Space-efficient representations of "non-linear" objects

Rajeev Raman, University of Leicester
ラアジブ　ラアマン、レスタ大學

Introduction

- We are increasingly processing very large data sets.
- Processing is often *complex* and *in-memory*.
  - String processing using suffix trees:
    - *n* characters › 10*n* bytes in main memory.
  - XML processing using standard libraries:
    - file of *n* bytes › 10*n* bytes in main memory ("XML bloat").
- Such files typically compress well (5x – 10x) so disk storage is two orders of magnitude smaller…
- Can we get *in-memory* data structures with *compressed* file size memory usage?

Processing Compressed Data

- Data compression looks for regularities/repeated substructures in data.
  - such regularities can be useful for tasks such as pattern matching, data mining etc.
  - these regularities may be expressed in a highly non-local or data-dependent way.
- Can these regularities be exploited in a way that is more amenable to processing while in compressed form?
- *Succinct and compressed data structuring.*

Succinct and Compressed Data Structures

- Field started in Jacobson's PhD thesis (CMU, 1989).
  - precursors in Elias (1975), Tarjan/Yao (1979), Chazelle (1985)…
- Significant acceleration in the last decade.
- Basic aim: given a combinatorial object x from a class of objects S, and a set of operations, represent x so that:
  - operations are supported rapidly (typically $O(1)$ time).
  - memory used is "close" to the information-theoretic = $\log_2|S|$ bits.
- Information-theoretic bounds are not *instance-sp*
  - full binary tree on n nodes = random binary tree on n nodes
  - ITLB corresponds to uniform distribution on set of objects S.
    - Reality – non-uniform!
    - "Compressed data structures."

Unit-cost Random-Access Machine (RAM) model

State of Art

- Many results on "linear" or "linearizable" objects:
  - strings, trees.
  - directions have been to extend functionality.
- Some results on graph representations:
  - typically for constrained families of graphs (e.g. planar).
  - general (random) graphs – not too interesting (Farzan, Munro, ESA 2009).
  - compressible graphs?
- Consider more "tree-like" objects:
  - Random access to grammar-compressed strings
  - Representing DAG/BDDs
  - Posets of small width

The Practice

- Many implementations
  - Pizza + Chili corpus (Pisa and U. Chile)
    - Opportunistic indices, Compressed Suffix Arrays.
  - Sux4J (Milan)
  - Aladdin project (CMU).
- XML indexing
  - SiXML project (Leicester)
    - Collaborating with FLWOR foundation to deliver a succinct XML storage for the Zorba Xquery processor.

Random Access in Grammar-Compressed Strings and Trees

(Bille, Landau, Raman, Rao, Sadakane, Weimann, SODA 2011)

Grammar Compression

Input: string S of length N in straight-line program (SLP) form with m lines:

