BGP Divergence Continued

Computer Networks
Dr. Jorge A. Cobb

Statically Solving an SPP

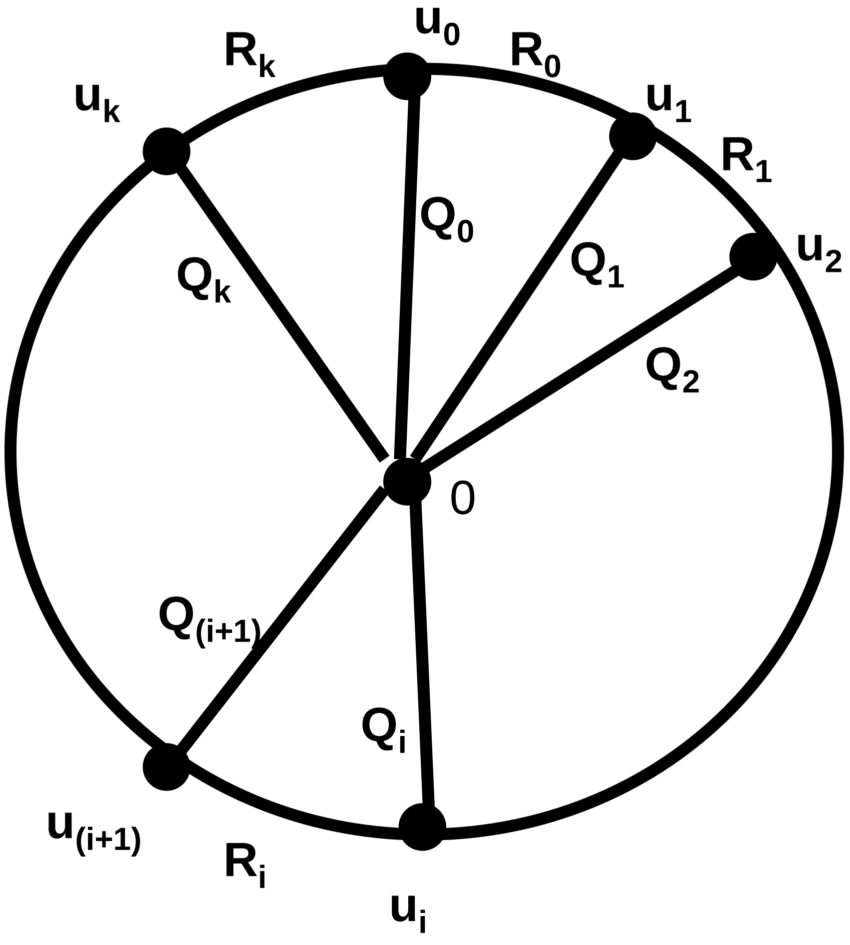
- Given an SPP, can we check if it is solvable?
- Sure,
  a) enumerate all possible states
  b) For each state, check if it is stable
- Checking for safety is even worse (check all sub-instances)
- Solvability of SPP is NP-Complete
  a) Exponential complexity!! (that we know of)

A Sufficient Condition for Sanity

- If an instance of SPP has no **dispute wheel**, then

Static (SPP)	Dynamic SPP
solvable	converges
unique solution	predictable restoration
all sub-problems uniquely solvable	robust with respect to link/node failures (safe)

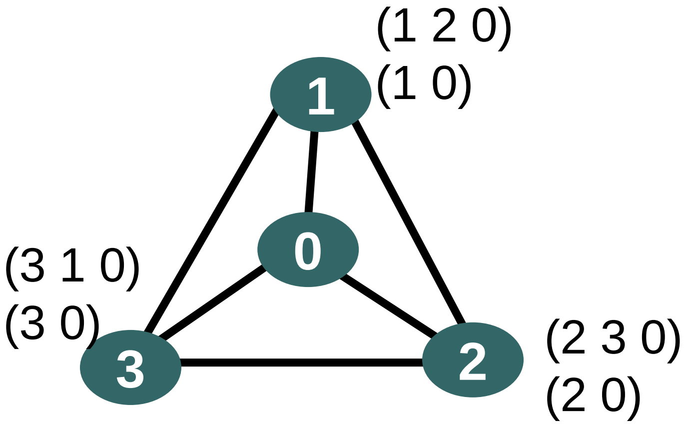
Dispute Wheels



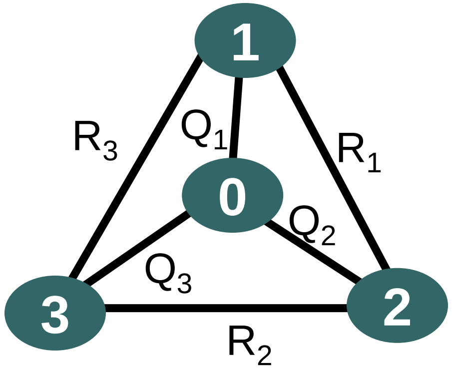
- $u_0, u_1, \ldots, u_k$ are nodes (not necessarily distinct)
- R_i is a path from u_i to $u_{(i+1)}$
- Q_i is a path from u_i to 0
- Q_i and $R_i\, Q_{(i+1)} \in \mathbf{P}^{u_i}$
- $\lambda^{u_i}(Q_i) < \lambda^{u_i}(R_i\, Q_{(i+1)})$

