2. Problemlösen und Suche

Was ist ein Problem?

gegeben: **Anfangszustand S**
Menge von Operatoren, überführen Zustand in Nachfolgezustand
Menge von Zielzuständen, oft definiert durch Zielprädikat
Kostenfunktion für Pfade (= Folgen von Operatoren)

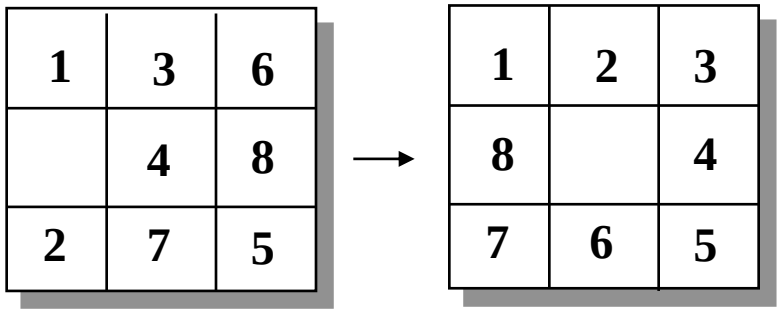
gesucht: **Pfad von S zu einem Zielzustand (häufig: mit minimalen Kosten)**

Die Menge aller von S aus durch Anwendung von beliebig vielen Operatoren erreichbaren Zustände heißt Zustandsraum

Der Zustandsraum lässt sich als Baum repräsentieren, der S als Wurzel hat, und in dem N_2 Nachfolger von N_1 ist gdw. es einen Operator gibt, der N_1 in N_2 überführt.

Wir gehen davon aus, dass es nur endlich viele Operatoren gibt!

Beispiel: 8-Puzzle



Zustände: 3 x 3 Integer Array mit entsprechenden Werten

Operatoren: blank left, right, up, down

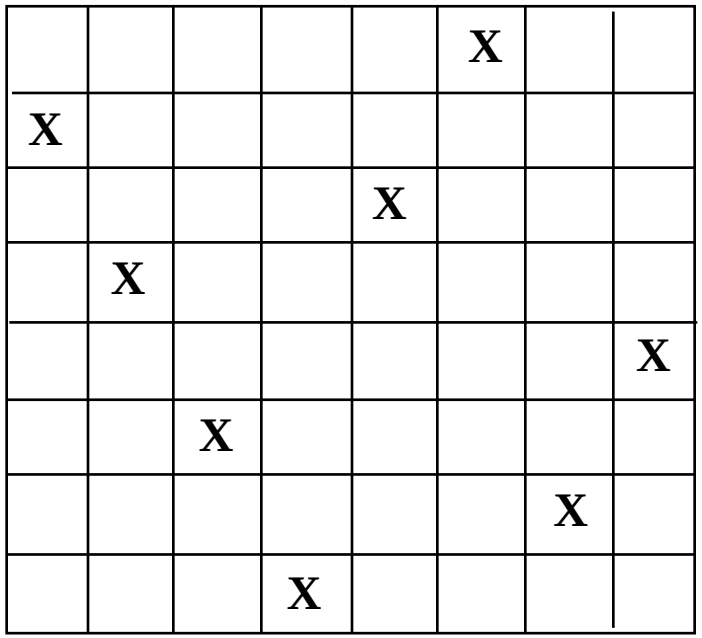
Zielzustand: siehe oben rechts

Kosten: Pfadlänge

Beispiel: 8 Damen

Positioniere 8 Damen so auf Schachbrett, dass keine eine andere bedroht

Startzustand:	**leeres Schachbrett**
Zielzustände:	**Schachbrett mit 8 Damen, keine bedroht**
Operatoren:	**Füge Dame auf freies Feld**
Kosten:	**0**



Beispiel: Kohl etc.

