# Seminararbeit

Thema : Suchverfahren

Autor : Sven Schmidt
ii4765, FH-Wedel

# Inhalt

# 1. GRUNDLAGEN

## 1.1 Problemlösen als Suche

### Zustandsraumpräsentation

Probleme bei der Künstlichen Intelligenz lassen sich meist in Probleme formulieren, die durch Suche zu lösen sind. So ist z.B.: bei einem Schach-Computer der nächste Zug zu berechnen.Es werden also aus der aktuellen Spielsituation alle möglichen Züge berechnet und bewertet. Dabei sucht sich der Schachcomputer aus allen Zügen den nächsten besten Zug heraus und führt ihn aus. Intelligente Schachcomputer Werten so mehere Züge im voraus aus, bevor sie den nächten Zug berechnen. Da der Computer so menschliches Verhalten imitiert, das dem Betrachter der Eindruck erweckt wird er spiele gegen einen Menschen, nennt man dieses Vorgehen bei einem Computer : Künstliche Intelligenz.

Ziel der Suche ist es

- einen konkreten Zielzustand zu erreichen
- Den Weg vom Startzustand zum Zielzustand auszugeben

Um ein Problem der Künstlichen Intelligenz in ein Problem der Suche zu überführen, muss es so formuliert werden, das eine Suche darauf angewendet werden kann. Hierbei kann eine Zustandsraumrepräsentation des Problems helfen. Dazu muss man das Problem zerlegen in

- Zustände
- Zustandsübergangsoperatoren

**Zustände** sind die zum Zeitpunkt der Suche vorliegende Information. Bei einem Schachspiel wäre das die Besetzung der Spielfiguren auf dem Spielbrett. Wenn eine Lösung des Problems gefunden wurde, dann repräsentiert die Reihenfolge der Zustände den Lösungsweg.
Der erste Zustand ist als **Startzustand** zu bezeichnen. Er repräsentiert den Zustand, der zu Anfang der Suche / des Problems steht. Der Zustand, der als Lösung des Problems angesehen wird (ein Problem kann auch mehrere Lösungen besitzen), wird als **Endzustand** bezeichnet. Wenn die Suche diesen Zustand erreicht hat, wird die Suche abgebrochen und als Erfolg vermerkt. Der Weg vom Startzustand zum Endzustand ist der Lösungsweg des Problems.

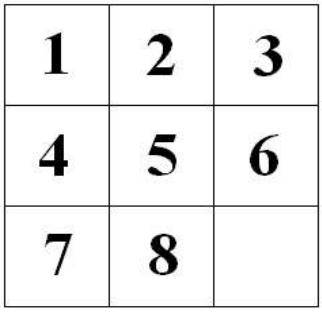


Bild 1: *Beispiel Schiebepuzzel* : Der Endzustand (Information wo sich welcher Stein befindet)

Zustände belegen Speicher! Jeder durchlaufene Zustand muss gespeichert werden, damit z.B.: Das Suchprogramm sich merken kann welche Zustände schon untersucht, und wie sie vom Startzustand erreicht wurden.

**Zustandsübergangsoperatoren** definieren den Übergang (Berechnung) von einem Zustand in seinen Folgezustand. Mithilfe der Zustandsoperatoren können alle Zustände berechnet (durchlaufen) werden. Dadurch lässt sich das Problem algorithmisch relativ einfach lösen. Bei dem Schiebepuzzel wären die Zustandsübergangsoperatoren die Berechnung, welche Steine in das freie Feld geschoben werden können. Es werden also im Konkreten Fall die Spielsituation des nächsten Zuges berechnet. Diese Berechnung belegt Speicher und dauert Zeit. Diese Zeit sollte möglichst gering gehalten werden, damit möglichst viele Zustände auf Zieleigenschaften überprüft werden können. Weiteres dazu im Kapitel : „Bewertung von Suchstrategien“.

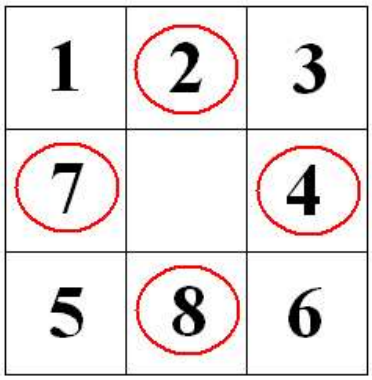


Bild 2 : *Beispiel Schiebepuzzel :* Die vier nächsten Zustände errechenbar über die Zustandsübergangsoperatoren.

Die **Zustandsraumrepräsentation** ist in aller Regel ein Baum, in dem die Knoten die Zustände, und die Kanten zwischen den Zuständen die Zustandsübergangsoperatoren darstellen. Der Startzustand ist die Wurzel des Baumes und der Endzustand (evtl. auch mehrere) kann im Baum an beliebiger Stelle liegen. Jeder Pfad von der Wurzel (Startzustand) zu einem der Zielzustände repräsentiert die Lösung des Problems.
Der Baum der Suche lässt sich mit folgenden Parametern beschreiben:

d = Tiefe des Baumes ()
b = Verzweigungsgrad des Baumes (z.B.: Jeder Zustand hat 2 Nachfolgezustände)

Ziel der Suchverfahren ist es mit wenig Ressourcenaufwand (Berechnung der Zustandsübergänge, sowie Anzahl zugleich gespeicherter Zustände) möglichst kurze Lösungspfade zu finden.