Example of Dispute Wheel

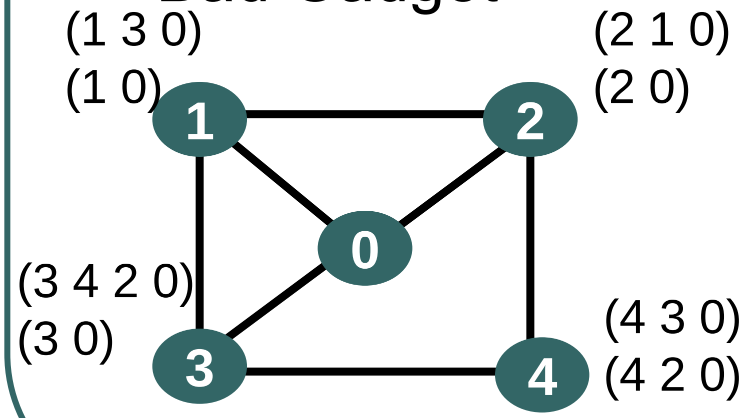
"Bad Triangle"



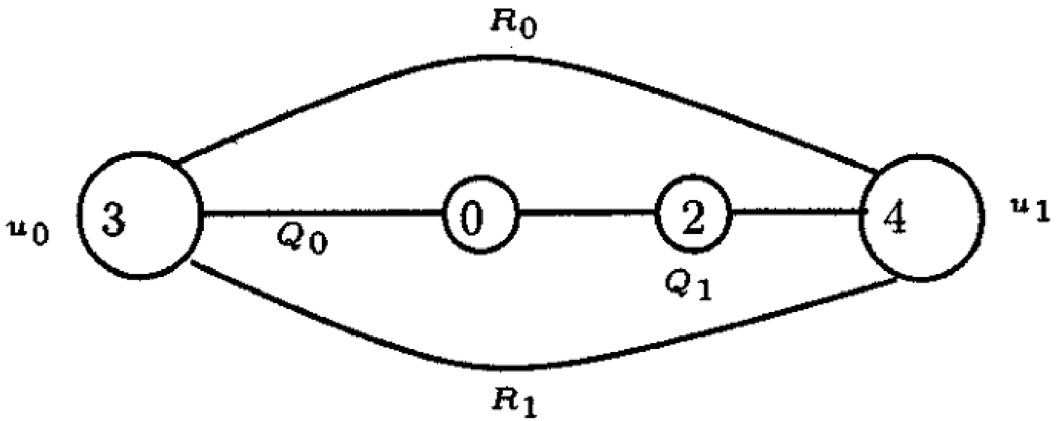
Its "dispute wheel"



"Bad Gadget"



A "dispute wheel"



Sufficient Condition for a Solution

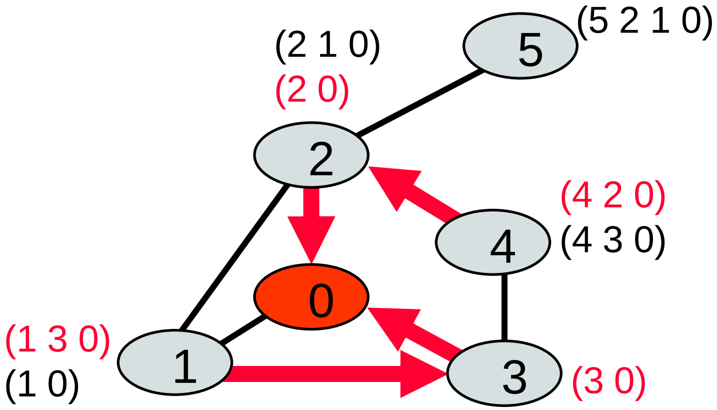
- **Theorem:** if an SPP instance does not have a dispute wheel, then it has a solution

