Disjoint Pattern Database Heuristics

Seminar im SS 10

Einleitung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Suche nach Möglichkeiten Probleminstanzen effizient zu lösen
- Verschiedene Ansätze möglich
- Hier: Anwendung von IDA* Algorithmus auf disjunkten Datenbanken, um eine erste optimale Lösung zu finden
- Lösungsansatz wird anhand von mehreren Spielen analysiert
- Vergleich der Leistungsfähigkeit von disjunkten Musterdatenbanken-Heuristiken, paarweise Distanz und Distanzen höherer Ordnung

Algorithmen

- A* Algorithmus
  - ‘Kürzester Wege’-Algorithmus
  - Optimalitätseigenschaft
  - Verwendet Schätzfunktion z.B. Luftlinienentfernung
  - kann mit Prioritätswarte-schlange implementiert werden
  - Terminert wenn Zielknoten erreicht oder wenn die Warteschlange leer ist
  - Laufzeit: $O(|V|^2)$

- IDA* Algorithmus
  - Iterative Tiefensuche auf A*
  - Festlegen eines Tiefenlimits der Knoten
  - Optimalitätseigenschaft bleibt wie in A* erhalten
  - Laufzeit abhängig von: [8]
    - Verzweigungsfaktor (branching factor)
    - Heuristischen Verteilung
    - Optimalen Lösungskosten
  - Kostenfunktion:

$$f(n) = g(n) + h(n)$$

A* Algorithmus

- jeder Knoten besitzt als Zusatzinformation den minimalen Weg zum Ziel

- Summe aus Abstand vom Start zum ausgewählten Knoten und Abstand vom Knoten zum Ziel (ist geschätzt) muss minimal sein

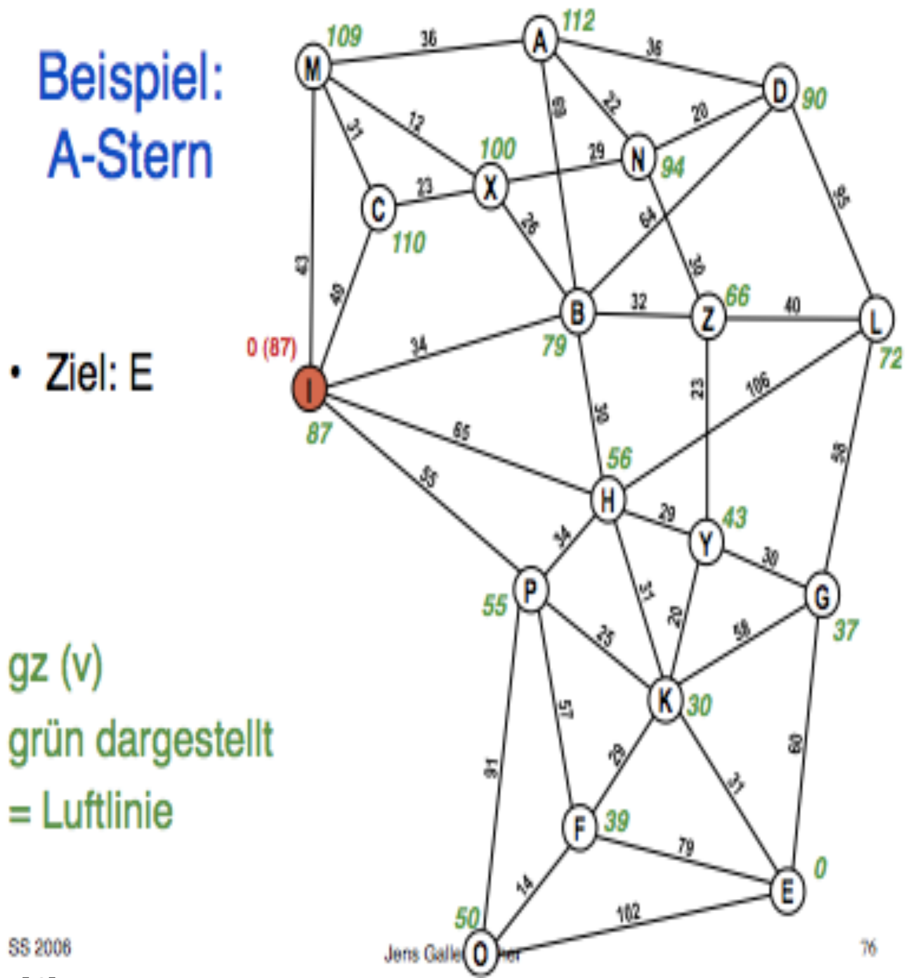
A-Stern-Algorithmus (vereinfacht)

```
/* Knotenmarkierung gs(v)=(errechneter) Abstand vom Start,
   gz(v)=(geschätzer) Abstand vom Ziel,
   M3 implizit definiert als V \ ( M1 ∪ M2 ) */
for (all v ∈V) { gz(v) = Luftlinie; if ( v ≠ s ) gs(v) = ∞; } /* Initialisierung */
M1 = { s }; M2 = { }; gs(s) = 0;
for (all v ∈ s.neighbors( ) ) {        /* fülle M2 initial */
    M2 = M2 ∪ { v };
    gs(v) = c( (s, v) );
}
while ( M2 ≠ { } ) {
    v = Knoten in M2 mit minimalem (gs (v) + gz (v));
    M2 = M2 \ { v };  M1 = M1 ∪ { v };         /* verschiebe v */
    for (all w ∈ v.neighbours( ) ) {
        if ( w ∈ M3 ) {    /* verschiebe Nachbarn und passe L an */
                M2 = M2 ∪ { w }; gs(w) = gs(v) + c( (v, w) );
        } else if ( gs(w) > gs(v) + c( (v, w) ) )  gs(w) = gs(v) + c( (v,w) );
    }
}
```