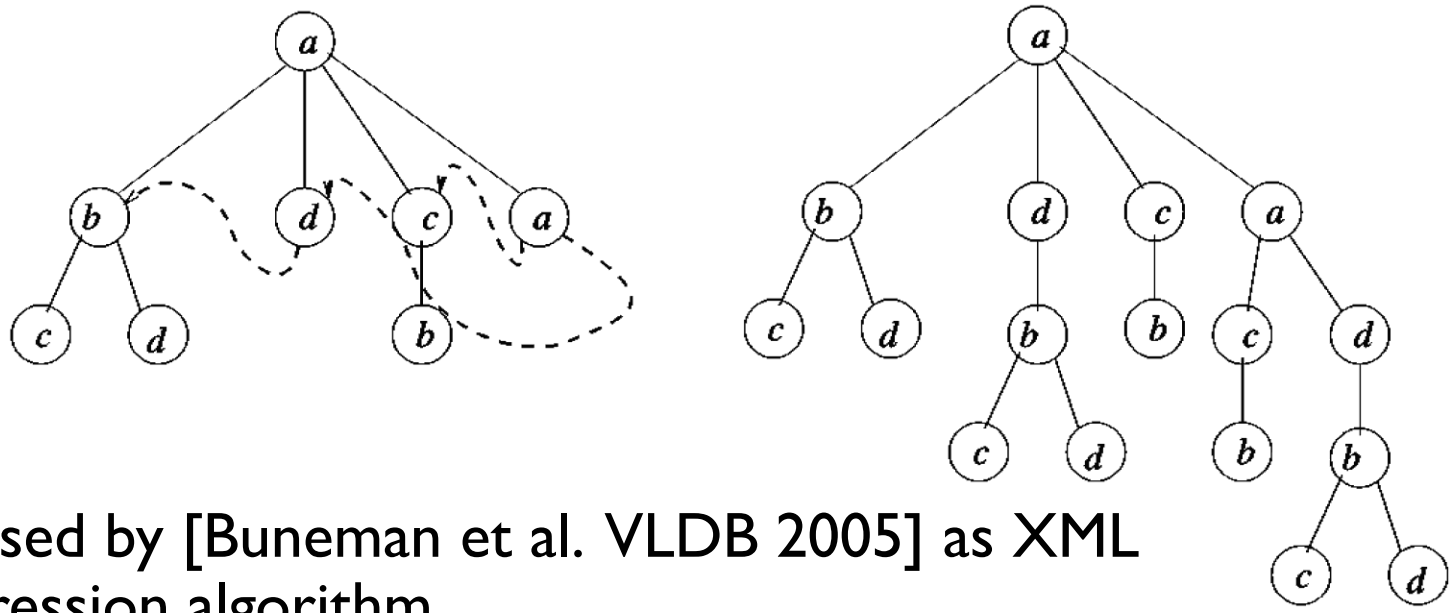
- $F_1 = b$
- $F_2 = a$
- $F_3 = F_2 F_1$
- $F_4 = F_3 F_2$
- $F_5 = F_4 F_3$
- $F_6 = F_5 F_4$

- $S = abaababa$

- SLP's capture many popular data compression algorithms: LZ* etc.
- Can give exponential compression.
- **Query**: access *i*th symbol of S

