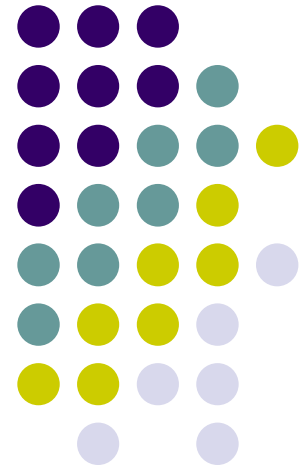


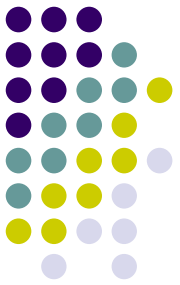
# Forbidden-set Routing

---

Andrew Twigg

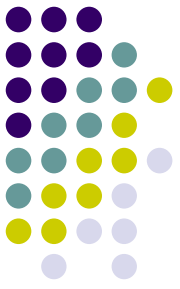
Cambridge University





# Outline

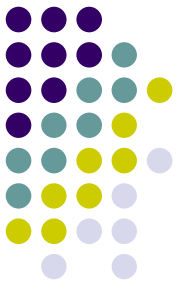
- Motivation: Policy-based routing in large, autonomous networks (eg Internet)
  - “forbidden-set” policies
- Stable routing trees
  - Some intractability results
- Compact routing approach
  - An alternative to routing trees
  - Distance separator labels



# Policy-based routing

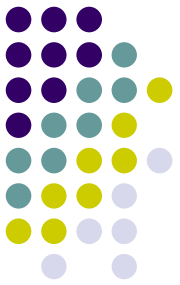
- Generalization of shortest-path routing
  - “Shorter paths are not necessarily better”
- Consider the subjective-cost model
  - Introduced by Feigenbaum, Karger et al.
  - Each node  $u$  has a cost function where  $c_u(v)$  is the cost of routing through node  $v$
- Setting  $c_u(v) \in \{0,1\}$  gives **forbidden-set routing**
  - Each node  $u$  has a forbidden set  $S(u) \subseteq V(G)$
  - The cost to  $u$  of a path  $P$  is

$$c_u(P) = |S(u) \cap P|$$



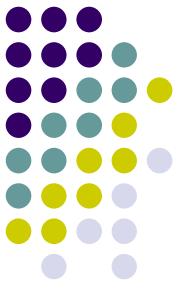
# Forbidden-set policies

- Why forbidden-set?
  - Simple to describe
  - Expressive enough to capture policies that cannot be expressed with next-hop preferences
    - X doesn't want its packets sent through Y or Z
  - Interesting graph-theoretic view
    - $S(u)$  is a  $uv$ -separator iff there is no path  $P$  from  $u$  to  $v$  with  $c_u(P)=0$



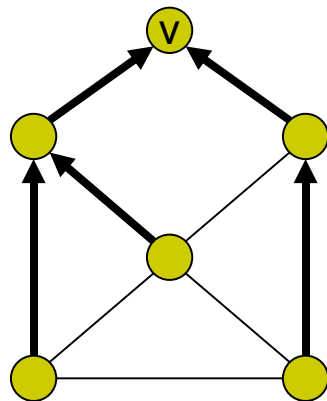
# Outline

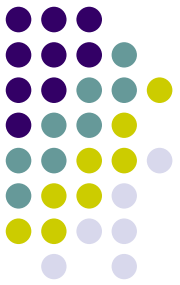
- Motivation: Policy-based routing in large, autonomous networks (eg Internet)
  - “forbidden-set” policies
- **Stable routing trees**
  - Some intractability results
- Compact routing approach
  - An alternative to routing trees
  - Distance separator labels



# Routing trees

- A *routing tree for  $v$*  is a spanning tree rooted at  $v$ 
  - Construct one routing tree for each destination
  - Packets are forwarded to the root of the tree for  $v$
  - Space  $\Omega(n)$  per node
- The set of shortest paths to  $v$  is a routing tree for  $v$ 
  - Easy to construct in a distributed way
- Routing trees used by all known policy routing algorithms





# Stable routing trees

- Stability property
  - A routing tree is *stable* if no node can switch to a lower cost path without creating a cycle
  - **Assumption:** We can only use stable routing trees
    - Why? *Autonomous* nodes will always choose the lowest cost path available
- For shortest-path policies, the shortest paths tree is always stable
  - Every subpath of a shortest path is a shortest subpath
- What about other routing policies?



# Problems with routing trees

- We can only use stable trees
  - For general policies, a stable routing tree may not exist and NP-complete to decide existence [Griffin et al. 02]
- High space requirements
  - $\Omega(n)$  bits per node
- What to do?
  - Use simple policies (e.g. forbidden-set routing)
  - Use simple networks (e.g. small tree width)

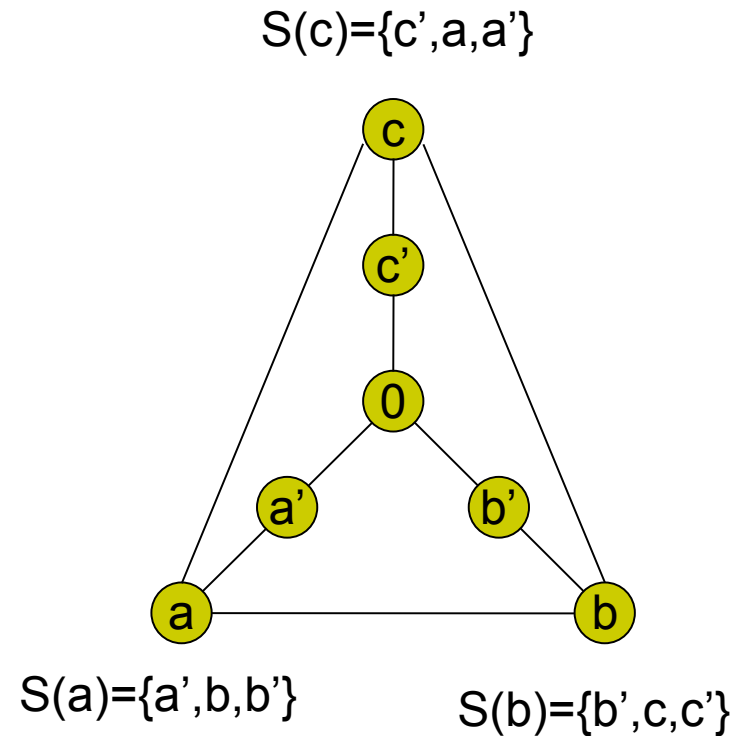
**Theorem:** Deciding if a stable routing tree exists is NP-complete using forbidden-set policies on graphs of tree width 7