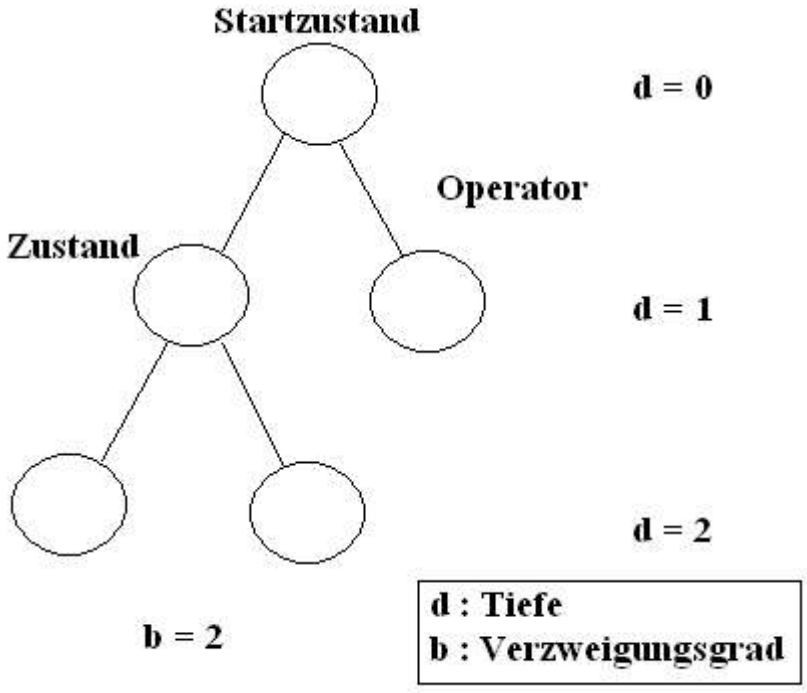


Bild 3 : Zustandsraumrepräsentation (Baum der Suche)

Um die Suche mit einem Computerprogramm durchzuführen wird der Anfangszustand, gültige Zielzustände und die Übergangsfunktion gebraucht. Die Übergangsfunktion legt durch Anwendung eines Operators fest, welche Nachfolge-Zustände vom aktuell Betrachteten zu erreichen sind. Die zu erreichenden Zustände können häufig danach bewertet werden, wie dicht sie dem Zielzustand sind und dabei die Kosten abschätzen die durch explorieren (durchsuchen) des Suchbaums angefallen sind oder noch anfallen könnten.

Definition: Expandieren:

Unter Expandieren versteht man, den nächsten Zustand durch die Übergangsfunktion zu berechnen. Dadurch wird Zeit für den Vorgang, sowie Speicherplatz verbraucht. In einer Liste heisst expandieren, das der „Vaterknoten“ vernichtet wird und an seiner Stelle seine Nachfolgeknoten entstehen.

Definition: Start-, Zielzustand:

Ein Start- und Zielzustand wird durch algorithmisch überprüfbare Eigenschaften charakterisiert. Ist diese Charakterisierung eine einfache endliche Auflistung der jeweiligen Elemente, so nennt man dies eine **explizit vorgegebene Zustandsmenge**. Wenn die Aufzählung nicht möglich st, so nennt man das eine **implizit vorgegebene Zustandsmenge**. In diesem Fall müssen die zu charakterisierenden Elemente noch errechnet werden, was zusätzlich Zeit in anspruch nimmt.

## 1.2 Generische Suche

Dieses Suchverfahren ist so allgemein gehalten, das aus ihm durch Spezialisierung und Konkretisierung beliebige Suchverfahren gewonnen werden können.

Algorithmus:

1. L ist Liste der Startknoten für das Problem.
2. Ist L Leer, so melde einen Fehlschlag, andernfalls wähle Knoten n aus L.
3. Ist n ein Zielknoten, so melde einen Erfolg und liefere den Pfad vom Startknoten zu n.
4. Ersetze in L den Knoten n durch seine Nachfolgeknoten.Markiere dabei die neuen Knoten mit dem jeweils zugehörigen Pfad vom Startknoten.
5. Weiter mit Schritt 2.

L (Agenda) = Liste der Knoten (Zuständen), die ausgehend vom Startzustand durch expandieren erreicht wurden. Die expandierten Knoten werden erst bei ihrer eigenen Expansion auf Zieleigenschaften geprüft und nicht schon beim Eintragen in die Agenda L. Der bisher beschrittene Suchpfad wird in jedem Knoten vermerkt. Es ist also ausreichend später nur den Zielknoten zu speichern, dem bei erreichen der Suchpfad eingetragen wurde. Die Liste L wird solange auf einen Zielknoten überprüft bis dieser gefunden wird, oder keine Knoten mehr in der Liste L sind.
Ist die Liste Leer und gab es keine Zielknoten auf dem Suchpfad, so wird die Suche nicht erfolgreich abgebrochen.

Die Suchstrategie ist das Verfahren nach dem der nächste Knoten n, ausgewählt wird. So kann z.B.: Wenn L durch eine lineare Liste repräsentiert wird, der vermeintlich beste Knoten weiter vorne in der Liste plaziert werden. Wenn die Knoten im Schritt 2 immer von vorne entnommen werden, so bedeutet dies für die Knoten weiter vorne in der Liste eine schnellere Überprüfung. Somit kann die Suche nach dem Zielknoten verbessert werden.

## Vorwärtssuche vs. Rückwärtssuche

Bei manchen Suchproblemen kann es besser sein vom Zielknoten zum Startknoten zu suchen (Zielorientierte Suche). Das ist unter den folgenden Bedingungen möglich:

- Die Zustandsübergangsoperatoren sind einfach invertierbar
- explizit vorgegebene Start- sowie Zielknoten oder Beide sind mit wenig Aufwand (im Vergleich zur Lösung des Suchproblems) aufzählbar.

Bei einer Zielorientierten Suche werden vom Zielknoten aus alle möglichen Vorgängerknoten generiert, bis man auf den Startknoten trifft. Es wird also eine Vorwärtssuche durchgeführt, nur das die Start- und Zielknoten vertauscht werden, die Operatoren dabei invertiert werden und der gefundene Lösungpfad verkehrtherum ausgegeben wird.
Beim Schiebepuzzel sind die Operatoren leicht umkehrbar und somit auch die Rückwärtssuche gut durchzuführen.

Hat ein Unterraum der Suche typischerweise einen kleineren Eingangs- als Ausgangsgrad, so ist für diesen Unterraum die Rückwärtssuche besser geeignet als die Vorwärtssuche.
Es kann auch je nach Gestalt des Unterraums mal vorwärts, mal rückwärts gesucht werden.

So wird z.B.: bei Inselgesteuerter Suche eine „Insel“ zwischen Startknoten und Zielknoten bestimmt. Dann wird versucht von beiden Seiten einen Lösungspfad zu finden der über diese Insel läuft. Lässt sich kein solcher Pfad finden, so wird die Insel ignoriert.