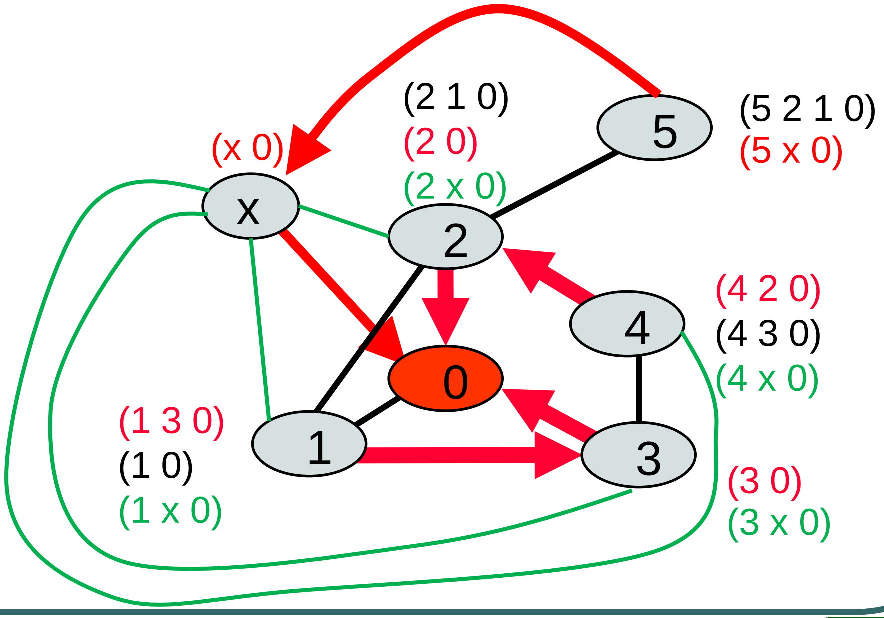
- Note: this is a sufficient but not necessary condition
  a) A dispute wheel could exist and still we have a solution (can you add nodes to bad triangle and make it converge while still having a dispute wheel???)

- We will build a spanning tree such that:
  a) If we complete the construction then there is a solution
  b) If we "get stuck", then there is a dispute wheel
     - Equivalently (contrapositive):
       no dispute wheel ➔ spanning tree built → solution exists

Ensuring solution IS a spanning tree

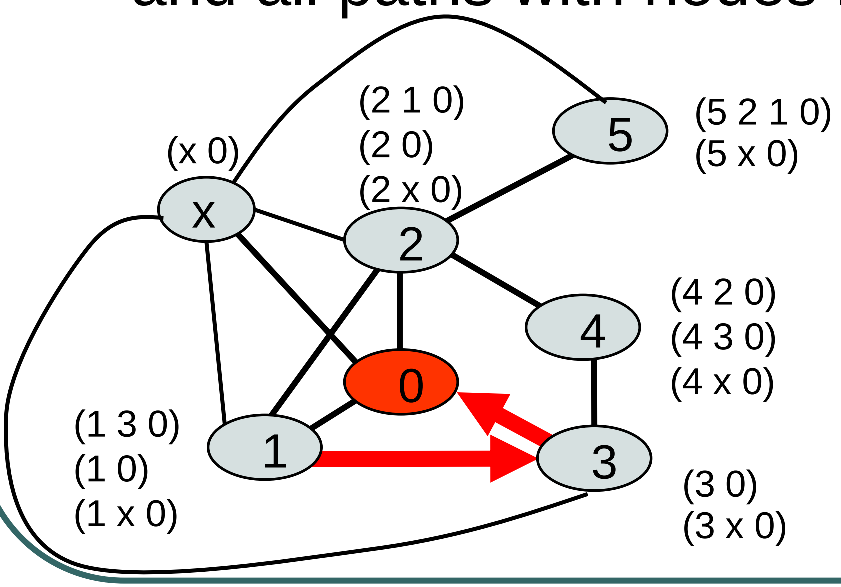
- Some solutions have some nodes with an empty path (not a spanning tree).
- We modify the SPP instance by adding an additional node x, and adding for each node u, the path (u x 0), that is ranked <u>lowest</u> among all paths at u.
- You can show <u>the solutions are the same as before</u>, except no one has an empty path.

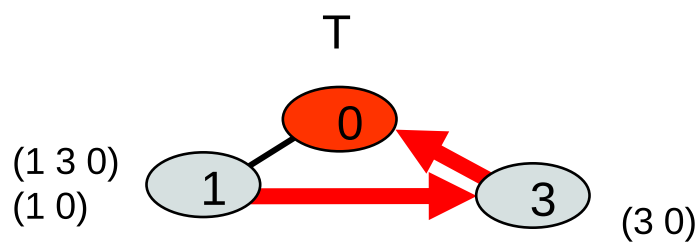




Tree construction

- For a tree T (not necessarily spanning), V(T) are its vertices (nodes)
- Let S be our SPP instance.
- At each step of our construction, the sub-instance S' obtained from S by <u>removing all nodes not in</u> V(T) and all paths with nodes not in V(T) **is stable.**