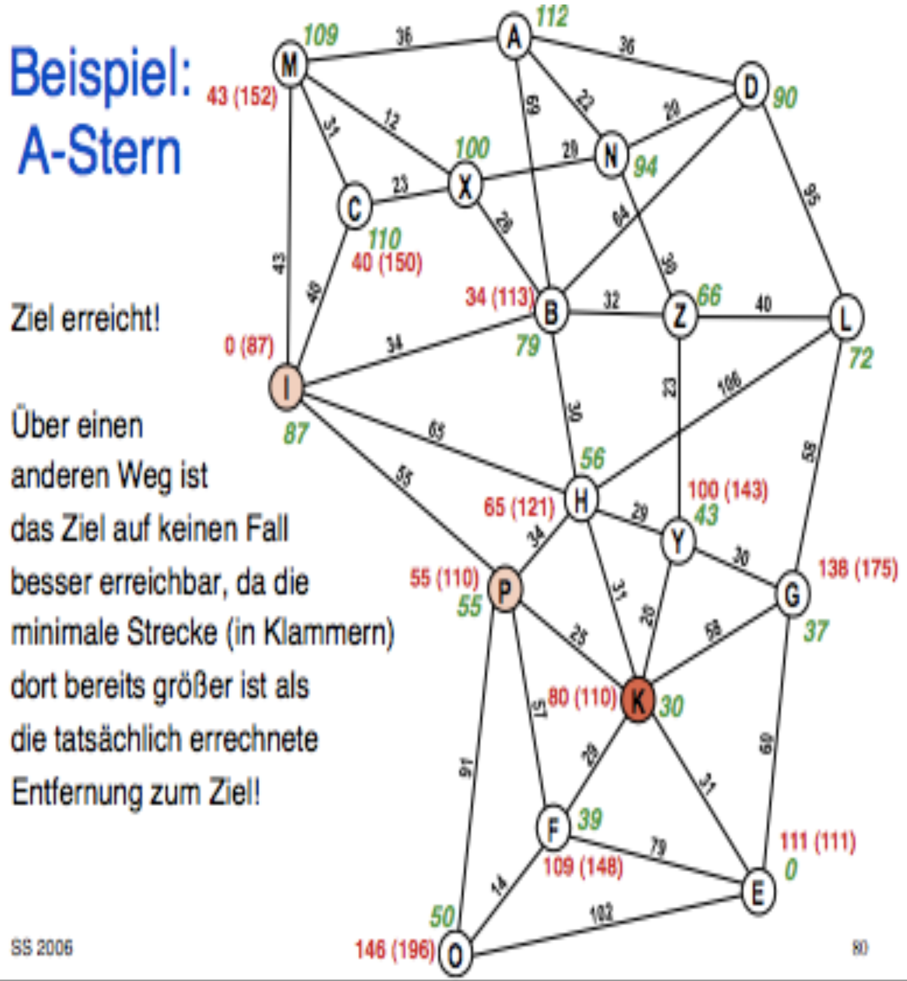
SS 2006 Jens Gallenbacher 1

[6]