## Probleme bei Zyklischen Pfaden

Verschiedene Knoten können obwohl sie an unterschiedlicher Stelle im Suchbaum stehen den gleichen Zustand repräsentieren. Beim Suchen kann meist auf verschiedenen Wegen zu demselben Zustand gelangt werden. Im Normalfall wird dann derselbe Pfad nochmals überprüft. Dieser Umstand kann zu unendlich langen Wegen führen (zyklische Suchpfade) und deshalb auch nicht terminieren.
Um unnötiges Prüfen von schon untersuchten Knoten zu verhindern werden alle untersuchen Knoten auf eine Closed-List C gesetzt. Diese Liste wird vor dem expandieren von Knoten durchsucht, ob der Knoten schon überprüft wurde. So wird verhindert, das die entsprechenden Teilräume nicht erneut durchsucht werden.
Dabei ist der Aufwand nicht zu unterschätzen, der mit dieser Methode betrieben wird:

- alle bisher überprüften Knoten (die Liste C) müssen zusätzlich im Speicher gehalten werden
- Der Zeitaufwand zum Überprüfen ob der zu expandierende Knoten schon in der Closed-List ist, wird je mehr Knoten schon überprüft wurden immer länger

## 1.3 Bewertung von Suchstrategien

Die Suchstrategie wird im Schritt 2 bestimmt (Aussuchen des nächsten Knotens n).Sie bestimmt in welchem Teil und in welcher Reihenfolge der Suchbaum durchsucht wird.

**Vollständiges Suchverfahren** (erschöpfende Suche):
Alle Knoten des Suchraums werden expandiert solange kein Zielknoten erreicht wurde. Dieses Suchverfahren findet garantiert einen Zielknoten solange es einen gibt.

**Unfaire Suchverfahren**:
Knoten werden zuerst in der Tiefe gesucht. Das kann zu Problemen führen, wenn der beschrittene Pfad unendlich lang ist. Knoten werden zwar auf die Liste gesetzt, aber nicht expandiert, da sie nicht auf dem gewünschten Suchpfad liegen.

Bei mehreren Lösungen will man natürlich die finden, die am nächsten ist damit die Kosten für die Suche möglichst gering gehalten werden. Man Spricht von der Güte einer Lösung. Die Kosten sind die Faktroren Speicher- und den Zeitaufwand.
Im einfachsten Fall ist die Güte die Länge des kürzesten Pfads vom Startknoten zum Zielknoten (Wenn die Kosten pro Berechnungsschritt Konstant sind).
Es werden 2 wichtige Faktoren bei der Bewertung von Suchstrategien genommen:

- SPACE(x) : Speicherbedarf den das Suchverfahren x für ein konkretes Suchproblem hat. Alle Gleichzeitig in der Agenda L gespeicherten Knoten.
- TIME(x) : Zeitbedarf der beim untersuchen der Knoten auf ihre Zieleigenschaft aufgewendet wird

Annahme bei den Betrachtungen der Suchverfahren x (Starke Vereinfachung):

- Suchraum ist **uniform** : konstanter Verzweigungsgrad und einheitliche Tiefe d.
- Mit gleicher Wahrscheinlichkeit liegt der Zielknoten an jeder Stelle der maximaltiefe d (also ganz unten im Suchbaum)

Abhängig von der Position des Zielknotens, wird in den folgenden Betrachtungen ein **best case**, **average case** und **worst case** des Speicherbedarfs und des Zeitbedarfs angegeben.

Die Suchverfahren lassen sich in zwei Bereiche unterteilen:

- Uninformierte Suche:
  - Die Auswahl eines Knotens aus der Agenda L erfolgt nur anhand seiner Position in der Liste.

- Heuristische Suche:
  - Bei der Auswahl des Knotens aus der Agenda L wird auch der Inhalt des Knotens berücksichtigt.

# 2. Uninformierte Suchverfahren (blinde Suche)

## 2.1 Breitensuche

Bei der Breitensuche wird die Agenda L als lineare Liste verwaltet (wie in den nachfolgenden Suchverfahren auch). Werden neue Knoten in die Agenda aufgenommen, so werden sie immer hinten angehängt. Wird ein Knoten aus der Liste auf Zieleigenschaften geprüft (Schritt 2), so wird er vom Anfang der Agenda L entnommen.Somit wird der Suchbaum Schicht für Schicht durchsucht. Es wird erst weiter in einer unteren Ebene gesucht, wenn alle Knoten der aktuellen Ebene auf Zieleigenschaft überprüft wurden.

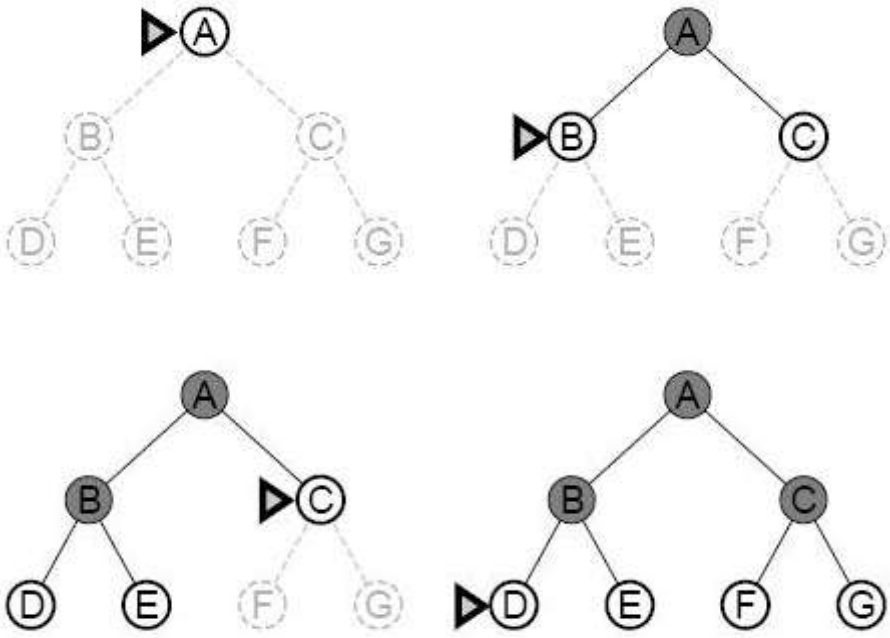


Bild 4 : Breitensuche : Diagramm

Aufwand für die Suche unter der Annahme, das der Suchbaum uniform ist und das der Zielknoten in der untersten Ebene liegt:

**Speicheraufwand**

Da Ebene für Ebene durchsucht wird und bevor der erste Knoten in der Tiefe k überprüft wird alle Knoten aus der Ebene k auf die Liste gesetzt werden ist der Speicheraufwand :
SPACE(Breitensuche) = $b^d$ (Exponentiell !!!)
Alle Knoten der zu untersuchenden Ebene! Die Knoten der vorangegangen Ebenen wurden komplett von der Liste durch Ihre Nachfolgeknoten ersetzt!

