An Improved Algorithm to Accelerate Regular Expression Evaluation

Michela Becchi and Patrick Crowley

ANCS 2007

Context

- *Regular expression matching* is a critical operation in networking
  - » Intrusion detection
  - » Context based billing
  - » Peer-to-peer traffic detection and prioritization
  - » Application level filtering
- Challenge: perform regular expression matching at line rate, given data-sets of hundreds (or thousands) of patterns
  - » Processing time
  - » Memory requirement (occupancy and bandwidth)

Background & Problem definition

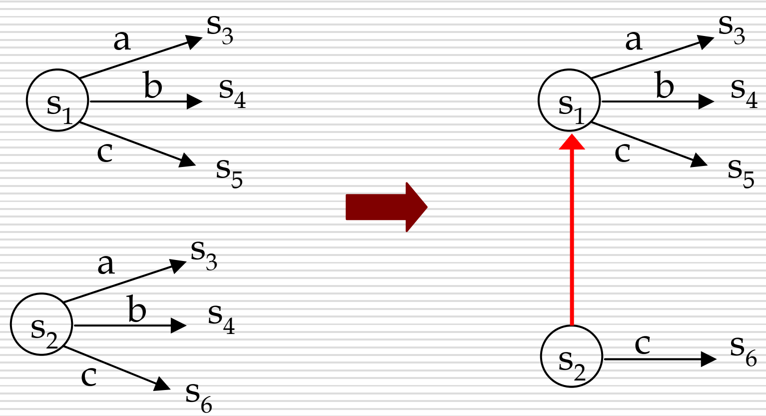
- Two algorithmic solutions
  - » Non deterministic finite automata (NFAs)
    - – High time complexity/memory bandwidth requirements
    - – Compact representation
  - » Deterministic finite automata (DFAs)
    - – Low time complexity
    - – Potentially high storage requirement

- Multiple implementation approaches
  - » FPGA [Sidhu 2001, Clark 2003, Moscola 2003]
  - » Software [Paxson 1998, Roesch 1999, Tuck 2004]
  - » Custom hardware [Kumar 2006]

- Problem: given a DFA, find a representation
  1. compact
  2. allowing an acceptable bound of memory bandwidth requirement/processing time

Background - D^2FA

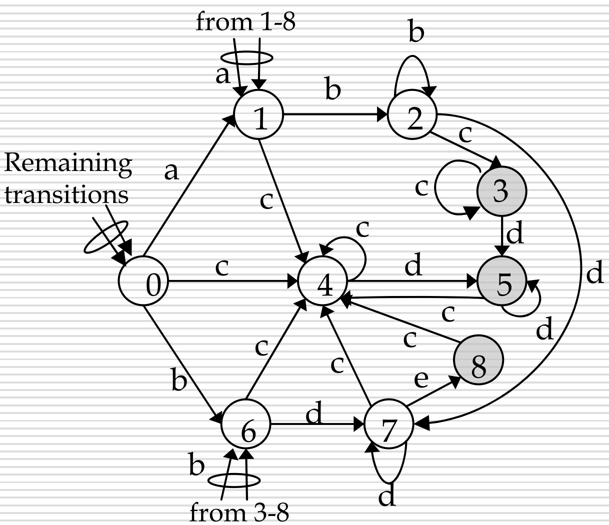
- Observation:
  - » DFAs from practical datasets have redundancy in state transitions
- Idea:
  - » *default transitions:* non-consuming transitions



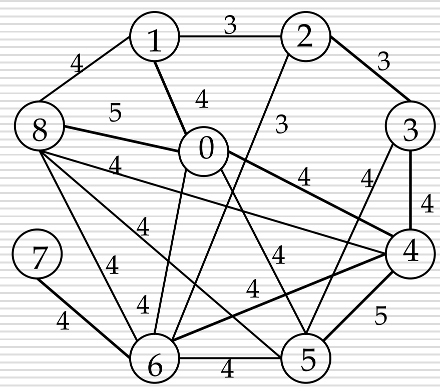
- Implication:
  - » Traversal time / memory bandwidth requirement dependent upon maximum default path length