Beispiel A* Algorithmus



[6]

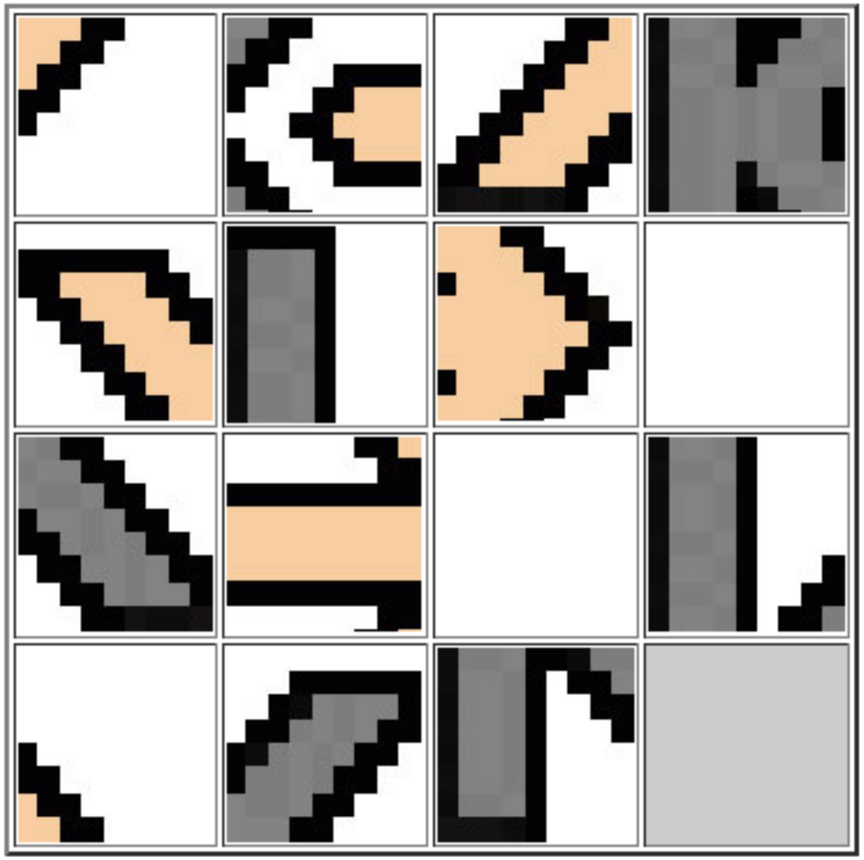


[6]

Betrachtete Spiele 1/2

- Schiebepuzzle
  - Ist der allgemeine Ausdruck für Puzzle mit beweglichen Fliesen
  - Die bekanntesten sind das Fifteen Puzzle und das Twenty-Four Puzzle