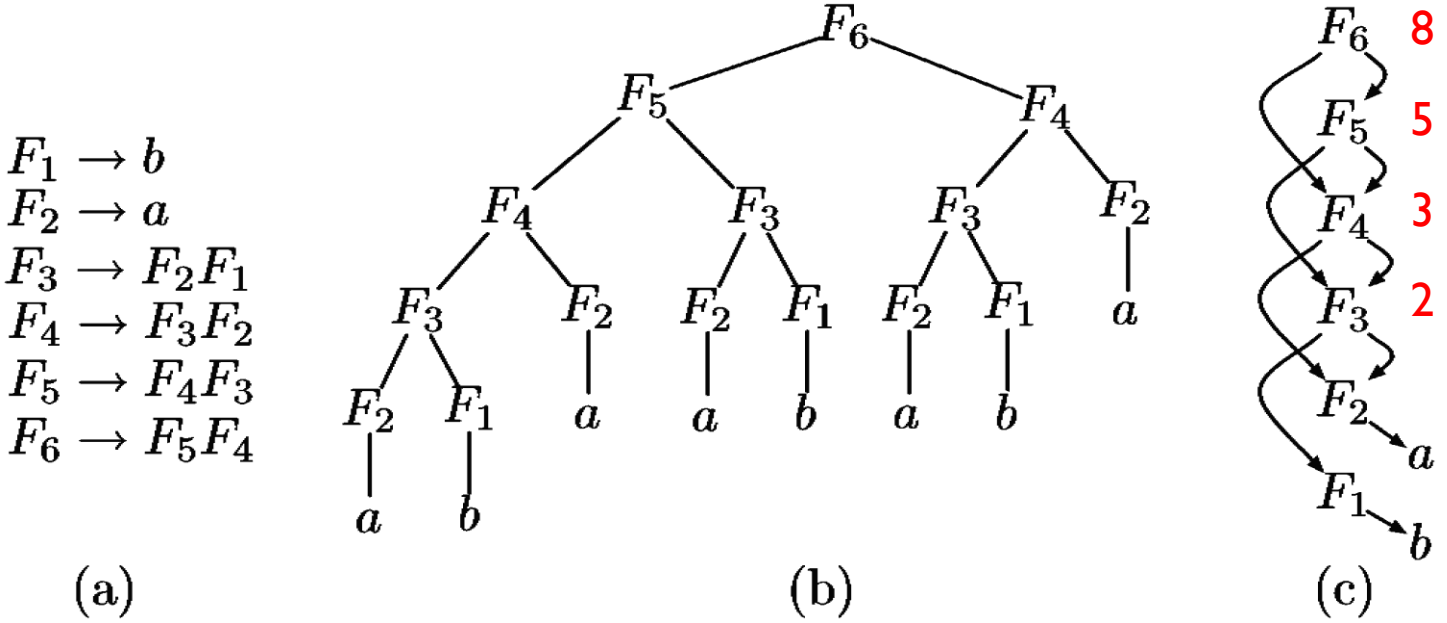
Related Problem

- Input: labelled tree *T* compressed by sharing subtrees:



- Proposed by [Buneman et al. VLDB 2005] as XML compression algorithm.
- [Grohe et. al] – aids Xquery processing.
- Operations on tree?
  - navigational (parent, next sibling, anc, desc, etc).
  - labelled operations?

Existing Solutions



Naïve algorithm: store size of string generated by each non-terminal, and walk down the DAG. Time: $O(h)$, space $O(m)$ words. But h can be $\Omega(m)$.

Alternatively: "balance" grammar [Rytter, TCS 2003, Charikar et al. IEEE Trans IT 2005], to give height $O(\log N)$, and use above algorithm. Time = $O(\log N)$. But the SLP size increases to $O(m \log N)$, so space is $O(m \log N)$ words.

New Solution

	Time	Space (words)
Naïve	$O(m)$	$O(m)$
Balanced Grammar	$O(\log N)$	$O(m \log N)$
New	$O(\log N)$	$O(m)$

m: grammar size, N: original string size. Words are O(log N) bits long.

- New solution
  - Uses a "heavy-path" decomposition of the DAG.
  - Observe that "logarithmic" biased search is ideal.
  - A novel linear-space data structure for optimal-time biased ancestor search in trees.

Heavy Path Decomposition

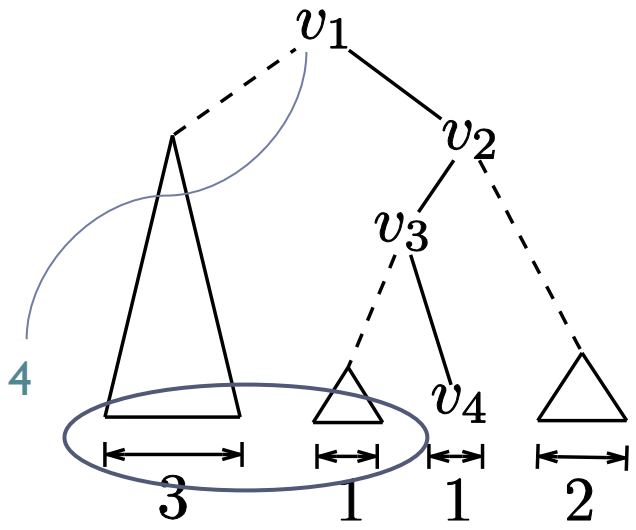
- Classify edges in the DAG as *heavy* or *light*:
  - If $X = YZ$ is an instruction, then we look at the size of the substring derived from the non-terminals X, Y and Z.
    - If $|Y| > |X|/2$ the edge from X to Y is **heavy** and to Z **light** (vv).
    - Y is **heavy** descendant, **light** descendant ow.
    - X has **at most** one heavy descendant in the DAG.
  - The graph induced by the heavy edges is a *forest*, it has one root for each terminal symbol.
- DAG becomes a heavy path forest + light edges.
  - No root-to-leaf path in the DAG follows more than $O(\log N)$ light edges [Tarjan 1983].

