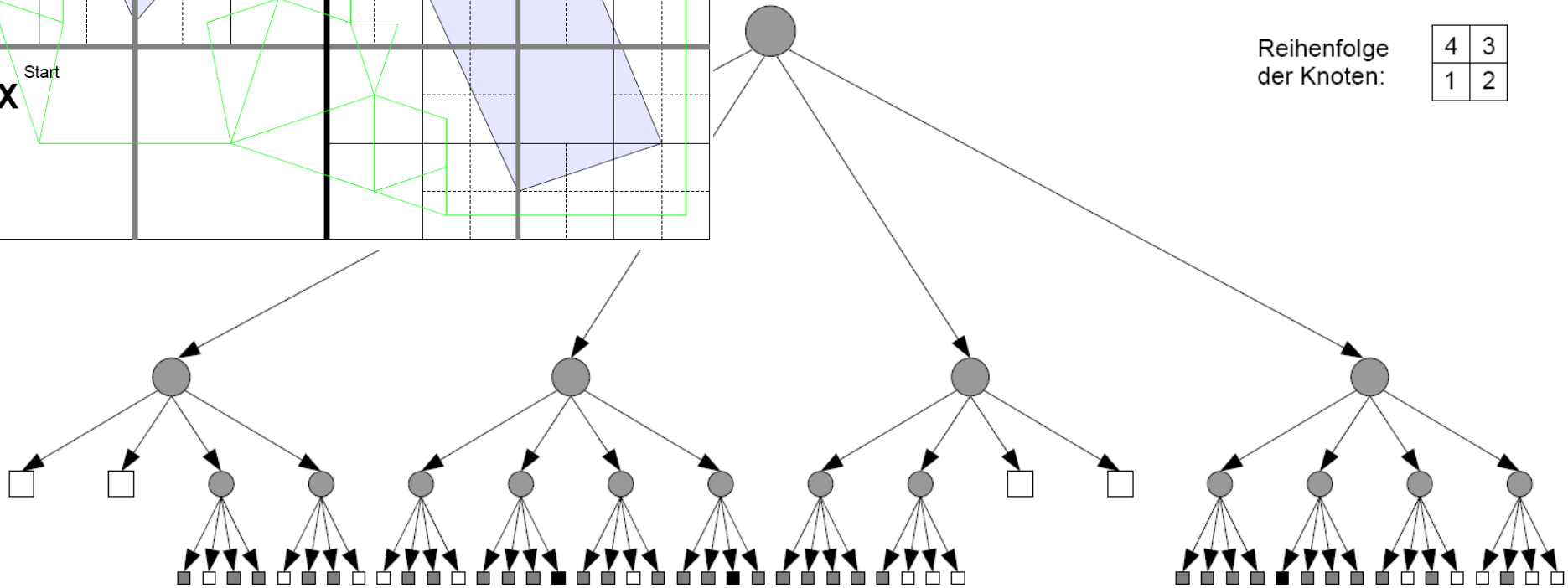
Hindernisraum

Gemischt – innerer Knoten

Gemischt (müsste weiter unterteilt werden)

Reihenfolge der Knoten:

4	3
1	2



Quadtrees

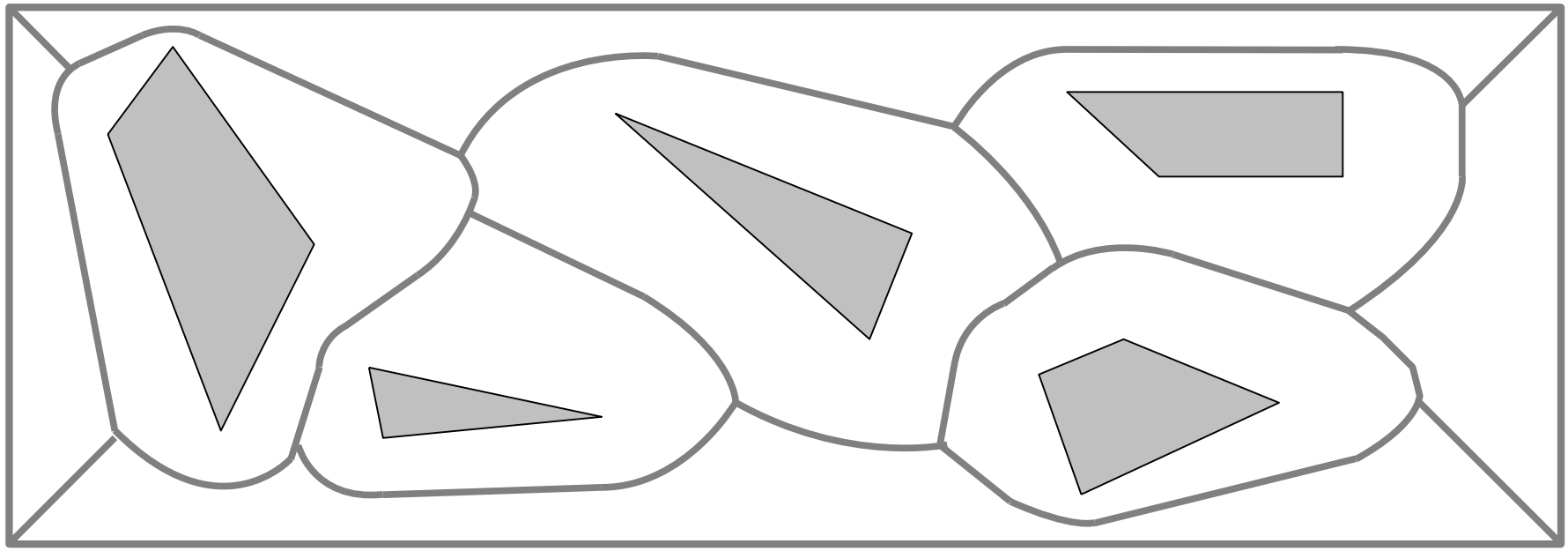
- Wo nötig, verfeinern, bis beliebige untere Genauigkeitsgrenze erreicht
- **Verbleibende gemischte Quadrate** als **Hindernisraum** behandeln
- Gesamter Raum in Quadrate eingeteilt, die frei oder belegt sind
- In jedem Bereich des Raumes nur so genau, wie nötig
- Einheitliche Raumeinteilung ohne Speicherverschwendung

Quadtrees

- Ob ein existierender Weg gefunden wird, hängt von **Feinheit der Einteilung** ab
- Pfade können geradeaus zwischen den Mittelpunkten der benachbarten Quadrate verlaufen
- Evtl. trotzdem durch Mittelpunkt der gemeinsamen Trenngerade fahren, um mehr Abstand von möglichen Hindernissen zu halten
- Umweg hängt von der Genauigkeit der Unterteilung und von Größe der durchquerten Quadrate ab

Voronoi-Diagramme

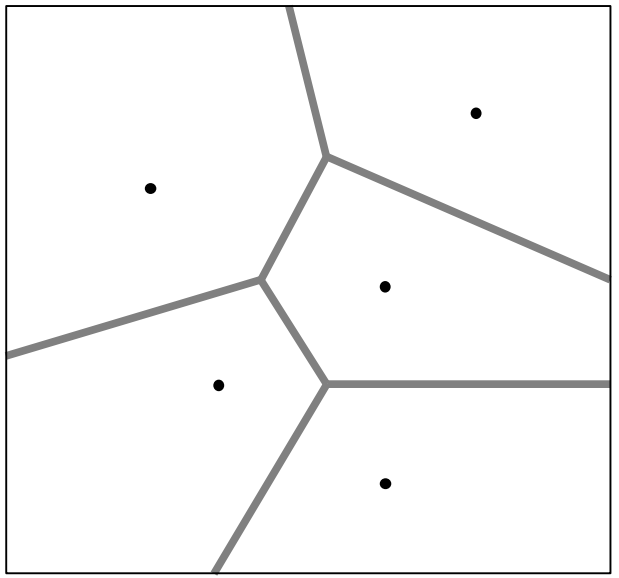
- Ziel: Maximalen Abstand zu Hindernissen halten
- Voronoi-Diagramm teilt Raum so in Flächen ein, dass jedes Hindernis in genau einer Fläche enthalten ist
- Fläche enthält alle Punkte, für die dieses Hindernis das nächstgelegene ist

