Solving Exact Cover Problems

Martin Nuss[1]

TU Graz, Austria

March 6, 2011

[1]Betreuung: Prof. Winfried Kernbichler

Table of contents

Wie sieht ein Exact Cover Problem aus?

- Jedes Exact Cover Problem lässt sich auf eine logische Matrix abbilden.

	1	2	3	4
A	0	0	1	1
B	1	1	0	0
C	1	1	1	0
D	1	0	0	1
E	0	1	0	0
F	1	0	1	0
G	0	0	1	0

Wie sieht ein Exact Cover Problem aus?

- Jedes Exact Cover Problem lässt sich auf eine logische Matrix abbilden.
- Finde alle Zeilen, sodass in jeder Spalte genau eine 1 vorkommt (oder vice versa): AB, DEF.

	1	2	3	4
A	0	0	1	1
B	1	1	0	0
C	1	1	1	0
D	1	0	0	1
E	0	1	0	0
F	1	0	1	0
G	0	0	1	0

Wie sieht ein Exact Cover Problem aus?

- Beispiel: Ein Schulbus hat in einer Reihe vier Sitzplätze.

Wie sieht ein Exact Cover Problem aus?

- Beispiel: Ein Schulbus hat in einer Reihe vier Sitzplätze.
- Es können nie zwei Schüler auf einem Platz sitzen, ebensowenig soll einer leer bleiben.

	1	2	3	4
A	0	0	1	1
B	1	1	0	0
C	1	1	1	0
D	1	0	0	1
E	0	1	0	0
F	1	0	1	0
G	0	0	1	0

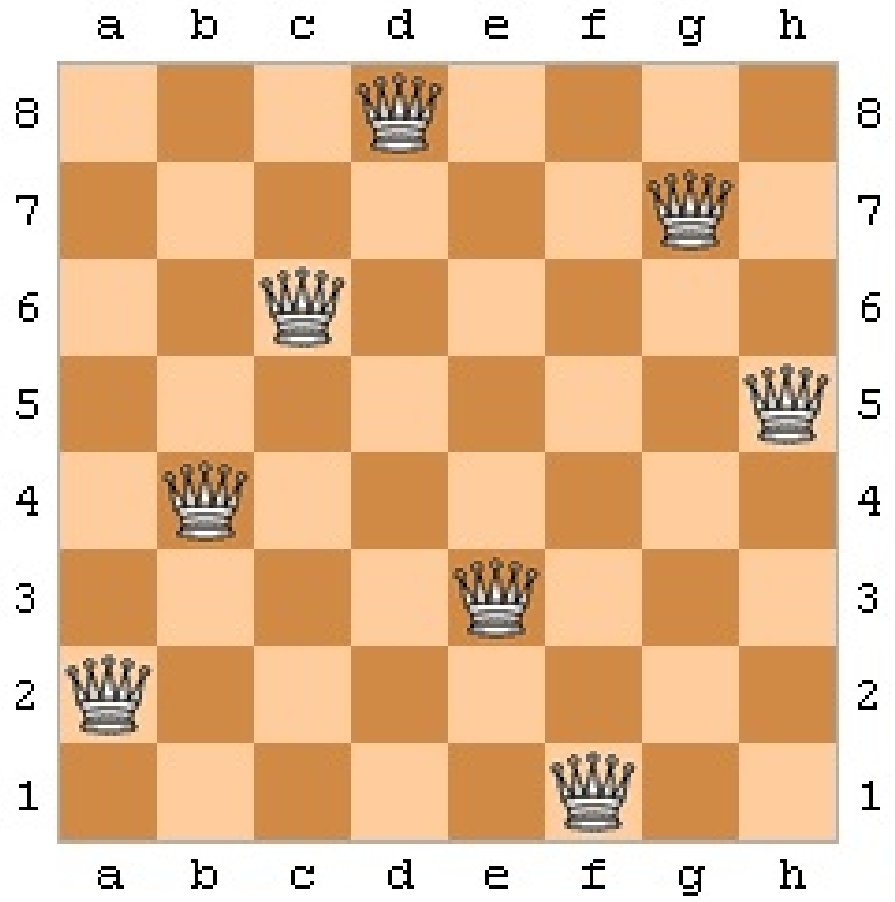
Wie sieht ein Exact Cover Problem aus?

- Beispiel: Ein Schulbus hat in einer Reihe vier Sitzplätze.
- Es können nie zwei Schüler auf einem Platz sitzen, ebensowenig soll einer leer bleiben.
- Es gibt gewisse Gruppen von Schülern, die unbedingt, auf keinen Fall nebeneinander sitzen wollen.

	1	2	3	4
A	0	0	1	1
B	1	1	0	0
C	1	1	1	0
D	1	0	0	1
E	0	1	0	0
F	1	0	1	0
G	0	0	1	0

Beispiele für Exact Cover Probleme

- N-Queens



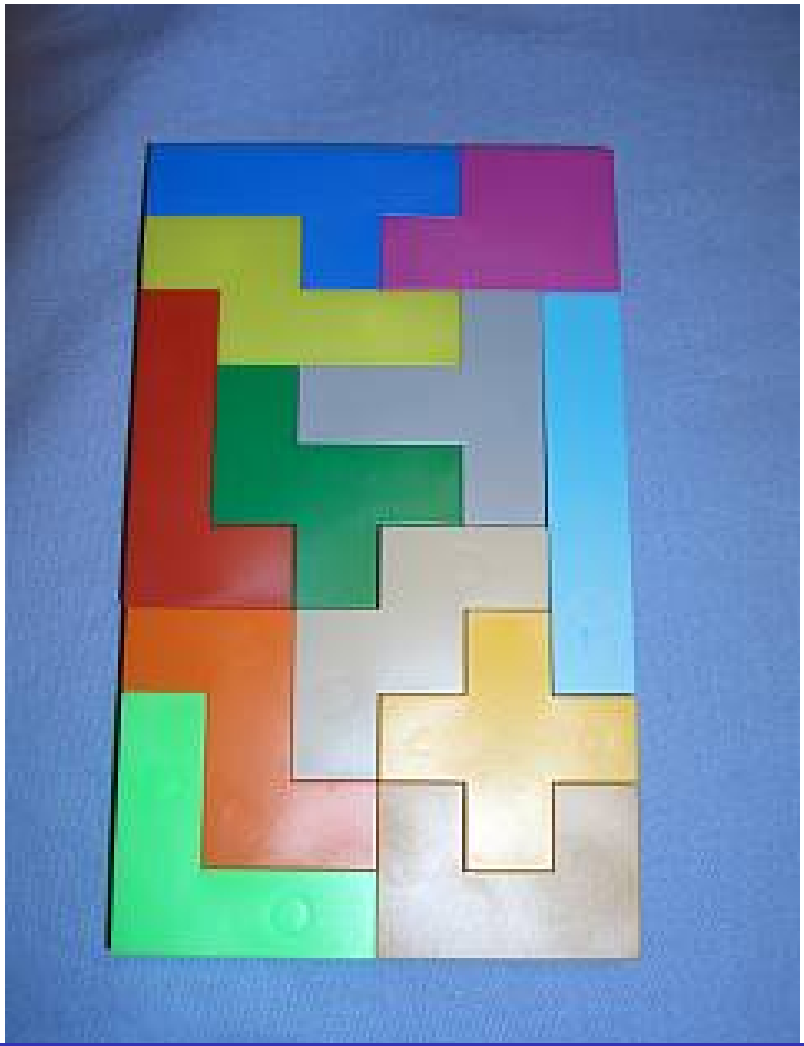
Beispiele für Exact Cover Probleme

- N-Queens
- Sudoku

3			2	4			6	
	4						5	3
1	8	9	6	3	5	4		
				8		2		
		7	4	9	6	8		1
8	9	3	1	5		6		4
		1	9	2		5		
2			3			7	4	
9	6		5			3		2