Search in the Heavy Path Forest

- Overall idea: walk down the DAG as in the naïve algorithm, but:
  - speed up on heavy paths.
  - root-to-leaf path passes through $O(\log N)$ heavy paths.
  - *desired running time is also O(log N) – constant time per path?*

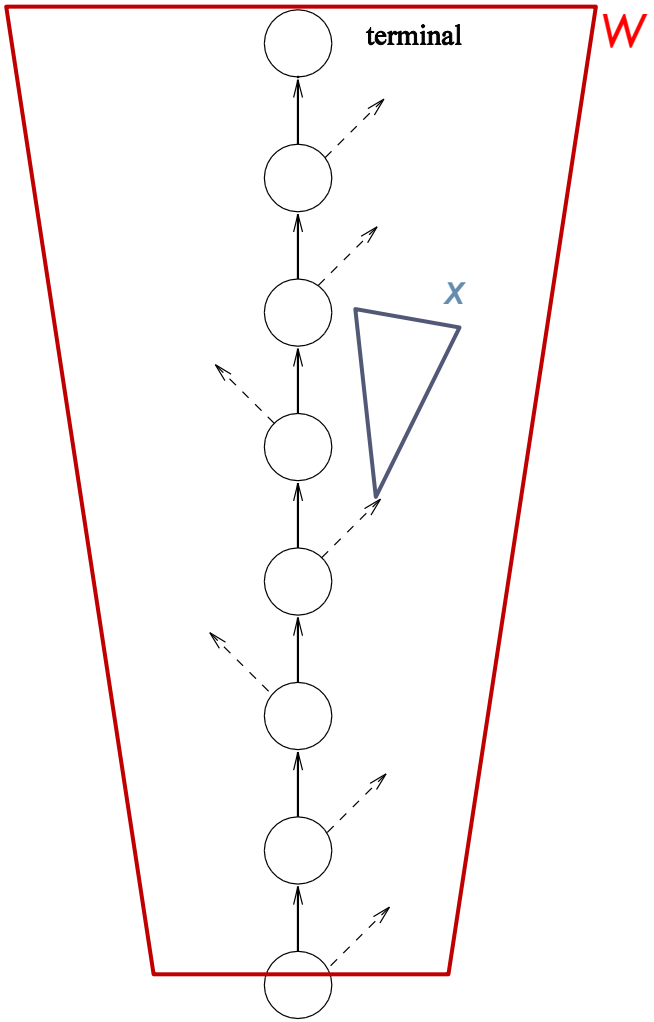
$$\text{size}(v_1) = 7 \qquad z = 5$$

$$\begin{aligned} l_0 &= 1 & r_0 &= 1 \\ l_1 &= 4 & r_1 &= 1 \\ l_2 &= 4 & r_2 &= 3 \\ l_3 &= 5 & r_3 &= 3 \\ l_4 &= 5 & r_4 &= 3 \end{aligned}$$



Search in Heavy Path Forest

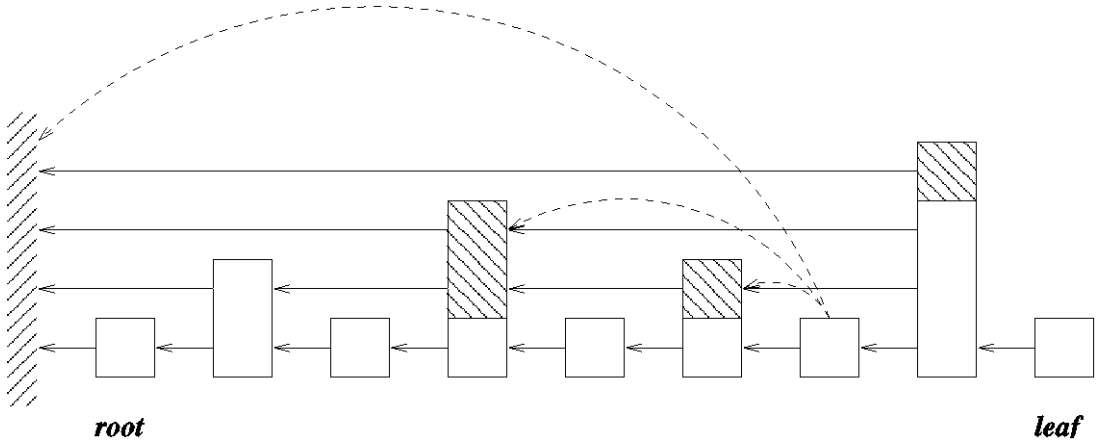
- Search $O(\log N)$ times in L+R "prefix sums" of heavy paths, to find the "light child" where the answer lies.