→ Routing trees are a bad choice for forbidden-set policies

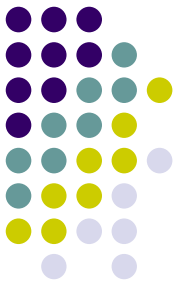
# Forbidden sets and stable trees



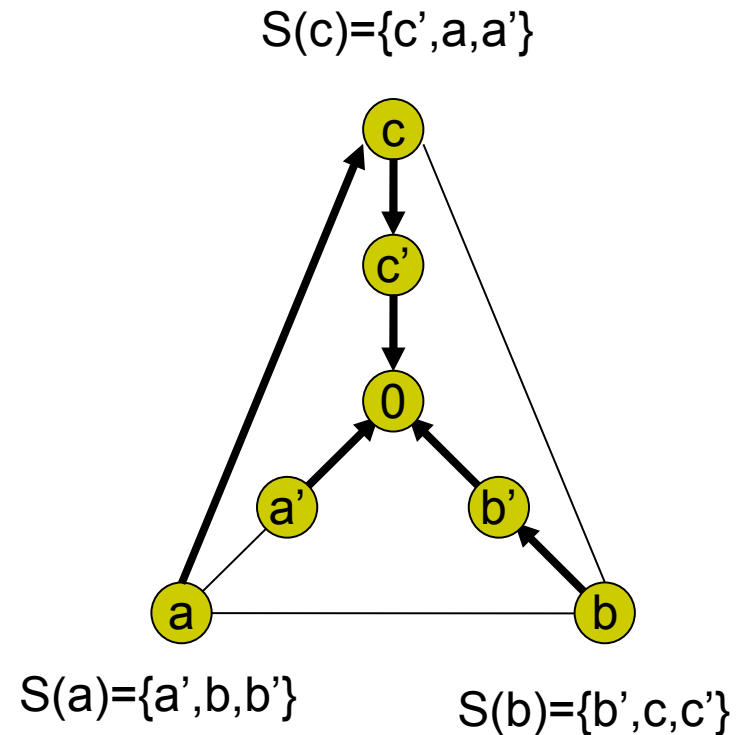
- No stable routing tree rooted at 0



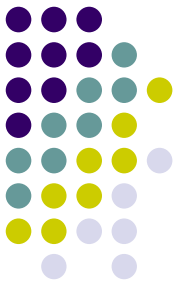
# Forbidden sets and stable trees



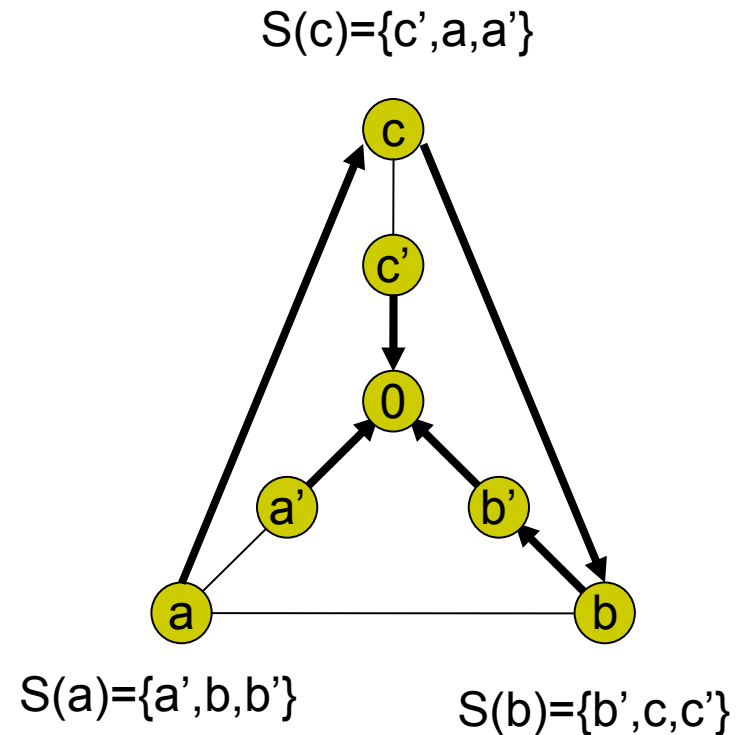
- No stable routing tree rooted at 0
- Why?
  - Pick any routing tree
  - But  $c$  prefers to go via  $b$



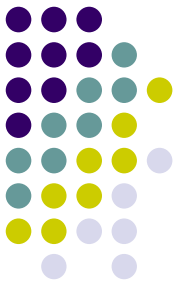
# Forbidden sets and stable trees



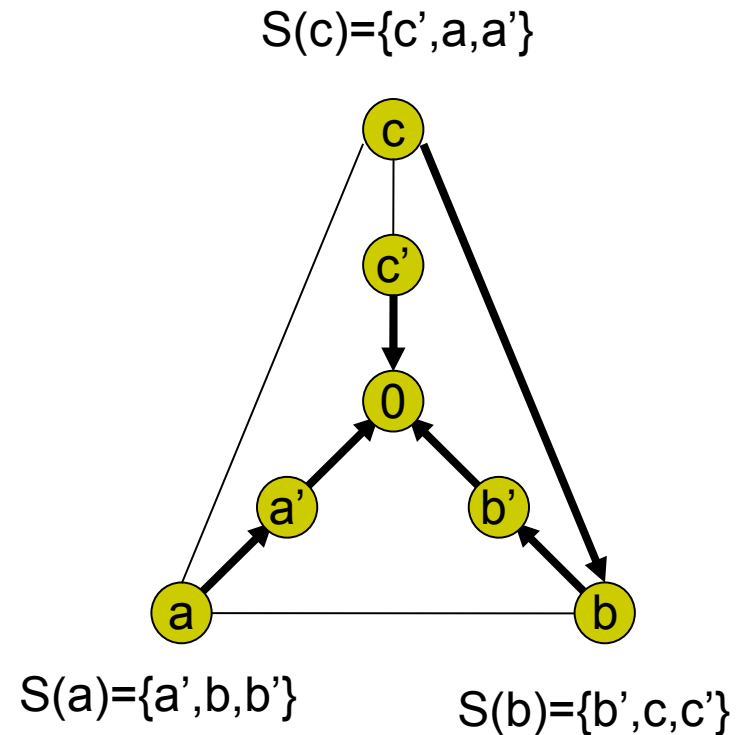
- No stable routing tree rooted at 0
- Why?
  - Pick any routing tree
  - Now  $b$  prefers to go via  $a$
  - Now  $a$  prefers to go via  $a'$

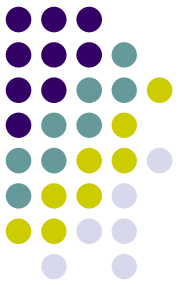


# Forbidden sets and stable trees



- No stable routing tree rooted at 0
- Why?
  - Pick any routing tree
  - Now  $b$  prefers to go via  $a$
  - Now  $a$  prefers to go via  $a'$
  - Now  $b$  prefers to go via  $a$
  - ...