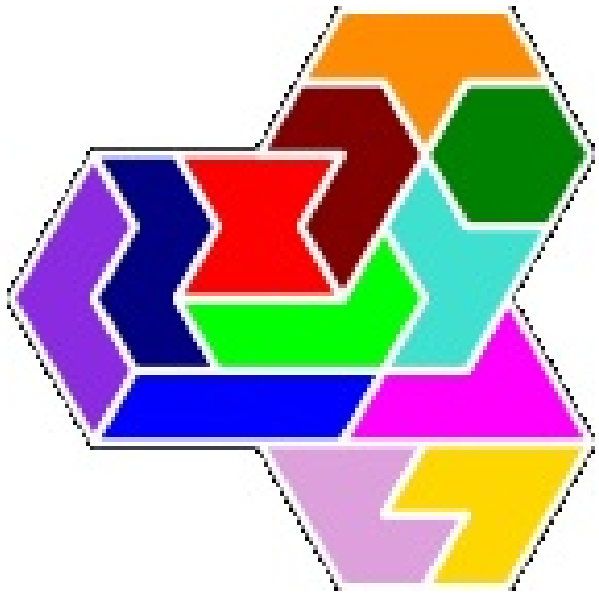
Beispiele für Exact Cover Probleme

- N-Queens
- Sudoku
- Pentomino (bzw. allgemein N-omino)



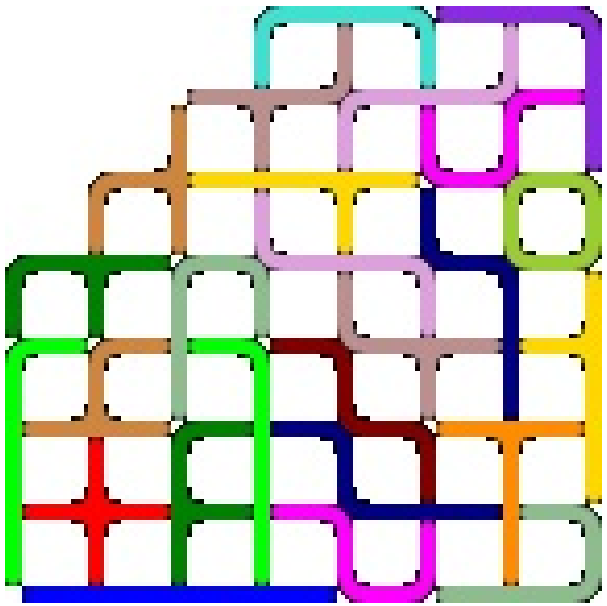
Beispiele für Exact Cover Probleme

- N-Queens
- Sudoku
- Pentomino (bzw. allgemein N-omino)
- Hexiamond (bzw. allgemein N-iamond)



Beispiele für Exact Cover Probleme

- N-Queens
- Sudoku
- Pentomino (bzw. allgemein N-omino)
- Hexiamond (bzw. allgemein N-iamond)
- Tetrastick (bzw. allgmein N-stick)



Beispiele für Exact Cover Probleme

⇒

Probleme dieser Art lassen sich auf eine Matrix wie eben beschrieben abbilden!

	1	2	3	4
A	0	0	1	1
B	1	1	0	0
C	1	1	1	0
D	1	0	0	1
E	0	1	0	0
F	1	0	1	0
G	0	0	1	0

Lösungsverhalten

NP-complete

- Sollte eine Lösung des Problems vorliegen, so kann diese in polinomialer Zeit verifiziert werden (NP).
- Zudem kann der Algorithmus zur Lösung dieses Problems in einen zur Lösung aller NP Proleme übersetzt werden (NP hard).

Nach derzeitigem Stand der Forschung (P$\neq$NP) bedeutet dies für die Praxis, dass das Problem nicht in polinomialer Zeit gelöst werden kann und somit die Rechenzeit exponentiell mit der Systemgröße wächst.

Brute Force Methode

- Probiere jede Zeile ob Lösung

Brute Force Methode

- Probiere jede Zeile ob Lösung
- Probiere jede Kombination aus zwei Zeilen ob Lösung

Brute Force Methode

- Probiere jede Zeile ob Lösung
- Probiere jede Kombination aus zwei Zeilen ob Lösung
- Probiere jede Kombination aus drei Zeilen ob Lösung

Brute Force Methode

- Probiere jede Zeile ob Lösung
- Probiere jede Kombination aus zwei Zeilen ob Lösung
- Probiere jede Kombination aus drei Zeilen ob Lösung
- ...

Brute Force Methode

- Probiere jede Zeile ob Lösung
- Probiere jede Kombination aus zwei Zeilen ob Lösung
- Probiere jede Kombination aus drei Zeilen ob Lösung
- ...
- Probiere jede Kombination aus N Zeilen ob Lösung (N = Anzahl der Zeilen)

Brute Force Methode

- Programmtechnisch ist dies nicht direkt umsetzbar

Brute Force Methode

- Programmtechnisch ist dies nicht direkt umsetzbar
- Selbstgenerierender Code (N ineinander geschachtelte *for* Schleifen)

Brute Force Methode

- Programmtechnisch ist dies nicht direkt umsetzbar
- Selbstgenerierender Code (N ineinander geschachtelte *for* Schleifen)
- Die Rechenzeit wird schon bei $N \approx 10$ enorm

Brute Force Methode

- Programmtechnisch ist dies nicht direkt umsetzbar
- Selbstgenerierender Code (N ineinander geschachtelte *for* Schleifen)
- Die Rechenzeit wird schon bei $N \approx 10$ enorm
- Wir wollen Probleme mit $N \approx 5000$ lösen

Direkte Matrixmanipulation

- Matlab als matrixbasierte Sprache

Direkte Matrixmanipulation

- Matlab als matrixbasierte Sprache
- Zeilen/Spalten direkt aus Matrix löschen

Direkte Matrixmanipulation

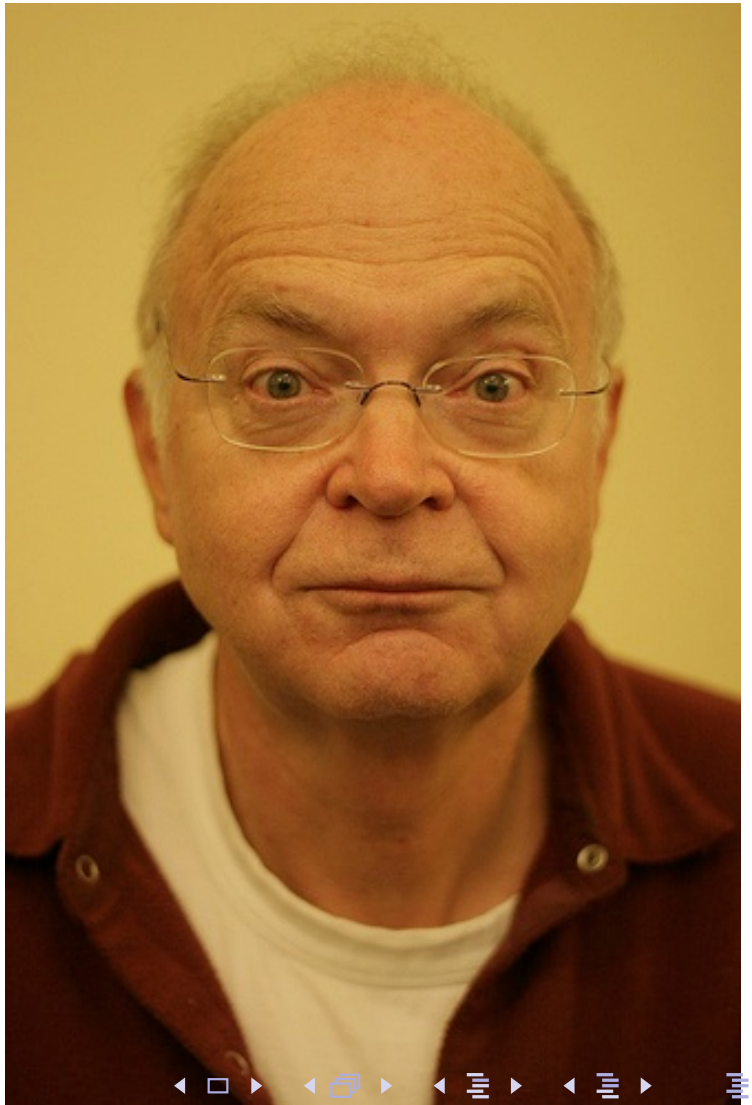
- Matlab als matrixbasierte Sprache
- Zeilen/Spalten direkt aus Matrix löschen
- Code einfach, jedoch ständige Reskalierung des Speichers langsam