Biased Ancestor Search

- **Input:** rooted tree with m nodes and (integer, poly) weights on vertices.
- **Query:** given vertex p and weight x find lowest ancestor q s.t. sum of weights from p to q exceeds x.
  - Running time: $O(\log(W/w_q))$
    - W = sum of weights from p to root. w_q = weight of q.
- Two new data structures:
  - $O(m\ \alpha(m))$ space in the pointer machine model.
    - $\alpha(m)$ is an inverse of Ackermann's function.
    - "interval-biased" search tree.
  - $O(m)$ space in the RAM model.
    - "biased skip list" [Bagchi, Buchsbaum, Goodrich, Algorithmica 2005]
    - generalized to trees.
    - space kept to linear by use of succinct data structures.

Biased Skip List



- Initial height: $\log w_v$, promote amortized $O(1)$ times.
- Generalizes to tree; each node-root sub-path is a BSL.
  - BSL supports finger search – we can't use that result.
  - leaf-root path is a BSL (amortized linear space), tree space non-linear?
- Succinct tree of $O(m)$ bits to follow height-l pointers.
  - $h = O(\log N)$ => $O(m \log N)$ bits => $O(m)$ words!

Summary of Results

- Main Theorem:
  - Given a string S of length N as an SLP with m instructions, can represent it so that the ith symbol of S can be randomly accessed in $O(\log N)$ time using:
    - $O(m\ \alpha(m))$ space and preprocessing in the pointer machine model.
    - $O(m)$ space and preprocessing in the RAM model.
- Decode j consecutive symbols:
  - $O(\log N + j)$
- New results for approximate pattern matching in SLP-compressed texts.
- Tree operations in $O(\log N)$ time and $O(m)$ space.

Open Problems

- There are two parameters:
  - m (size of SLP), N (size of string)
  - $N = \Omega(m)$, $N \leq 2^m$
- Space bound is not tight:
  - $O(m)$ words = $O(m \log N)$ bits
    - SLP takes $O(m \log m)$ bits.
  - Of course, not succinct…
- Lower bounds (Chen, Verbin, Yu, 2011)
  - $\Omega(m^{1/2-\varepsilon})$, $O((m \log m)^{1/2})$ upper bound in cell probe model.
  - $\Omega(\log N / \log \log N)$
  - Upper bounds that depends on m and N?

Space-efficient representations of DAGs and BDDs

Maruyama, Nakahara, Kishiue, Sakamoto (SPIRE 2011)

Hansen, Rao, Teidemann (ECAI 2008)

Binary DAGs

- Binary DAG
  - all nodes except sink have out-degree 2, each child labelled L or R.
- Previous result can be viewed as evaluating a function f $[1..n] \rightarrow \Sigma$ (consider the string $f(0), f(1), \ldots, f(n-1)$).
  - a bit like BDD's?
  - does improving running time from $O(h)$ to $O(\log N)$ matter? Can we get $o(h)$?
- We now consider representations that focus on the DAG itself.