**Zeitaufwand**

Nachdem alle Knoten auf die Agenda L gesetzt wurden, ist im besten Fall der erste Knoten in der neuen Ebene der Zielknoten, im schlimmstenfall der letzte der Ebene(Es sind dann noch $b^d$ Knoten zu untersuchen!).
Es müssen also mindestens alle Knoten aus den oberen Ebenen überprüft worden sein :

$$\sum_{0}^{(d-1)} b^k = \frac{(b^d-1)}{(b-1)}$$

Im günstigsten Fall: Alle Knoten der Ebene (d-1) + 1 Knoten:

$$\frac{(b^d-1)}{(b-1)}+1=\frac{(b^d-1+(b-1))}{(b-1)}=\frac{(b^d+b-2)}{(b-1)}=O(b^{(d-1)})$$

Im ungünstigsten Fall: Alle Knoten der Ebene (d-1) + $b^d$ Knoten der aktuellen Ebene:

$$\frac{(b^d-1)}{(b-1)}+b^d=\frac{(b^d-1)}{(b-1)}+\frac{b^d*(b-1)}{(b-1)}=\frac{(b^{(d+1)}-1)}{(b-1)}=O(b^d)$$

Der mittlere Zeitbedarf ist also (arithmetisches Mittel):

$$TIME(Breitensuche)=\frac{(b^{(d+1)}+b^d+b-3)}{(2*(b-1))}=O(b^d)$$

(Exponentiell !!)

Die Breitensuche ist auf jedenfall vollständig. Leider ist der Zeitaufwand und der Speicheraufwand sehr hoch, doch wird wenn es eine Lösung gibt, wird die kürzeste (mit dem kleinsten Lösungspfad vom Start- zum Zielknoten) als erstes gefunden. Was nicht heisst, das die kürzeste Lösung die Kostenoptimalste ist.
(Siehe nächstes Kapitel : „Gleiche-Kosten-Suche“)

**Zeit und Speicheranforderungen für die Breitensuche**

| Tiefe | Knoten | Zeit | Speicher |
|---|---|---|---|
| 2 | 1100 | 1 Sekunde | 1 Megabyte |
| 4 | 111100 | 11 Sekunden | 100 Megabyte |
| 6 | 10^7 | 19 Minuten | 10 Gigabyte |
| 8 | 10^9 | 31 Stunden | 1 Terabyte |
| 10 | 10^11 | 129 Tage | 100 Terabyte |
| 12 | 10^13 | 35 Jahre | 100 Petabyte |
| 14 | 10^15 | 3523 Jahre | 1 Exabyte |

Verzweigungsfaktor b = 10
10000 Knoten / Sekunde
1000 Byte / Knoten

Aus dieser Tabelle ist ersichtlich, das sich die Breitensuche nur für wenig Anwendungen mit kleinen Suchbäumen eignet. Der Speicheraufwand, sowie der Zeitaufwand steigen exponentiell an.

## 2.2 Gleiche-Kosten-Suche

Bei der Gleichen-Kosten-Suche müssen die Kosten der Übergangsfunktion von einem Zustand in seinen Folgezustand bekannt sein. Es handelt sich nicht um eine heuristische Suche, da nicht die Zustände sondern die Kosten der Zustandsübergänge mit in die Suche einbezogen werden. Allerdings ganz uninformiert ist das Verfahren auch nicht, da schon vor der Suche Informationen mit einfliessen.
Die Zustandsübergänge (Operatoren) verursachen nur Positive kosten! Keine Operation ist umsonst oder bringt sogar Kostengewinn!

Die Kosten liegen also bei
c(ni-->ni+1) > 0

Die Kostenfunktion eines Suchpfades (strikt monotone Kostenfunktion g(nk)) lässt sich dann durch einfaches Aufsummieren der vorangegangenen Kosten vom Startknoten aus (n0,..,nk) ermitteln:

$$g(n_k) = \sum_{0}^{k-1} c(n_i \rightarrow n_{(i+1)})$$

Gleiche-Kosten-Suche expandiert die Knoten die die kleinsten Operator-Kosten haben. Wenn die Gleiche-Kosten-Suche mit einem optimaler-Kosten-Suchpfad durchlaufen wurde (keine weiteren Knoten wurden auf die Agenda gesetzt), wird an oberer Stelle weitergesucht. Somit ist diese Art der Suche vollständig!
Bei Suchräumen die nicht eine Baumstruktur aufweisen oder sogar Zyklen enthalten, wird der Pfad expandiert der die geringsten Kosten aufweist. Um Zyklen entgegenzuwirken wird vor dem Einfügen eines Knoten in die Agenda geprüft, ob sich bereits ein Pfad mit diesem Knoten mit höheren Kosten in ihr befindet. In diesem Fall wird der Mehrkostenpfad destruktiv aus der Agenda entfernt und durch den kostengünstigeren Pfad ersetzt.Somit sind alle Knoten in der Agenda L über kostenoptimale Pfade zu erreichen.

## 2.3 Tiefensuche

Bei der Tiefensuche wird im Gegensatz zur Breitensuche die Knoten die neu auf die Agenda L kommen am Anfang eingesetzt und von vorne auf ihre Zieleigenschaft untersucht.
Bevor also die Geschwister eines Knotens expandiert werden dürfen, müssen zuerst alle Kinder des Knotens expandiert worden sein.

