




Informatik II Backtracking



G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de

Allgemeines Problem-Statement

- Gegeben:
 1. Eine Reihe von Variablen, für die man eine **Belegung** sucht
 2. **Constraints** zwischen den Variablen, d.h., Belegungen, die sich gegenseitig ausschließen
- Gesucht: eine (oder alle) Belegungen der Variablen, die keinen Constraint verletzen (a.k.a. **Konflikt**)
- Dabei bestehen oft folgende Probleme:
 - Falls man eine konfliktfreie Teilbelegung hat, hat man i.A. nicht genug Informationen, um die restlichen Variablen sofort konfliktfrei zu belegen → man muß ausprobieren
 - Manche Teilbelegungen können gar nicht zu einer konfliktfreien Belegung erweitert werden (eine "**Sackgasse**")
- **Backtracking** ist der methodische Weg, verschiedene Folgen von Belegungen zu **probieren**, bis eine gefunden wird, die konfliktfrei ist

G. Zachmann Informatik 2 — SS 11 Backtracking 2




Chatsworth Garden Maze

G. Zachmann Informatik 2 — SS 11


Backtracking 3

Beispiele

- Ausweg aus einem Labyrinth:
 - Finde einen Weg vom Startpunkt zum Zielpunkt
 - An jeder Kreuzung gibt es 2-3 Entscheidungsmöglichkeiten: geradeaus, rechts, links
 - "Faden der Ariadne" = früheste Beschreibung des Backtracking-Verfahrens



Theseus in the Labyrinth
Sir Edward Burne-Jones (1862)

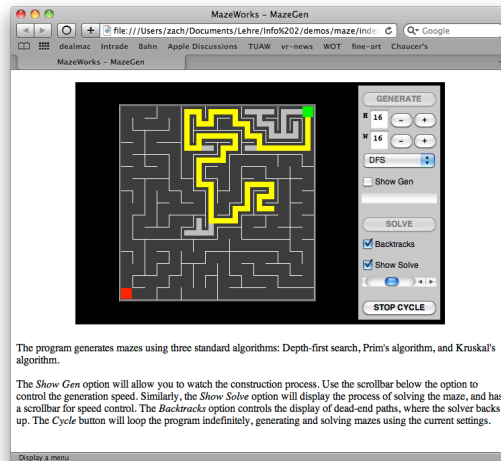


Ariadne
John William Waterhouse (1898)

G. Zachmann Informatik 2 — SS 11

Backtracking 4

- Der Algorithmus ist so einfach, dass eine Demo reicht:



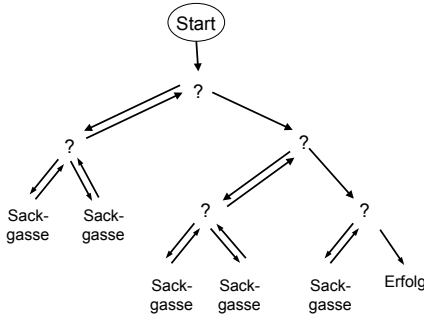
<http://www.mazeworks.com/mazegen/>

- Solitaire (Brettspiel):
 - Alle Löcher, bis auf eins, sind mit Stiften besetzt
 - Man darf mit einem Stift über einen anderen springen
 - Übersprungene Stifte werden entfernt
 - Ziel: alle Stifte, bis auf den letzten, zu entfernen



- Internet-Browsing

Der abstrakte Backtracking-Algorithmus



```


explore( node ):
if node_ist Blatt:
    if node == Zielknoten:
        return "Erfolg"
    else:
        return "Sackgasse"
    foreach Kind c von node:
        explore( c )
    if c erfolgreich:
        return "Erfolg"
    return "Sackgasse"

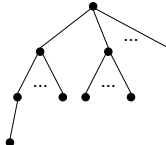

```

- Backtracking kann als Suche nach einem speziellen "Zielblatt" in einem sog. **Such-Baum** gesehen werden
- Achtung: dieser Baum wird beim Backtracking nie real aufgebaut!
- Wesentliche Komponente: ein Kriterium, so daß Sackgassen möglichst weit oben erkannt werden
- Realisierung der Pfadrückverfolgung durch Rekursion

