Dancing Links
an educational pearl

Massimo Nocentini, PhD.
massimo.nocentini@gmail.com

ESUG2019 – August 28, 2019.

SCHMIDT GmbH
Ingenieurbüro für Bauwesen

```
outline
    ^ LinkedList new
      add: 'me and the core idea';
      add: 'DoubleLink objs';
      add: 'exact cover problem';
      add: 'AlgorithmX';
      add: 'covering and uncovering columns';
      add: 'N-Queens and Sudoku problems';
      yourself
```

Hi!

```
$ whoami
Massimo Nocentini, PhD
Mathematician (algebraic combinatorics, formal methods for algs)
Programmer (automated reasoning, logics and symbolic comp)
https://github.com/massimo-nocentini/dancinglinksst
```

In Donald's words[1]: *Suppose* x *points to an element of a doubly linked list; let* $L[x]$ *and* $R[x]$ *point to the predecessor and successor of that element. Then:*

$$L[R[x]] \leftarrow L[x], \quad R[L[x]] \leftarrow R[x] \quad (1)$$

remove x *from the list; every programmer knows this. But comparatively few programmers have realized that*

$$L[R[x]] \leftarrow x, \quad R[L[x]] \leftarrow x \quad (2)$$

will put x *back again, with no refs to the whole list at all.*

[1] https://arxiv.org/abs/cs/0011047

Space for sketching

Main ideas

SCHMIDT GmbH
Ingenieurbüro für Bauwesen

- Operation (2) arises in **backtrack programs**, which enumerate all solutions to a given set of constraints and it was introduced in 1979 by Hitotumatu and Noshita.
- The beauty of (2) is that operation (1) can be undone by knowing only the value of x.
- We can apply (1) and (2) repeatedly in complex data structures that involve large numbers of interacting doubly linked lists.
- Knuth: "*This process causes the pointer variables inside the global data structure to execute an exquisitely choreographed dance; hence I like to call* (1) *and* (2) *the **technique of dancing links**.*"
- Minato et al. [2] constructs a Zero-suppressed BDD (ZDD) that represents the set of sols and it enables the efficient use of memo cache to speed up the search.

---

[2] https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14907

DoubleLinks and DoubleLinkedLists

DoubleLink objects respond to messages

```
remove
  nextLink ifNotNil: [ :next | next previousLink: previousLink ].
  previousLink ifNotNil: [ :previous | previous nextLink: nextLink ]
```

and

```
restore
  nextLink ifNotNil: [ :next | next previousLink: self ].
  previousLink ifNotNil: [ :previous | previous nextLink: self ]
```

that implement operations (1) and (2), respectively; moreover, we extend DoubleLinkedList objects with the message

```
makeCircular
  head
    ifNotNil: [
      head previousLink: tail.
      tail nextLink: head ]
```

to introduce circular, doubly connected, lists.

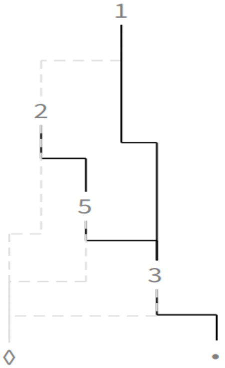
Exact Cover $\in \mathcal{NP}$

Given a matrix of 0*s and* 1*s, does it have a set of rows containing* ***exactly one*** *symbol* 1 *in each column?*

The problem with matrix

$$\begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \end{pmatrix}^T \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}^T$$

is solved by two sets of rows, namely
$\{r_1 = 1, r_3 = 1\}$ and $\{r_2 = 1, r_5 = 1, r_3 = 1\}$.



We can think of the columns as elements of a universe, and the rows as subsets of the universe; then the problem is to cover the universe with disjoint subsets, $\mathcal{NP}$-complete even when each row contains exactly three 1s.

Space for sketching

AlgorithmX instance side

```
searchDepth: k forDLRootObject: h partialSelection: cont
  ^ (h isFixPointOf: [ :ro | ro right ])
     ifTrue: [ self yieldNode: top onBlock: cont ]
     ifFalse: [
       memo
         at: h columns
         ifPresent: [ :tree | self yieldNode: tree onBlock: cont ]
         ifAbsentPut: [
           self
             searchDepth: k
               forDLColumnObject: h chooseColumn
               partialSelection: cont ] ]
```

AlgorithmX instance side

```
searchDepth: k forDLColumnObject: c partialSelection: sel
  ^ self
      onEnter: [ c cover ]
      do: [
        c
          untilFixPointOf: [ :co | co up ]
          foldr: [ :r :x |
            | y |
            y := self searchDepth: k
                      forDLDataObject: r
                      partialSelection: sel.
            y isZDDBottom
                ifTrue: [ x ]
                ifFalse: [
                  self
                    uniqueNodeWithDLDataObject: r
                      withLowerNode: x
                      withHigherNode: y ] ]
          init: bottom ]
      onExit: [ c uncover ]
```