T is stable

Note that (1 0) remains since both 1 and 0 are in T

Adding a node to the tree

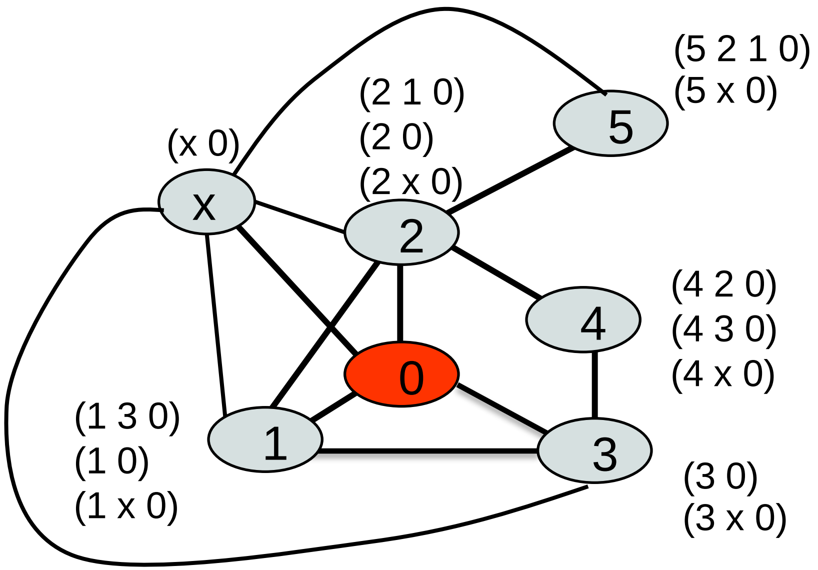
- A path P is said to be *consistent* with tree T if once P encounters a node in T, the rest of P is along T
  - I.e., once in T you remain in T
- Notation: for a node v in tree T, let T(v) be the path from v to 0 along T.

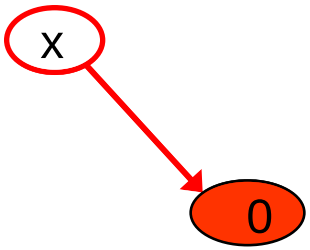
Adding a node to the tree (continued)

- Consider any node u such that:
  a) It has a neighbor v in T, and (u, v)T(v) is allowed at u. (notation: v is the first node in T(v))

  b) Of all the paths P in $\mathcal{P}^u$ that are "consistent with T" path (u, v)T(v) (i.e. directly into T) is the highest ranked of these paths.
- If such u is found, add u to T
  - Note: such u may not exist.
  - In this case, we are "stuck", and T does not become spanning.

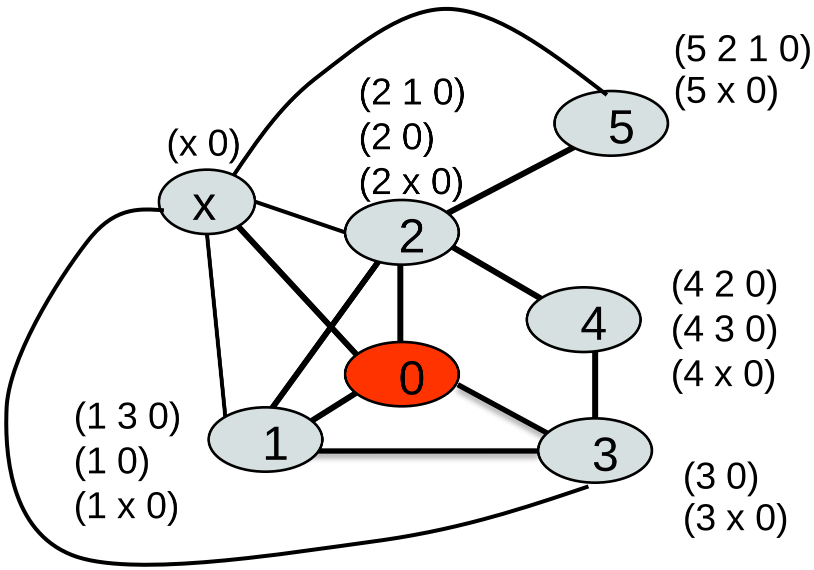
Example: build the tree

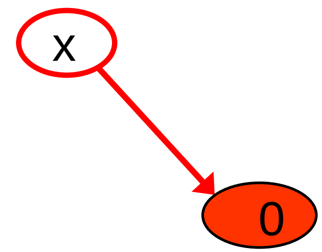




- First: T = {0, x} (x satisfies both (a) and (b) above)
- Note that, because of this, now ALL nodes u always have at least one path consistent with T, i.e., (u, x, 0)