G. Zachmann Informatik 2 — SS 11 Backtracking 7

Zusammenhang zu Depth-First-Search

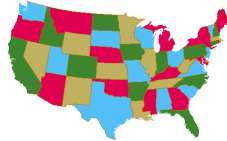
- Ein Knoten wird als **nützlich** bezeichnet, falls er zu einer Lösung führen **könnte**, sonst **nicht nützlich**
- Hauptidee: Backtracking besteht aus der Anwendung von DFS auf den Such-Baum, wobei für jeden Knoten entschieden wird
 - ob er nützlich ist → weiter absteigen;
 - ob er nicht nützlich ist → zurück zum Vaterknoten (**back-tracking**)
- Weitere Betrachtungsweise:
 - Der Suchraum wird durch den Baum ausgefüllt und mit Hilfe von Rekursion und Backtracking durchsucht, um eine Lösung zu finden



G. Zachmann Informatik 2 — SS 11 Backtracking 8

Das Färbungsproblem

- Färben einer Landkarte:
 - Eine Karte soll mit nur 4 Farben gefärbt werden
 - Bedingung: benachbarte Länder (gemeinsame Grenze) müssen verschieden gefärbt werden

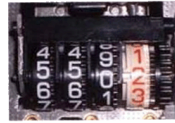


- Der **Vier-Farben-Satz** [1976]:
Jede Landkarte in der Ebene lässt sich mit höchstens 4 Farben so färben, daß keine 2 Länder mit gemeinsamer Grenze die gleiche Farbe haben.

G. Zachmann Informatik 2 — SS 11 Backtracking 9

Naive Lösung: erschöpfende Suche (*exhaustive search*)

- Länder werden durchnummeriert, dito die Farben
- Erzeugung aller möglichen Färbungen durch die "Tacho-Mehode":
 - Jedes Rädchen = ein Land
 - Starte mit Zählerstand 0...0
 - Drehe rechtes Zählrädchen einmal durch alle Ziffern (= Farben)
 - Bei Übergang von 3 → 0 im rechten Rädchen: drehe zweitrechtes Rädchen 1 Ziffer weiter
 - Bei jedem Zählerstand: überprüfe Konsistenz (4-Farben-Bedingung)



G. Zachmann Informatik 2 — SS 11 Backtracking 10

Diagram illustrating the backtracking process for a 3x8 grid with 8 numbered blocks. The blocks are: 1 (top row), 2 (row 2, col 2-3), 3 (row 2, col 4-5), 4 (row 3, col 4-5), 5 (row 3, col 1-2), 6 (row 3, col 6-7), 7 (row 3, col 7-8), and 8 (bottom row). The diagram shows four stages of block placement with corresponding binary vectors below each grid. The vectors represent the state of the grid cells (1-8) where 0 is empty and a number indicates the block occupying that cell. The process shows how blocks are placed and then removed (backtracked) when they lead to a dead end.

Stage 1: All cells are empty. Vector: 0 0 0 0 0 0 0 0

Stage 2: Block 8 is placed at (3,7). Vector: 0 0 0 0 0 0 0 1

Stage 3: Block 8 is removed. Block 2 is placed at (2,2). Vector: 0 0 0 0 0 0 0 2

Stage 4: Block 2 is removed. Block 3 is placed at (2,4). Vector: 0 0 0 0 0 0 0 3

Stage 5: Block 3 is removed. Block 4 is placed at (3,4). Vector: 0 0 0 0 0 0 1 0

Stage 6: Block 4 is removed. Block 5 is placed at (3,1). Block 6 is placed at (3,6). Block 7 is placed at (3,7). Block 8 is placed at (3,8). Vector: 0 0 0 3 0 1 2 1

Stage 7: Block 8 is removed. Block 2 is placed at (2,2). Block 3 is placed at (2,4). Block 4 is placed at (3,4). Block 5 is placed at (3,1). Block 6 is placed at (3,6). Block 7 is placed at (3,7). Block 8 is placed at (3,8). Vector: 0 1 2 0 2 0 1 3

G. Zachmann Informatik 2 — SS 11 Backtracking 11

- Nachteile:
 - Im ungünstigsten Fall müssen fast alle Kombinationen durchprobiert werden
 - Im average case müssen die Hälfte aller Kombinationen probiert werden
 - Der Algorithmus zählt weiter, obwohl klar ist, daß einige Kombinationen übersprungen werden können

G. Zachmann Informatik 2 — SS 11 Backtracking 12

Eine Backtracking-Lösung

- Idee: ein Land färben und dann versuchen, das nächste zu färben, so daß folgende Bedingungen erfüllt sind:
 - Die benachbarten, bereits gefärbten Länder haben unterschiedliche Farben
 - Deines der benachbarten, noch ungefärbten Länder wird un färbbar
- Diese lokalen Bedingungen können erfüllt sein, aber dennoch zu keiner Lösung führen (Sackgasse)

