Algorithm Theory

(08 exact cover, backtracking, multiply linked lists, dancing links)

Alois Heinz
Heilbronn University,
Max-Planck-Str. 39, 74081 Heilbronn
heinz@hs-heilbronn.de

Nov 17 2022

Overview: Exact Cover Problem

- common computational core of many practical application problems
- can be used to model decision, optimization, constraint satisfaction, or counting problems
- Ex.: partitions, tilings, time tables, graph colorings, map colorings, n-queens, sudoku
- challenge: find / any / the best / all / the # / of the potential solutions
- can be described with the help of the subsets of a set, matrices (tables) or bipartite graphs
- $\mathcal{NP}$-complete problem, therefore solutions are often difficult to find
- dancing links (DLX) is a recursive, nondeterministic, depth-first backtracking algorithm, that finds solutions to the exact cover problem efficiently.
- DLX is able to search the complete potential solution space, the efficiency is a direct result of a most intelligent use of list structures to model and adapt matrices fast

Definition: Exact Cover Problem

Exact Cover problem:

Given: finite set X of elements, and $S \subseteq 2^X$, set of subsets of X

Sought: one / the best / all / the $\#$ of subsets $S^* \subseteq S$, which allow a disjunctive decomposition of X:

$$\biguplus_{M_j \in S^*} M_j = X \quad \wedge \quad (\forall M_k, M_\ell \in S^*) \;\; M_k \cap M_\ell = \emptyset$$

Example: $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $S = \{\{1, 5, 9\}$, $\{1, 6, 8\}$, $\{2, 4, 9\}$, $\{2, 5, 8\}$, $\{2, 6, 7\}$, $\{3, 4, 8\}$, $\{3, 5, 7\}$, $\{4, 5, 6\}\}$

Variant: In some cases it is sufficient to cover only a subset $\widetilde{X} \subseteq X$ disjunctively: $\biguplus_{M_j \in S^*} M_j = \widetilde{X}$

Question: How can solutions be found — as fast as possible?

Ex.: Set Partitions

Find the number $a(n)$ of all disjunctive decompositions of $X = \{1, 2, \ldots, 6n+3\}$ into $(2n+1)$ 3-element subsets of X having the same element sum.

If $n = 1$ we have $X = \{1, 2, \ldots, 9\}$ and

$S = \{\{1,5,9\}, \{1,6,8\}, \{2,4,9\}, \{2,5,8\}, \{2,6,7\}, \{3,4,8\}, \{3,5,7\}, \{4,5,6\}\}$

The cover-matrix of the problem is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$S_1 = \{\{1,5,9\}, \{2,6,7\}, \{3,4,8\}\}$

$S_2 = \{\{1,6,8\}, \{2,4,9\}, \{3,5,7\}\}$ $\qquad a(1) = 2$

Ex.: Tilings using 2×2-Tiles and L-Trominoes

Let $b(n)$ be the number of tilings in a $4 \times n$-rectangle, e.g. in a 4×4-rectangle:
$X = \{1, \ldots, 16\}$, $|S| = 45$

$S = \{\{1,2,5\}, \{1,2,6\}, \{1,5,6\}, \{2,3,6\}, \{2,3,7\}, \{2,5,6\}, \{2,6,7\}, \{3,4,7\},$
$\{3,4,8\}, \{3,6,7\}, \{3,7,8\}, \{4,7,8\}, \{5,6,9\}, \{5,6,10\}, \{5,9,10\}, \{6,7,10\},$
$\{6,7,11\}, \{6,9,10\}, \{6,10,11\}, \{7,8,11\}, \{7,8,12\}, \{7,10,11\}, \{7,11,12\},$
$\{8,11,12\}, \{9,10,13\}, \{9,10,14\}, \{9,13,14\}, \{10,11,14\}, \{10,11,15\},$
$\{10,13,14\}, \{10,14,15\}, \{11,12,15\}, \{11,12,16\}, \{11,14,15\}, \{11,15,16\},$
$\{12,15,16\}, \{1,2,5,6\}, \{2,3,6,7\}, \{3,4,7,8\}, \{5,6,9,10\}, \{6,7,10,11\},$
$\{7,8,11,12\}, \{9,10,13,14\}, \{10,11,14,15\}, \{11,12,15,16\}\}$

```
._______.  ._______.  ._______.  ._______.  ._______.  ._______.
|1 2|3 4|  |  _|_  |  |  _|   |  |  _|_  |  |  _|_  |  |   |_  |
|5_6|7_8|  |_|   |_|  |_| |___|  |_|  _|_|  |_|_  |_|  |___| |_|
|9 A|B C|  | |___| |  | |___| |  | |_|   |  |   |_| |  | |___| |
|D_E|F_0|  |___|___|  |___|___|  |___|___|  |___|___|  |___|___|
   S1         S2         S3         S4         S5         S6
```