Basics

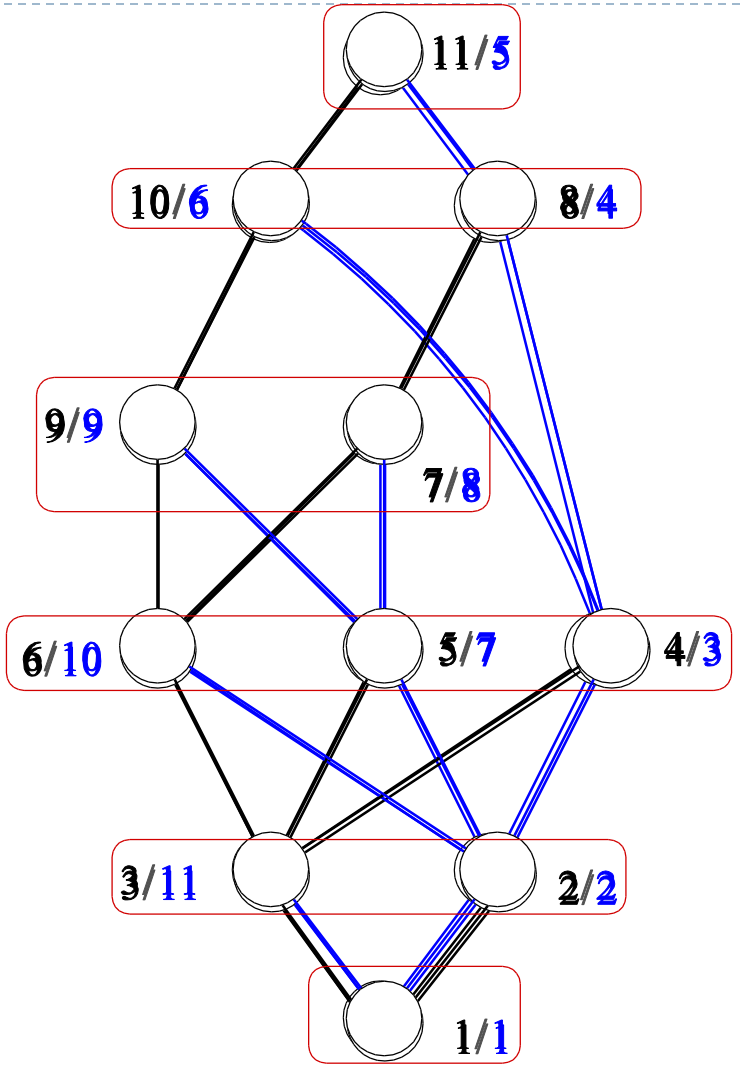
- Given static sequence of bits $X = x_1, \ldots, x_n$ support:
  - $\text{rank}_1(i)$ – number of **1**s in $x_1, \ldots, x_i$
  - $\text{select}_1(i)$ – position of i^{th} **1**.

  X = **0 1 1 0 1 0 0 1**

  $\text{rank}_1(4) = 2, \text{select}_1(4) = 8$
  - Rank/Select in $O(1)$ time using $n + o(n)$ bits (Clark 1996).
- Given permutation π on $[1..n]$, support $\pi(i)$ and $\pi^{-1}(i)$.
  - Can be done in $(1 + \varepsilon)\, n \log n$ bits to support $\pi(i)$ in $O(1)$ time and $\pi^{-1}(i)$ in $O(\varepsilon^{-1})$ time (MRRR, 2003), compressed (BN,2009)
- Given binary tree on n nodes, support navigation.
  - Navigation in $O(1)$ time and $2n + o(n)$ bits (Jacobson 1989).