Background – D^{2}FA construction

RegEx: ab^+c^+, cd^+ and bd^+e

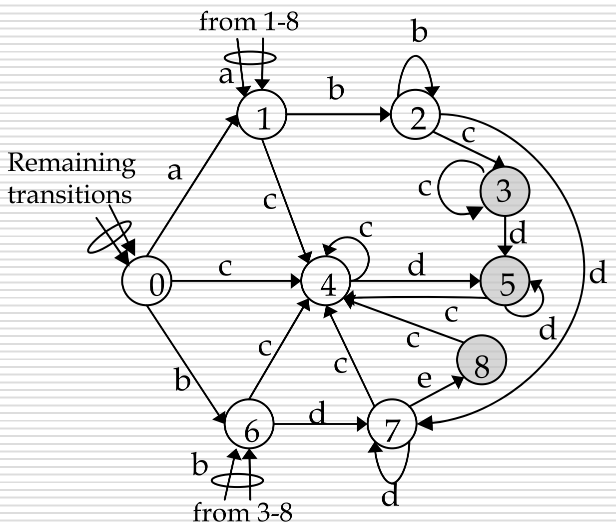


Space reduction graph

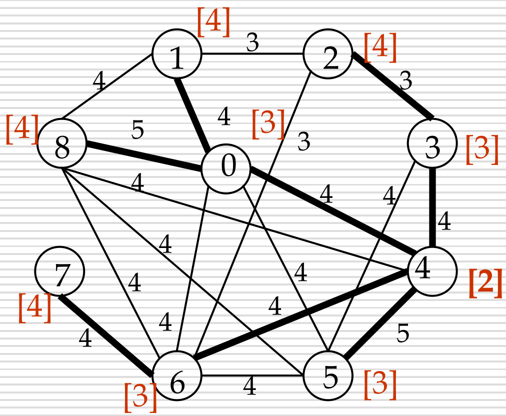


Background – D²FA construction

RegEx: *ab⁺c⁺*, *cd⁺* and *bd⁺e*



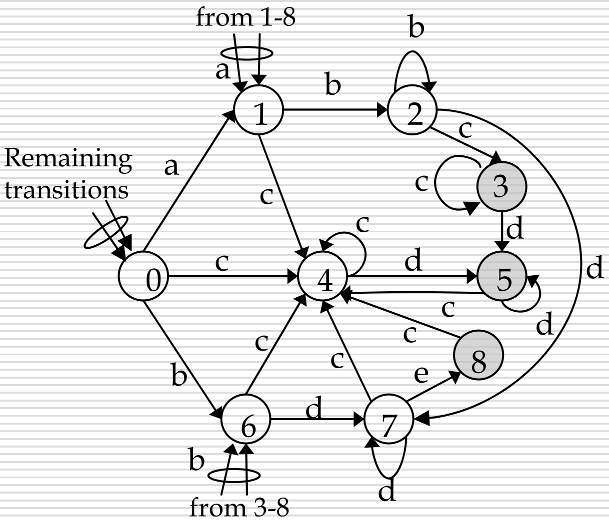
Space reduction graph



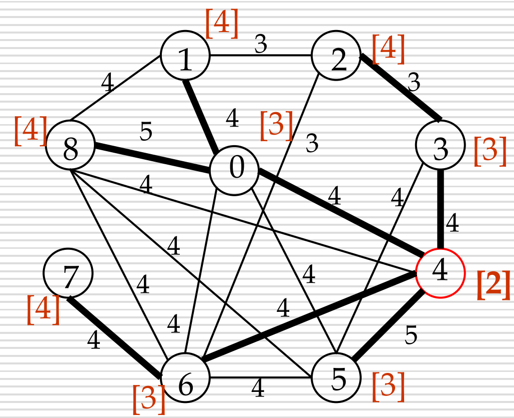
Diameter bound=4 ⇨
removed transitions=33