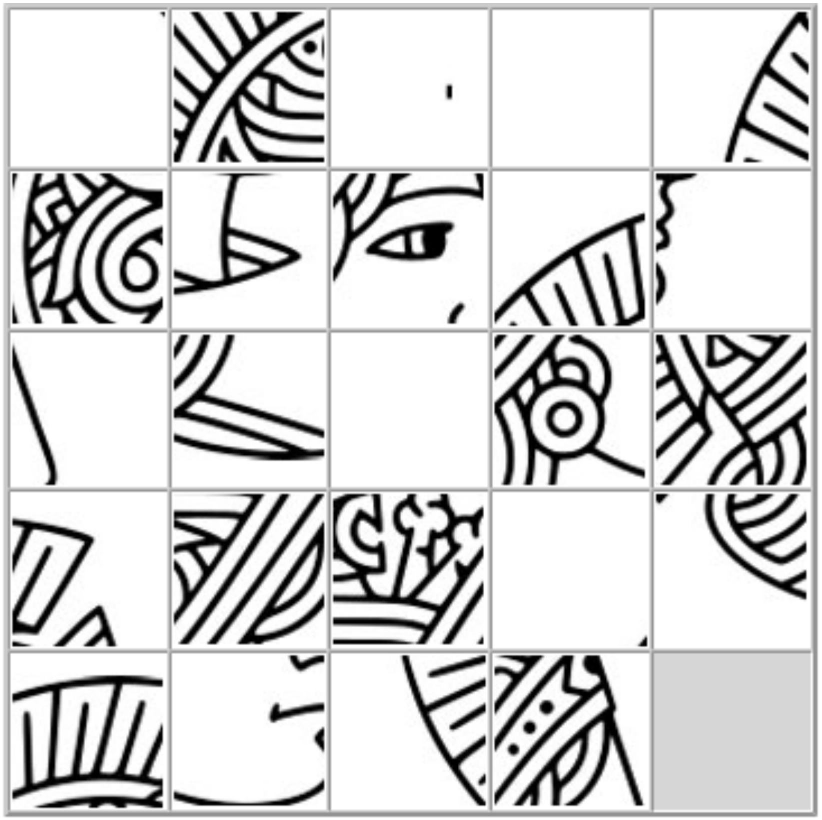
- 15er Puzzle



Quelle: eigene Darstellung

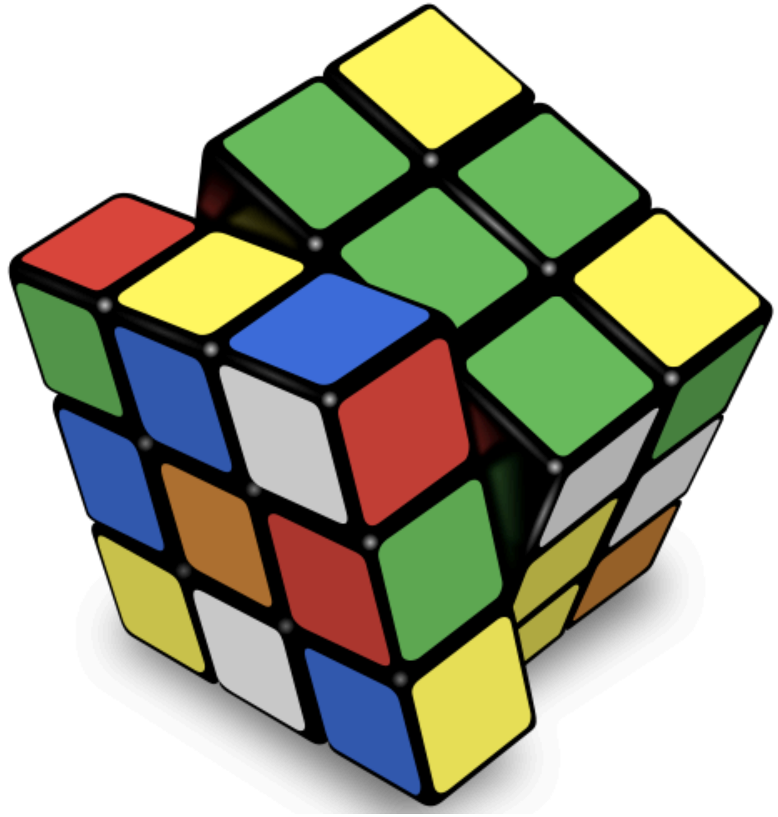
Betrachtete Spiele 2/2

- 24er Puzzle



Quelle: eigene Darstellung

- Rubik’s Cube (Zauberwürfel)



Quelle: http://upload.wikimedia.org/wikipedia/commons/b/b6/Rubik%27s_cube_v3.svg

Heuristiken

TECHNISCHE UNIVERSITÄT DARMSTADT

- Manhattan Distanz
  - Einfachste und am meisten verwendete Heuristik
  - Berechnet mit der Formel: $d(x_1,x_2) = \sum_A d_A(v_{1,A},v_{2,A})$
  - 2 Möglichkeiten der Nutzung:
    - Vereinfachung des Problems, indem Leerheitsbedingung gelockert
    - Jede Fliese ist ein Teilproblem
  - Kosten der Züge entsprechen genau der Manhattan Distanz von Anfangsposition zur Zielposition