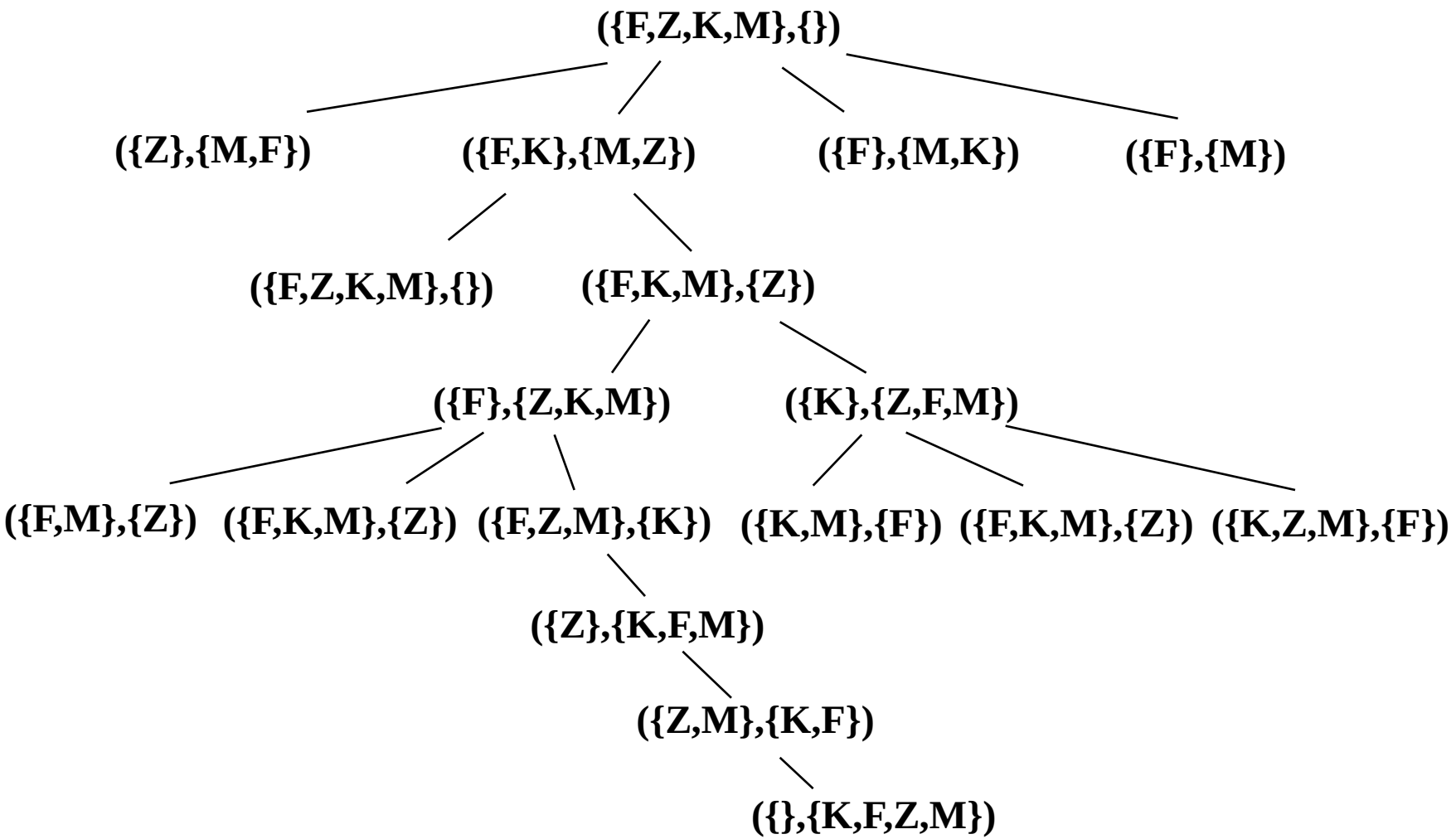
S:	**Fuchs** **Ziege** **Kohl** **Mensch**	**Fluss**

Mensch kann in Boot immer 1 Objekt an anderes Ufer mitnehmen.
Wenn Mensch an anderem Ufer ist, frisst Fuchs Ziege bzw. Ziege Kohl

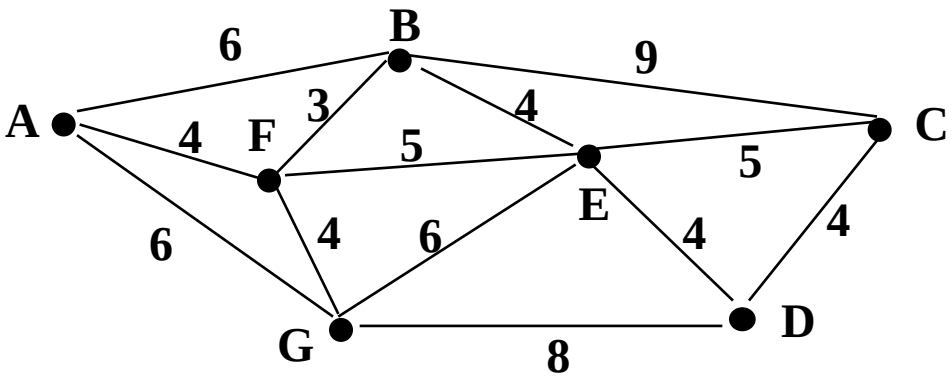
Ziel:	**Fluss**	**Fuchs** **Ziege** **Kohl** **Mensch**

formal:
Zustand:
Mengenpaar (L,R), S = ({F,Z,K,M},{}), Ziel = ({},{F,Z,K,M})
Operatoren:
transportiere_O mit O = {F}, {Z}, {K} oder {}, geeignet zu definieren