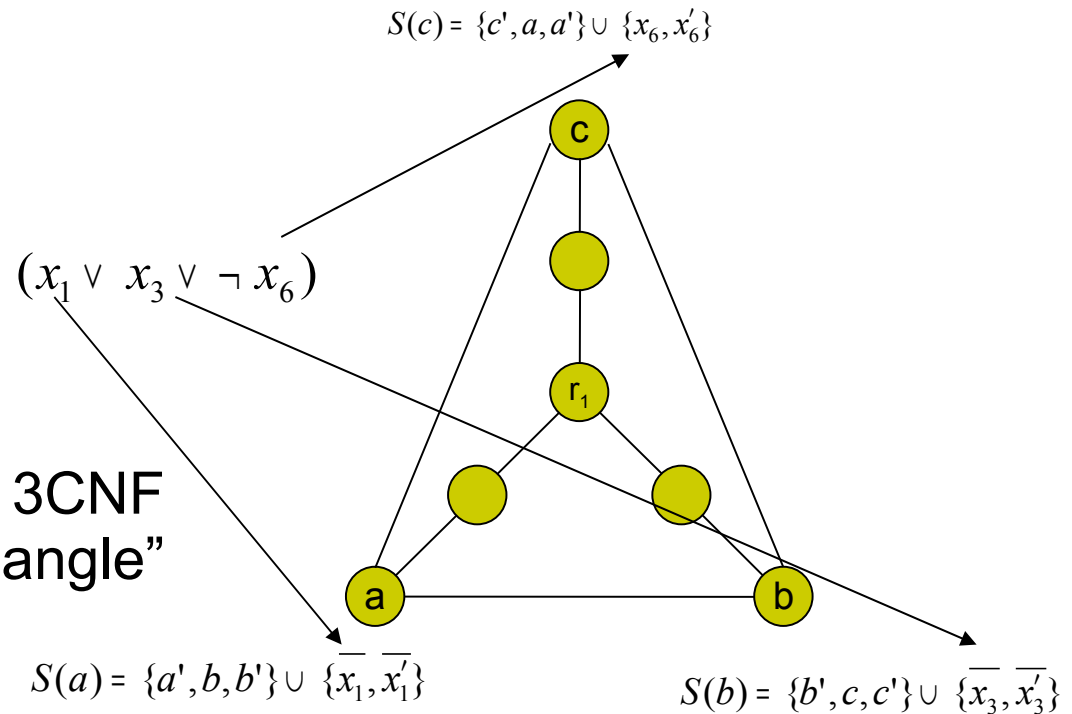


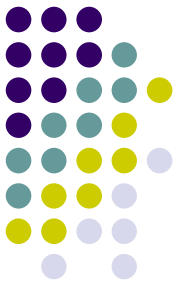


# Proof outline

**Theorem:** Deciding if a stable routing tree exists is NP-complete using forbidden-set policies on graphs of tree width 7

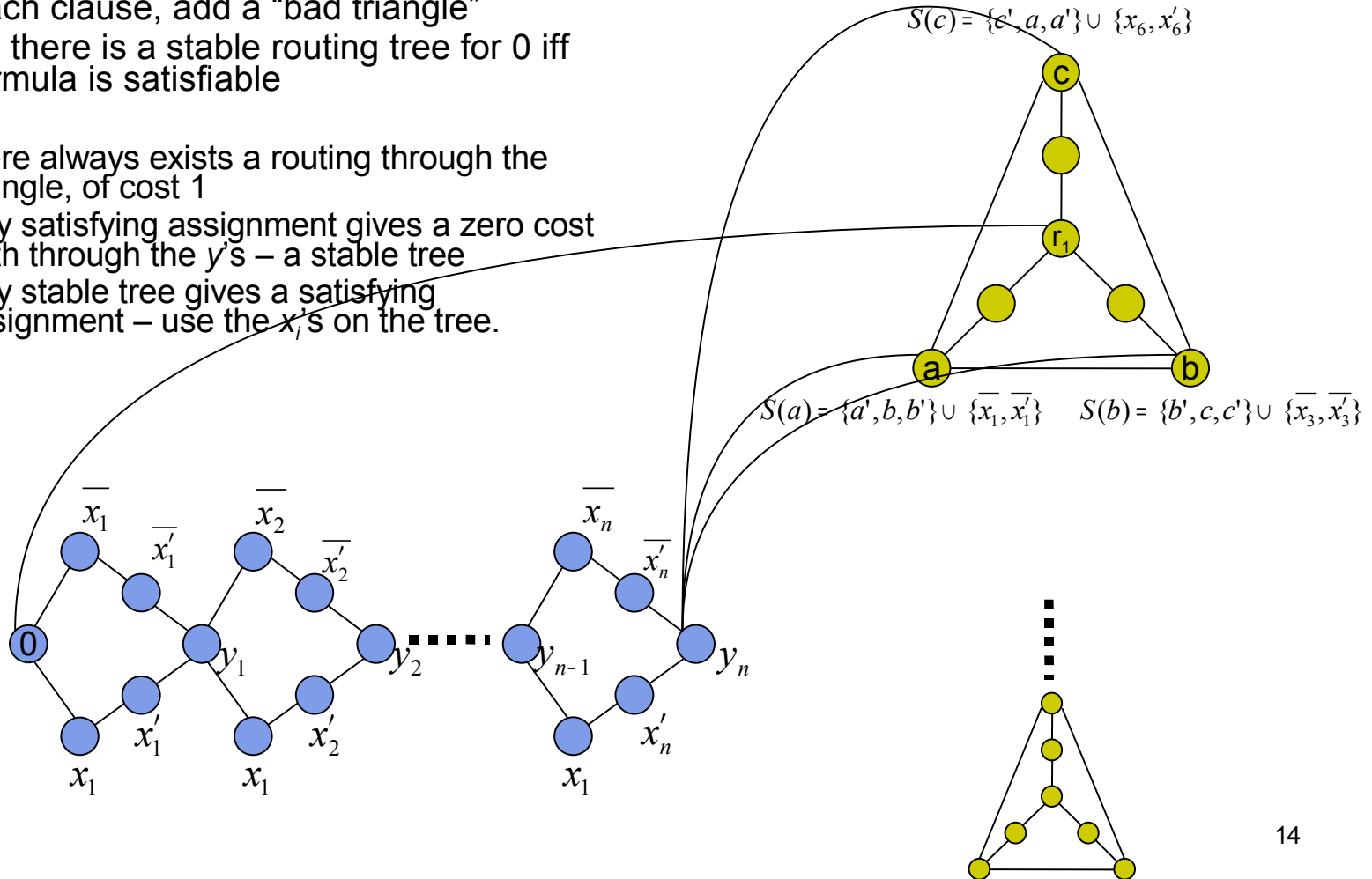
- Encode a clause of a 3CNF formula into a “bad triangle”