Heuristiken

- Pattern database (Muster-Datenbanken)[3]
  - Pattern = partielle Spezifikation eines Zustands (oder Permutation)
  - Blank = leere Fliese
  - Pattern database = Menge aller pattern, die durch Permutation eines Zielmusters gewonnen werden kann
- Nicht-additive Musterdatenbanken
  - Randfliesen und die leeren Fliese werden zur Bestimmung eines Zustands benötigt
  - Anwendung des IDA* Algorithmus um eine optimale Lösung zu erreichen
- Disjunkte Muster-Datenbanken
  - Einteilung in disjunkte Gruppen
  - Idee: Summe der Werte für Lösung benutzen

Nicht-additive Musterdatenbank

- Bsp.: 15er Puzzle

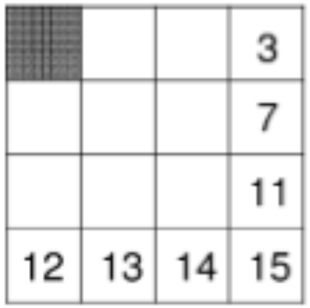


Fig. 2. The fringe pattern for the Fifteen Puzzle.

- Es gibt 7 Randfliesen und eine leere Fliese

- 16 verschiedene Standorte
- Es ergeben sich

  16!/(16-8)! = 518.918.400 Permutationsmöglichkeiten
- Für jede Permutation wird die Anzahl der Züge gespeichert
- Speichergröße:
  - pro Permutation 1Byte
  - insgesamt 495 MB
- Optimale Lösung über Anwendung des IDA*

Nicht-additive Musterdatenbank

- Bsp.: Rubic's cube

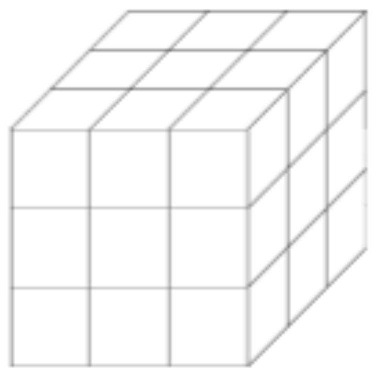


Fig. 3. 3 × 3 × 3 Rubik's Cube.

  - Enthält 4,3252x10^{19} verschiedene erreichbare Zustände
  - 8 Eckwürfelchen, 12 Randwürfelchen

- Speicherbedarf:
  - Eckwürfelchen je 4 Bits
    => gesamt: ca. 42 MB
  - 6 Randwürfelchen
    => gesamt: ca. 20 MB
  - Insgesamt:
    42MB + 20MB + 20MB = 82MB

- Optimale Lösung durch Anwendung des IDA* auf das Maximum der drei Werte

- Mittlere optimale Lösungslänge ist 18 Züge pro Gruppe

Nicht-additive Musterdatenbank

- Größte Einschränkung der nicht-additiven Musterdatenbanken:

  ⇒ Große Probleme können nicht gelöst werden

  - Beispiel:
    - 24er Puzzle die Musterdatenbank hätte 25!/(25-n-1)! Einträge
    - Datenbank mit 6 Fliesen und einer leeren benötigt 2,4 Billionen Einträge

- Bei mehreren Musterdatenbanken wird das Maximum der Werte genommen

  ⇒ da die Werte alle erforderlichen Züge zum Zielzustand und somit zur Lösung des Problems enthalten

Disjunkte Musterdatenbank

- Grundidee:
  Anstatt des Maximums wird die Summe der Werte aus den Musterdatenbanken genutzt
- Dazu Aufteilung der Fliesen in disjunkte Gruppen
- Vorberechnung der Anzahl der Züge der Fliesen pro Gruppe um ihren Zielzustand / Zielposition zu erreichen
- Abspeichern der Ergebnisse pro Gruppe in einer Tabelle
- Lösung für ein Problem: Berechnugn der Werte für jede Fliese in der Gruppe und Addition dieser zu einem Lösungswert

Unterschied zwischen nicht-additiver und disjunkter Musterdatenbank

- Die nicht-additive Musterbank umfasst alle erforderlichen Züge zur Lösung des Fliesenmusters
  - Bei mehreren Musterdatenbanken wird das Maximum der Werte gewählt
- In der disjunkten Musterdatenbank werden die Anzahl der Zügen der Fliessen in der Gruppe gezählt
  - Berücksichtigt die leere Position nicht (geringere Größe)
  - Einfaches Beispiel: Manhattan Distanz