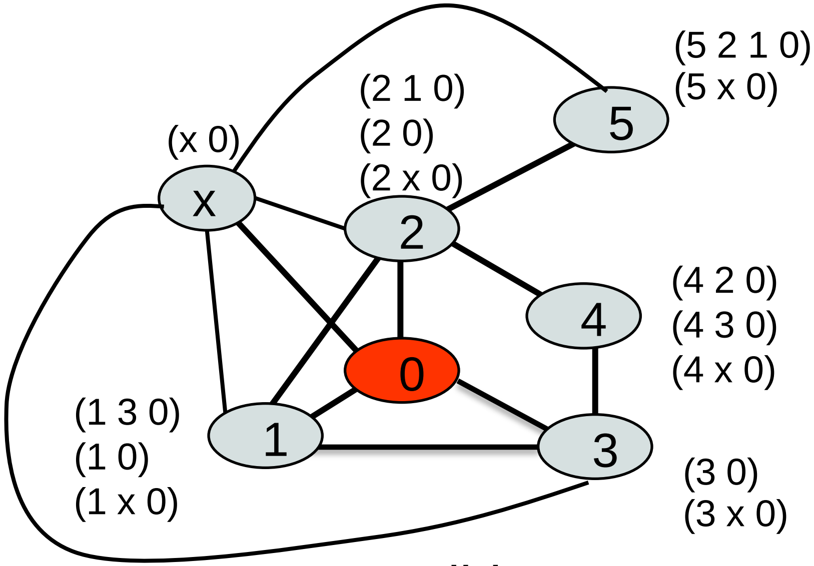
Example: build the tree

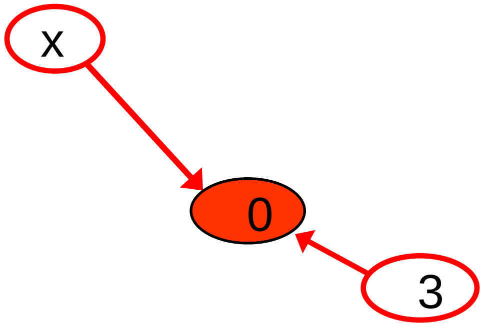




- Next candidates:
  - a) Because the tree is still small, <u>all paths in all nodes are consistent</u> with T.
  - b) E.g., 1 has three consistent paths: (1 3 <u>**0**</u>), (1 <u>**0**</u>), (1 <u>**x**</u> <u>**0**</u>)
  - c) However, in only node 3, the highest ranked one, i.e. (3 0), has its next hop in T
  - d) We thus add 3 to the tree.