G. Zachmann Informatik 2 — SS 11 Backtracking 13

7 ist un färbbar geworden!
→ Backtracking

7 wieder un färbbar

G. Zachmann Informatik 2 — SS 11 Backtracking 14

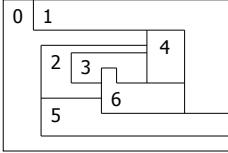
- Vorteil: wesentlich effizienter als erschöpfende Suche
- Nachteil: abhängig von der Nummerierung, d.h. bei ungünstiger Nummerierung ist der Aufwand ebenfalls enorm hoch
- Die Datenstruktur muß folgendes speichern:
 - Länder sind Instanzen einer Klasse **Land**
 - Zu jedem Land alle seine Nachbarn (Liste von Zeigern)
 - Ein Array der Länge 4 :
 - Wert 1 an der Stelle i = Land **ist** in der Farbe i gefärbt
 - Wert -1 = Land **kann** mit Farbe i gefärbt werden
 - 0 = Land kann **nicht** mit der Farbe i gefärbt werden
 - Initialisiere alle Stellen (= Farben) mit -1

G. Zachmann Informatik 2 — SS 11 Backtracking 15

Beispiel-Implementierung in Python

- Achtung:
 - Im folgenden wird die "Unfärbbar"-Heuristik weggelassen
 - Es gibt nur 1 Klasse für alle Länder zusammen
 - Bessere Implementierung à la vorige Folie → Übungsaufgabe
- Datenstruktur:
 - Klasse **Map** mit 2 internen Arrays
 - Array **colors**, wobei **colors[i]** die Farbe des i -ten Landes ist
 - Array **neighbors** mit den benachbarten Ländern
 - Beispiel: **neighbors[5][3] == 8** bedeutet: Land 8 ist zu Land 5 benachbart (hier ist es zufällig das dritte Land in der Liste der zu Land 5 benachbarten Länder)

G. Zachmann Informatik 2 — SS 11 Backtracking 16



```

class Map(object):
    def __init__(self):
        self.neighbors = 7 * [None]
        self.neighbors[0] = [ 1,4,2,5 ]
        self.neighbors[1] = [ 0,4,6,5 ]
        self.neighbors[2] = [ 0,4,3,6,5 ]
        self.neighbors[3] = [ 2,4,6 ]
        self.neighbors[4] = [ 0,1,6,3,2 ]
        self.neighbors[5] = [ 2,6,1,0 ]
        self.neighbors[6] = [ 2,3,4,1,5 ]
        self.colors = 7 * [None]

```

G. Zachmann Informatik 2 — SS 11 Backtracking 17

```



def main():
    map = Map()
    print map.explore(0, "red" )
    print map.colors

# Fortsetzung von class Map

def explore( self, country, color ):
    if country >= len( self.neighbors ):
        return True
    if okToColor( country, color ):
        self.colors[country] = color
        for c in [ "red", "yellow", "green", "blue" ]:
            if explore( country+1, c ):
                return True
    return False

```

G. Zachmann Informatik 2 — SS 11 Backtracking 18



```

# Teste, ob irgend ein Nachbar von 'country'
# die Farbe 'color' hat

def okToColor( self, country, color ):
    for n in map[country]:
        if self.colors[ map[country][n] ] == color :
            return False
    return True

```


G. Zachmann Informatik 2 — SS 11 Backtracking 19

Erläuterung

- Es werden alle Länder rekursiv durchlaufen, begonnen wird mit Land 0
- Für jedes Land wird eine Farbe gewählt:
 - Die Farbe muß von denen aller Nachbarn verschieden sein
 - Wenn keine Farbe gefunden werden kann, wird ein Fehler gemeldet
 - Wenn eine Farbe gefunden wird, wird sie benutzt und mit dem nächsten Land fortgefahren
 - Wenn alle Länder durchlaufen sind (alle wurden gefärbt), wird Erfolg gemeldet
- Nach der Rückkehr vom obersten Aufruf ist der Algorithmus fertig


G. Zachmann Informatik 2 — SS 11 Backtracking 20



Komplexität

- Exponentieller Aufwand:
 - Anzahl der Kombinationen steigt exponentiell
 - Anzahl der möglichen Sackgassen steigt ebenfalls exponentiell, daher ist Backtracking oftmals deutlich schneller als erschöpfende Suche
- Fällt in die Klasse NP-vollständig:
 - D.h., es gibt wahrscheinlich keinen effizienten (= polynomiellen) Algorithmus