Direkte Matrixmanipulation

- Matlab als matrixbasierte Sprache
- Zeilen/Spalten direkt aus Matrix löschen
- Code einfach, jedoch ständige Reskalierung des Speichers langsam
- Funktioniert gut, jedoch benötigt man zur Lösung eines *echten* Problems $O(Stunden)$

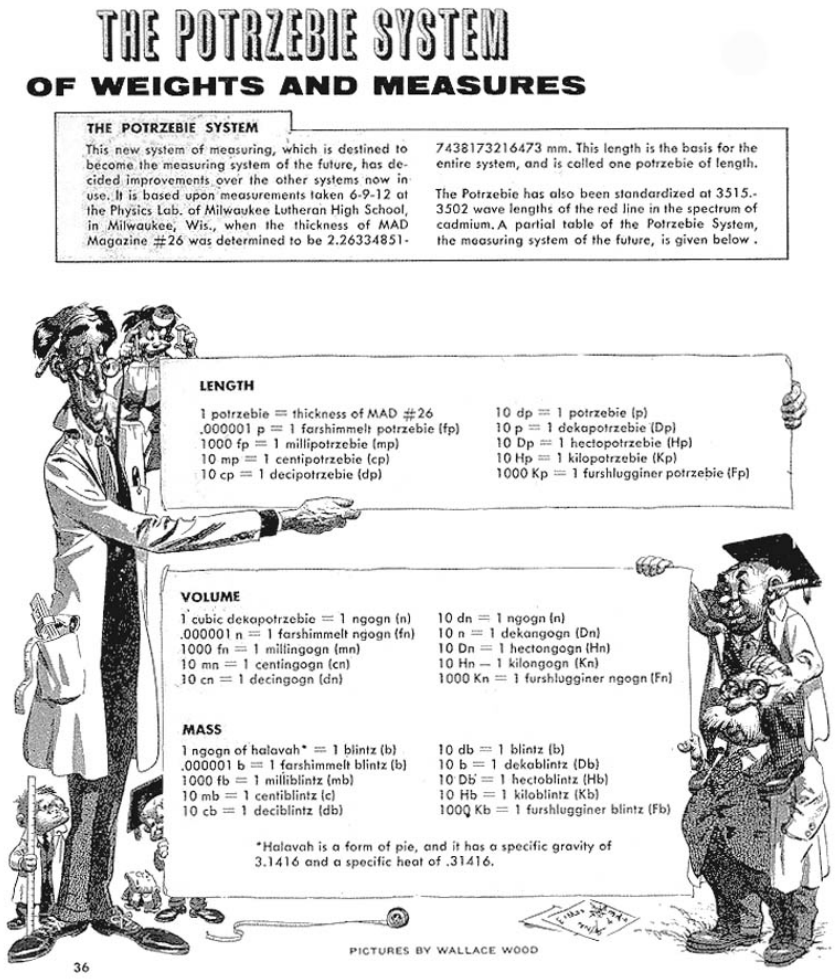
Donald Knuth

- Donald Knuth, Prof. Em. from Stanford



Donald Knuth

- Donald Knuth, Prof. Em. from Stanford
- Potrzebie System



THE POTRZEBIE SYSTEM

OF WEIGHTS AND MEASURES

THE POTRZEBIE SYSTEM

This new system of measuring, which is destined to become the measuring system of the future, has decided improvements over the other systems now in use. It is based upon measurements taken 6-9-12 at the Physics Lab. of Milwaukee Lutheran High School, in Milwaukee, Wis., when the thickness of MAD Magazine #26 was determined to be 2.26334851-7438173216473 mm. This length is the basis for the entire system, and is called one potrzebie of length.

The Potrzebie has also been standardized at 3515.-3502 wave lengths of the red line in the spectrum of cadmium. A partial table of the Potrzebie System, the measuring system of the future, is given below.

LENGTH

1 potrzebie = thickness of MAD #26
.000001 p = 1 farshimmelt potrzebie (fp)
1000 fp = 1 millipotrzebie (mp)
10 mp = 1 centipotrzebie (cp)
10 cp = 1 decipotrzebie (dp)
10 dp = 1 potrzebie (p)
10 p = 1 dekapotrzebie (Dp)
10 Dp = 1 hectopotrzebie (Hp)
10 Hp = 1 kilopotrzebie (Kp)
1000 Kp = 1 furshlugginer potrzebie (Fp)

VOLUME

1 cubic dekapotrzebie = 1 ngogn (n)
.000001 n = 1 farshimmelt ngogn (fn)
1000 fn = 1 millingogn (mn)
10 mn = 1 centingogn (cn)
10 cn = 1 decingogn (dn)
10 dn = 1 ngogn (n)
10 n = 1 dekangogn (Dn)
10 Dn = 1 hectongogn (Hn)
10 Hn = 1 kilongogn (Kn)
1000 Kn = 1 furshlugginer ngogn (Fn)

MASS

1 ngogn of halavah* = 1 blintz (b)
.000001 b = 1 farshimmelt blintz (b)
1000 fb = 1 milliblintz (mb)
10 mb = 1 centiblintz (c)
10 cb = 1 deciblintz (db)
10 db = 1 blintz (b)
10 b = 1 dekablintz (Db)
10 Db = 1 hectoblintz (Hb)
10 Hb = 1 kiloblintz (Kb)
1000 Kb = 1 furshlugginer blintz (Fb)

*Halavah is a form of pie, and it has a specific gravity of 3.1416 and a specific heat of .31416.

PICTURES BY WALLACE WOOD

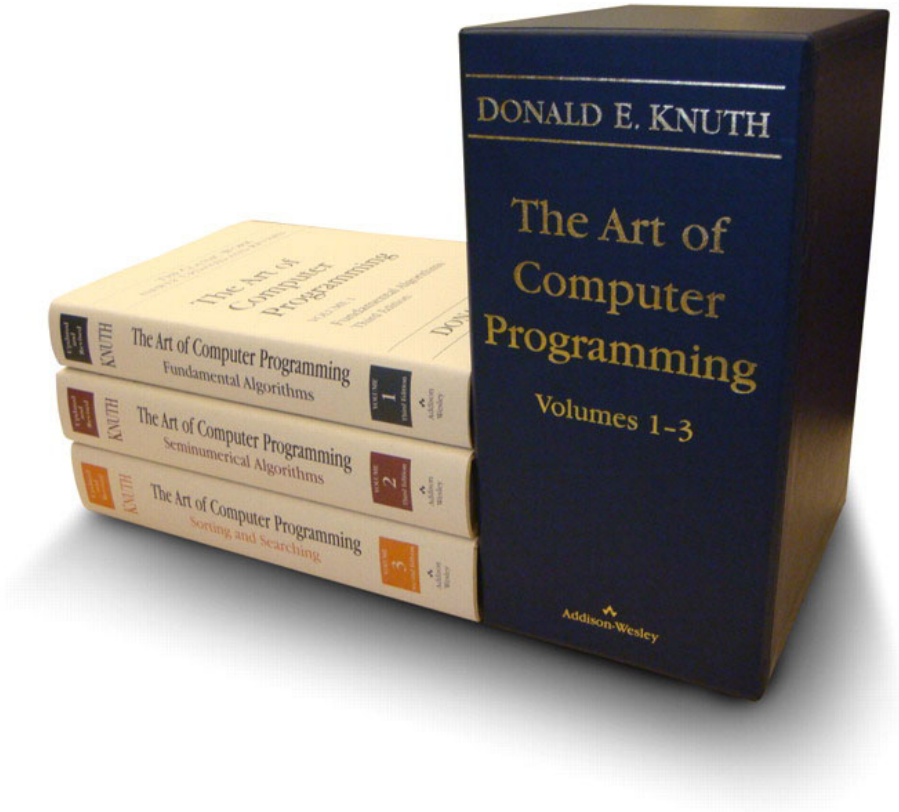
36

Donald Knuth

- Donald Knuth, Prof. Em. from Stanford
- Potrzebie System
- TEX (he made this presentation possible)