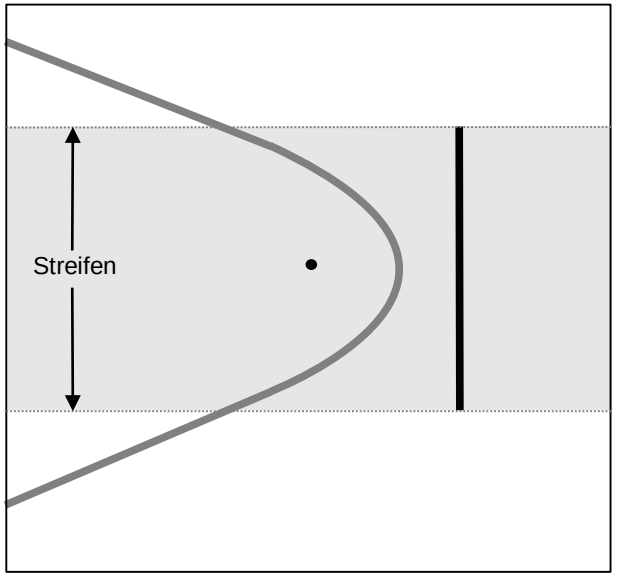


Voronoi-Diagramme erstellen

Die **Grenzlinie** ist zwischen ...

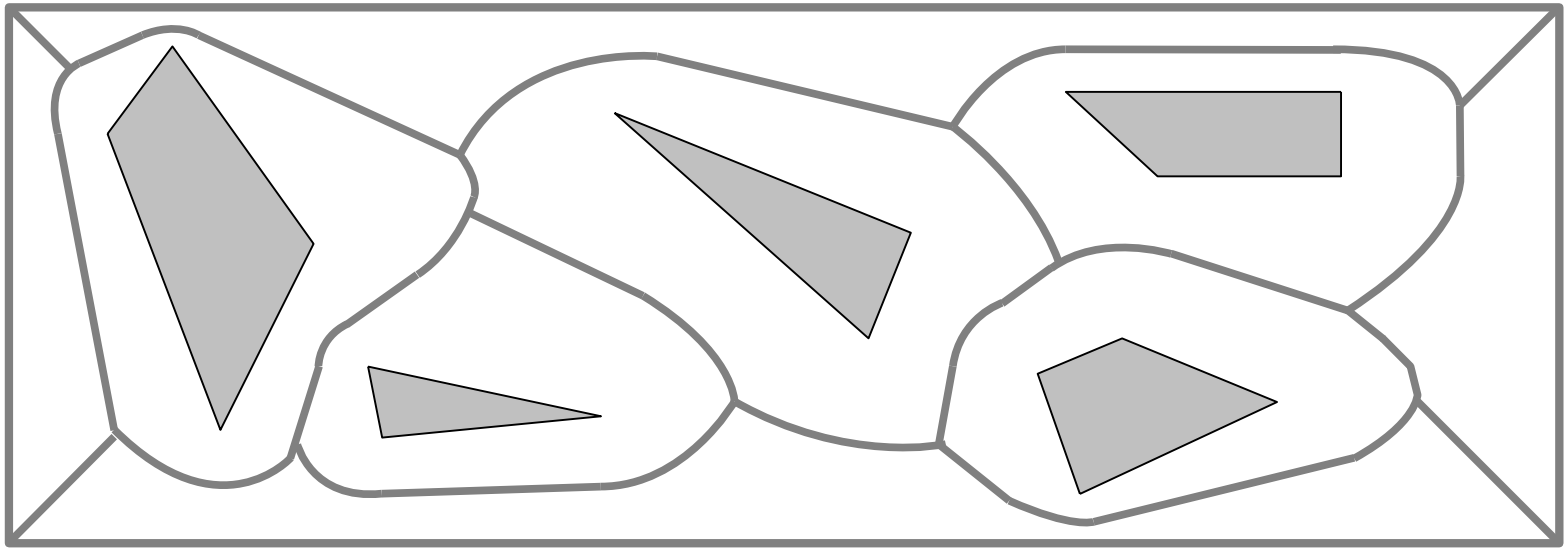
- zwei **punktförmigen Hindernissen** eine Gerade
- einem **Punkt** und einer **Gerade** eine Parabel
- einem **Punkt** und einer **endlichen Strecke** eine Parabel im Streifen der Strecke, außerhalb Halbgerade
- zwei **Strecken**
  - Gerade im Schnitt der Streifen,
  - Parabel bzgl. Endpunkt in einzelnem Streifen,
  - Gerade bzgl. Endpunkten außerhalb der Streifen