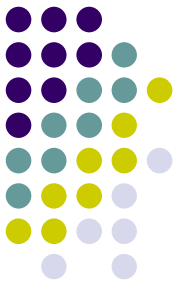




# Proof outline

- Reduction from 3SAT
  - For each clause, add a “bad triangle”
  - Claim: there is a stable routing tree for 0 iff the formula is satisfiable
  - Proof:
    - there always exists a routing through the triangle, of cost 1
    - Any satisfying assignment gives a zero cost path through the  $y$ 's – a stable tree
    - Any stable tree gives a satisfying assignment – use the  $x_i$ 's on the tree.

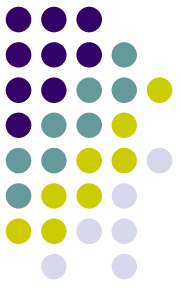




# Outline

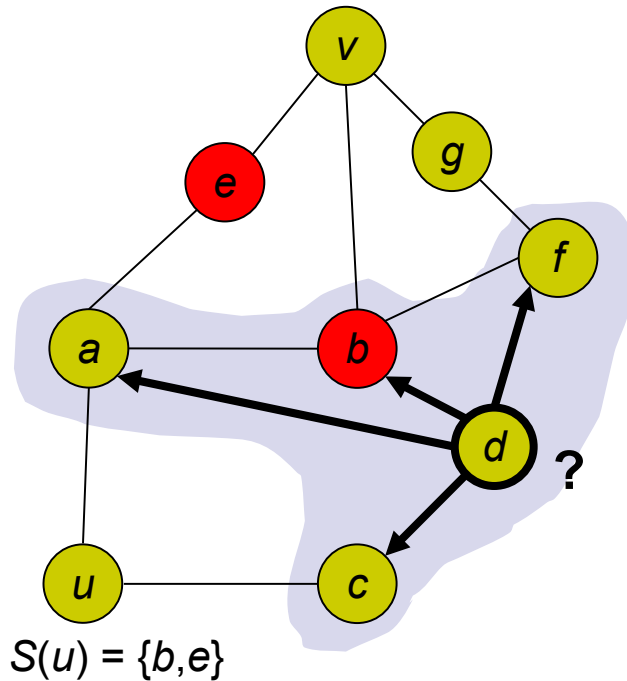
- Motivation: Policy-based routing in large, autonomous networks (eg Internet)
  - “forbidden-set” policies
- Stable routing trees
  - Some intractability results
- Compact routing approach
  - An alternative to routing trees
  - Distance separator labels

# Compact routing

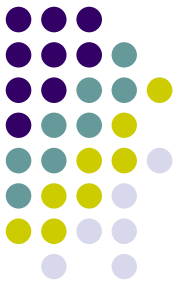


- Terminology
  - Nodes are identified by labels  $L(v)$
  - Edges are identified by port numbers at the nodes
  - Each packet has a header that can store e.g. destination
  - Each router has a **local routing function**
    - Given an incoming packet with destination  $L(v)$ , decide which port it should be forwarded on
  - Goal: Small labels ( $\text{polylog}(n)$ ) and small routing tables ( $o(n)$ )
- Compact routing known to be almost-optimal for shortest-path routing [Cowen, Thorup-Zwicky]
  - Stretch-3 routing scheme using  $\tilde{O}(n^{1/2})$ -bit tables
  - Routing trees use space  $\Omega(n)$

# Distance separator labels

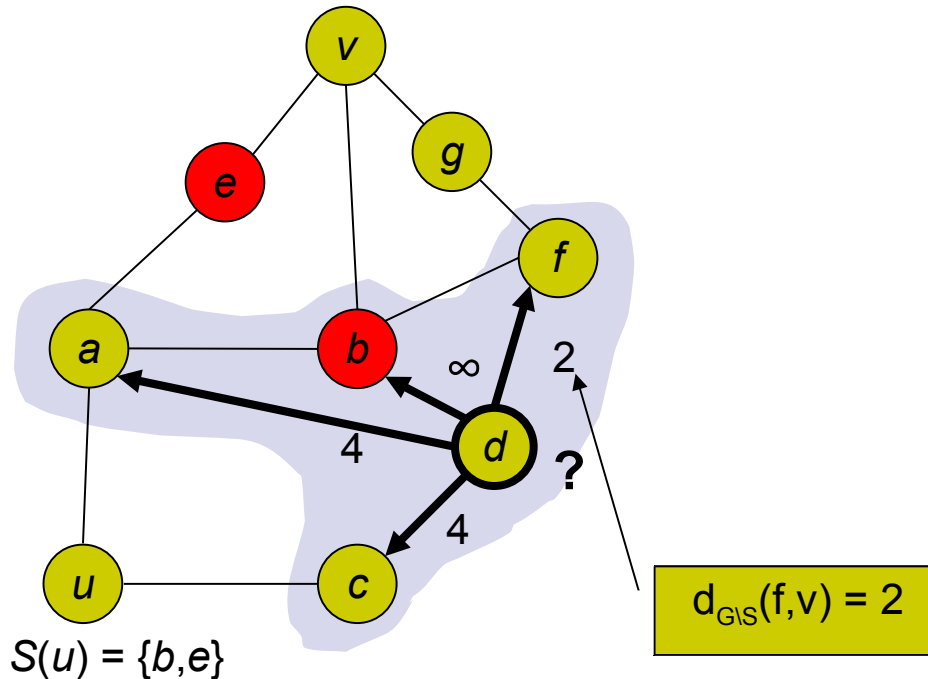


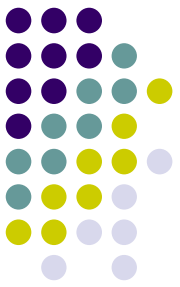
- Consider a packet arriving at  $d$  from  $u$  with destination  $v$
- Where should  $d$  forward it to?
- If  $d$  sends it to  $a$  or  $c$  then the packet must return to  $d$  later