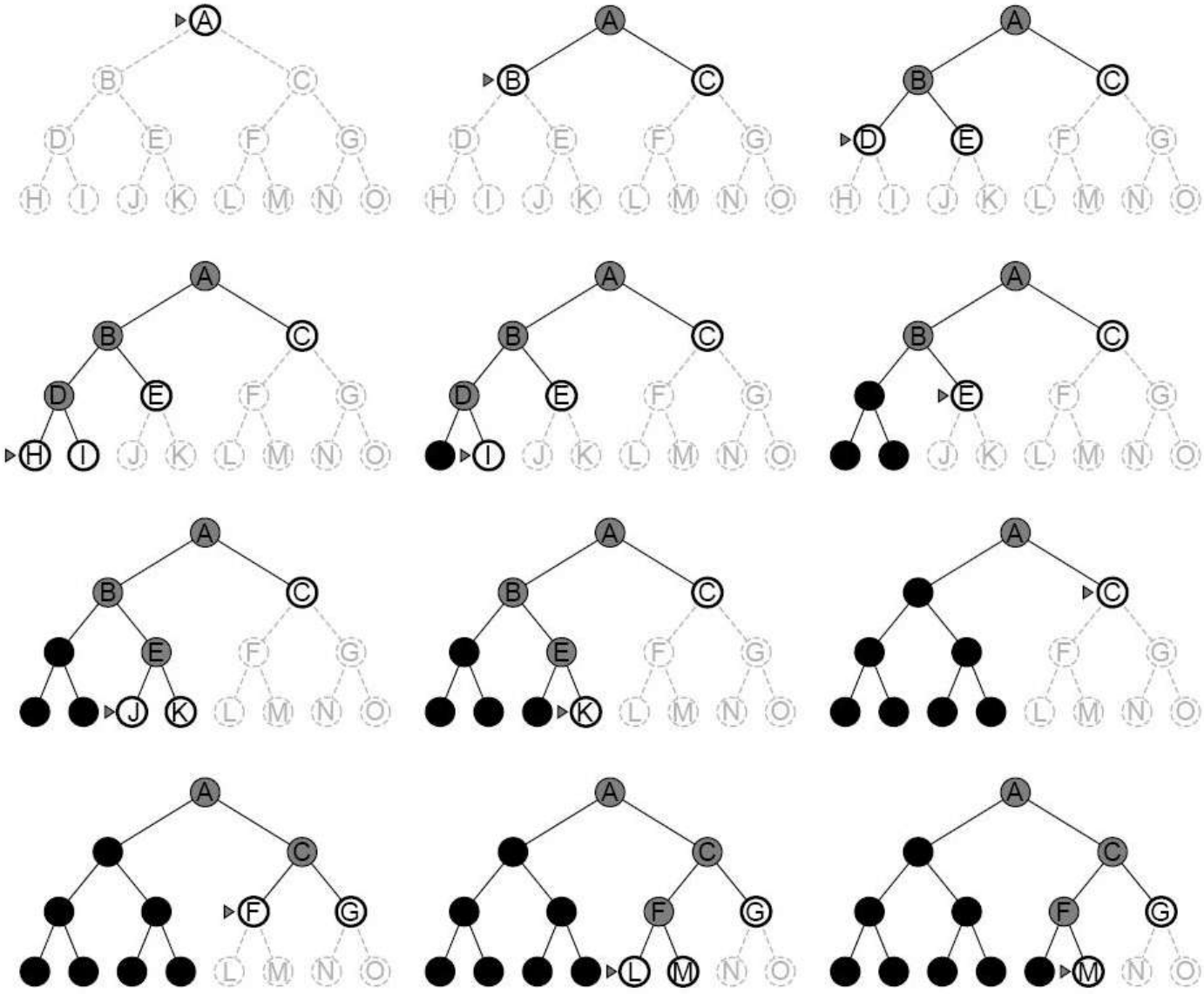


Bild 5 : Tiefensuche : Diagramm

**Speicheraufwand**

In der Agenda L werden alle unexpandierten Knoten der Geschwisterknoten gespeichert, die auf dem bisherigen Suchpfad durchlaufen wurden. (Sie werden erst beim Aufstieg expandiert).
Es ergibt sich dadurch der Speicherbedarf:
SPACE(Tiefensuche) = d(b-1)+1 = O(b*d)
Er ist LINEAR zur Tiefe d!!

**Zeitaufwand**

Der Suchbaum wird von links unten nach rechts oben durchsucht.
Die Lösung liegt in der Betrachtung des Aufwandes am Ende des Suchbaums (in der untersten Ebene). Im best case ist die Lösung also links unten (erster Knoten in der untersten Ebene), und im worstcase rechts „oben“ (letzter Knoten in der untersten Ebene).
Deshalb ist der minimale Zeitaufwand besonders gut, da der Algorithmus zuerst nach links unten sucht.

TIME(Bestcase Tiefensuche) = d+1 = O(d)  LINEAR !!!

Im Worstcase muss jedoch der komplette Baum vorher durchsucht werden:

TIME(Worstcase Tiefensuche) = $\sum_{0}^{d} b^k = \frac{(b^{(d+1)}-1)}{(b-1)} = O(b^d)$  Exponentiell !!!

Daraus ergibt sich ein mittlerer Zeitbedarf von:

Time(Tiefensuche) = $\frac{(b^{(d+1)}+b*d+b-d-2)}{(2*(b-1))} = O(b^d)$  Exponentiell !!!

Damit hat die Tiefensuche im Zeitbedarf keinen Vorteil zur Breitensuche.

Wenn der Zielknoten in einem der linken der langen Wurzelpfade liegt hat die Tiefensuche einen Zeitvorteil. Liegt der Zielknoten jedoch weit oben Rechts der Wurzel, dann wird die Breitensuche eher zum Ziel kommen. Es ist also abzuschätzen, wie Tief der Zielknoten im Suchbaum liegt und ungefähr an welcher Stelle.
Sollte jedoch die Lösung rechts oberhalb des Suchraums liegen und es einen unendlichen Wurzelpfad geben, so würde die Tiefensuche den unendlichen Pfad verfolgen und nicht terminieren. Die Tiefensuche ist also nicht fair. Sie ist auch nicht optimal, das sie zuerst die Lösung finden wird die links unterhalb der anderen liegt.

## 2.4 Schrittweise vertiefende Suche

Die Schrittweise vertiefende Suche ist ein Optimum zwischen der Tiefensuche und der Breitensuche. Sie hat den geringen Speicherbedarf der Tiefensuche, terminiert, ist optimal und hat den Zeitbedarf der Breitensuche.
Im wesentlichen wird eine Iterierte Tiefensuche durchgeführtbis zu einer maximalen Suchtiefe c, die bei jedem neuen Suchvorgang um 1 erhöht wird.
Die Schrittweise Vertiefende Suche wird nach jedem erhöhen der maximalen Suchtiefe c von vorne ausgeführt. Das lohnt sich wenn die Zeit für das erneute Erzeugen von Knoten geringer ist als die Überprüfung auf die Zieleigenschaft.