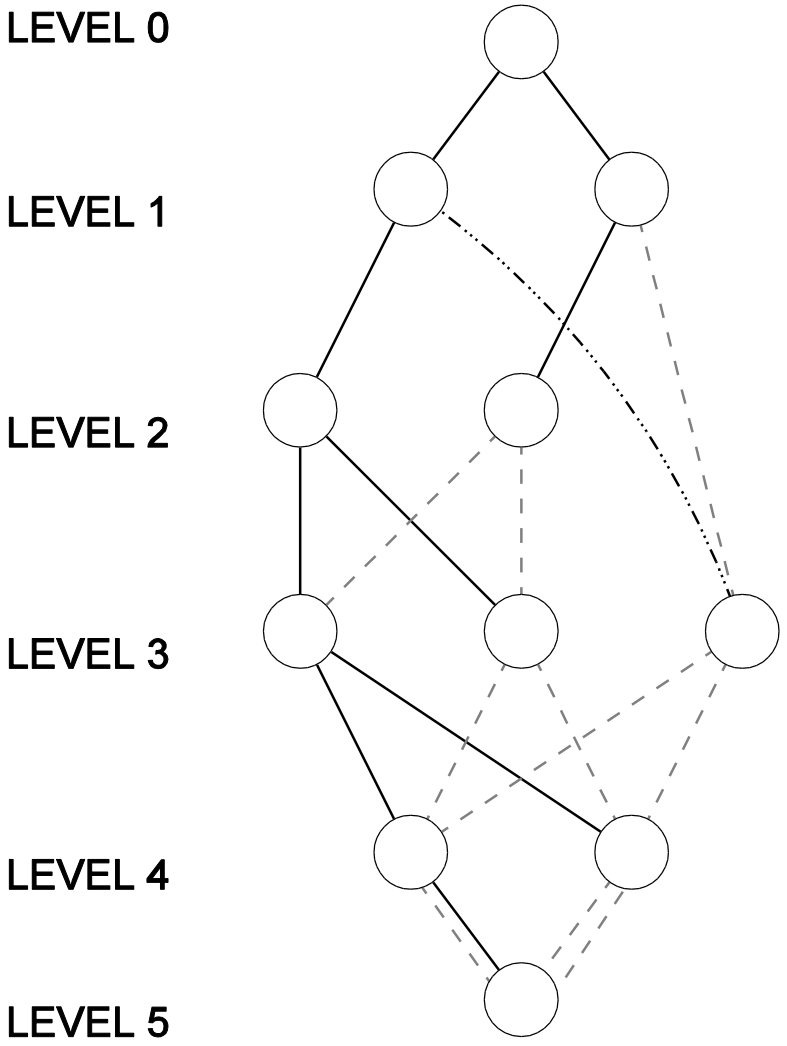
Navigating Binary DAGs (Maruyama et al.)

- Partition given binary DAG into right tree and left tree.
- Represent each tree in $2n + o(n)$ bits
- Numbering not the same!
- Store mapping between the two numberings
- ~ $(1+\varepsilon)\, n \log n + 4n$ bits, navigation in $O(1/\varepsilon)$ time
- BDD? Level info?



BDD compression (Hansen et al.)

- BDD: every node in a layer, and edges only from lower layer to higher.
- Encode a spanning tree (note: "long edges")
- Encode non-tree edges
  - worst-case space seems to be ~ $2n \log n$ bits.
  - reported: 1-2 bits per node
  - navigation? model?



Succinct Representations of Posets

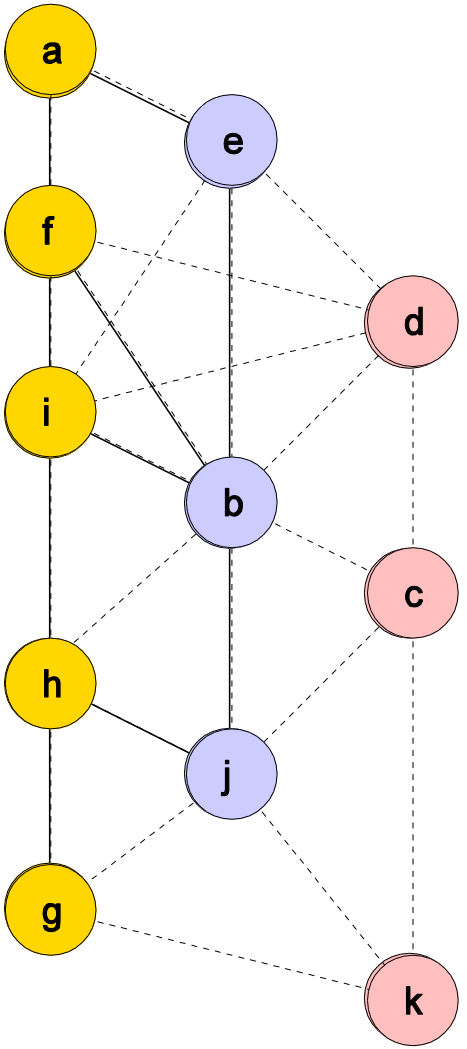
Farzan, Fischer (ISAAC 2011)