Voronoi-Diagramme

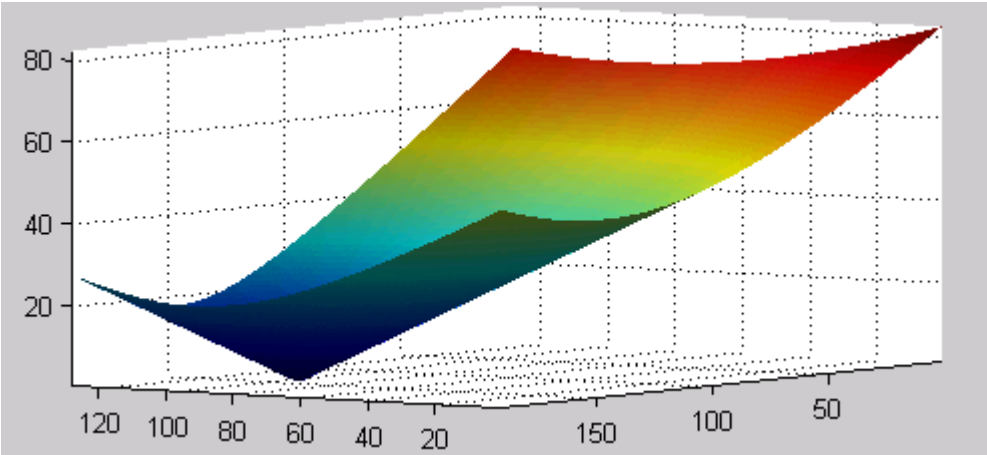
- Grenzen der Voronoi-Flächen bilden Wegenetz zwischen den Hindernissen
- Kreuzungen sind Knoten des Suchgraphen, Kante zwischen zwei direkt verbundenen Kreuzungen
- So ermittelter Weg garantiert **maximale Freiheit von Hindernissen**, aber ist evtl. großer Umweg
- Konstruktion bei komplexen Hindernissen ➔ Potentialfeld