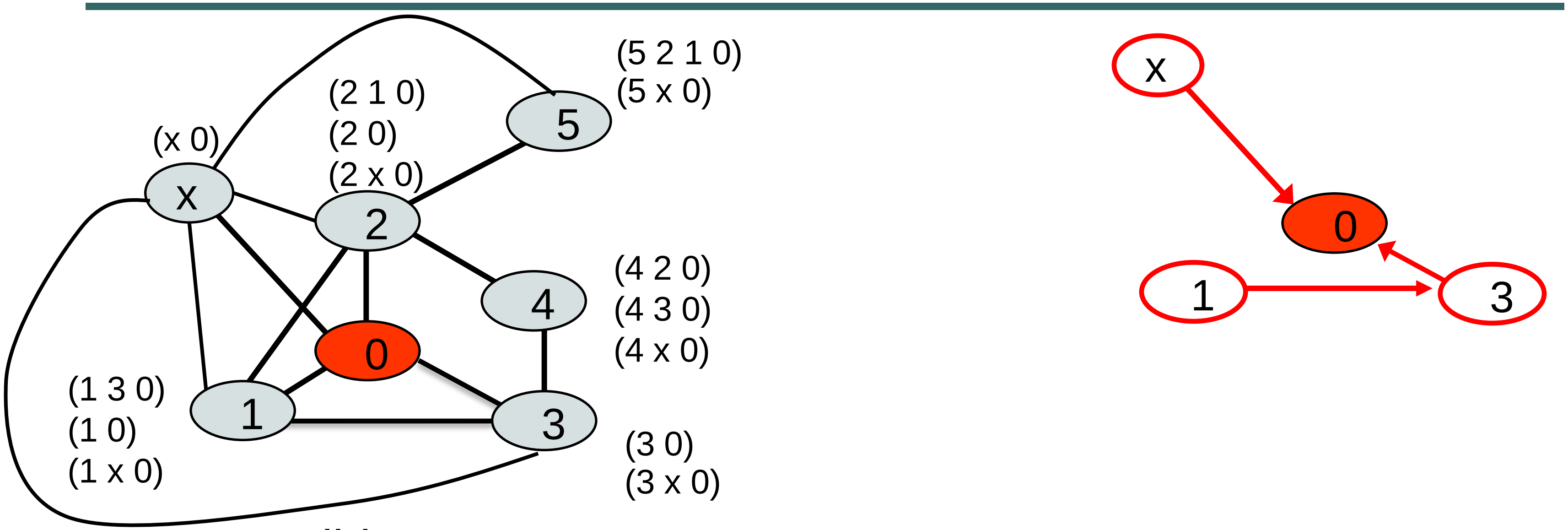
Example continued …





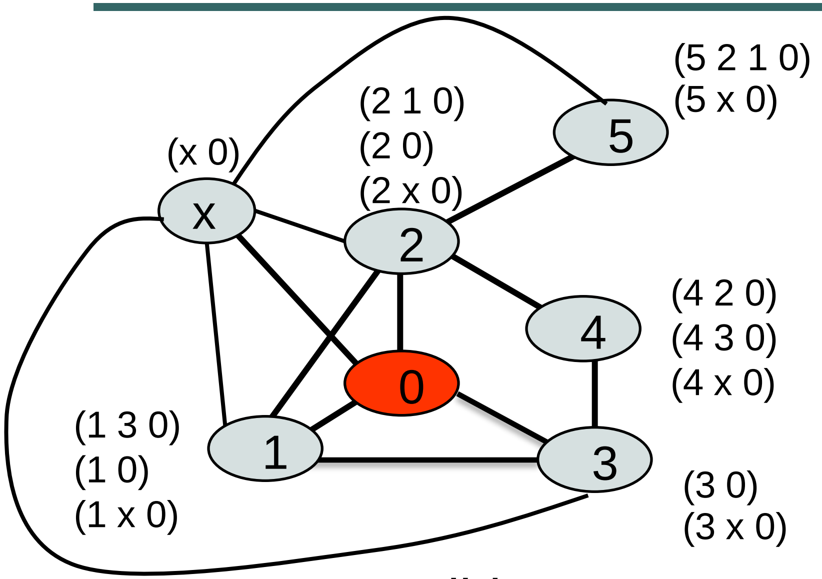
- Next candidates:
  a) Again, because the tree is still small, all paths in every node (not in T, i.e. in nodes 1, 2, 4, and 5) are consistent with T.
  b) E.g., 1 has three consistent paths: (1 **3** **0**), (1 **0**), (1 **x** **0**)
  c) However, of these nodes, only node 1, the highest ranked one, i.e. (1 3 0), has its next hop in T
  d) We thus add 1 to the tree.

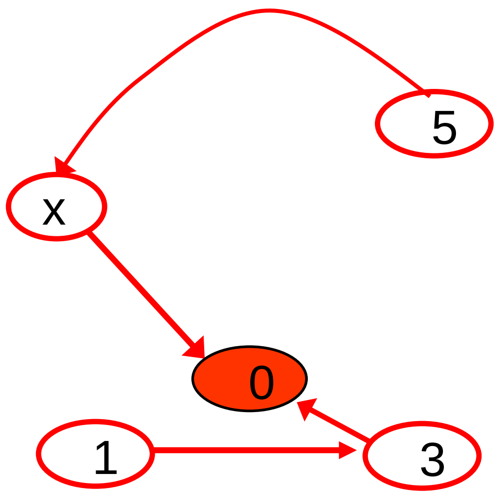
Example continued …



- Next candidates:
  a) All nodes not in T have a consistent path, but, not all paths in all nodes not in T (in nodes 2, 4, and 5) are consistent with T. E.g., (5 2 1 0) and (2 1 0) are not consistent since 1 has a tree path (1 3 0).
  b) Consistent paths: (2 **0**) (2 **x 0**) (4 2 **0**) (4 **3 0**) (4 **x 0**) (5 **x 0**)
  c) Of these nodes, 2 and 5 have their highest ranking path with a next hop in T. We can add either 2 or 5 to T.
  d) We arbitrarily choose to add 5 to the tree.