Donald Knuth

- Donald Knuth, Prof. Em. from Stanford
- Potrzebie System
- TEX (he made this presentation possible)
- The Art of Computer Programming

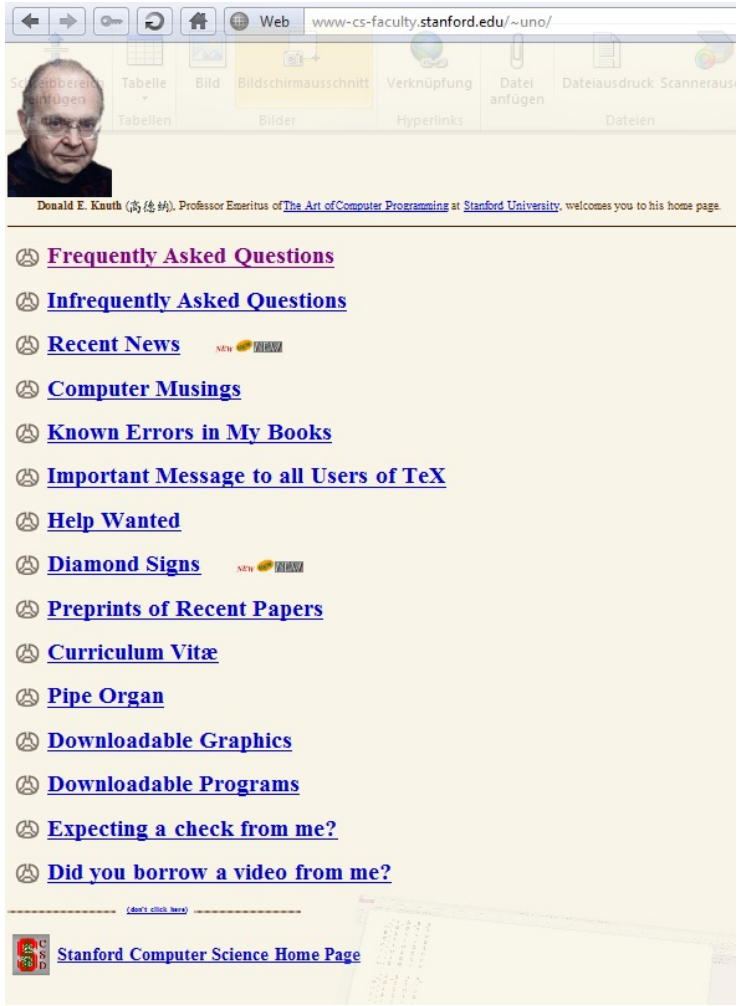


Donald Knuth

- Donald Knuth, Prof. Em. from Stanford
- Potrzebie System
- TEX (he made this presentation possible)
- The Art of Computer Programming
- Lots of Prices: John von Neumann Medal, Turing Award, National Medal of Science,...

Donald Knuth

- Donald Knuth, Prof. Em. from Stanford
- Potrzebie System
- TEX (he made this presentation possible)
- The Art of Computer Programming
- Lots of Prices: John von Neumann Medal, Turing Award, National Medal of Science,...
- Look at his homepage (http://www-cs-faculty.stanford.edu/ uno/)



Algorithm X

Dancing Links[1]

Die hinter Algorithm X stehende Idee basiert auf dem Löschen von Elementen aus einer mehrfach verketteten Liste

$$L[R[x]] \leftarrow L[x], \qquad R[L[x]] \leftarrow R[x]$$

Das wiedereinhängen in die Liste

$$L[R[x]] \leftarrow x, \qquad R[L[x]] \leftarrow x$$

ist dabei ein zentraler Punkt des Verfahrens.

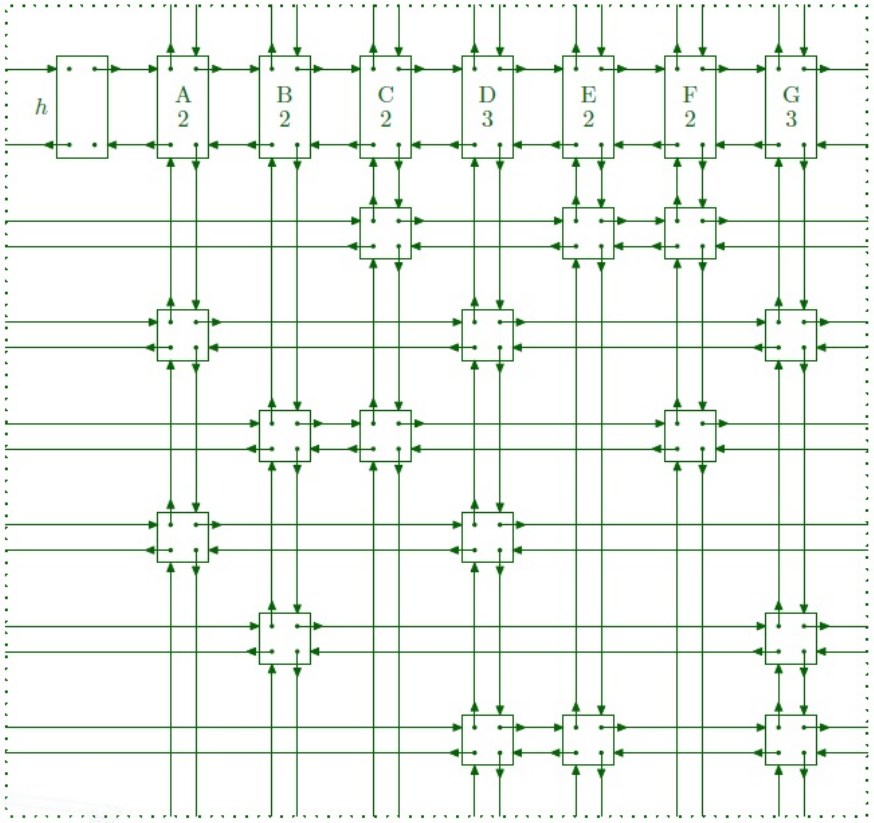
[1] Knuth E. Donald, Dancing Links, 2000, http://arxiv.org/abs/cs/0011047 or http://www-cs-faculty.stanford.edu/ uno/preprints.html

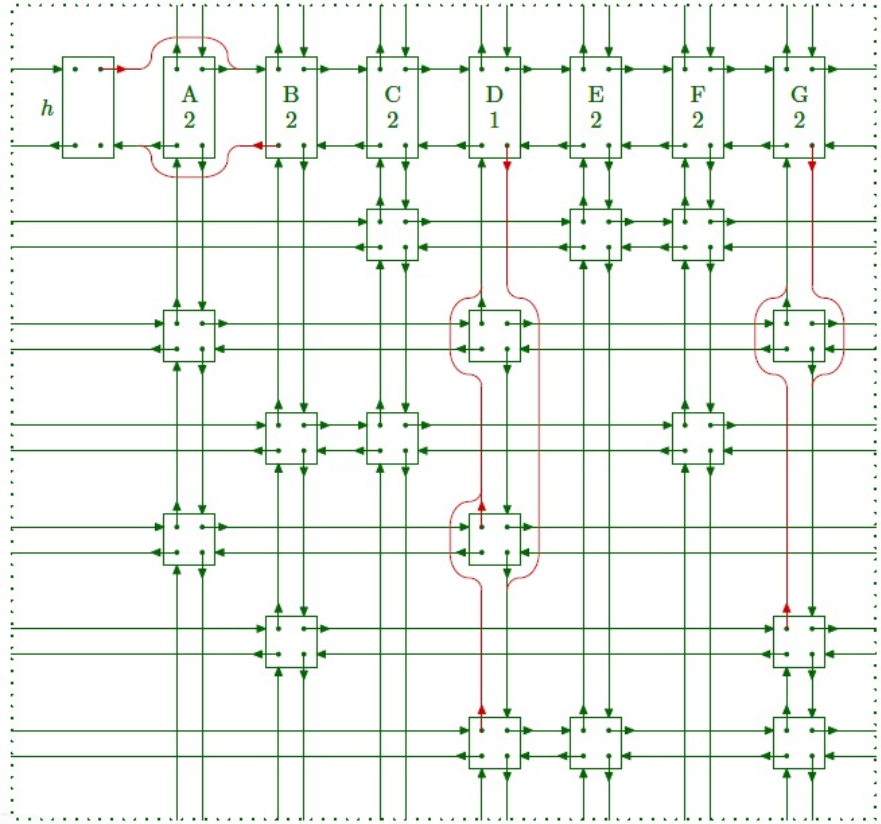
Algorithm X

Der Algorithmus besteht dabei aus drei wesentlichen Teilen:

- Der rekursiven Routine *search()*, die auf eine Matrix angewandt, diese reduziert.
- Der Funktion *cover()*, welche eine Spalte der Matrix, abdeckt.
- Der Funktion *uncover()*, welche die Pointeroperationen von cover() in exakt rückwärtiger Reihenfolge rückgängig macht.