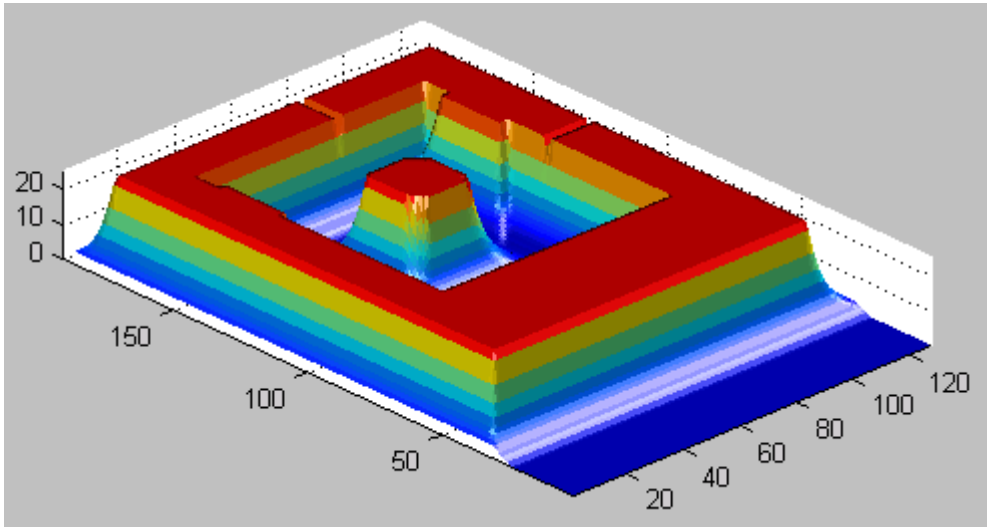


Potentialfeldmethode

- Völlig anderer Ansatz: Keine Überführung der Planungsaufgabe in Graphen
- Stattdessen: Potentialfeld im Konfigurationsraum definieren, darin lokale Suche durch Gradientenabstieg
- Potentialfeld hat globales Minimum im Zielpunkt, mit der Entfernung in alle Richtungen linear ansteigend
- Hindernisse haben sehr hohes Potential, von Hindernisrändern aus mit der Entfernung kontinuierlich abnehmend

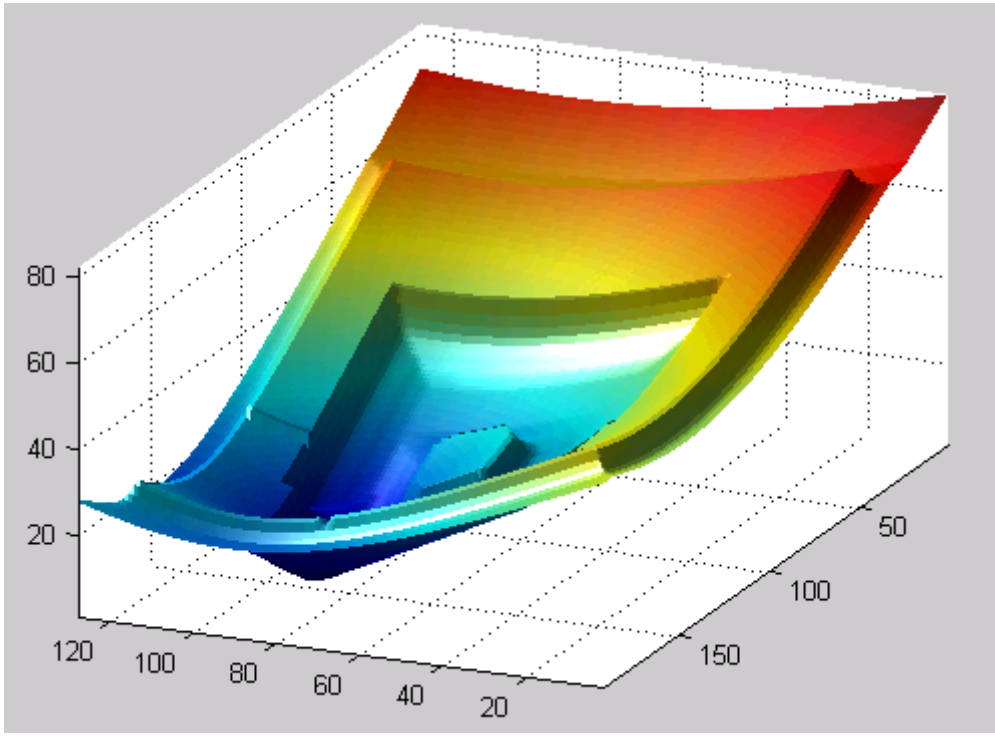
Potentialfeldmethode





Potentialfeldmethode

- Die beiden so entstandenen Potentialfelder werden addiert
- In endgültigem Potentialfeld vom Startpunkt aus Gradientenabstieg durchführen
- Gradientenabstieg folgt den „Tälern“ des Potentialfeldes zum Zielpunkt



Potentialfeldmethode

- Gefundene Wege bilden Kompromiss zwischen kurzem Weg und Abstand zu Hindernissen
- Meistens sehr gute Pfade
- **Problem**: lokale Minima ➔ Methode zum Erkennen und Verlassen nötig
- Ohne Modifikationen keine Garantie, dass ein Weg gefunden wird, wenn einer existiert
- Besser: Wellenausbreitung (künstliches Potentialfeld)