AlgorithmX instance side

SCHMIDT GmbH
Ingenieurbüro für Bauwesen

```
searchDepth: k forDLDataObject: r partialSelection: cont
  ^ self
     onEnter: [ r untilFixPointOf: [ :ro | ro right ]
                  do: [ :j | j column cover ] ]
     do: [ self
             searchDepth: k + 1
               forDLRootObject: r column root
               partialSelection: [ :sel |
                 cont
                   value:
                     (ValueLink new
                       value: r model;
                       nextLink: sel;
                       yourself) ] ]
     onExit: [ r untilFixPointOf: [ :ro | ro left ]
                 do: [ :j | j column uncover ] ]
```

AlgorithmX instance side

The entry point is the message

```
searchDLRootObject: h onSolutionDo: aBlock
  ^ self
      searchDepth: 0
      forDLRootObject: h
      partialSelection: [ :selLink |
        aBlock value: (LinkedList new add: selLink; asSet) ]
```

Knuth advices to use the heuristic (provided by `DLRootObject` objs)

```
chooseColumn
  ^ self chooseColumnWithWeight: #size withOpt: #<
```

```
chooseColumnWithWeight: weightBlock withOpt: optBlock
  ^ self
      untilFixPointOf: [ :ro | ro left ]
      foldr: [ :j :r |
        (optBlock value: (weightBlock value: j) value: (weightBlock value: r))
          ifTrue: [ j ]
          ifFalse: [ r ] ]
      init: self right
```

that *minimizes* the search tree's branching factor.

DLColumnObject instance side

The operation of *covering* column c removes c from the header list and removes all rows in c's own list from the other column lists they are in.

```
cover
  we remove.
  self
    untilFixPointOf: [ :co | co down ]
    do: [ :i |
      i
        untilFixPointOf: [ :do | do right ]
        do: [ :j |
          j nsLink remove.
          j column updateSize: [ :s | s - 1 ] ] ]
```

Operation (1) is used here to remove objects in both the horizontal and vertical directions.

DLColumnObject instance side

Finally, we get to the operation of *uncovering* a given column c. Here is where the links do their dance:

```
uncover
  self
    untilFixPointOf: [ :co | co up ]
    do: [ :i |
      i
        untilFixPointOf: [ :do | do left ]
        do: [ :j |
          j nsLink restore.
          j column updateSize: [ :s | s + 1 ] ] ].
  we restore
```

Notice that uncovering takes place in precisely the reverse order of the covering operation, using the fact that (2) undoes (1).

Sudoku to Exact Cover reduction

SCHMIDT GmbH Ingenieurbüro für Bauwesen

```
emptySudokuIndicators
  | ones start end |
  start := 0. end := 8. ones := LinkedList new.
  start to: end do: [ :row |
    start to: end do: [ :column |
      start to: end do: [ :value |
        | rowIndex cellConstraint rowConstraint
          columnConstraint boxConstraint model |
        model := {(#x -> row). (#y -> column). (#v -> value)} asDictionary.
        rowIndex := 81 * row + (9 * column) + value.
        cellConstraint := rowIndex @ ((end + 1) * row + column).
        rowConstraint := rowIndex @ (9 * row + value + 81).
        columnConstraint := rowIndex @ (9 * column + value + (81 * 2)).
        boxConstraint := rowIndex
            @ (27 * (row // 3) + (9 * (column // 3)) + value + (81 * 3)).
        ones
          add: ((cellConstraint + 1) asDLPoint primary: true) -> model;
          add: ((rowConstraint + 1) asDLPoint primary: true) -> model;
          add: ((columnConstraint + 1) asDLPoint primary: true) -> model;
          add: ((boxConstraint + 1) asDLPoint primary: true) -> model ] ] ].
  ^ ones
```

Soduku solutions

```
testDLXonSudoku
  | grid sols chain matrices |
  grid := DLDataObject gridOn: DLDataObjectTest new emptySudokuIndicators.
  chain := Generator
    on: [ :g | AlgorithmX new
              searchDLRootObject: (grid at: #root)
              onSolutionDo: [ :sel | g yield: sel ] ].
  sols := (chain next: 2) contents.
  matrices := sols collect: [ :sol | "build the corresponding matrix" ].
  self
    assert: matrices first printString
    equals:
      '(9 8 7 6 5 4 3 2 1
        6 5 4 3 2 1 9 8 7
        3 2 1 9 8 7 6 5 4
        8 9 6 7 4 5 2 1 3
        7 4 5 2 1 3 8 9 6
        2 1 3 8 9 6 7 4 5
        5 7 9 4 6 8 1 3 2
        4 6 8 1 3 2 5 7 9
        1 3 2 5 7 9 4 6 8 )'.
```

N-Queens to Exact Cover reduction

SCHMIDT GmbH
Ingenieurbüro für Bauwesen

```
NQueensIndicators: n
  | ones |
  ones := LinkedList new.
  0 to: n - 1 do: [ :row |
    0 to: n - 1 do: [ :column |
      | rowIndex rowConstraint columnConstraint
        diagonalConstraint antiDiagonalConstraint model |
      model := Dictionary new at: #x put: row; at: #y put: column; yourself.
      rowIndex := n * row + column.
      rowConstraint := rowIndex @ row.
      columnConstraint := rowIndex @ (n + column).
      diagonalConstraint := rowIndex @ (2 * n + (row + column)).
      antiDiagonalConstraint := rowIndex
            @ (2 * n + (2 * n) - 1 + (n - 1 - row + column)).
      ones
        add: ((rowConstraint + 1) asDLPoint primary: true) -> model;
        add: ((columnConstraint + 1) asDLPoint primary: true) -> model;
        add: ((diagonalConstraint + 1) asDLPoint primary: false) -> model;
        add: ((antiDiagonalConstraint + 1) asDLPoint primary: false) -> model
  ^ ones
```