G. Zachmann Informatik 2 — SS 11 Backtracking 21



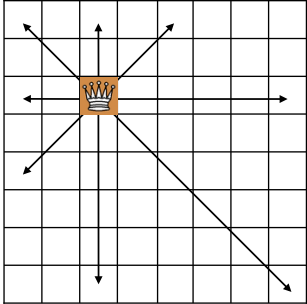
Anwendungen

- Stundenplanproblem
 - Veranstaltungen = Länder
 - Verant., die nicht gleichzeitig ablaufen können, sind "benachbart"
 - Anzahl der Farben, die zum Einfärben benötigt werden = Anzahl der verschiedenen Zeitfenster (minimale Anzahl = *chromatic number* des Graphen)
- Registerzuweisungs-Probleme
- Bandbreitenzuweisungs-Probleme
- Viele mathematische Probleme lassen sich als Knotenfärbeproblem formulieren

G. Zachmann Informatik 2 — SS 11 Backtracking 22

Das Acht-Damen-Problem

- Aufgabe: stelle 8 Damen so auf ein Schachbrett, daß keine Dame eine andere angreift
 - Schachbrett = 8x8 Felder
 - Eine Dame kann jede andere Figur angreifen, die in derselben Zeile, derselben Spalte, oder der Diagonalen steht
 - (Die Dame ist die mächtigste Figur beim Schach)



G. Zachmann Informatik 2 — SS 11 Backtracking 23

Die naive Strategie

- Naive Strategie: alle möglichen Belegungen ausprobieren
 - Anzahl Möglichkeiten = $64 \cdot 63 \cdot 62 \cdot 61 \cdot 60 \cdot 59 \cdot 58 \cdot 57$
 - Zu viele → nicht praktikabel
- Idee:
 - Platziere von vornherein nur eine Dame pro Spalte
 - Starte mit der ersten Spalte; suche für jede eine Zeile, die noch nicht belegt ist
 - Es müssen nur noch $8! = 40\,320$ Damen-Anordnungen auf Diagonalangriffe überprüft werden

G. Zachmann Informatik 2 — SS 11 Backtracking 24

Beispiel

(a) (b) (c)

- Die 5 Damen können sich nicht gegenseitig angreifen, aber jede Position in Spalte 6 wird bedroht
- Backtracking** zu Spalte 5 um ein anderes Feld für die Dame zu probieren
- Backtracking** zu Spalte 4 um ein anderes Feld für die Dame zu probieren und dann Spalte 5 erneut zu betrachten

G. Zachmann Informatik 2 — SS 11 Backtracking 25

Eine Lösung des Problems

1 2 3 4 5 6 7 8

G. Zachmann Informatik 2 — SS 11 Backtracking 26

Der Algorithmus im Detail (hier **zeilenweise**)

```

# Teste, ob Dame in (neue_reihe,neue_spalte) in Konflikt mit
# einer der schon stehenden Damen in 'loesung' ist.
# loesung = Array mit neue_reihe-1 Eintraegen,
# wobei in Zeile i eine Dame auf Feld (i,loesung[i]) steht.

def konflikt( neue_reihe, neue_spalte, loesung ):
    # Stelle sicher, dass neue Dame mit keiner existierenden
    # Dame auf gleicher Spalte oder Diagonale steht
    for reihe in 0,...,neue_reihe-1:
        if ( loesung[reihe] == neue_spalte      # gleiche Spalte
            or
            (neue_reihe - reihe) +             # gleiche Diagonale
            (neue_spalte - loesung[reihe]) == 0
            or
            (neue_reihe - reihe) -             # gleiche Diagonale
            (neue_spalte - loesung[reihe]) == 0
        ):
            return True                        # Konflikt
    return False # alles OK

```

G. Zachmann Informatik 2 — SS 11 Backtracking 27

```

# Berechne alle Loesungen fuer ein Brett der Groesse
# neue_reihe x spalten.
# teilloesungen = alle Loesungen fuer
# (neue_reihe-1) x spalten
# Eine loesung = definiert wie in Fkt konflikt()
# Jedes Array in teilloesungen hat (neue_reihe-1)
# Eintraege, jedes Array im Output hat neue_reihe Elem.

def dame_dazu( neue_reihe, spalten, teilloesungen ):
    neue_loesungen = []
    for loesung in teilloesungen:
        # versuche jede Spalte von neue_reihe
        for neue_spalte in 0,...,spalten:
            if not konflikt( neue_reihe, neue_spalte,
                             loesung ):
                neue_loesung = loesung + [neue_spalte]
                neue_loesungen.append( neue_loesung )
    return neue_loesungen