Here we have exactly $b(4) = 6$ solutions.

Ex.: N-Queens Problem

Find the number $c(n)$ of placements of n queens on the $n \times n$ chessboard such that no queen is able to capture any other using the standard chess queen's moves.

This problem can be coded into a cover-matrix with one row for each queen position, one column for each of the rows, columns and diagonals of the chessboard. $c(4) = 2$:

```
(r,c) || row  | col  | up diag | down dg || elements  || solutions |
      || 1234 | 1234 | 1234567 | 1234567 ||   of X    ||           |
------++-!!!!-+-!!!!-+---------+---------++-----------++-----------+
(1,1) || 1    | 1    | 1       |    1    || 1 5  9 19 ||           |    (1) 1 2 3 4
(1,2) || 1    |  1   |  1      |     1   || 1 6 10 20 ||      (2)  |       +-------+
(1,3) || 1    |   1  |   1     |      1  || 1 7 11 21 || (1)       |     1 |    Q  |
(1,4) || 1    |    1 |    1    |       1 || 1 8 12 22 ||           |     2 |Q      |
(2,1) ||  1   | 1    |  1      |   1     || 2 5 10 18 || (1)       |     3 |      Q|
(2,2) ||  1   |  1   |   1     |    1    || 2 6 11 19 ||           |     4 |  Q    |
(2,3) ||  1   |   1  |    1    |     1   || 2 7 12 20 ||           |       +-------+
(2,4) ||  1   |    1 |     1   |      1  || 2 8 13 21 ||      (2)  |
(3,1) ||   1  | 1    |   1     |  1      || 3 5 11 17 ||      (2)  |    (2) 1 2 3 4
(3,2) ||   1  |  1   |    1    |   1     || 3 6 12 18 ||           |       +-------+
(3,3) ||   1  |   1  |     1   |    1    || 3 7 13 19 ||           |     1 |  Q    |
(3,4) ||   1  |    1 |      1  |     1   || 3 8 14 20 || (1)       |     2 |      Q|
(4,1) ||    1 | 1    |    1    | 1       || 4 5 12 16 ||           |     3 |Q      |
(4,2) ||    1 |  1   |     1   |  1      || 4 6 13 17 || (1)       |     4 |    Q  |
(4,3) ||    1 |   1  |      1  |   1     || 4 7 14 18 ||      (2)  |       +-------+
(4,4) ||    1 |    1 |       1 |    1    || 4 8 15 19 ||           |
```

DLX: Basic Ideas

- Store only 1-elements of the (sparsely populated) cover matrix within a multiply linked list structure (preprocessing)
  - for each row there is a circular doubly linked list of 1's
  - for each column there is a circular doubly linked list of 1's plus a column header element (with additional information)
  - a further circular doubly linked list contains all these column headers plus a special root element h
  - each element has references (pointers) for `left`, `right`, `up`, `down` and an additional reference `column` pointing to the column header element
- find a subset of the rows such that all their 1-elements cover at least the mandatory columns (and each column is covered not more than once)
- the temporary removal of a column c from the doubly linked row list is done like this:

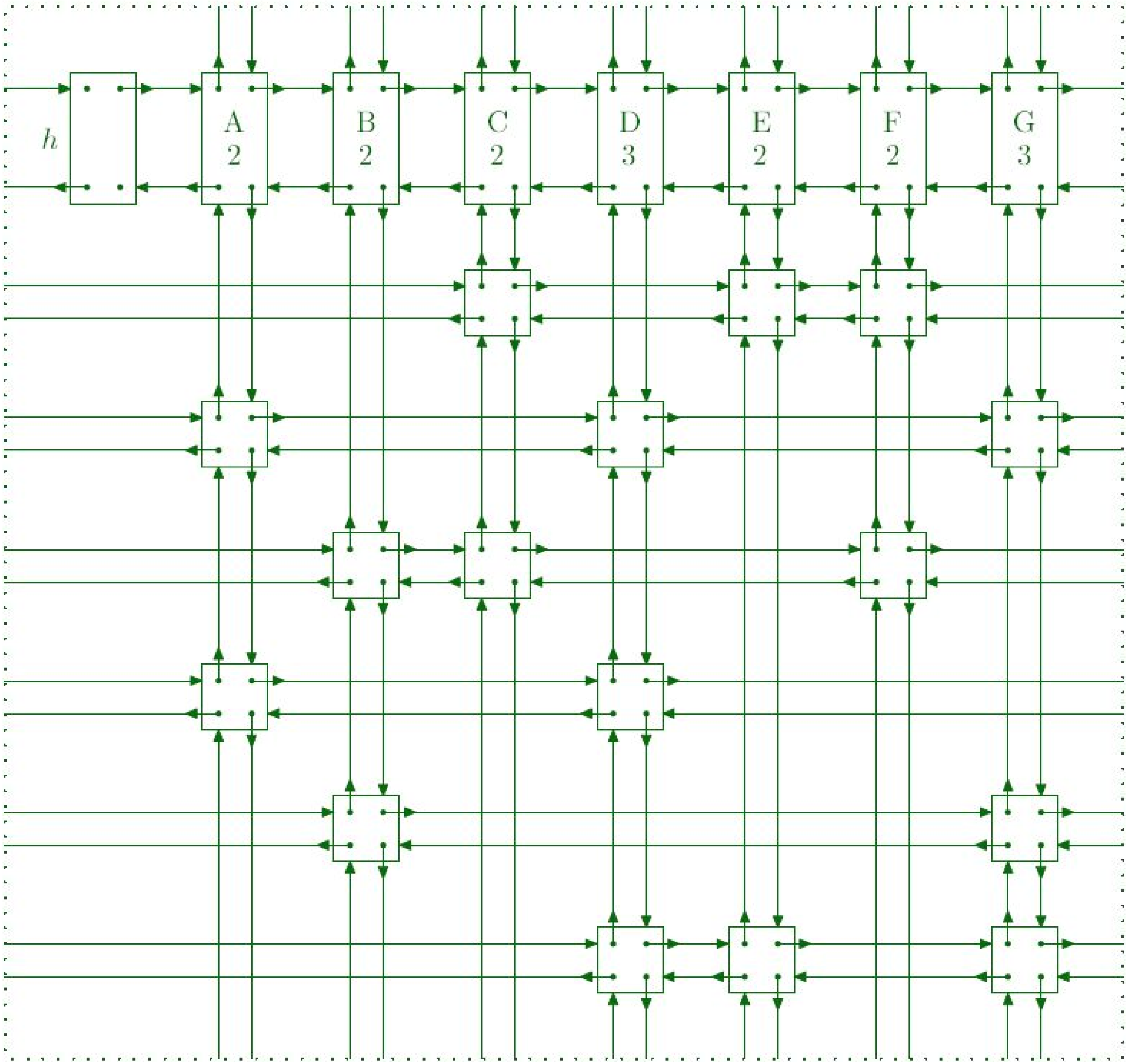
```
c.left.right = c.right;
c.right.left = c.left;
```

- re-inserting c into the doubly linked row list is done like this:

```
c.left.right = c;
c.right.left = c;
```