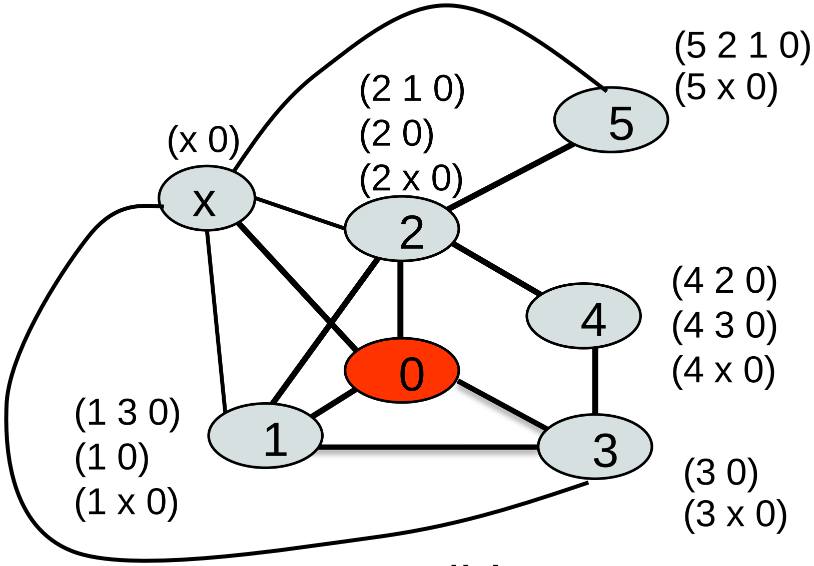
Example continued …

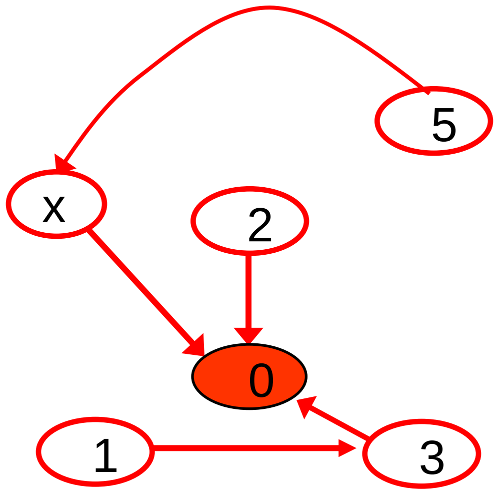




- Next candidates:
  a) All nodes not in T have a consistent path, but, not all paths in all nodes not in T (in nodes 2, 4) are consistent with T. E.g., (2 1 0) is not consistent since 1 is taking the path (1 3 0).
  b) Consistent paths: (2 **0**) (2 **x 0**) (4 2 **0**) (4 **3 0**) (4 **x 0**)
  c) Of these nodes, 2 has its highest ranking path with a next hop in T.
  d) We add 2 to the tree.

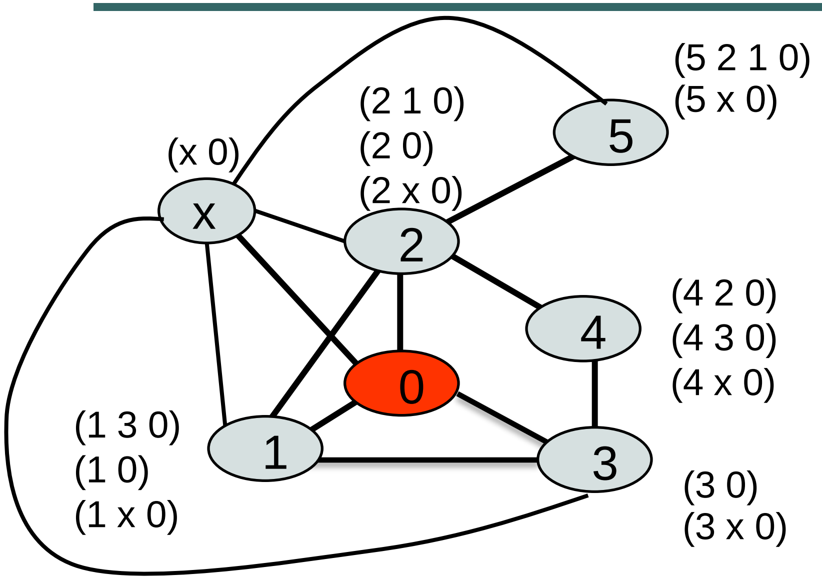
Example continued …

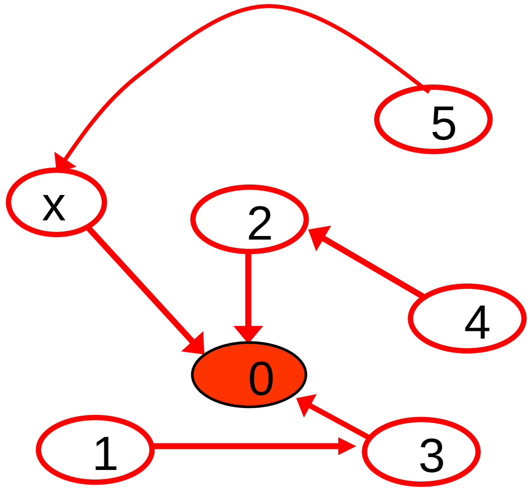




- Next candidates:
  a) Only 4 remains. It has three consistent paths (4 **2 0**) (4 **3 0**) (4 **x 0**)
  b) Its highest ranked consistent path has its next hop in T.
  c) We add 4 to the tree