# Distance separator labels

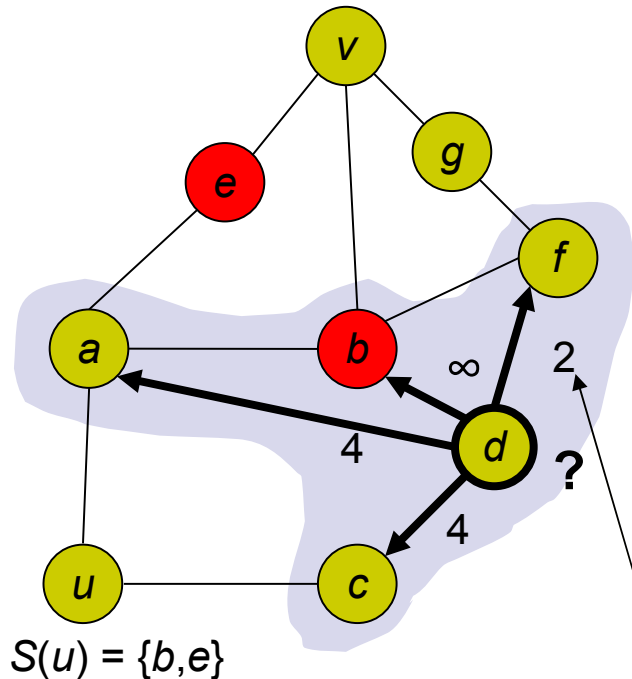
**Distance separator labels:** Assign labels  $L(v)$  to nodes so that given  $L(u), L(v)$  and  $L(s_1), \dots, L(s_k)$ , we can compute  $d(u, v)$  in  $G \setminus \{s_1, \dots, s_k\}$





# Distance separator labels

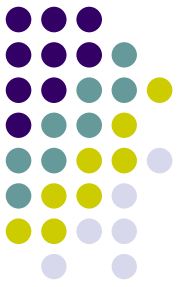
**Distance separator labels:** Assign labels  $L(v)$  to nodes so that given  $L(u), L(v)$  and  $L(s_1), \dots, L(s_k)$ , we can compute  $d(u, v)$  in  $G \setminus \{s_1 \dots s_k\}$



- Using distance separator labels
  - $d$  sends the packet to its neighbour that minimizes the distance to  $v$  in  $G \setminus \{s_1 \dots s_k\}$
  - This will guarantee that the packet travels from  $u$  to  $v$  on the shortest path that avoids  $\{s_1 \dots s_k\}$

We can route on paths avoiding a set  $S$  if we have distance separator labels

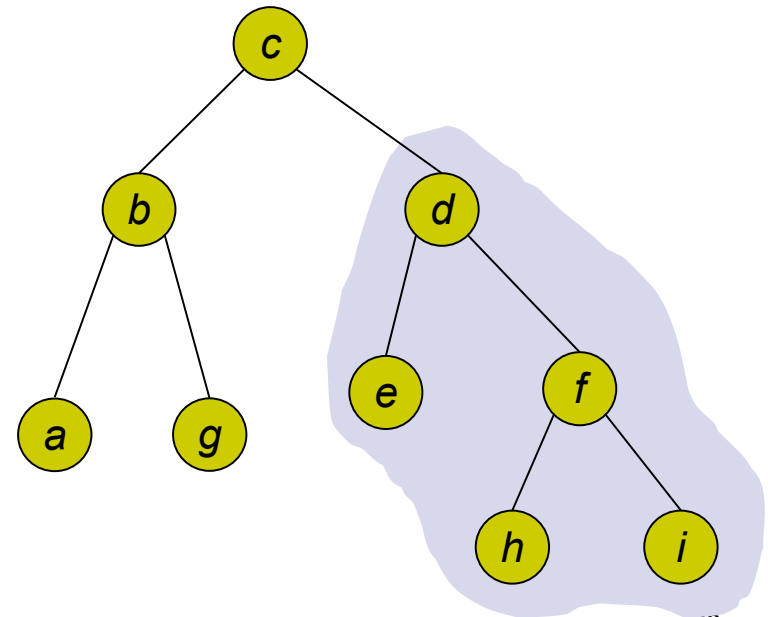
# Separator labels for trees



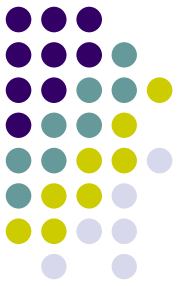
Observation: A node  $s$  is a  $uv$ -separator iff  $s \leq LCA(u, v)$  and  $(u \leq s$  or  $v \leq s)$

( $u \leq v$  means that  $u$  is a descendant of  $v$ )

So to construct separator labels, we need to be able to decide if  $u$  is an ancestor of  $v$  by only consulting  $L(u), L(v)$



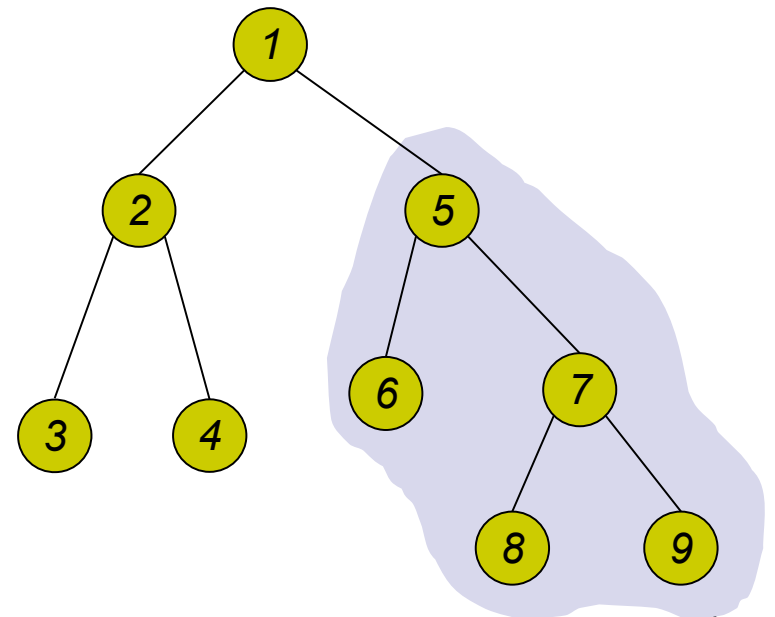
# Separator labels for trees

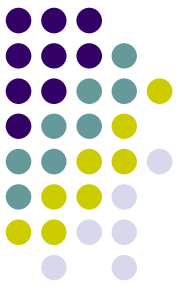


Observation: A node  $s$  is a  $uv$ -separator iff  $s \leq LCA(u, v)$  and  $(u \leq s$  or  $v \leq s)$

( $u \leq v$  means that  $u$  is an ancestor of  $v$ )