Background – D²FA construction

RegEx: *ab⁺c⁺*, *cd⁺* and *bd⁺e*

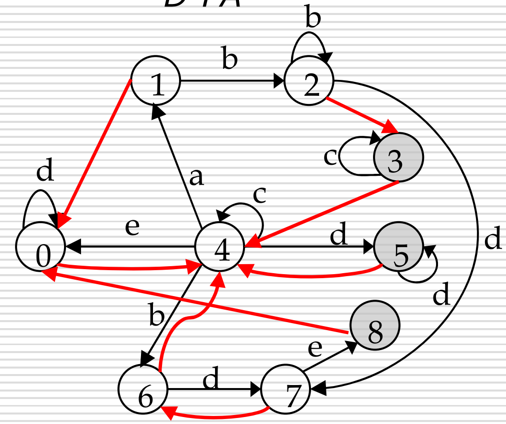


Space reduction graph



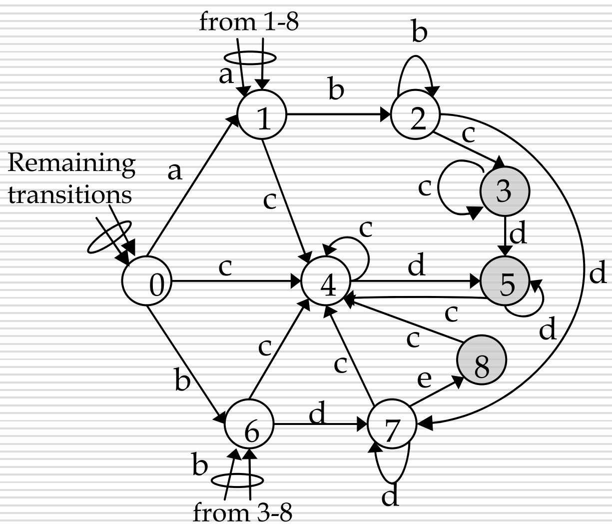
Diameter bound=4 ⇨ removed transitions=33

D²FA

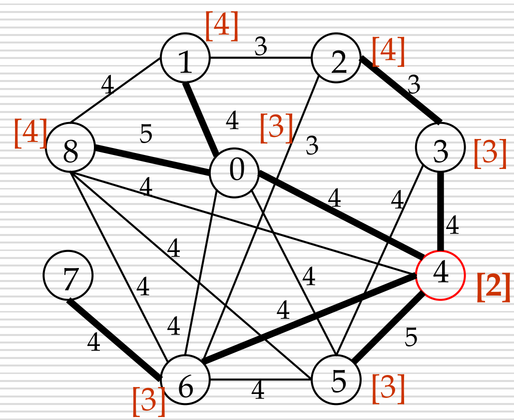


Background – D^{2}FA construction

RegEx: *ab$^+$c$^+$*, *cd$^+$* and *bd$^+$e*

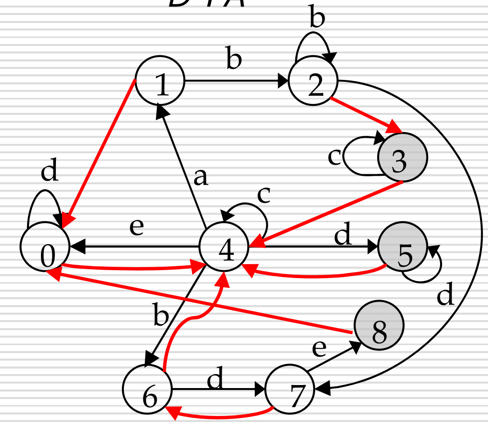


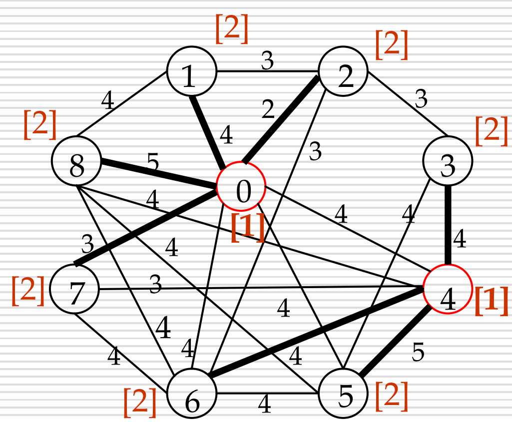
Space reduction graph



Diameter bound=**4** ⇨
removed transitions=33

D^{2}FA



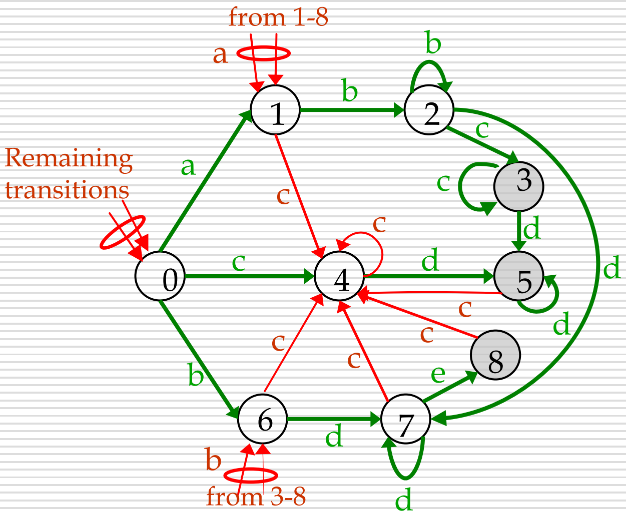


Diameter bound=**2** ⇨
removed transitions=28

Traversal time=O((D/2+1)N)
Time complexity=O(n^{2}logn)
Space complexity=O(n^2)

Transition redundancy: why?

