Solving Sudoku with Dancing Links

Rob Beezer
beezer@pugetsound.edu

Department of Mathematics and Computer Science
University of Puget Sound
Tacoma, Washington USA

African Institute for Mathematical Sciences
October 25, 2010

Available at http://buzzard.pugetsound.edu/talks.html
Example: Combinatorial Enumeration

Create all permutations of the set \( \{0, 1, 2, 3\} \)

- Simple example to demonstrate key ideas
- Creation, cardinality, existence?
- There are more efficient methods for this example
Brute Force Backtracking

BLACK = Forward
BLUE = Solution
RED = Backtrack

<table>
<thead>
<tr>
<th>root</th>
<th>0 1 2 0 1 3 3 0 2 1 0 2 3 3 0 3 1 0 3</th>
<th>1 0 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1 2 2 0 1 3 0 2 1 0 2 3 0 3 1 0 3</td>
<td>1 0 2 1</td>
</tr>
<tr>
<td>0 0</td>
<td>0 1 2 0 1 0 2 0 2 2 0 3 0 3 2 root 1 0 2 3</td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 1 0 2 0 0 2 2 0 3 0 3 2 1 0 1 0 2</td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1 3 0 2 1 0 2 3 0 3 1 0 3 2 1 1 0 0</td>
<td>:</td>
</tr>
<tr>
<td>0 1 2</td>
<td>0 1 3 0 2 1 0 2 3 0 3 1 0 3 2 1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>0 1 2 0</td>
<td>red 0 1 3 0 2 1 0 2 3 0 3 1 0 3 2 1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>0 1 2 0</td>
<td>0 1 3 0 2 1 0 2 3 0 3 1 0 3 2 1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>0 1 2 0</td>
<td>0 1 3 0 2 1 0 2 3 0 3 1 0 3 2 1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>0 1 2 0</td>
<td>0 1 3 0 2 1 0 2 3 0 3 1 0 3 2 1 1 0 1</td>
<td>:</td>
</tr>
<tr>
<td>0 1 2 1</td>
<td>0 1 3 0 2 1 0 2 3 0 3 1 0 3 2 1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>0 1 2 0</td>
<td>0 1 3 0 2 1 0 2 3 0 3 1 0 3 2 1 1 0 1</td>
<td>:</td>
</tr>
</tbody>
</table>
A Better Idea

- Avoid the really silly situations, such as: 1 0 1
- “Remember” that a symbol has been used already
- Additional data structure: track “available” symbols
- Critical: must maintain this extra data properly
- (Note recursive nature of backtracking)
Sophisticated Backtracking

<table>
<thead>
<tr>
<th>BLACK = Forward</th>
<th>BLUE = Solution</th>
<th>RED = Backtrack</th>
</tr>
</thead>
<tbody>
<tr>
<td>root {0,1,2,3}</td>
<td>{0 2 1 3 }</td>
<td>{0 3 2 1 }</td>
</tr>
<tr>
<td>0 {1,2,3}</td>
<td>{0 2 1 } 3</td>
<td>{0 3 2 } 1</td>
</tr>
<tr>
<td>0 1 {2,3}</td>
<td>{0 2 } 1,3</td>
<td>{0 3 } 1,2</td>
</tr>
<tr>
<td>0 1 2 {3}</td>
<td>{0 2 3 } 1</td>
<td>{0 1,2,3}</td>
</tr>
<tr>
<td>0 1 2 3 {}</td>
<td>{0 2 3 } 1</td>
<td>{0 1,2,3}</td>
</tr>
<tr>
<td>0 1 2 {3}</td>
<td>{0 2 3 } 1</td>
<td>{0 1,2,3}</td>
</tr>
<tr>
<td>0 1 {2,3}</td>
<td>{0 2 } 1,3</td>
<td>{0 1,2,3}</td>
</tr>
<tr>
<td>0 1 3 {2}</td>
<td>{0 1,2,3}</td>
<td>{0 1,2,3}</td>
</tr>
<tr>
<td>0 1 3 2 {}</td>
<td>{0 3 } 1,2</td>
<td>{0 1,2,3}</td>
</tr>
<tr>
<td>0 1 3 {2}</td>
<td>{0 3 1 } 2</td>
<td>{0 1,2,3}</td>
</tr>
<tr>
<td>0 1 {2,3}</td>
<td>{0 3 1 } 2</td>
<td>{0 1,2,3}</td>
</tr>
<tr>
<td>0 {1,2,3}</td>
<td>{0 3 1 } 2</td>
<td>{0 1,2,3}</td>
</tr>
<tr>
<td>0 2 {1,3}</td>
<td>{0 3 1 } 2</td>
<td>{0 1,2,3}</td>
</tr>
<tr>
<td>0 2 1 {3}</td>
<td>{0 3 2 } 1</td>
<td>{0 1,2,3}</td>
</tr>
</tbody>
</table>