Man könnte nun denken, das dieses Vorgehen sehr Aufwändig ist. Wenn man jedoch bedenkt, das die meisten Kosten in den untersten Ebene des Suchbaums entstehen kann das erneute erzeugen der oberen Knoten fast vernachlässigt werden. Im Vergleich zur Breitensuche hat die Schrittweise vertiefende Suche bei einem Verzweigungsgrad von 10 ca. 11% mehr arbeit.

### Speicheraufwand

Aufgrund der Realisierung wie bei der Tiefensuche hat die Schrittweise vertiefende Suche denselben Speicheraufwand wie der Tiefensuche:

SPACE(Tiefensuche) = SPACE(Schrittweise vertiefende Suche) = O(b*d)  (LINEAR!!)

### Zeitaufwand

Der Zeitaufwand verhält sich pro Iterationsschritt genauso wie bei der Tiefensuche.
Der Zusätzliche Zeitaufwand, der durch das erneute expandieren des Suchbaums entsteht ist nicht wirklich ein Problem:
Vereinfacht:

$$\text{TIME(Schrittweise vertiefende Suche)} = \frac{((b+1)*b^{(d+1)})}{(2*(b-1)^2)}$$

Ein Verhältnis zum mittleren Zeitverhalten der Tiefensuche ist:

$$1 \leqslant \frac{(TIME(Tiefensuche))}{(TIME(Schrittweise\,vertiefende\,Suche))} = \frac{(b+1)}{(b-1)} \leqslant 3$$

Für Große d ist das Iterieren also nicht wirklich ein Problem.

| Kriterium | Breiten-suche | Einheitliche Kosten | Tiefen-suche | Beschränk-te Tiefen-suche | Iterative Ver-tiefung | Bidirek-tional (falls möglich) |
|---|---|---|---|---|---|---|
| Vollständig? | Ja[a] | Ja[a, b] | Nein | Nein | Ja[a] | Ja[a, d] |
| Zeit | $O(b^{d+1})$ | $O(b^{\lceil C^*/\varepsilon \rceil})$ | $O(b^m)$ | $O(b^l)$ | $O(b^d)$ | $O(b^{d/2})$ |
| Speicher | $O(b^{d+1})$ | $O(b^{\lceil C^*/\varepsilon \rceil})$ | $O(bm)$ | $O(bl)$ | $O(bd)$ | $O(b^{d/2})$ |
| Optimal? | Ja[c] | Ja | Nein | Nein | Ja[c] | Ja[c, d] |

Bild 6 : Uniformierte Suchverfahren im Überblick

Wie in Bild 6 zu sehen, ist bei jedem uninformierten Suchverfahren der Zeitaufwand exponentiell.

# 3. Heuristische Suche

## 3.1 Heuristische Schätzfunktionen

Für jedes Blinde Suchverfahren gilt das Zeitverhalten:

$$TIME(X) = O(b^d)$$

Das stellt jedoch ein Problem dar, das diese Verfahren in ihrer Reinform praktisch nicht angewandt werden, da schon bei kleinen d und kleinen b der Zeitaufwand zu groß wird. Setzt man allerdings Suchverfahren ein, die ein Vorabwissen über das konkrete Suchproblem mit einbeziehen, so kann der Zeitaufwand erheblich verkleinert werden. Aam Besten wäre es wenn man mit einer Funktion h()* den kleinsten Suchpfad im vorraus berechnen könnte. Also die geringsten Kosten und geringste Nähe vom Start- zum Zielknoten ermitteln würde. Doch der Aufwand der dafür betrieben werden müsste, ist ungefähr gleich dem einer uninformierten Suche. Es muss also ausreichen die geringsten Kosten und nähesten Abstand zum Zielknoten zum nächsten Knoten mit der Funktion h() abzuschätzen. Somit kann der nächst beste zu expandierender Knoten bestimmt werden. Die Funktion h() sollte nicht zu Aufwändig sein und sollte dennoch genau genug sein, um die Suchfunktion nicht in die Irre zu führen. Die Funktion h hängt von dem individuellen Problem ab, das es zu lösen gilt.

So können z.B.: beim Schiebepuzzel 2 verschiedene Funktionen h unterschiedlich gut zum Erfolg führen:

- h1() : Alle Steine werden gezählt, die sich nicht auf ihrem Platz befinden.
- h2() : Der Abstand jedes Steines zu seiner Zielposition werden gezählt und aufaddiert (Manhattan Distanz)

Bei h1() findet nur eine Abschätzung Aufgrund der Fehlposition statt. Diese Funktion ist mit sehr wenig Aufwand betrieben und führt doch relativ schnell zum Ziel.
In die Funktion h2() fließt jedoch noch die Entfernung der Fehlsteine mit ein. Je weiter die Steine von ihrem Platz weg sind, desto höher ist die Zahl. Wie man leicht erkennen kann ist der Aufwand für h2() größer als bei h1(), dafür ist h2() aber effektiver. Es ist nun der Aufwand für die Umwege abzuschätzen, die h1() wahrscheinlich länger brauch. Bei kleinen Suchbäumen wird sich der zusätzliche Aufwand für h2() wohl nicht lohnen, aber bei größeren Suchbäumen allerdings schon.
Es sollten nur positive Werte zur Hand genommen werden. In der Regel heisst das: Je kleiner der Wert ist, desto geringer ist die Entfernung zum Zielknoten.