Anwendung einer disjunkten Datenbank auf ein 15er Puzzel

- Einteilung in Gruppe von 7 und 8 Fliesen
- Einträge: 7 Fliesen 57.657.600 mit Null bis 33 Zügen

  8 Fliesen 518.918.400 mit Null bis 38 Zügen
- In keinem Fall ist die leere Position ein Teil des Indexes der Datenbank
- Allgemeine Regel: die Partitionierungen der Fliesen zu einer Gruppe von Fliesen, die nahe beieinander in den Zielzustand gehen

  => da diese Fliesen miteinander interagieren

Experimentelle Ergebnisse 1/3

TECHNISCHE UNIVERSITÄT DARMSTADT

15er Puzzle

- Anwendung des IDA* auf 1.000 zufällig gewählte Probleminstanzen
- Durchschnittliche optimale Lösungslänge 52,522 Züge

Table 1
Experimental results on the Fifteen Puzzle

Heuristic function	Value	Nodes	Nodes/sec	Seconds	All solutions
Manhattan distance	36.940	401,189,630	7,269,026	55.192	1,178,106,819
Linear conflicts	38.788	40,224,625	4,142,193	9.710	144,965,491
Disjoint database	44.752	136,289	2,174,362	0.063	472,595
Disjoint + reflected	45.630	36,710	1,377,630	0.027	130,367

Experimentelle Ergebnisse 2/3

24er Puzzle

- Lösung mit dem IDA*, verwenden von disjunkten Datenbanken und seine Reflexion über die Hauptdiagonale
- Jede Gruppe besteht aus 6 Fliesen mit 127.512.000 Einträgen
- Allgemeine Heuristik: Maximum der Werte von originalen und reflektierten Datenbank
- Lösung über IDA* und die Verwendung von disjunkten Datenbank und seiner Reflexion über die Hauptdiagonale

Experimentelle Ergebnisse 3/3

- Teilung des 24er Puzzle's

R.E. Korf, A. Felner / Artificial Intelligence 134 (2002) 9–22

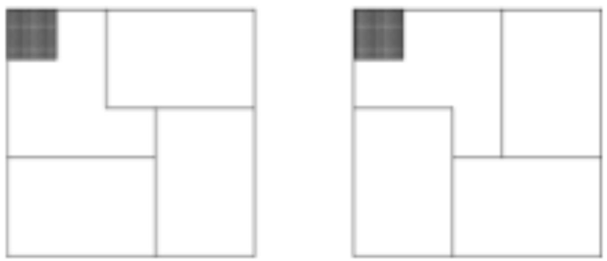


Fig. 5. Disjoint databases for Twenty-Four Puzzle.

- 3 Blöcke sind gleich, 1 Block unregelmäßig (leere Fliese)
- Nur 2 verschiedene Tabellen
- Anzahl benötigeter Züge:
  gleich -> je 0 bis 35 Züge
  unregelm. -> 0 bis 34 Züge
- Speicher: 1 Byte pro Eintrag

=> Insgesamt 243 MB

Ergebnisse

aus 50 zufällig ausgewählten Probleminstanzen

- Durchschnittliche optimale Lösungslänge: 100,78 Züge
- Durchschnittliche Anzahl erzeugter Knoten: 360.892.479.671
- Laufzeit pro Problem: 18 Sek. - 23 Tage

(durschnittl. 2 Tage pro Problem)

Table 2
Twenty-Four Puzzle data

No	Initial state	Length	Nodes
1	14 5 9 2 18 8 23 19 12 17 15 0 10 20 4 6 11 21 1 7 24 3 16 22 13	95	2,031,102,635
2	16 5 1 12 6 24 17 9 2 22 4 10 13 18 19 20 0 23 7 21 15 11 8 3 14	96	211,884,984,525
3	6 0 24 14 8 5 21 19 9 17 16 20 10 13 2 15 11 22 1 3 7 23 4 18 12	97	21,148,144,928
4	18 14 0 9 8 3 7 19 2 15 5 12 1 13 24 23 4 21 10 20 16 22 11 6 17	98	10,991,471,966
5	17 1 20 9 16 2 22 19 14 5 15 21 0 3 24 23 18 13 12 7 10 8 6 4 11	100	2,899,007,625
6	2 0 10 19 1 4 16 3 15 20 22 9 6 18 5 13 12 21 8 17 23 11 24 7 14	101	103,460,814,368
7	21 22 15 9 24 12 16 23 2 8 5 18 17 7 10 14 13 4 0 6 20 11 3 1 19	104	106,321,592,792