Algorithm X





Algorithm X search() Pseudocode

```
 1 if(h.getRight() == h){printSolution(); return;}
 2 else {
 3    ColumnNode column = chooseNextColumn();
 4    cover(column);
 5    for(node row = column.getDown() ;
      rowNode!=column;rowNod =rowNode.getDown()){
 6       solutions.add(rowNode);
 7       for(Node rightNode = row.getRight();
         otherNode!=row;rightNode=rightNode.getRight())
 8       { cover( rightNode ); }
 9       Search(k+1);
10       solutions.remove(rowNode);
11       column = rowNode.getColumn();
12       for(Node leftNode = rowNode.getLeft();
         leftNode!=row;leftNode=leftNode.getLeft())
13       { uncover( leftNode ); }
14    }
15    uncover( column );          }
```

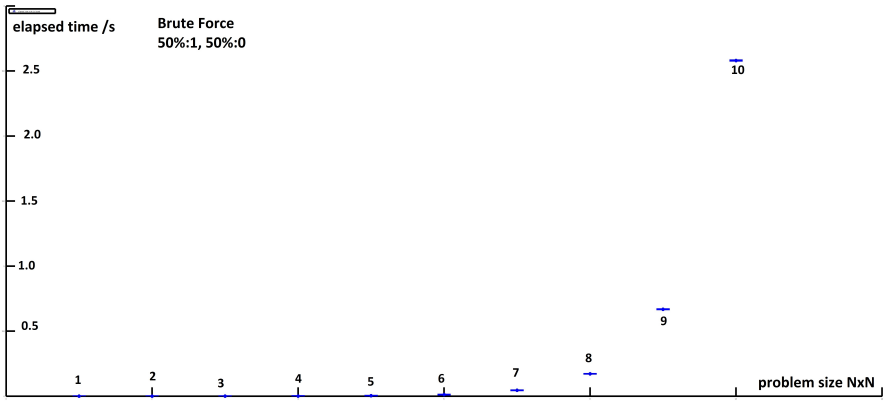
Algorithm X cover() Pseudocode

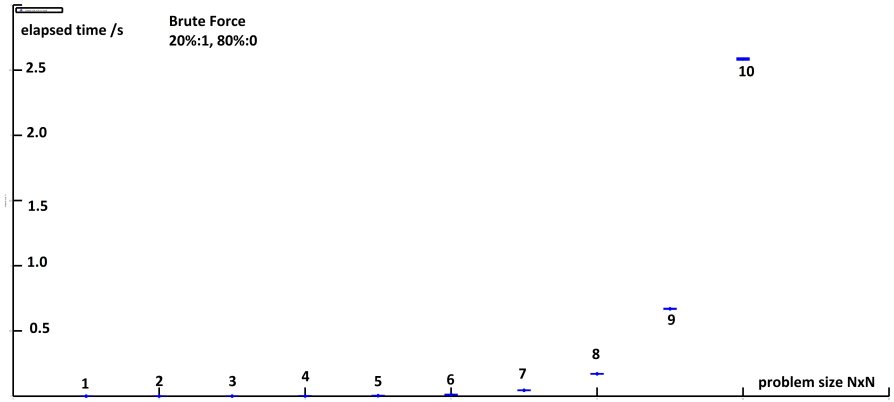
```
1 column.getRight().setLeft( column.getLeft() );
2 column.getLeft().setRight( column.getRight() );
3 for Noderow=column.getDown();
  row!=column;roe=row.getDown())
4   for(Node rightNode=row.getRight() ;
    rightNode!=row;rightNode=rightNode.getRight()){
5     rightNode.getUp().setDown(rightNode.getDown());
6     rightNode.getDown().setUp(rightNode.getUp());
7   }
8 }
```

Algorithm X uncover() Pseudocode

```
1 Node column = dataNode.getColumn();
2 for(Node row=column.getUp();
  row!=column;row=row.getUp())
3   for(Node leftNode=row.getleft();
    leftNode!=ro;leftNode=leftNode.getRight()) {
4     leftNode.getUp().setDown( leftNode.getDown() );
5     leftNode.getDown().setUp( leftNode.getUp() );
6   }
7   column.getRight().setLeft(column.getLeft());
8   column.getLeft().setRight(column.getRight());
9 }
```

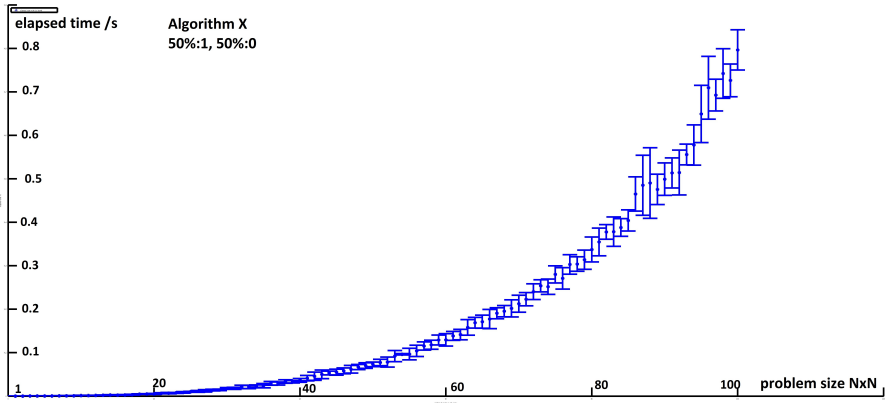
Brute Force Performance

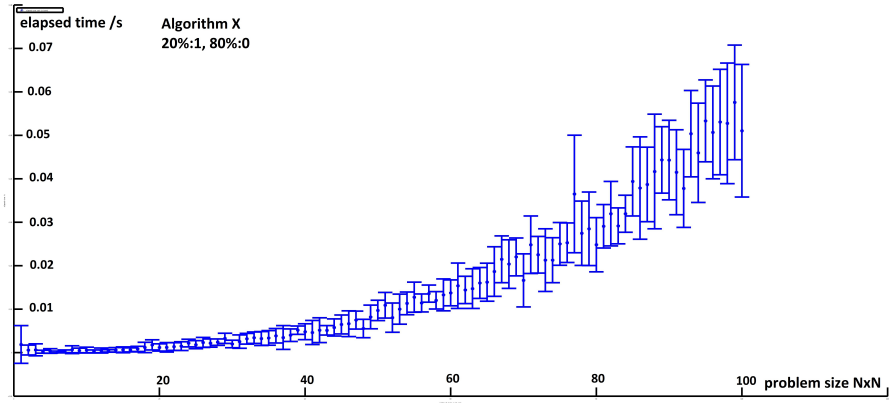




- Test mit NxN Zufallsmatrizen, mit verschiedenem Besetzungsgrad: 50% und 20%
- Der einzige Vorteil von Brute Force: es ist unabhängig von allen Systemeigenschaften (außer der Systemgröße).
- Das 10x10 Problem dauert $\approx$ 2.5 Sekunden.