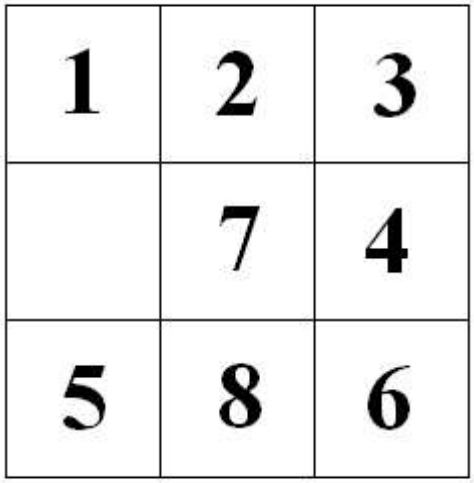


Bild 7 : aktuelle Spielsituation

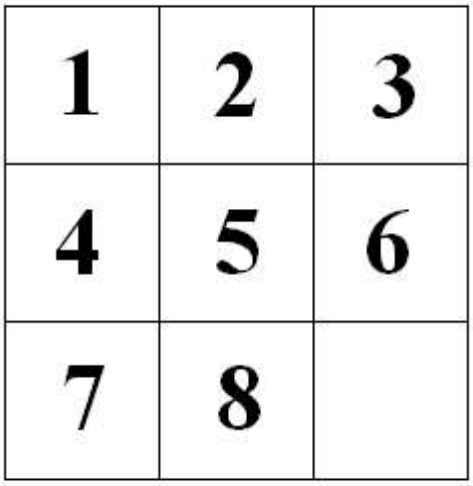


Bild 8 : Endsituation

Beispiel Schiebepuzzel:

h1() = 4 (7,4,5,6 sind nicht auf ihrer Position)
h2() = 7 (Die Entfernungen fließen mit ein)

h2() hat eine größere Zahl für denselben Zustand als h1(). Dadurch lässt sich allein Zahlenmäßig feststellen, das es wohl mit h2() feinere Unterschiede in der Bewertung der Spielsituation gibt als bei h1(). (Ähnlich dem Schulnotenunterschied zwischen der Realschule (1-6) und dem Gymnasium (0-15))

| | Search Cost | | | Effective Branching Factor | | |
|---|---|---|---|---|---|---|
| $d$ | IDS | A*($h_1$) | A*($h_2$) | IDS | A*($h_1$) | A*($h_2$) |
| 2 | 10 | 6 | 6 | 2.45 | 1.79 | 1.79 |
| 4 | 112 | 13 | 12 | 2.87 | 1.48 | 1.45 |
| 6 | 680 | 20 | 18 | 2.73 | 1.34 | 1.30 |
| 8 | 6384 | 39 | 25 | 2.80 | 1.33 | 1.24 |
| 10 | 47127 | 93 | 39 | 2.79 | 1.38 | 1.22 |
| 12 | 364404 | 227 | 73 | 2.78 | 1.42 | 1.24 |
| 14 | 3473941 | 539 | 113 | 2.83 | 1.44 | 1.23 |
| 16 | – | 1301 | 211 | – | 1.45 | 1.25 |
| 18 | – | 3056 | 363 | – | 1.46 | 1.26 |
| 20 | – | 7276 | 676 | – | 1.47 | 1.27 |
| 22 | – | 18094 | 1219 | – | 1.48 | 1.28 |
| 24 | – | 39135 | 1641 | – | 1.48 | 1.26 |

Empirische Evaluation (Durchschnitt über 100 Beispiele, $d$: Lösungstiefe)

Bild 9 : Gegenüberstellung der Kosten der Schrittweisen vertiefenden Suche und einer Heuristiksuche mit der Heuristik h1() und h2().

In Bild 9 ist zu erkennen, das die Heuristik h2() schneller zum Ziel gelangt als h1(). Um eine Bewertung zu uniformierten Suchverfahren zu bekommen wurde hier noch die schrittweise vertiefende Suche angezeigt.
Aufgezeigt sind die durchschnittlichen Schritte, nach welchen der zufällig „versteckte“ Endzustand gefunden wurde (linke Seite). Auf der rechten Seite sind die durchschnittlichen Kosten (Speicher- und Zeitaufwand) pro Sprung aufgeführt. Bei der IDS (iterated depth search) wird nach einer Tiefe von 14 der Versuch abgebrochen, da er zu lange dauern würde.

**Suche mit schrittweiser lokaler Verbesserung**

Mit diesen Heuristischen Funktionen und deren Gewichtung können blinde Suchverfahren verbessert werden (Die Auswahl des Knotens der als nächstes Expandiert wird). Gut bewertete Knoten werden weiter vorne in der Agenda eingefügt.Wird bei der Expansion eines Knotens die Kosten seiner Nachfolgeknoten ermittelt und sie aufgrund dieser Bewertung sortiert in die Agenda eingefügt (nicht unbedingt alle Nachfolger), so spricht man von: „Suche mit schrittweiser lokaler Verbesserung“.

## 3.2 Bergsteigen

Beim Bergsteigen wird eine Tiefensuche durchgeführt, bei der alle Nachfolgeknoten nach ihrer Bewertung sortiert an den Anfang der Agenda eingefügt werden.
Dadurch gelangt dieses Suchverfahren

## 3.3 optim. Bergsteigen (randomisiert)

Beim optimistischen Bergsteigen wird im Prinzip vorgegangen wie beim Bergsteigen, nur das diesmal beim expandieren eines Knotens (Setzen der Nachfolgeknoten auf die Agenda L), nur der jeweils beste Nachfolgeknoten auf die Liste gesetzt wird. Das wird solange durchgeführt bis entweder ein Endzustand erreicht ist, oder das Verfahren keine größeren Veränderungen in der Heuristik aufweist. Das Bedeutet im Detail: Es wird nur in die Tiefe gesucht, nach der besten Bewertung der Heuristik. Ein Aufsteigen im Suchbaum ist nicht mehr möglich, da die Geschwisterknoten des besten Nachfolgeknotens gelöscht wurden. Stellt man sich die Suchlandschaft als 3 Dimensionale Berglandschaft vor, so wird bei der optimistischen Suche immer abwärts Richtung Tal gegangen. Niemals wird ein Berg wieder bestiegen. Das optimistische Bergsteigen ist sehr schnell und Speicherfreundlich, da es keine Extraknoten speichern muss. Gesucht wird das globale Minimum der Suchlandschaft (der Tiefste Punkt). Die heuristische Schätzfunktion muss möglichst genau arbeiten, damit auch ein Endzustand erreicht werden kann, darf allerdings auch nicht zu viel Kosten!