Paarweise Distanz und Distanzen höherer Ordnung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Einteilung in 2-Fliesen- und 3-Fliesen Musterdatenbanken
- Vorteil:
  - Wechselwirkungen zwischen Fliesen werden aufgefangen (nicht nur in einer Gruppe)
  - Wenig Speicherbedarf (jeweils nur 3 MB)
- Nachteil:
  - Operationskosten sind teurer
- Ergebnisse:

15er Puzzle	Durchschnittliche Laufzeit	Durchschnittlich erzeugte Knoten
Manhattan Distanz	53 s	< 700.000
Disjunkte Datenbank	27 ms	< 700.000
2-bzw. 3-Fliesen-Datenbank	5 s	700.000

Paarweise Distanz und Distanzen höherer Ordnung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Im 24er Puzzle:
  - 2- und 3-Fliesen-Datenbank erzeugt weniger Knoten als die disjunkte
  - Aber Heuristik schwieriger zu berechnen, daher Laufzeit länger

  => Disjunkte Datenbank hat hier bessere Leistung (weiterführende Informationen unter [5])

- Ergebnis:
  - Leistungsunterschied zwischen disjunkten Datenbanken und 2- und 3-Fliesen-Datenbank im 15er Puzzel größer
  - Disjunkte Datenbank Heuristiken für diese beiden Probleme effektiver und einfacher

  => **Aber** nicht für größere Versionen der Probleme oder in einem anderen Gebiet geeignet

Fazit

- Optimale Lösungen für 50 zufällige Probleminstanzen des 24er Puzzle's
  - Durchschnittlich 100 Züge für Lösung
- Optimale Lösung für das 15er Puzzle gefunden (27 ms)
- Vewendete Heuristik: Musterdatenbank mit IDA* Algorithmus
- Musterdatenbank prüft:
  - Kosten der berechneten Lösung für unabhängige Teilziele
  - Kosten der berechneten Lösung mehrerer Teilziele unter Berücksichtigung der Wechselwirkungen untereinander

Fazit

- Disjunkte Musterdatenbank:
  - Um Werte aus verschiedenen Datenbanken zusammenfügen zu können
  - Einteilung der Teilziele in disjunkte Gruppen
  - Zusammenfügen der gemeinsamen Kosten der Lösung für alle Teilprobleme einer Gruppe
  - Kosten teurer in der Suche, aber weniger erzeugte Knoten

Fragen

Literaturverzeichnis

[1] Felner, Ariel, Korf, Richard E., *Disjoint pattern database heuristics.* Artificial Intelligence 134 (2002) 9-22.

[2] Felner, Ariel, Korf, Richard E., *Disjoint pattern database heuristics.* September 5,2001.

[3] Culberson, Joseph C., Schaeffer, Jonathan, *Pattern Database. Computational Intelligence.* Volume 14, Number 3, 1998.

[4] Felner,Ariel, Korf, Richard E., Hanan, Sarit *Additive Pattern Database Heuristics.* Journal of Artificial Intelligence Research 22 (2004) 279-318.

[5] Felner, Ariel, *Improving search techniques and using them on different environments.* Ph.D. Thesis, Dept. of Mathematics and Computer Science, Bar-Ilan University, Ramat-Gan, Israel, 2001.

[6] Gallenbacher,Jens, *Abenteuer Informatik:IT zum Anfassen – von Routenplaner bis Online Banking*. Spektrum Akademischer Verlag, Auflage 2, 2008.

[7] Edelkamp,Stefan, *Data Structures and Learning Algorithms in State Space Search*.AKA,DISKI, Volume 201, 1999.

[8] Korf, Richard E., Edelkamp, Stefan, Reid, Michael, *Time Complexity of Iterative-Deepening-A**. Artificial Intelligence 129 (2001), no. 1-2, pp.199-218.

Sobald nicht anders angegeben sind alle Darstellungen aus (Korf, Fellner 2002)