Algorithm X Performance

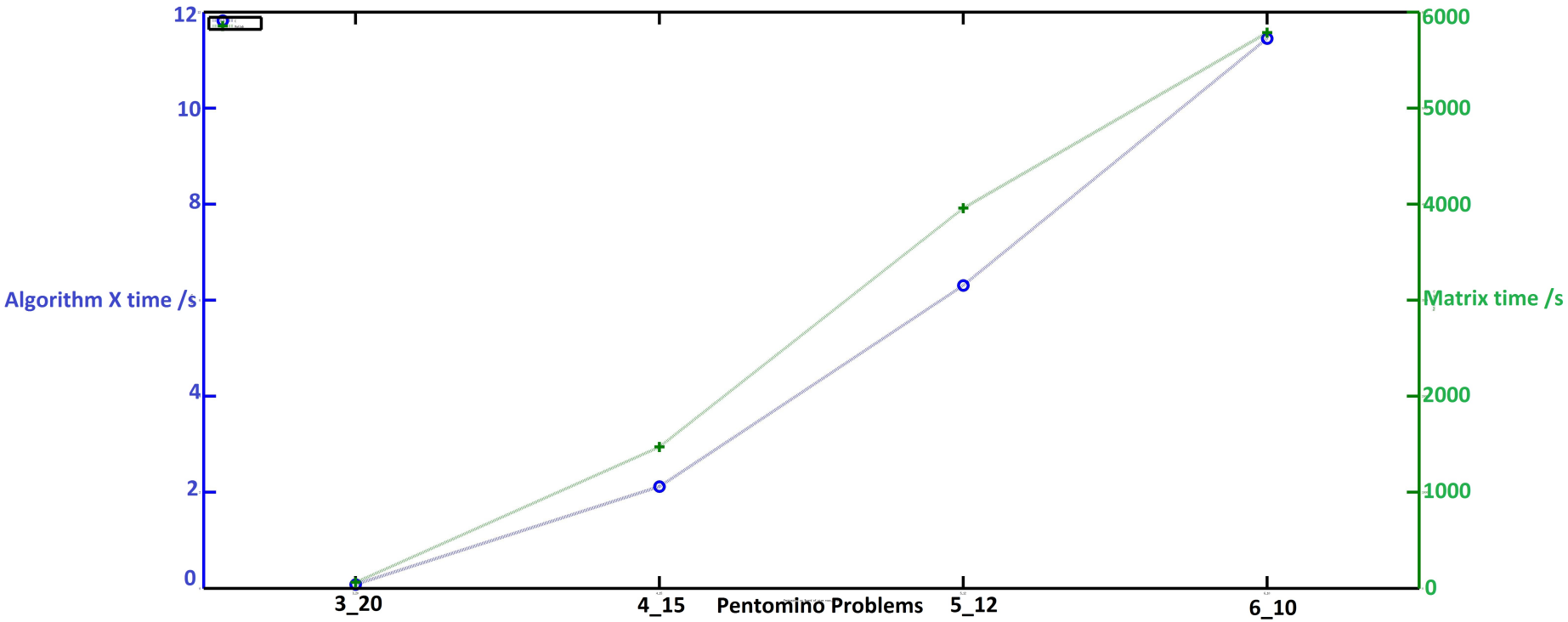




- Test mit NxN Zufallsmatrizen, mit verschiedenem Besetzungsgrad: 50% und 20%
- Algorithm X arbeitet sehr schnell, ist jedoch von der genauen Systemstruktur abhängig.
- Das 100x100 Problem dauert ≈ 0.08 Sekunden bzw. ≈ 0.8 Sekunden.

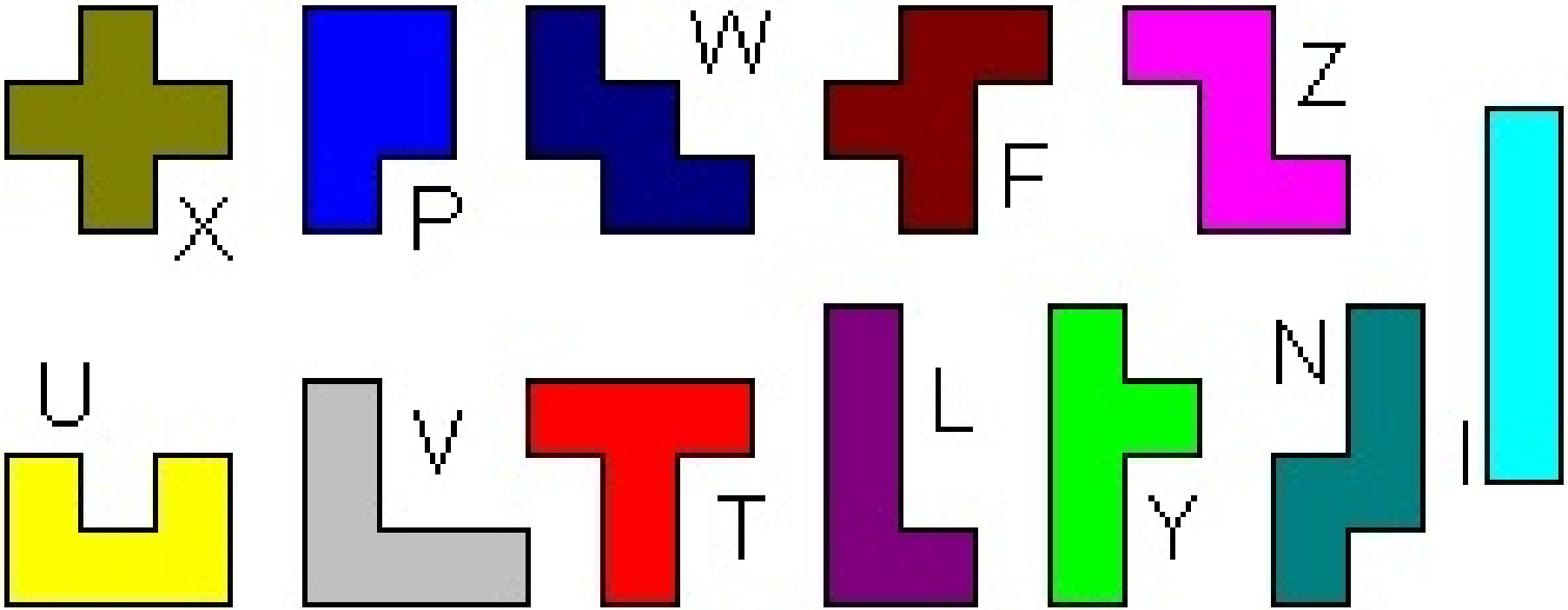
- Genaues Laufzeitverhalten nicht vorhersehbar!
- Die Laufzeit steigt exponentiell mit der Systemgröße.
- Die Laufzeit steigt im inversen Besetzungrad.
- Genau ist der Zusammenhang zwischen Laufzeit und Besetzungsgrad nicht quanitfizierbar, da er sehr stark von der genauen Struktur der Matrix abhängt. Dieser Zusammenhang ist außerdem von der Strategie, wie Zeilen und Spalten im Algorithm X ausgesucht werden abhängig.

Dancing Links vs Direkte Matrixmanipulation



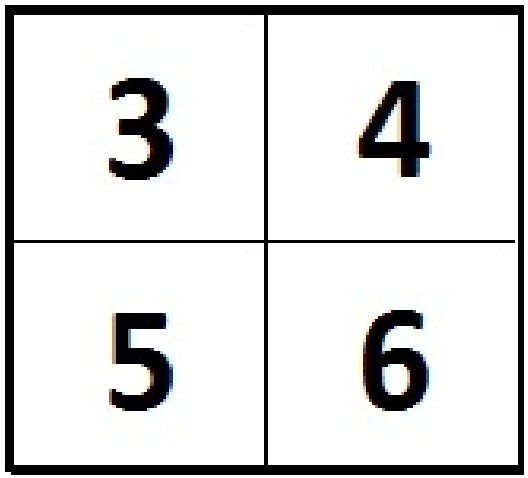
The Pentomino Puzzle

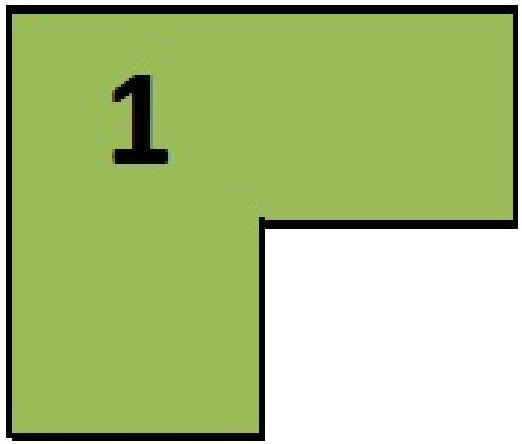
- Ordne 12, aus je 5 Einheiten bestehende *Tetrissteine* auf einem Spielfeld mit 60 Plätzen an