RegEx: ab^+c^+, cd^+ and bd^+e

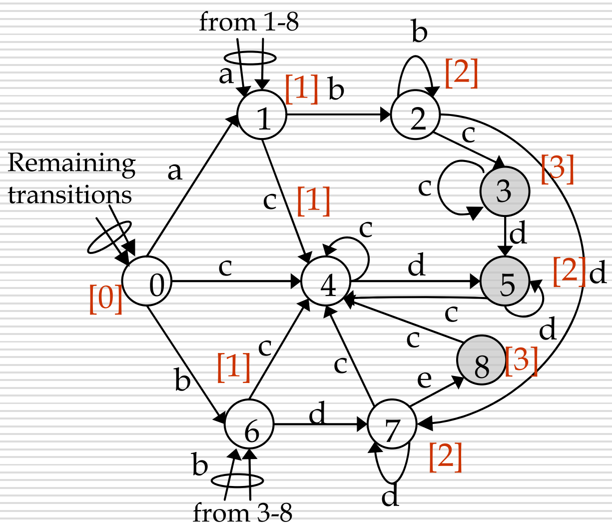


- **Forward** transitions:
  - » Matches
  - » State specific
- **Backward** transitions:
  - » Mismatch
  - » Shared by multiple states

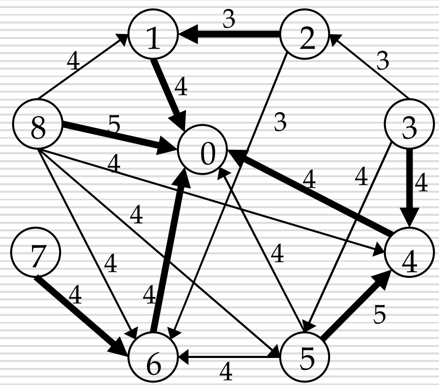
- Idea:
  - » Introduce *state depth:* minimum distance from entry state
  - » Orient default transitions only backwards (towards decreasing depth)
- Pros:
  - » Traversal time **O(2N)** *independent of the maximum default path length*
  - » Generality: no need of diameter bound parameter
- Cons:
  - » Possible compression loss

Our scheme

RegEx: *ab⁺c⁺*, *cd⁺* and *bd⁺e*

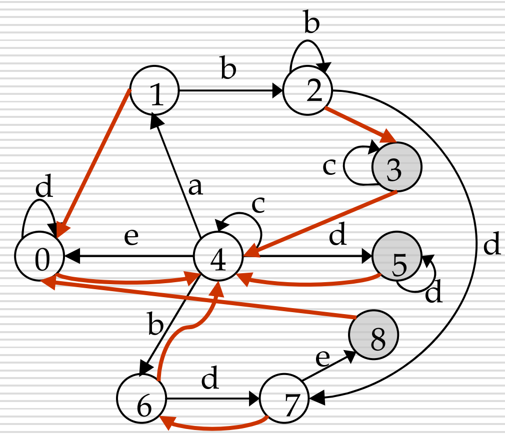


Oriented space reduction graph



NO diameter bound
Time complexity=O(2N)
removed transitions=33

Reduced graph



- **Observations:**
  - » Maximum spanning tree on oriented graph: Edmonds and Chu solutions
    - – 2 steps: edge selection and cycle resolution
  - » No cycles:
    - – Space reduction graph not necessary
    - – Simple breath-first traversal algorithm

Traversal time=O(2N)
Time complexity=O(n^2)
Space complexity=O(n)

Discussion

- Generalization (Jon Turner's observation)
  - » Allowing default transitions only from depth *d* to depth ≤*d*-*k*, w/ *k*≥*1*, leads to worst case traversal time $O\left(\frac{k+1}{k}N\right)$
    - – Time and space complexity of the construction algorithm still $O(n^2)$ and $O(n)$
    - – Examples:
      - *k=1* ⇨ *traversal O(2N)*
      - *k=2* ⇨ *traversal O(1.5N)*
      - *k=3* ⇨ *traversal O(1.33N)*
      - *k=4* ⇨ *traversal O(1.25N)*

- Compression
  - » D²FA:
    - – Constraint: diameter bound
    - – Heuristic
  - » Our algorithm:
    - – Constraint: orientation (may be not a problem for RegEx originated DFAs)
    - – Optimal solution

Discussion (cont'd)

- Default transitions and depth computation through breath-first traversal
  - Default transitions can be computed <u>during subset construction</u>, that is, <u>at DFA creation time</u>.