root \{0,1,2,3\} | root \{0,1,2,3\} | root \{0,1,2,3\} | root \{0,1,2,3\} | root \{0,1,2,3\} | root \{0,1,2,3\} | root \{0,1,2,3\} | root \{0,1,2,3\} | root \{0,1,2,3\} |
| \{0 3 2 1 \}  | \{1 2 3 0 \}  | \{1 3 0 2 \}  | \{1 3 2 0 \}  | \{1 3 0 2 \}  | \{1 3 0 2 \}  | \{1 3 0 2 \}  | \{1 3 0 2 \}  | \{1 3 0 2 \}  |
Depth-First Search Tree

0
  /  \
 01 02
   /  \
 012 021
   /  \
 0123 0213
  
03
  /  \
 031 032
   /  \
 0312 0321
Algorithm

n=4
available=[True]*n  # [True, True, True, True]
perm=[0]*n          # [0, 0, 0, 0]

def bt(level):
    for x in range(n):
        if available[x]:
            available[x]=False
            perm[level]=x
            if level+1 == n:
                print perm
            bt(level+1)
            available[x]=True

bt(0)
Sudoku Basics

- $n^2$ symbols
- $n^2 \times n^2$ grid
- $n^2$ subgrids ("boxes") each $n \times n$
- Classic Sudoku is $n = 3$
- Each symbol once and only once in each row
- Each symbol once and only once in each column
- Each symbol once and only once in each box
- The grid begins partially completed
- A Sudoku puzzle should have a unique completion
### Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>8</th>
<th></th>
<th>4</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td></td>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>5</td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

$\Rightarrow$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>5</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4</td>
<td>9</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>8</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>2</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>1</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

$\Rightarrow$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>5</th>
<th>1</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>1</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>
Sudoku via Backtracking

- Fill in first row, left to right, then second row, ... 
- For each blank cell, maintain possible new entries 
- As entries are attempted, update possibilities 
- If a cell has just one possibility, it is forced 
- Lots to keep track of, especially at backtrack step
Sudoku via Backtracking

- Fill in first row, left to right, then second row, . . .
- For each blank cell, maintain possible new entries
- As entries are attempted, update possibilities
- If a cell has just one possibility, it is forced
- Lots to keep track of, especially at backtrack step

Alternate Title: “Why I Don’t Do Sudoku”
Top row, second column: possibilities?

\[
\begin{array}{ccc}
5 & 6 & 7 \\
8 & 3 & 1 \\
4 & 9 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
5 & 6 & 7 \\
1 & 8 & 2 \\
4 & 9 & 3 \\
\end{array}
\]

\[
\{1, 2, 4, 7, 8\} \cap \{1, 2, 3, 6, 7\} = \{1, 2, 7\}
\]
Suppose we try 2 first.
Seventh row, second column: possibilities?

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

{2, 3, 6, 8, 9}

{1, 4, 7, 8} \rightarrow \{1, 4, 7, 8\} \cap \{2, 3, 6, 8, 9\} = \{8\}

One choice!
This may lead to other singletons in the affected row or column.
Exact Cover Problem

- Given: matrix of 0’s and 1’s
- Find: subset of rows
- Condition: rows sum to exactly the all-1’s vector
- Amenable to backtracking (on columns, not rows!)
- Example: (Knuth)

```
0 0 1 0 1 1 0
1 0 0 1 0 0 1
0 1 1 0 0 1 0
1 0 0 1 0 0 0
0 1 0 0 0 0 1
0 0 0 1 1 0 1
```
Solution

Select rows 1, 4 and 5:

\[
\begin{bmatrix}
0 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
\]
Sudoku as an Exact Cover Problem

- Matrix rows are per symbol, per grid location \((n^2 \times (n^2 \times n^2) = n^6)\)
- Matrix columns are conditions: \((3n^4\) total)
  - Per symbol, per grid row: symbol in row \((n^2 \times n^2)\)
  - Per symbol, per grid column: symbol in column \((n^2 \times n^2)\)
  - Per symbol, per grid box: symbol in box \((n^2 \times n^2)\)

Place a 1 in entry of the matrix if and only if matrix row describes symbol placement satisfying matrix column condition

Example:
Consider matrix row that places a 7 in grid at row 4, column 9
  - 1 in matrix column for “7 in grid row 4”
  - 1 in matrix column for “7 in grid column 9”
  - 1 in matrix column for “7 in grid box 6”
  - 0 elsewhere
Sudoku as an Exact Cover Problem