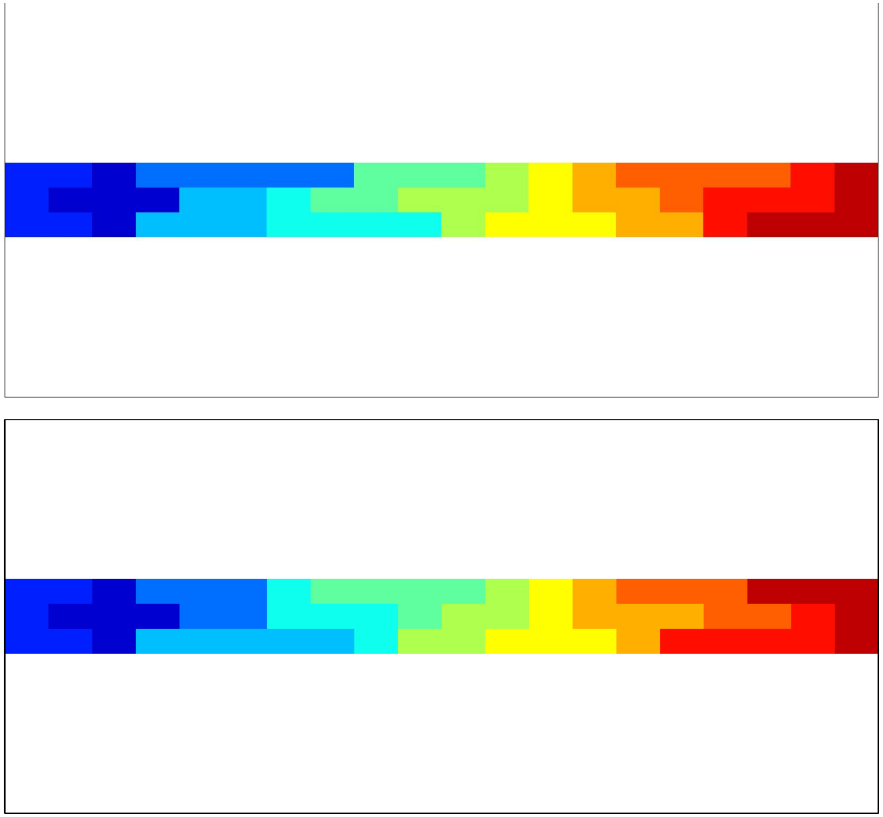
The Pentomino Puzzle - Umsetzung

- Beispiel zur Umsetzung von Pentomino in die logische Matrix
- Verschieben und drehen $\Rightarrow$ alle unterschiedlichen Konfigurationen