Problem

- Given poset $(P, <)$, $|P| = n$.
- Equivalent to DAG $G = (P, E)$
- Answer reachability queries:
  - given x, y in P, is y reachable from x?
- Uninteresting for arbitrary DAGs: $\Theta(n^2)$ bits
- DAGs/posets of small *width* w.
  - width of poset: length of longest antichain.
  - but there are posets of width 1 with size $\Theta(n^2)$ bits.
- Encode the *transitive reduction* G_r of G – the subgraph of E with the fewest edges but same reachability.
  - Shown that width w DAGs/posets require ~ $2nw + n \log n$ bits.
  - Adjacency query: given x, y in P is (x, y) an edge in G_r

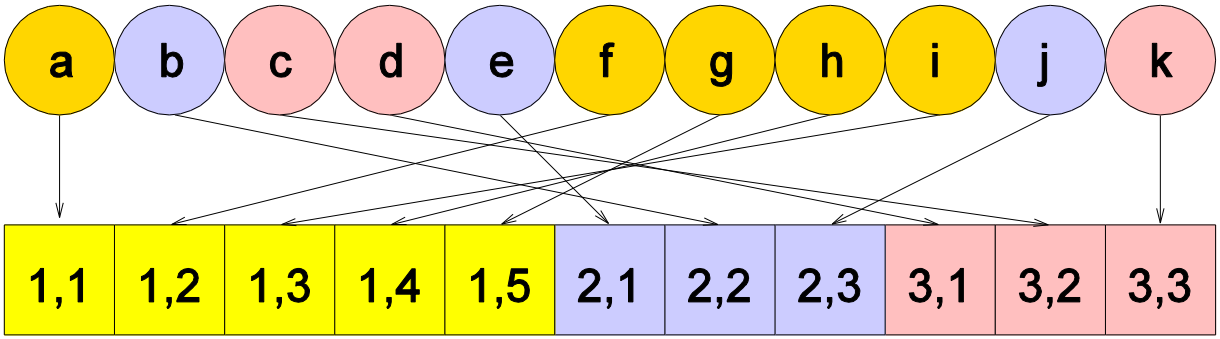
Representing Posets

- decompose poset of width *w* into w chains (Dilworth's Thm)
- Store "merge" info:
  - 0 1 00 1 0 1 0
  - rank/select: find position of element in one chain in the other.
- Number nodes (chain#, chain pos), e.g, b = (2, 2)
- Repeat for all: $2nw + o(nw)$ bits.
- reach(d, g): find d = (3,1), g = (1, 5), find rank of d in chain 1.



Posets, concluded

- Need to store mapping from node names to node numbers:
  - permutation data structure, needs $(1+\varepsilon)\, n \log n$ bits
  - Total $(1+\varepsilon)\, n \log n + 2nw$ bits
    - $n \log n + 2nw$ lower bound



$succ_G(x)$, $pred_G(x)$	$O(w + k)$
$adj_{Gr}(x,y)$	$O(w)$
GUB(x,y)	$O(w^2 + k)$

Conclusion

Problem

- Potentially interesting problems in tree-like objects.
  - Techniques usually based on decomposing the object into trees or other linear objects.
  - List of operations efficiently supported is still small.
  - Succinct space bounds not always achieved.
- Many applications of such objects in data mining, constraint satisfaction, verification etc.
  - Have to deal with large objects, in-memory processing.
  - sCUDD library?