- Puzzle is “pre-selected” matrix rows
- Can delete these matrix rows, and their “covered matrix columns”
- $n = 3$: 729 matrix rows, 243 matrix columns
- Previous example: Remove 26 rows, remove $3 \times 26 = 78$ columns
- Select $81 - 26 = 55$ rows, from 703, for exact cover (uniquely)
- Selected rows describe placement of symbols into locations for Sudoku solution
Dancing Links

- Manage lists with frequent deletions and restorations
- Perfect for descending, backtracking in a search tree
  - “pointers of each already-used element are still active while... removed”
  - Two pages, N queens problem
  - Donald Knuth listed in the Acknowledgement
- Popularized by Knuth, “Dancing Links” (2000, arXiv)
  - Algorithm X = “traditional” backtracking
  - Algorithm DLX = Dancing Links + Algorithm X
  - 26 pages, applications to packing pentominoes in a square
Doubly-Linked List
Remove Node “C” From List

\[ R[x] \]
Remove Node “C” From List

\[ L[R[x]] \]
Remove Node “C” From List

L[R[x]]       L[x]
Remove Node “C” From List

\[ L[R[x]] \leftarrow L[x] \]
Two Assignments to Totally Remove “C”

\[ L[R[x]] \leftarrow L[x] \quad \text{and} \quad R[L[x]] \leftarrow R[x] \]
Two Assignments to Totally Remove “C”

A [ B C D E]

L[R[x]] ← L[x]  R[L[x]] ← R[x]

DO NOT CLEAN UP THE MESS
List Without “C”, Includes Our Mess
Restore Node “C” to the List

R[x]
Restore Node “C” to the List

\[ L[R[x]] \]
Restore Node “C” to the List

\[ \text{L}[\text{R}][x]] \quad x \]
Restore Node “C” to the List
Restore Node “C” to the List

L[R[x]] ← x
R[L[x]] ← x
Restore Node “C” to the List

L[R[x]] ← x
R[L[x]] ← x

WE NEED OUR MESS, IT CLEANS UP ITSELF
DLX for the Exact Cover Problem

- Backtrack on the columns
- Choose a column to cover, this will dictate a selection of rows
DLX for the Exact Cover Problem

- Backtrack on the columns
- Choose a column to cover, this will dictate a selection of rows
- Loop over rows, for each row choice remove covered columns
DLX for the Exact Cover Problem

- Backtrack on the columns
- Choose a column to cover, this will dictate a selection of rows
- Loop over rows, for each row choice remove covered columns
- Recursively analyze new, smaller matrix
- Restore rows and columns on backtrack step
Exact Cover Example (Knuth, 2000)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Exact Cover Representation (Knuth, 2000)
Exact Cover Representation (Knuth, 2000)

- Cover column A
- Remove rows 2, 4

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Exact Cover Representation (Knuth, 2000)

- Loop through rows
- Row 2 covers D, G
- D removes row 4, 6
- G removes row 5, 6

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Recurse on $2 \times 4$ matrix
It has no solution, so will soon backtrack
Implementation in Sage

The games module only contains code for solving Sudoku puzzles, which I wrote in two hours on Alaska Airlines, in order to solve the puzzle in the inflight magazine. — William Stein, Sage Founder

- Sage, open source mathematics software, sagemath.org
Implementation in Sage

The games module only contains code for solving Sudoku puzzles, which I wrote in two hours on Alaska Airlines, in order to solve the puzzle in the inflight magazine. — William Stein, Sage Founder

- Sage, open source mathematics software, sagemath.org
- Stein (UW): naive recursive backtracking, run times of 30 minutes
- Carlo Hamalainen (Turkey/Oz): DLX for exact cover problems
- Tom Boothby (UW): Preliminary representation as an exact cover
- RAB: Optimized backtracking
  - lots of look-ahead
  - automatic Cython conversion of Python to C
- RAB: new class, conveniences for printing, finished DLX approach
Timings in Sage

Test Examples:
- Original doctest, provenance is Alaska Airlines in-flight magazine?
- 17-hint “random” puzzle (no 16-hint puzzle known)
- Worst-case: top-row empty, top-row solution 987654321
- All ~48,000 known 17-hint puzzles (Gordon Royle, UWA)

Equipment: R 3500 machine, 3 GHz Intel Core Duo

<table>
<thead>
<tr>
<th>Puzzle</th>
<th>Time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Naive</td>
</tr>
<tr>
<td>Alaska</td>
<td>34</td>
</tr>
<tr>
<td>17</td>
<td>1,494,000</td>
</tr>
<tr>
<td>Worst</td>
<td>4,798,000</td>
</tr>
<tr>
<td>48K 17</td>
<td></td>
</tr>
</tbody>
</table>
Talk available at:
buzzard.pugetsound.edu/talks.html