- D^2FA space complexity can be an issue for big DFAs
  - *$O(n^2)$:* space reduction graph
  - Using adjacency list 17B/edge

```
struct wgedge {
  vertex    l,r;      // endpoints of the edge
  weight    wt;       // edge weight
  edge      lnext;    //link to next edge incident to l
  edge rnext;         //link to next edge incident to r
} *edges;
```

  - Fully connected graph w/ ~11K nodes will require 1GB storage
  - Possible solutions: partial graphs based on weight
    - Multiple scans
    - Effect on algorithm's execution time

Discussion (cont’d)

- Traversal locality
  - » DFA traversal exhibits locality
  - » Average traffic tends to mismatch
  - » States at low depths tend to be traversed more
  - » Backward default transition reiterate the traversal of likely states

Alphabet reduction

- Observation:
  - Some symbols are treated in the same way over the whole DFA [$\delta(s,c_i)= \delta(s,c_j)$ for each state s ϵ DFA]
  - Example:
    - Ignore case
    - \n, \r
    - unused characters
- Idea:
  - Group characters into classes
  - Mapping filter
- Algorithm:
  - Sequence of clustering operations
  - Breath-first traversal w/ $O(n^2)$ complexity
  - Applicable at DFA creation time

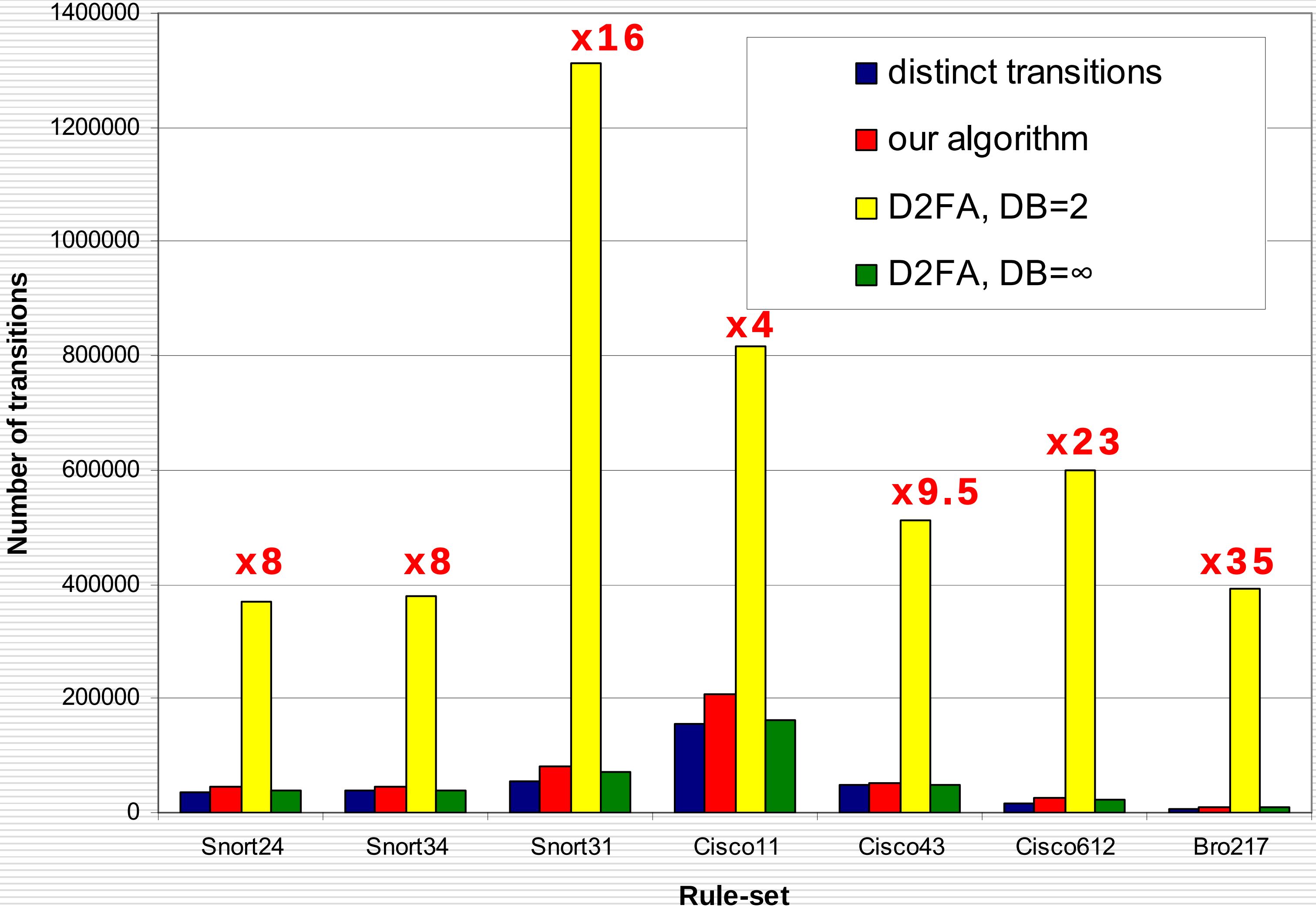
Evaluation: rule-sets

Data-set	ASCII length range	% RegEx w/ wild-cards (*,+)	% RegEx w/ char ranges ≥ 5
Snort24	6..70	37.5	50
Snort34	15..99	38.2	32.4
Snort31	16..120	41.9	93.5
Cisco11	9..13	90.9	9.1
Cisco43	15..73	32.6	27.9
Cisco612	3..50	0	1.6
Bro217	5..76	1.4	13.4

Evaluation - compression

Dataset	Original DFA		D^2FA algorithm						Our algorithm	
			Compression (as a function of the diameter bound)							
	# of states	*% duplicates*	*DB=2*	*DB=6*	*DB=10*	*DB=14*	*DB=∞*	*max def. length*	*Compression*	*max def. length*