Partieller Suchbaum



Travelling Salesman



Verkäufer hat n Städte zu besuchen, jede genau 1 mal, muss dann zum Ausgangspunkt A zurück.
Ziel: minimiere zurückgelegte Gesamtstrecke

Zustände: Liste der bisher von A aus besuchten Städte (anfangs leer)
Zielzustand: Liste, die alle Städte enthält und mit A endet, Strecke minimal
Operatoren: Füge noch nicht besuchte Stadt am Ende der Liste ein

Klassifikation von Suchmethoden

	uninformiert	informiert (heuristisch)
systematisch (exhaustive)	Breitensuche Tiefensuche iteratives Vertiefen	A* greedy Verfahren
nicht.-system.	blindes Suchen	Verbesserungsverfahren: Hill Climbing genetische Algorithmen

Heuristik: Strategie zum schnelleren Finden einer (guten) Lösung

Uninformiertes systematisches Suchen

```
Procedure Suchen
begin
QUEUE := (Startknoten);
while QUEUE ≠ ( ) do
    begin
    KNOTEN := First( QUEUE);
    QUEUE := Rest( QUEUE);
    if ZIEL( KNOTEN ) then return Pfad zu KNOTEN
    else füge alle Nachfolger von KNOTEN zu QUEUE
    end;
print("keine Lösung")
end
```

Breitensuche: Nachfolger jeweils am Ende von QUEUE eingefügt
Tiefensuche: Nachfolger jeweils am Anfang von QUEUE eingefügt

Kriterien für Suchstrategien

Vollständigkeit:	**wird Lösung gefunden, wenn es eine gibt?**
Zeitkomplexität:	**wie lange dauert die Suche?**
Speicherkomplexität:	**wieviel Speicherplatz wird benötigt?**
Optimalität:	**wird die "beste" Lösung gefunden?**

Breitensuche:
vollständig; optimal, wenn Kosten von Länge des Pfades bestimmt werden; exponentieller Speicher- und Zeitaufwand

Tiefensuche:
unvollständig; nicht optimal; geringerer Speicherbedarf, da nur ein Pfad gespeichert werden muss sowie die an ihm liegenden offenen Knoten

Iterative Deepening

<u>Tiefenbeschränkte Suche</u>: Tiefensuche mit Tiefenbeschränkung
- vermeidet möglicherweise unendliches Suchen in falschem Pfad,
- aber: Schranke kann falsch gewählt werden

<u>Iteratives Vertiefen</u> verallgemeinert tiefenbeschränkte Suche
- Idee: tiefenbeschränkte Suche mit wachsender Tiefe 0, 1, 2, 3 usw.
- verbindet Vorteile von Breiten- und Tiefensuche
- Knoten zwar mehrfach überprüft, aber Speicherersparnis wiegt das auf

"In general, iterative deepening is the preferred search method when there is a large search space and the depth of the solution is unknown"

Informierte Suche: A*

verwendet Evaluierungsfunktion f, die für jeden Knoten n die Kosten des besten Pfades von Startknoten s über n zu einem Zielknoten schätzt

es gilt: $f(n) = g(n) + h(n)$

wobei g(n) die Kosten von s zu n sind, h(n) die geschätzten minimalen Kosten von n zu einem Zielknoten. h heißt auch heuristische Funktion.

es wird jeweils ein Knoten expandiert, für den f den minimalen Wert liefert

A* ist vollständig und optimal, falls gilt: geschätzte minimale Kosten h(n) von n zu Zielknoten ≤ tatsächliche minimale Kosten,
d.h., h darf Kosten unter- aber nicht überschätzen!!

Beispiel:
gesucht kürzester Weg von A nach B, Kosten jeweilige (Straßen-) Entfernung
Wähle h(n) als die Länge der Luftlinie von n nach B.

Beweis der Optimalität von A*

Sei G ein optimaler Zielknoten mit Pfadkosten f*,

G' ein suboptimaler Zielknoten mit f(G') = g(G') > f*
(da G' Zielknoten ist, ist h(G') = 0).

Angenommen, A* würde den Pfad zu G' als Lösung liefern. Dann muss G' im letzten Schritt aus der Queue geholt worden sein.

Sei n ein Blattknoten auf einem optimalen Pfad zu G, es gilt

$$f^* \geq f(n)$$

Da n nicht zur Expansion ausgewählt wurde, gilt auch

$$f(n) \geq f(G')$$

und damit $f^* \geq f(G')$.

Daraus folgt $f^* \geq g(G')$, im Widerspruch zur Annahme.