- removal from and re-insertion into column list is done analogously

DLX: Matrix Link Structure



Source: Donald E. Knuth: Dancing Links, P. 6

DLXNode und search

```
class DLXNode{                  // represents 1 element or header
  DLXNode C;                      // reference to column-header
  DLXNode L, R, U, D;             // left, right, up, down references
  DLXNode(){ C=L=R=U=D=this; } // supports circular lists
}
public static void search (int k){ // finds & counts solutions
  if (h.R == h) {cnt++; return;}      // if empty: count & done
  DLXNode c = h.R;                    // choose next column c
  cover(c);                           // remove c from columns
  for (DLXNode r=c.D; r!=c; r=r.D){  // forall rows with 1 in c
     for (DLXNode j=r.R; j!=r; j=j.R) // forall 1-elements in row
        cover(j.C);                     // remove column
     search(k+1);                     // recursion
     for (DLXNode j=r.L; j!=r; j=j.L) // forall 1-elements in row
        uncover(j.C);                   // backtrack: un-remove
  }
  uncover(c);                         // un-remove c to columns
}
```

cover und uncover

```
public static void cover (DLXNode c){ // remove column c
  c.R.L = c.L ;                       // remove header
  c.L.R = c.R ;                       // .. from row list
  for (DLXNode i=c.D; i!=c; i=i.D)    // forall rows with 1
     for (DLXNode j=i.R; i!=j; j=j.R){  // forall elem in row
        j.D.U = j.U ;                     // remove row element
        j.U.D = j.D ;                     // .. from column list
     }
}
public static void uncover (DLXNode c){//undo remove col c
  for (DLXNode i=c.U; i!=c; i=i.U)    // forall rows with 1
     for (DLXNode j=i.L; i!=j; j=j.L){  // forall elem in row
        j.D.U = j ;                       // un-remove row elem
        j.U.D = j ;                       // .. to column list
     }
  c.R.L = c ;                         // un-remove header
  c.L.R = c ;                         // .. to row list
}
```

A Selection of Solutions

n	$a(n)$	$b(n)$	$c(n)$
0	1	1	1
1	2	0	1
2	11	1	0
3	84	4	0
4	1296	6	2
5	24293	16	10
6	703722	37	4
7	24212879	92	40
8	1157746949	245	92
9	63552536107	560	352
10	?	1426	724
27		9564393972	234907967154122528
50		16392547758672574777	?
100		1943247477519075960641089333020587192662	

$$
\begin{aligned}
b(n) \quad &= \quad b(n-1) + b(n-2) + 9\,b(n-3) + b(n-4) \\
&\qquad - 3\,b(n-5) - 22\,b(n-6) - 16\,b(n-7) - 4\,b(n-9)
\end{aligned}
$$

Exercises

Write a Java program that reads a number $n \in \mathbb{N}$ and computes $a(n)$.
Here $a(n)$ is the number of set partitions of $\{1, 2, \ldots, 5n\}$ into 5-element subsets $\{i, i+k, i+2k, i+3k, i+4k\}$ with $1 \le k \le n$.
Use your program to compute a table with $n \to a(n)$ as large as possible.

Hint: $a(4) = 10$:

```
{{1,2,3,4,5}, {6,7,8,9,10}, {11,12,13,14,15}, {16,17,18,19,20}},
{{1,3,5,7,9}, {2,4,6,8,10}, {11,12,13,14,15}, {16,17,18,19,20}},
{{1,2,3,4,5}, {6,8,10,12,14}, {7,9,11,13,15}, {16,17,18,19,20}},
{{1,4,7,10,13}, {2,5,8,11,14}, {3,6,9,12,15}, {16,17,18,19,20}},
{{1,2,3,4,5}, {6,7,8,9,10}, {11,13,15,17,19}, {12,14,16,18,20}},
{{1,3,5,7,9}, {2,4,6,8,10}, {11,13,15,17,19}, {12,14,16,18,20}},
{{1,5,9,13,17}, {2,4,6,8,10}, {3,7,11,15,19}, {12,14,16,18,20}},
{{1,2,3,4,5}, {6,9,12,15,18}, {7,10,13,16,19}, {8,11,14,17,20}},
{{1,3,5,7,9}, {2,6,10,14,18}, {4,8,12,16,20}, {11,13,15,17,19}},
{{1,5,9,13,17}, {2,6,10,14,18}, {3,7,11,15,19}, {4,8,12,16,20}}.
```