Snort24	13886	98.97	89.59	98.48	98.91	98.92	98.92	16	98.71	12
Snort34	13825	98.91	89.33	98.48	98.85	98.86	98.86	16	98.69	10
Snort31	20052	98.93	74.42	97.18	98.42	98.6	98.63	13	98.44	6
Cisco11	24011	97.45	86.73	97.08	97.37	97.38	97.38	12	96.63	8
Cisco43	20320	99.06	90.16	98.46	99	99.05	99.05	14	98.97	8
Cisco612	11309	99.5	79.3	97.46	98.93	99.18	99.25	12	99.09	5
Bro217	6533	99.57	76.49	97.9	99.07	99.4	99.41	9	99.33	9

Evaluation – number of transitions



Alphabet reduction's effect

Dataset	**# of nodes**	**alphabet size**	**D^2FA, DB=2** *compression %* *BAR*	*AAR*	*transitions after AR*	**D^2FA, DB=∞** *compression %* *BAR*	*AAR*	*Trans. after AR*	**Our algorithm** *Compression %* *BAR*	*AAR*	*Trans. after AR*
Snort24	13886	46	89.59	97.87	75752	98.92	99.49	18095	98.71	99.4	21504
Snort34	13825	51	89.33	97.63	84046	98.86	99.47	18856	98.69	99.43	20342
Snort31	20052	53	74.42	94.48	283339	98.63	99.21	40347	98.44	99.13	44819
Cisco11	24011	38	86.73	97.74	138922	97.38	99.24	46689	96.63	99.09	55955
Cisco43	20320	65	90.16	97.09	151161	99.05	99.31	36037	98.97	99.27	37784
Cisco612	11309	115	79.3	90.46	276110	99.25	99.33	19316	99.09	99.2	23139
Bro217	6533	111	76.49	89.59	174035	99.41	99.43	9526	99.33	99.34	10957

Further decreasing the traversal time

Rule-set	# of states	% of dupli-cates	D^2FA		Our algorithm			
			DB=2	DB=∞	k=1	k=2	k=3	k=4
Bro217	6533	99.57	89.59	98.92	99.33	97.61	91.74	84.57
Cisco11	24011	97.45	89.33	98.86	96.63	81.92	69.08	56.74
Cisco43	20320	99.06	74.42	98.63	98.97	97.15	92.03	87.19
Cisco613	11309	99.5	86.73	97.38	99.09	98.23	94.5	88.38
Snort24	13886	98.97	90.16	99.05	98.71	95.42	90.66	85.82
Snort34	13825	98.91	79.3	99.25	98.69	95.45	91.85	88.13