Hinweis: für Vollständigkeit muss gelten:
Es gibt positive Konstante d, so dass jeder Operator mindestens d kostet.

Heuristiken am Beispiel 8-Puzzle

h1: Anzahl der Plättchen in falscher Position

h2: Summe der Entfernungen aller Plättchen von ihrer Zielposition gemessen in notwendigen Operationen (Manhattan distance)

1	3	6
	4	8
2	7	5

h1(n) = 6
h2(n) = 0 + 3 + 1 + 1 + 0 + 3 + 1 + 2 = 11

h2 ist näher am aktuellen Wert und führt schneller zum Ziel

Greedy Verfahren

Lösen von Optimierungsproblemen:

Gegeben: Gütefunktion w für (Teil-) Lösungen

Optimale Lösung konstruiert durch schrittweises Erweitern einer Teillösung (beginnend mit leerer Lösung)

Aus Erweiterungsmöglichkeiten die gewählt, die zu größtem w-Wert führt (greedy = gefräßig, der größte Bissen wird geschluckt)

Kanonischer Algorithmus:

Ordne Lösungskomponenten, so dass : $w(e1) \geq w(e2) \geq \ldots \geq w(ek)$;
A := ∅;
For i = 1 to k do
if $A \cup \{e_i\}$ zulässige Teillösung then $A := A \cup \{e_i\}$;
Output A.

Wann funktioniert das?

Sei E endliche Menge, U Menge von Teilmengen von E.

(E,U) heißt Teilmengensystem falls gilt:

1. $\varnothing \in U$,

2. $A \subseteq B$ und $B \in U$ impliziert $A \in U$.

Ein Teilmengensystem heißt Matroid, falls gilt:

$X, Y \in U, |X| < |Y|$ impliziert $\exists x \in Y \setminus X: X \cup \{x\} \in U$.

Theorem: Sei (E,U) Teilmengensystem, w: $E \rightarrow R$ Wertefunktion.
Der kanonische Greedy-Algorithmus liefert eine optimale Lösung für das zugehörige Optimierungsproblem (finde $X \in U$ mit maximalem Gewicht) gdw. (E,U) ein Matroid ist.

Anmerkung: Wert von X ist die Summe der Werte der Elemente von X.

Hill Climbing Verfahren

Operieren im Raum vollständiger (auch suboptimaler) Lösungen

- **Modifizieren (zufällig) aktuelle Lösung und ersetzen sie durch neue, wenn dadurch bessere Qualität erreicht wird**
- **Terminierung, falls längere Zeit keine Verbesserung**
- **Vorteil: geringer Speicherbedarf (nur jeweils aktuelle Lösung)**

Probleme:

- **lokale Maxima, alle erreichbaren Zustände schlechter, aber keine optimale Lösung gefunden**
- **Plateaus, Nachbarzustände alle gleichwertig, zielloses Hin- und Herspringen**

Lösung: random restart, wenn keine merkliche Verbesserung mehr eintritt, wird zufällig neuer Startpunkt ausgewählt. Jeweils beste bisher erreichte Lösung gespeichert.

Beispiel GSAT

gegeben: aussagenlogische Formel F in Klausenform
gesucht: Wahrheitsbelegung, die F wahr macht

$$(P \vee Q \vee \neg S) \wedge (\neg P \vee Q \vee R) \wedge (\neg P \vee \neg R \vee \neg S) \wedge (P \vee \neg S \vee T)$$

GSAT ist random restart hill climbing Verfahren

startet mit zufällig erzeugter Wahrheitsbelegung

Operatoren ändern den Wahrheitswert genau einer Aussagenvariable

Qualität einer Wahrheitsbelegung ist die Anzahl der Klausen, die sie wahr macht.

als Parameter werden maximale Anzahl der Schritte pro climbing und Anzahl der restarts vorgegeben

Verfahren stoppt, wenn alle Klausen erfüllt oder vorgegebene Schranken überschritten sind

Genetische Algorithmen

- **Versuch, natürliche Evolutionsprozesse nachzubilden**
- **Nicht einzelne Lösung verbessert, sondern *Population* von Lösungen**
- **Sowohl zufällige Änderungen (Mutationen) als auch Kreuzungen (Cross-over)**
- **Lösungen mit besten Bewertungen (Fitness) überleben**

Kanonischer Algorithmus

Erzeuge zufällige Anfangspopulation von Lösungen P = {a_1, …, a_m};
Repeat
    erzeuge bestimmte Anzahl zufälliger Mutationen der Lösungen in P;
    erzeuge bestimmte Anzahl zufälliger Kreuzungen von Paaren in P;
    wähle die m besten Lösungen aus und weise sie P zu
Until keine weitere Verbesserung der Fitness;
Gib die beste Lösung in P aus.