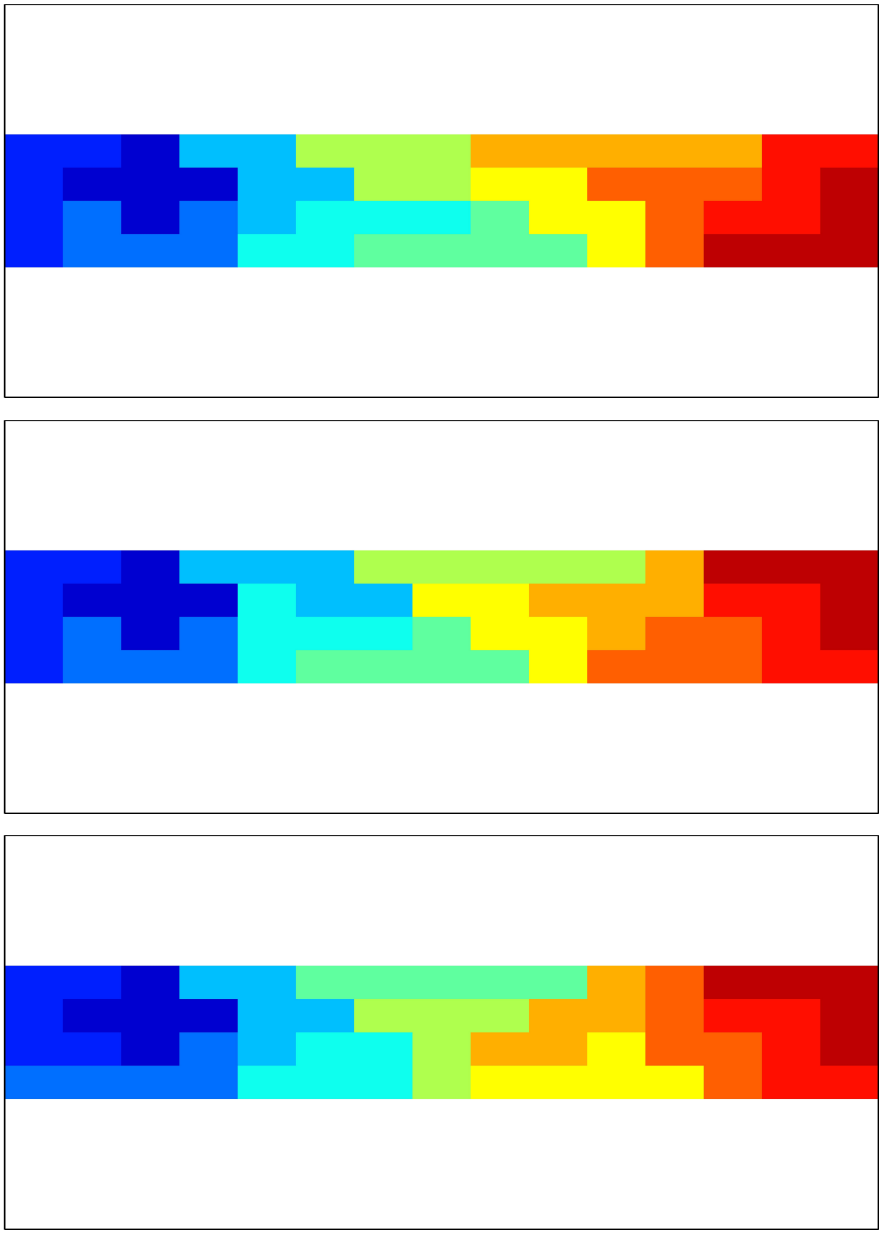


3x20 Puzzle



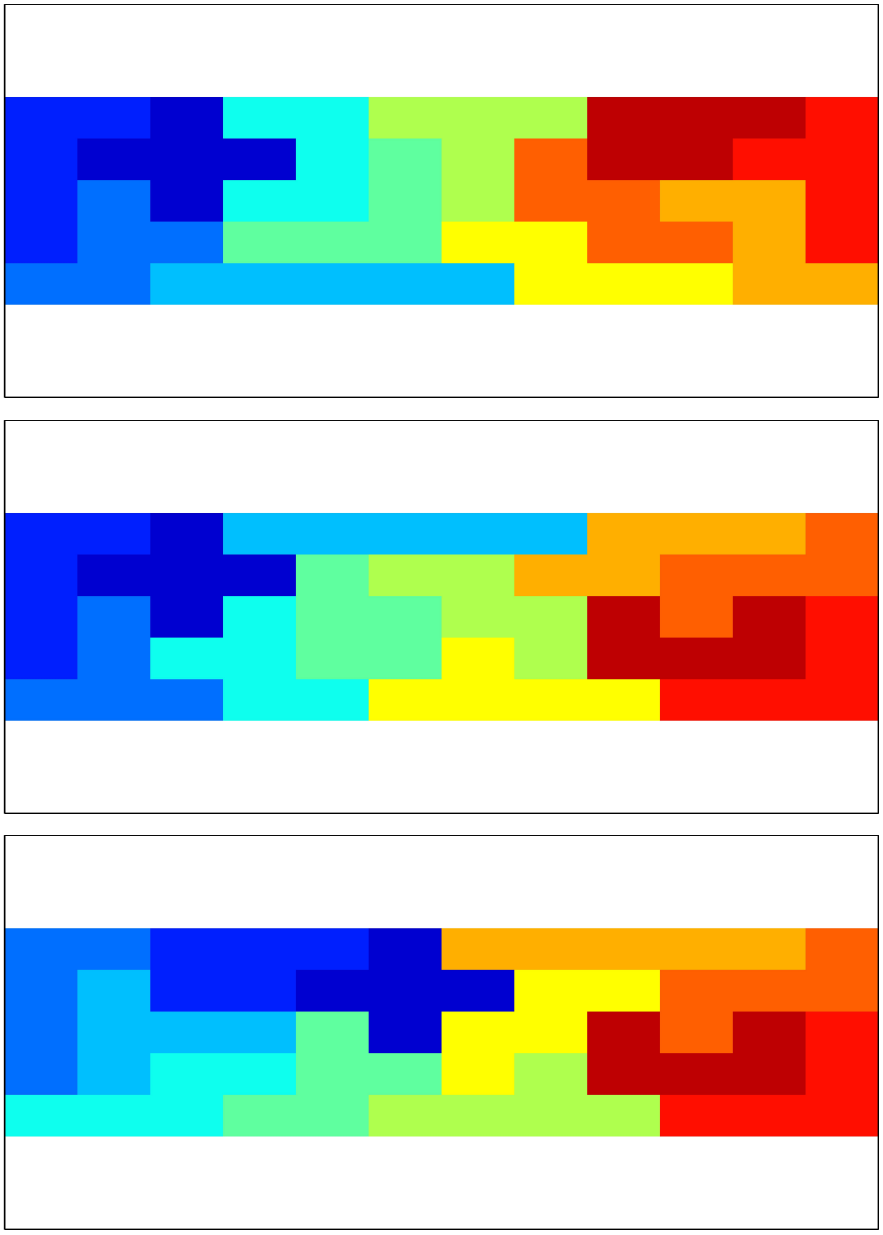
- Matrixgröße: 1236x72
- 2 Lösungen existieren
- Dancing Links benötigt 0.1s um alle zu finden
- Direkte Matrixmanipulation benötigt 1min um alle zu finden

4x15 Puzzle



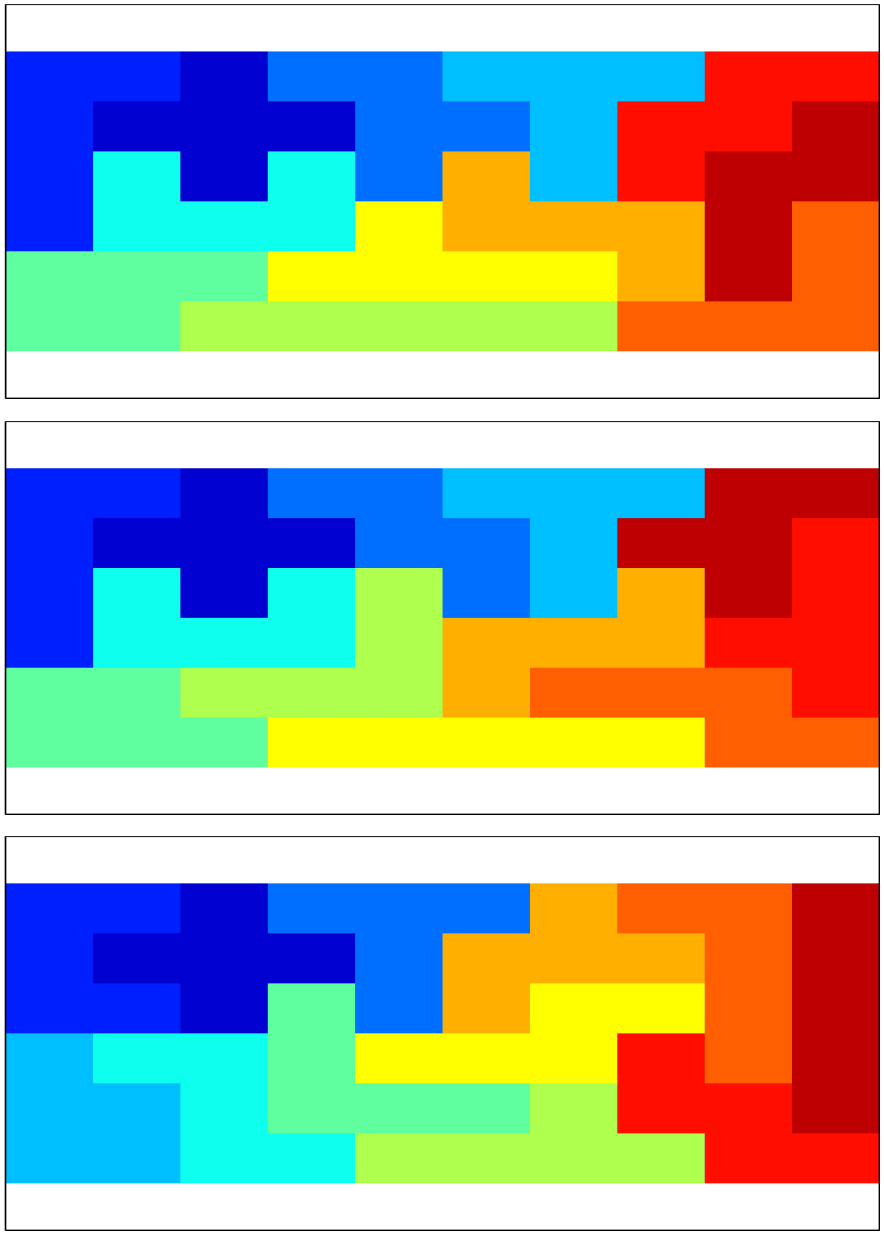
- Matrixgröße: 1696x72
- 368 Lösungen existieren
- Dancing Links benötigt 2s um alle zu finden
- Direkte Matrixmanipulation benötigt 24min 30s um alle zu finden

5x12 Puzzle



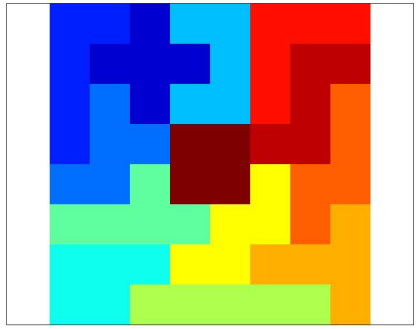
- Matrixgröße: 1936x72
- 1010 Lösungen existieren
- Dancing Links benötigt 6s um alle zu finden
- Direkte Matrixmanipulation benötigt 1h 6min um alle zu finden

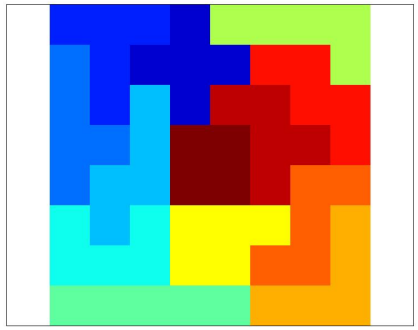
6x10 Puzzle

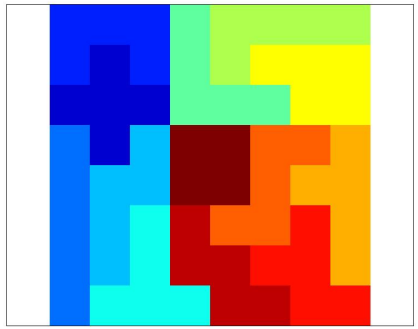


- Matrixgröße: 2056x72
- 2339 Lösungen existieren
- Dancing Links benötigt 11s um alle zu finden
- Direkte Matrixmanipulation benötigt 1h 37min um alle zu finden

8x8 mit 2x2 Loch Puzzle







- Matrixgröße: $1568x72$
- 130 Lösungen existieren
- Dancing Links benötigt 0.8s um alle zu finden
- Direkte Matrixmanipulation benötigt 8min 30s um alle zu finden

Literatur

Vielen Dank für Ihre Aufmerksamkeit!

- Exact Cover Problems: Wikipedia
- Dancing Links: Knuth E. Donald, Dancing Links, 2000, http://arxiv.org/abs/cs/0011047 or http://www-cs-faculty.stanford.edu/ uno/preprints.html