N-Queens solutions

```
testDLXon_NQueens_sequence
  | seq elapsedTime |
  elapsedTime := [ seq := (1 to: 10)
    collect: [ :i | (self runDLXonNQueens: i next: nil) size ] ] timeToRun.
  self assert: elapsedTime < 2 asSeconds.
  self assert: seq "also known as https://oeis.org/A000170"
        equals: {1 . 0 . 0 . 2 . 10 . 4 . 40 . 92 . 352 . 724}

testDLXon_8Queens
  | matrices |
  matrices := self runDLXonNQueens: 8 next: 1.
  self
    assert: matrices first printString
    equals:
      '(0 0 0 0 0 0 0 1
        0 0 0 1 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 1 0 0 0 0 0
        0 0 0 0 0 1 0 0
        0 1 0 0 0 0 0 0
        0 0 0 0 0 0 1 0
        0 0 0 0 1 0 0 0 )'.
```

Final remarks

- We presented a vanilla implementation of DLX with the ZDD extension in pure Smalltalk, with an educational savor.
- It is designed to be easy to understand and to play with still remaining efficient and robust.
- Dancing links are considered the state-of-the-art heuristic for EC
- :) Knuth is still actively working on this (a new fascicle is in prep [3])
- :(Constraints are verbose and rigid to express (currently we use `Point` objs), looking for a DSL that makes coding constraints easier
- TODO
  - Group column objects using different colours to gain expressivity
  - Write reductions to Exact Cover (from 3SAT, Knapsack, TSP, ...)

[3] https://cs.stanford.edu/~knuth/fasc5c.ps.gz

Thanks!

```
yieldNode: tree onBlock: cont
  tree sets
    collect: [ :each | (each collect: #model) as: LinkedList ]
    thenDo: [ :sel |
      | link |
      link := sel isEmpty ifTrue: [ nil ] ifFalse: [ sel firstLink ].
      cont value: link ].
  ^ tree


uniqueNodeWithDLDataObject: r withLowerNode: x withHigherNode: y
  | key |
  key := Array with: r with: x with: y.
  ^ zDDTree
    at: key
    ifAbsentPut: [ | z |
      z := ZDDNode new model: r; lower: x; higher: y; yourself.
      x parent: z.
      y parent: z.
      z ]
```

DLDataObject class side

SCHMIDT GmbH
Ingenieurbüro für Bauwesen

```
gridOn: aCollection
  | rootObj columns rows headers allObjs |
  aCollection
    sort: [ :vAssoc :wAssoc |
      | v w |
      v := vAssoc key.
      w := wAssoc key.
      v y <= w y and: [ v x <= w x ] ].
  allObjs := Dictionary new.
  headers := DoubleLinkedList new.
  columns := Dictionary new.
  rows := Dictionary new.
  rootObj := DLRootObject new
    addInDoubleLinkedList: headers direction: #we;
    yourself.
  allObjs at: #root put: rootObj.
  "to be contd..."
```

DLDataObject class side

```
gridOn: aCollection
  "...contd..."
  aCollection
    do: [ :anAssociation |
      | aPoint columnObj dataObj column row |
      aPoint := anAssociation key.
      column := columns
        at: aPoint y
        ifAbsentPut: [ | headerObj newColumn |
          headerObj := DLColumnObject new size: 0; root: rootObj; yourself.
          aPoint primary
            ifTrue: [ headerObj addInDoubleLinkedList: headers
                                direction: #we ]
            ifFalse: [ DoubleLinkedList
                circular: [ :dll | headerObj addInDoubleLinkedList: dll
                                             direction: #we ] ].
          newColumn := DoubleLinkedList new.
          headerObj addInDoubleLinkedList: newColumn direction: #ns.
          allObjs at: aPoint y put: headerObj.
          newColumn ].
  "..to be contd further..."
```

DLDataObject class side

SCHMIDT GmbH
Ingenieurbüro für Bauwesen

```
gridOn: aCollection
  "...contd"
      columnObj := column first.
      dataObj := DLDataObject new
        column: columnObj;
        point: aPoint;
        model: anAssociation value;
        yourself.
      row := rows at: aPoint x ifAbsentPut: [ DoubleLinkedList new ].
      dataObj
        addInDoubleLinkedList: column direction: #ns;
        addInDoubleLinkedList: row direction: #we.
      columnObj updateSize: [ :s | s + 1 ].
      allObjs at: aPoint put: dataObj ].
  headers makeCircular.
  columns valuesDo: #makeCircular.
  rows valuesDo: #makeCircular.
  ^ allObjs
```