Dieses Vorgehen führt jedoch auch zu einer Reihe von Problemen:
Wird in der Suchlandschaft ein lokales Minimum oder eine Ebene gefunden, so hält dieses Suchverfahren an diesen Stellen an, da es keine Verbesserung zeigt. Es ist also nicht klar, ob das globale Minimum gefunden wurde, wenn die Suche anhält. Stellt man sich eine

Wüstenlandschaft vor in denen viele kleine lokale Minima vorkommen, so ist dieses Verfahren gänzlich ungeeignet. Doch für Suchgebiete mit relativ wenig lokalen Minima und Ebenen kann die Zeit, die eingespart wird bei dem Verfahren noch genutzt werden, die Suche an einer zufällig gewählten Stelle nocheinmal zu beginnen. Man spricht vom „**randomisierten optimistischen Bergsteigen**“.
Das Verfahren wird solange wiederholt, bis angenommen werden kann, das das globale Minimum gefunden wurde. Dabei wird pro Iteration der beste Endzustand gespeichert und zum Schluss der beste Zustand daraus ermittelt.

## 3.4 Abschließendes Beispiel: Heuristische Schätzfunktionen: TIC TAC TOE:

Bei dem Brettspiel TIC TAC TOE geht es darum drei gleiche Steine in eine Reihe zu bekommen. Das Spiel wird zu Zweit gespielt. Die Spieler sind abwechselnd an der Reihe.

Um nun den nächsten Zug zu errechnen und zu bewerten wurde eine Tabelle aufgestellt. Dabei werden die Spielzustände wie folgt gewichtet:

Kostenfunktion h() = Bewerung des nächsten Spielzugs

| ***Kosten*** | ***Situation des nächsten Zuges*** |
|---|---|
| +1*n | Eigener ALLEIN liegender Stein in Horizont./Vertikal./Diagonal. |
| +5*n | Eigene ZWEI hintereinander lieg. Steine in Horizont./Vertikal./Diagonal. |
| +20. | Gewinnsituation |
| -1*n | Fremder ALLEIN liegender Stein in Horizont./Vertikal./Diagonal. |
| -5*n | Fremde ZWEI hintereinander lieg. Steine in Horizont./Vertikal./Diagonal. |
| -20. | Verlierersituation |

Diese Werte werden aufsummiert und ergeben eine Berwertung der nächsten Spielsituation.

!HIER! : Je höher die Zahl ist, desto besser ist der Zug.

Wenn die Funktion aus beiden Blickwinkeln (also auch vom Gegner aus) betrachtet wird, kann die nächste Situation noch besser eingeschätzt werden.

Bild 10 und Bild 11:

Ein X ganz links oben im Spielfeld sorgt dafür, das 3 x mit 5 Punkten bewertet wird. Das sorgt für einem guten Wert, der sonst nicht übertroffen wird. Stellt das Programm nun auch den Zug auf, den der Gegenspieler nun setzen würde, so erhält es einen sehr niedrigen Wert, wenn ein O auf die linke obere Ecke gesetzt würde. Der Gegenspieler hätte dadurch gewonnen. Dadurch erhält das Programm die Information, das oben links der beste Zug für sich UND für den Gegenspieler wäre. Es ist also doppelt Ratsam dort seinen X Stein zu plazieren!

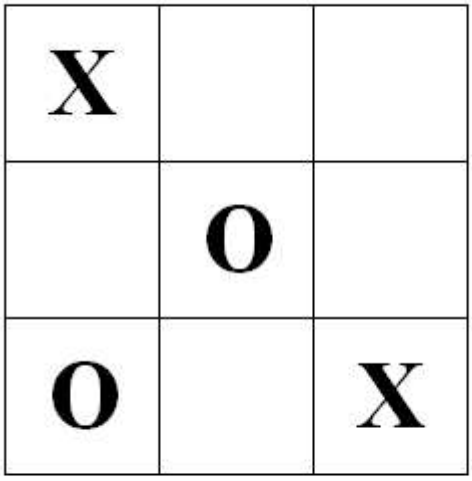


Bild 10: Ausgangssituation: O = Gegenspieler, X = Computer

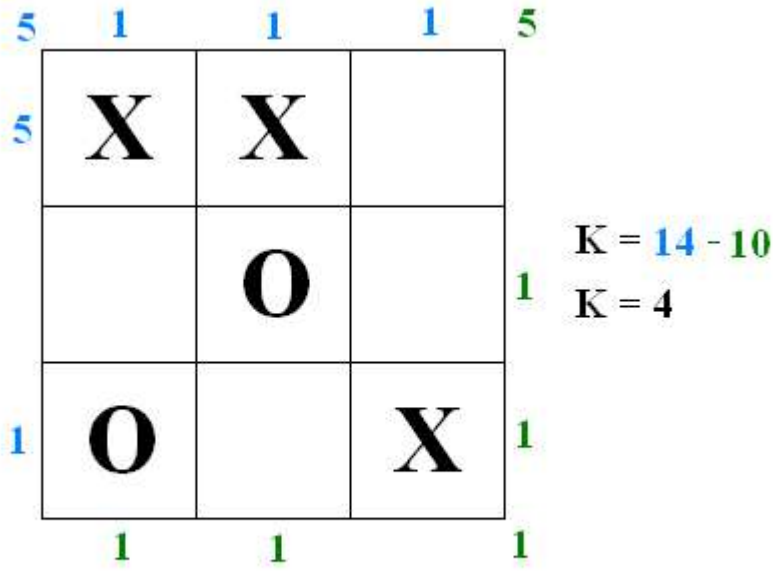


Bild 11 : Bewertung einer möglichen Spielsituation (K = Kosten)

# Literatur:


**Buch:**

Handbuch der Künstlichen Intelligenz: Kaptiel 4.1 – 4.2

**Internet:**

http://www.ki.informatik.uni-frankfurt.de/lehre/WS2002/KI/skript/KI-2Suche.pdf
(leider aus dem Internet entfernt worden)

http://www.techfak.uni-bielefeld.de/~jung/01ki2/Termin01klein.pdf

http://www.pearson-studium.de/media_remote/katalog/bsp/3827370892bsp.pdf

http://nakula.rvs.uni-bielefeld.de/~mirco/download/KI-Zusammenfassung.pdf

http://www.iicm.edu/greif/images/node1.html

http://www.informatik.uni-ulm.de/ki/Edu/Vorlesungen/GdKI/WS0304/skript/04-Infsuche.pdf

http://fstolzenburg.hs-harz.de/ki/folien/heuristik.pdf