```

G. Zachmann Informatik 2 — SS 11 Backtracking 28

```


# Erzeuge eine Liste von Lösungen auf einem Brett der
# Groesse (reihen x spalten)
# Eine Lösung wird durch eine Liste der Spaltenpositionen,
# indiziert durch die Reihennummer, angegeben.
# Die Indizes beginnen mit 0!

def damenproblem( reihen, spalten ):
    if reihen <= 0 or spalten <= 0:
        return [ [] ]
        # Ende der Rekursion; Lösung =
        # keine Dame auf dem nicht-existierenden Brett
    else:
        teilloesungen = damenproblem( reihen-1, spalten )
        return dame_dazu(reihen-1, spalten, teilloesungen )

```

G. Zachmann Informatik 2 — SS 11 Backtracking 29

Demo



Backtracking: Die Acht-Damen Aufgabe
(erfordert JavaScript)

Ziel: Acht Damen sollen so auf ein Schachbrett gestellt werden, dass keine Dame von einer anderen Dame "bedroht" wird.

Ein Mausklick auf ein leeres Feld des Schachbretts bringt eine Dame auf dieses Feld. Die "bedrohten" Felder werden angezeigt, das Feld mit der zuletzt gesetzten Dame ist grün markiert.

Jeweils die zuletzt gesetzte Dame kann durch Anklicken wieder vom Schachbrett entfernt werden.

Aktivieren des Schaltknopfs "automatisch" bewirkt den Ablauf eines einfach gehaltenen Zurückverfolgungsalgorithmus ("backtracking") zum Auffinden einer Lösung. Sein Ablauf kann durch Aktivieren der Option "Mausklick" unterbrochen werden. Der Algorithmus endet, wenn eine Lösung gefunden wurde, oder - bei vorangegangener ungünstiger Positionierung von Damen per Mausklick - mit dem leeren Schachbrett. In diesen Fällen kann mit dem Knopf "neue Lösung" nach weiteren Lösungen gesucht werden. Mit Einstellen einer Verzögerung (in Millisekunden) kann die Geschwindigkeit der Bildausgabe angepasst werden.

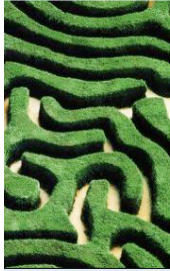
(© H.B. Meyer)

<http://www.faust.fr.bw.schule.de/mhb/backtrack/achtdamen/eight.htm>

G. Zachmann Informatik 2 — SS 11 Backtracking 30

Eine unterhaltsame Lehreinheit zum Thema

Math^o (Prism)^o



Backtracking
"nach vorn wenn möglich, zurück wenn nötig"

[Inhaltsverzeichnis](#) [Arbeitsblatt](#)

[Los geht's!](#)

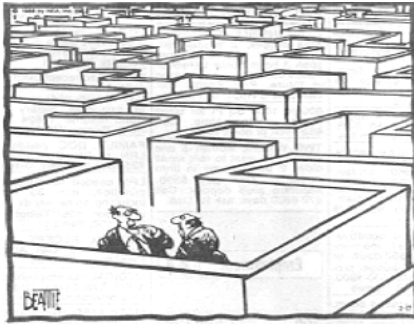
Autoren: Andreas Frommer - 10. Dezember 2004

<http://www.matheprisma.uni-wuppertal.de/Module/BackTr/index.htm>

G. Zachmann Informatik 2 — SS 11 Backtracking 31

Abschließende Bemerkungen

- Aufgaben für Backtracking sind i.A. **nicht** Optimierungsaufgaben, sondern Ja-/Nein-Aufgaben (**Entscheidungsprobleme**)
- Backtracking = **trial & error**
 - Verwenden meistens Rekursion
- Falls Backtracking keine Lösung findet → es gibt keine Lösung!
- Die Algorithmentechnik "Backtracking" stammt von Edsger Dijkstra
 - Das 8-Damen-Problem diente zur Illustration des "structured programming" (1972)



BEATLE
"The exit? Sure...take a right, then left, left again...no, wait...a right, then...no, wait..."

20

G. Zachmann Informatik 2 — SS 11 Backtracking 33