- Do a DFS, assigning ids to nodes





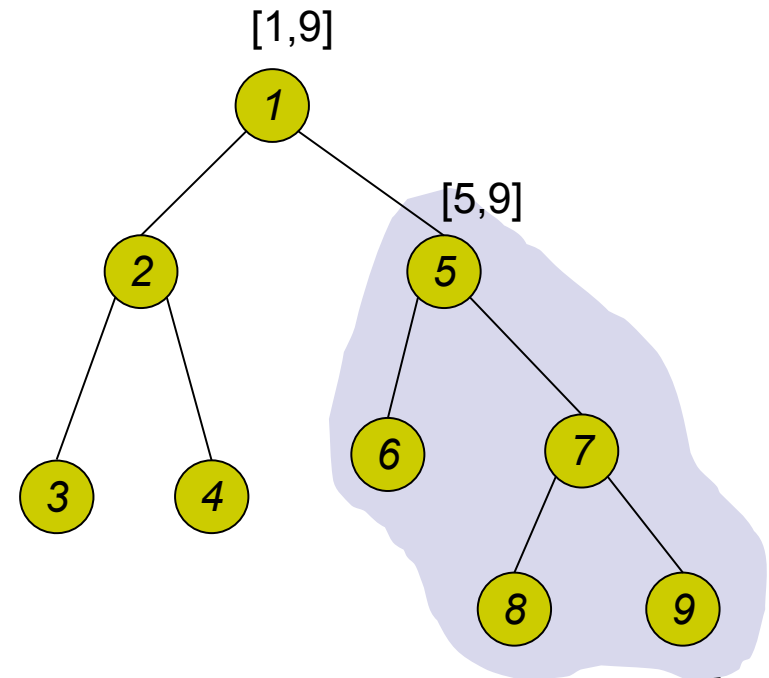
# Separator labels for trees

Observation: A node  $s$  is a  $uv$ -separator iff  $s \leq LCA(u,v)$  and  $(u \leq s$  or  $v \leq s)$

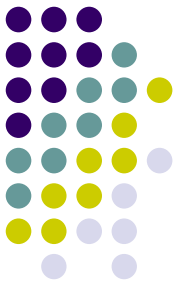
( $u \leq v$  means that  $u$  is an ancestor of  $v$ )

- Do a DFS, assigning ids to nodes
- Let  $f_v$  be the largest id of a descendant of  $v$
- $[id(v), f_v]$  is the *ancestor label* of  $v$

$v$  is an ancestor of  $u$   $id(v) \leq id(u) \leq f_v$



# Example

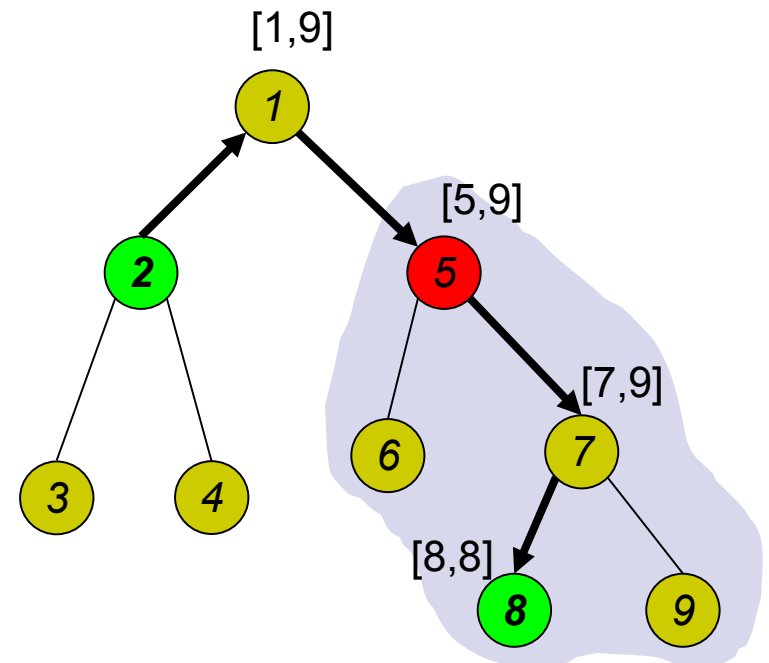


Observation: A node  $s$  is a  $uv$ -separator iff  $s \leq LCA(u,v)$  and  $(u \leq s$  or  $v \leq s)$

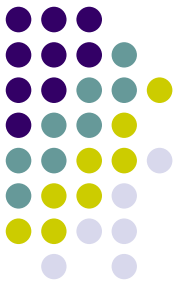
**Does node 5 separate 2 and 8 ?**

- 3)  $LCA(2,8)$  has label  $[1,9]$
- 4)  $5 \in [1,9]$ , so  $LCA(2,8)$  is an ancestor of 5
- 5)  $8 \in [5,9]$ , so 5 is an ancestor of 8

Therefore, 5 separates 2 and 8



# Distance separator labels: more general networks

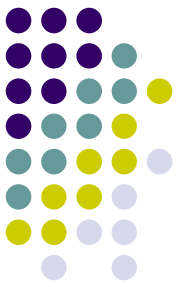


- In general there may be an exponential number of possible separators, but we can handle some types of networks

**Theorem:** Treewidth- $k$  graphs have  $O(k^2 \log^2 n)$ -bit distance separator labels

- This gives a routing scheme using  $O(\text{Deg}(G) k^2 \log^2 n)$ -bit tables

# Distance separator labels: cographs



As a warm up, consider cographs (having cwd 2)

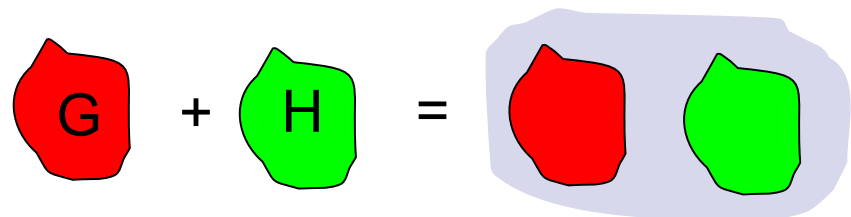
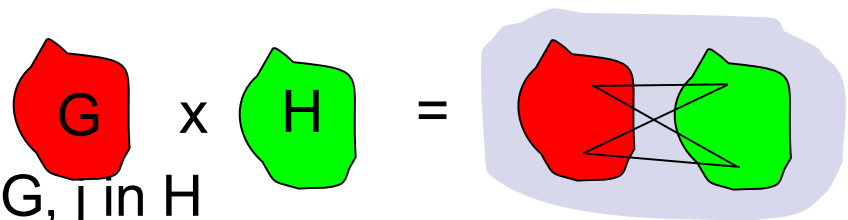
Idea: represent the graph by an algebraic expression

- Cographs can be constructed using two operations:

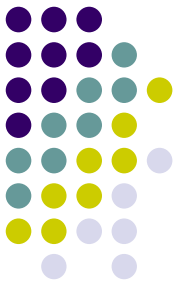
- Product:  $G \times H$

- Add edges  $(i,j)$  for  $i$  in  $G$ ,  $j$  in  $H$

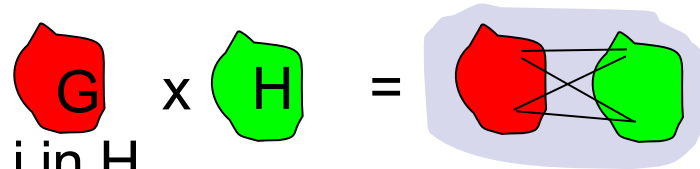
- Disjoint union:  $G + H$



# Distance separator labels: cographs

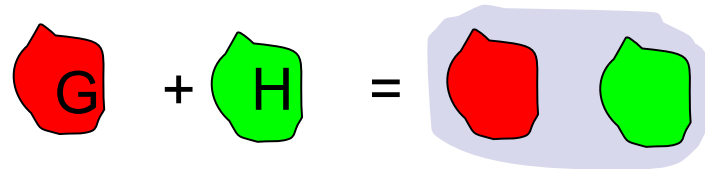


- Product:  $G \times H$

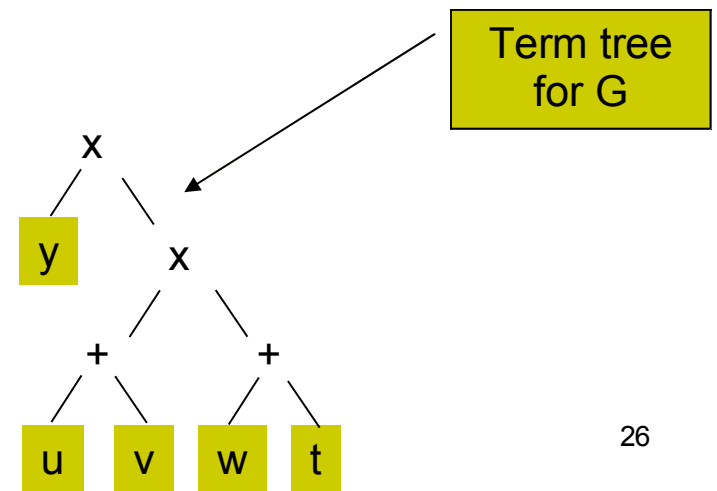
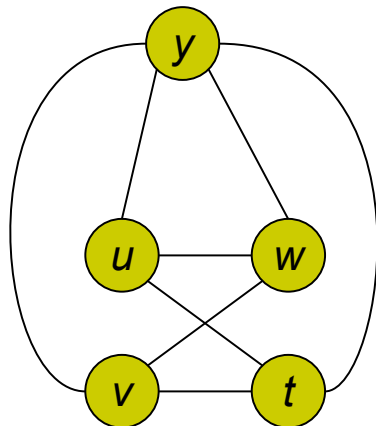


- Add edges  $(i,j)$  for  $i$  in  $G$ ,  $j$  in  $H$

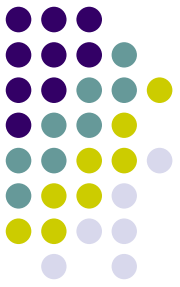
- Disjoint union:  $G + H$



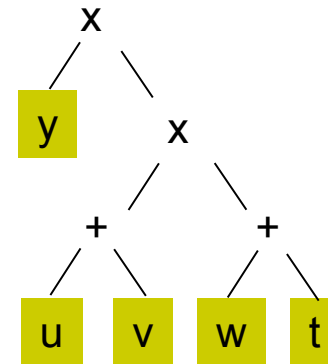
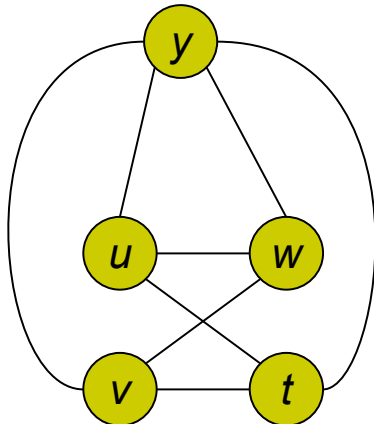
E.g.  $X(y, X(z, X(+ (u, v), + (w, t))))$



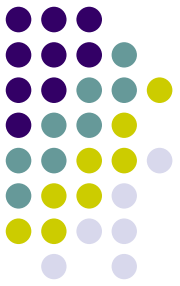
# Distance separator labels: cographs



- Each node stores its **access path** from the root
  - Eg  $L(u) = x \ 2 \ x \ 1 \ + \ 1$
- The set  $S$  is a separator of  $u, v$  iff
  - $w = LCA(u, v)$  is labelled with “+”
  - For every ancestor  $z$  of  $w$  labelled “x”, let  $z'$  be the child of  $z$  not on path from  $w$  to the root. Then all the descendants of  $z'$  must be in  $S$ .

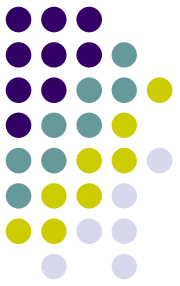


# Distance separator labels: cographs



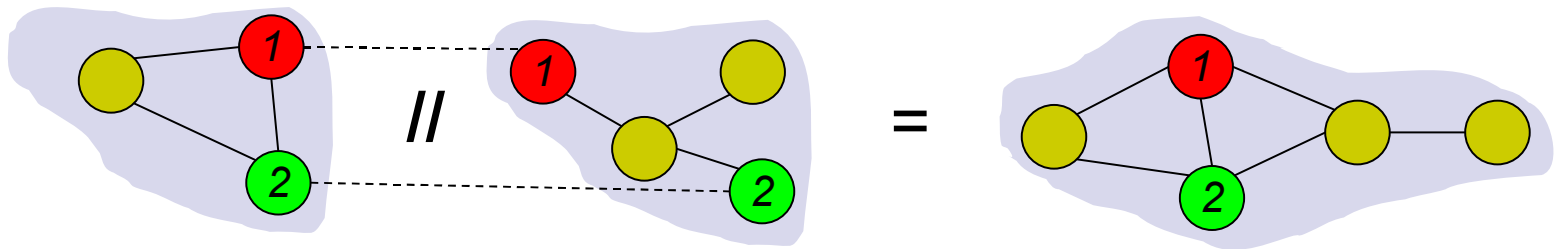
- If the term tree has height  $h$  then labels are  $O(h)$  bits
- This can be extended to graphs of treewidth  $k$