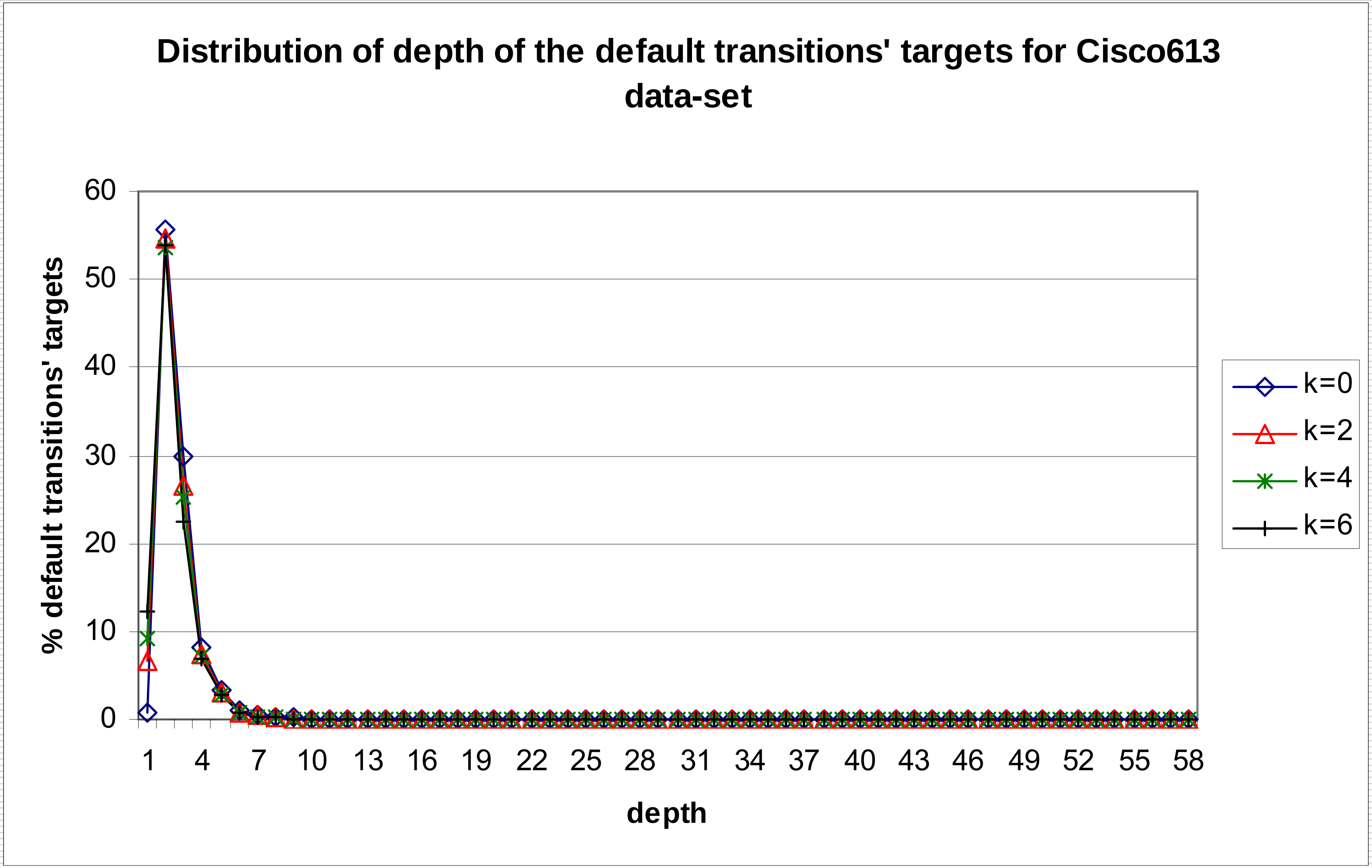
Conclusion

- DFA exhibit transition redundancy exploitable through default transitions
- D^2FA: algorithm trading off compression w/ traversal time
- In this work, we propose generic algorithm:
  - » With limited time and space complexity
  - » Allowing O(2N) traversal time (or less) when processing input text
  - » Leading to compression level similar to (or better than) D^2FA