# Distance separator labels: treewidth $k$ graphs

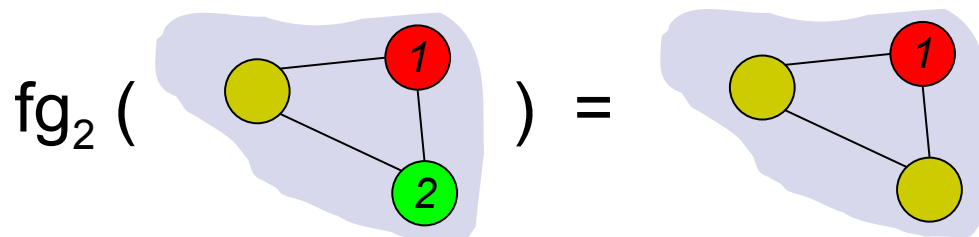


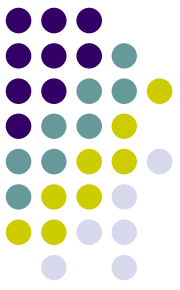
## Algebraic expressions for treewidth $k-1$ graphs

- Leaves are graphs containing  $\leq k$  distinct **sources**  $\{1 \dots k\}$
- Parallel composition:  $G // H$ 
  - Source with the same label in  $G, H$  are fused together



- Erasure:  $fg_a(G)$ 
  - The source labeled  $a$  is no longer a source





# A result for planar graphs

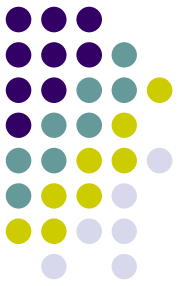
- Eppstein (2000) showed how to construct a centralized data structure of linear size so that for any constant  $c \geq 0$ , we can either route from  $u$  to  $v$  or return that  $d(u,v) > c$

**Theorem:** for any constant  $c \geq 0$ , we can assign labels of size  $\tilde{O}(|S(u)| \deg(G))$  so that for any nodes  $u, v$  we can either

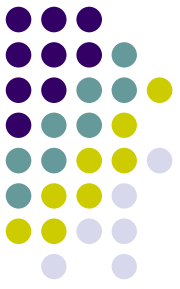
- route from  $u, v$  avoiding  $S(u)$ , or
- return that  $d_{G \setminus S}(u, v) > c$

- Cover a planar graph by graphs of small treewidth, then apply the previous result to each of these graphs

# Handling intermediate nodes

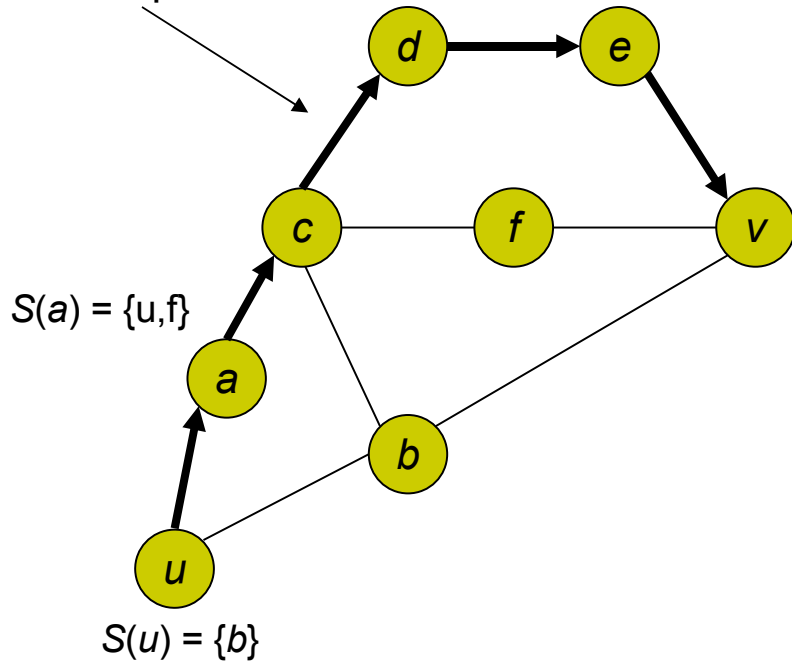


- So far we assumed that intermediate nodes always forward packets, even if the path is costly to them
- What if they just drop the packet?
  - Taking into account the preferences of intermediate nodes is hard



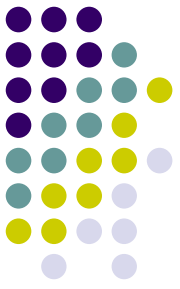
# Intermediate nodes

A good u-v path



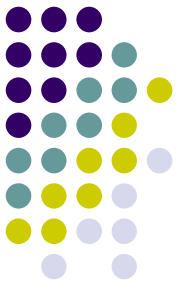
- Call a path 'good' if every subpath has zero cost
  - So, a packet can be sent over  $P$  at zero cost **to all nodes on  $P$**
- Without having to route around  $S(a)$ , the path would have been shorter
- Problem has a more complicated structure than before

# Handling intermediate nodes is hard

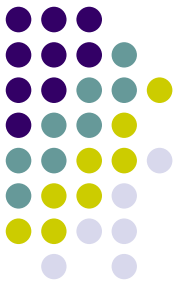


- We show that in order to decide if there exists a good path between two nodes, **labels of size  $\Omega(n^{1/2})$  bits are needed**
  - Even in trees!
  - Compare with  $O(\log n)$  for ancestor labels
- Conclusion?
  - Ignore the intermediate nodes?
  - Provide alternative incentives for them?
  - We *can* route on good paths using  $O(k)$  bit labels, but at the expense of time (a packet may traverse  $\Omega(n)$  edges before being returned if a good path does not exist)

# Summary



- **Routing trees** have many problems for policy routing
  - Intractable on simple networks (tree-like) and simple policies (forbidden-set)
- **Compact routing** is optimal for approx. shortest-path routing
  - Nothing was known for policy routing
    - Routing trees use space  $\Omega(n)$ , and have to be stable
  - Studied the problem of routing on paths avoiding a set  $X$ 
    - Space  $\tilde{O}(k^2)$  for tree width  $k$  networks
- Natural extension of compact routing to forbidden-set policies
  - Each node says “I want to route on **my** lowest-cost path”
  - Taking into account intermediate nodes’ costs is hard



# Some problems

- Extend the results to other classes of graphs and policies
  - Cliquewidth  $k$  graphs
  - Can things be improved if the forbidden-set policies are symmetric?
- Improve the routing scheme
  - Remove the linear dependence on the number of neighbours
- Can we do efficient routing without nodes revealing their policies?
- Consider other policies, e.g. transit policies