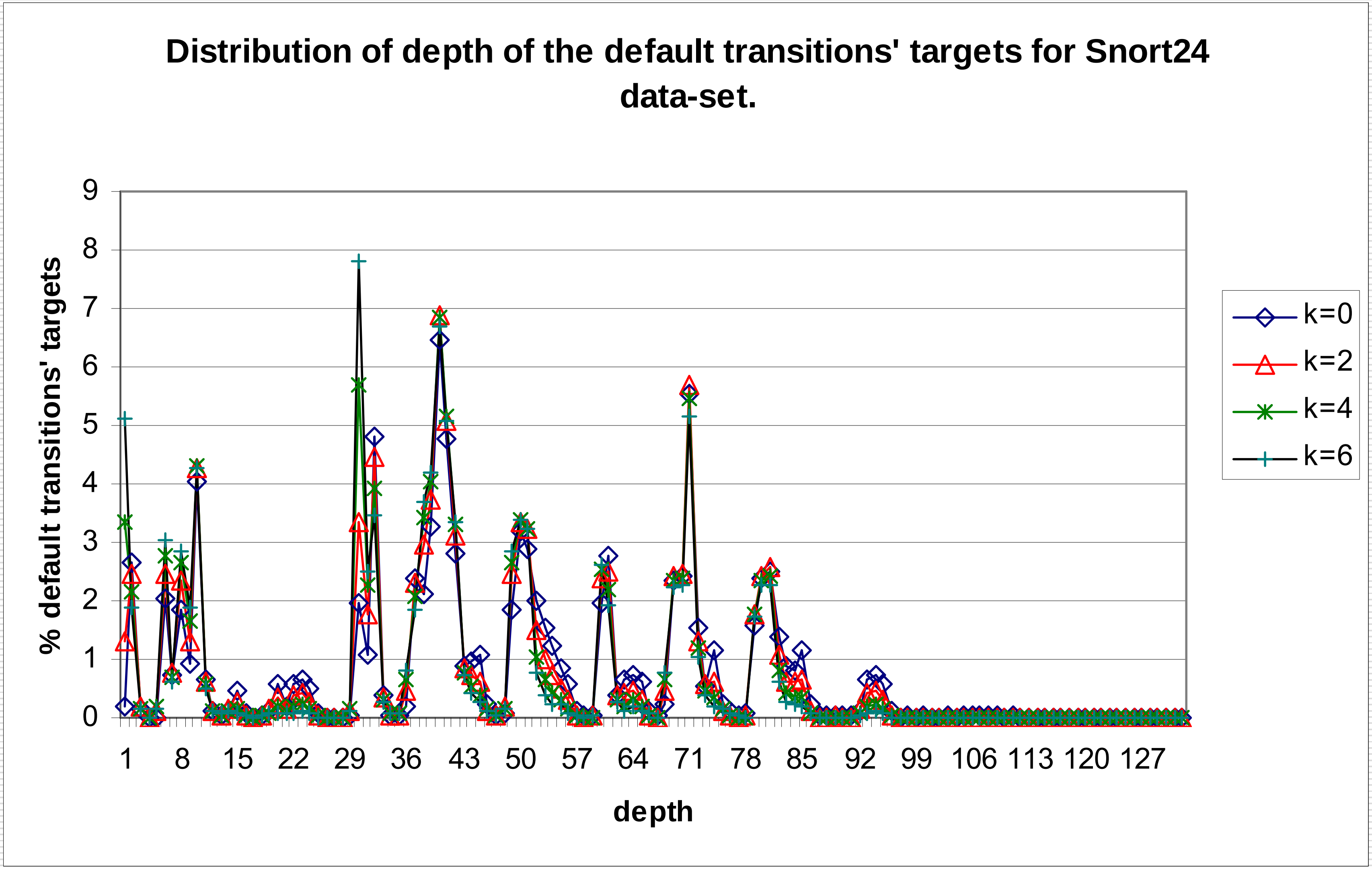
Thank you!

Questions?

Default transitions targets



Default transitions targets (cont'd)



Our algorithm

```
procedure default_transition (DFA dfa=(n, δ(states, Σ)),
                                   modifies set default);
    list queue; set depth[n];
    for state s ∈ states  ⇒ depth[s]=n; default[s]=s; rof
    depth[0]=0;queue.push(0);
    while (!queue.empty())⇒
      state s= queue.pop();
      int saving=0;
      for char c ∈ Σ ⇒
       if (depth[δ(s,c)]=n) ⇒
          depth[δ(s,c)]= depth[s]+1; queue.push(δ(s,c));
       fi
      rof;
      for (state t ∈ states & depth[t]<depth[s]) ⇒
          int common:=# common transitions btw. s and t;
          if (common > 1 &&  (common>saving ||
               (common=saving && depth[t]<depth[default[s]])))
               saving:=common;
               default[s]=t;
          fi
      rof;
    end while;
end;
```