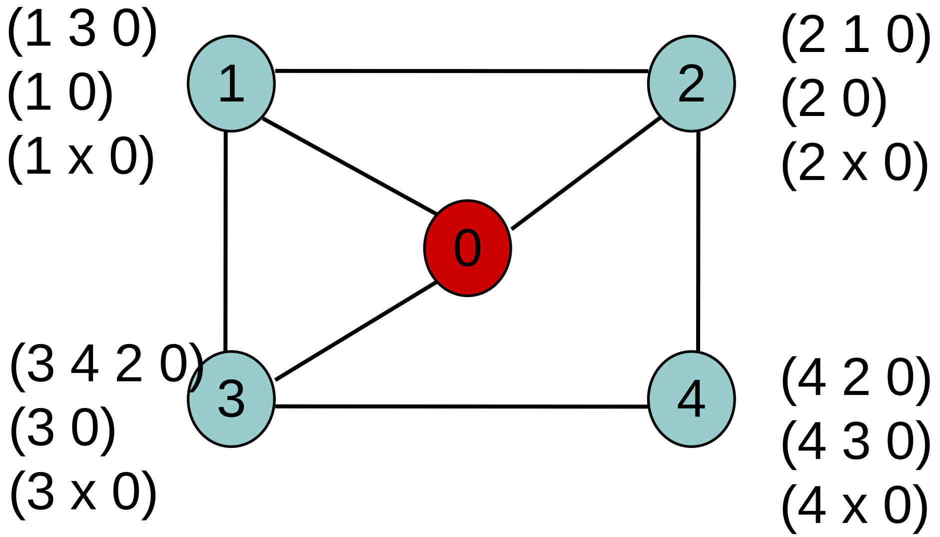
Example continued …





- T is a spanning tree
- T is stable (every step in construction yields a stable tree) (prove it!)
- Hence, there is a solution (T itself!)

What if we get stuck!



- Initially, T = {0, x} (and all paths are consistent)
- Note, we can’t add a single node more!
- We are stuck!
- All nodes prefer consistent paths not direct to x or 0.

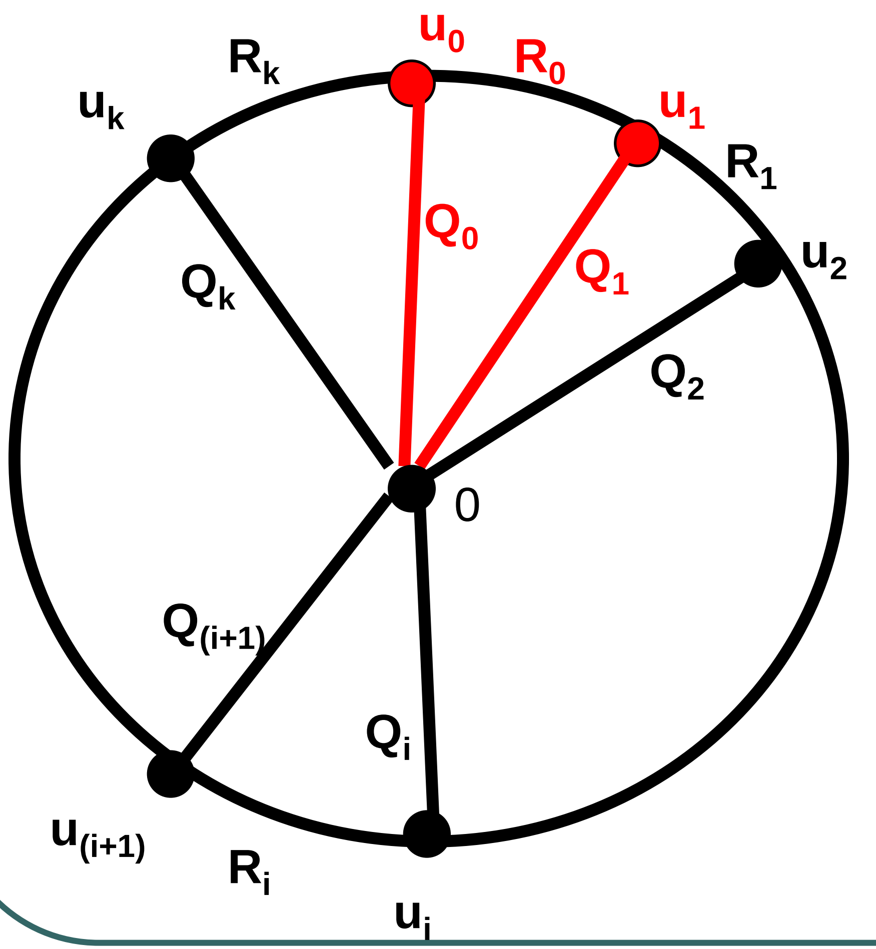
Stuck implies a dispute wheel . . .

- First, I am going to assume that in our SPP instance, if there is a path $(u_0, u_1, u_2, \ldots, u_n)$ allowed at u_0, then
  a) path $(u_1, u_2, \ldots, u_n)$ is allowed at u_1, and
  b) each edge (u_i, u_{i+1}), where $0 \le i < n$, exists in graph G.

- Note that otherwise, path could be removed from the list in u_0 and it would not affect the converge behavior of the system (the path would never be taken)

Stuck implies a dispute wheel (contd)

- We will build one “spoke” of the wheel



- $u_0, u_1, \ldots, u_k$ are nodes not on the tree T.
- $\lambda^{u_0}(Q_0) < \lambda^{u_0}(R_0\ Q_1)$

Stuck implies a dispute wheel (contd)

- Due to x, every u_0 not in T has at least one neighbor v_0 in T such that $(u_0, v_0)T(v_0)$ is allowed at u_0.
  a) Let $Q_0 = (u_0, v_0)T(v_0)$
- u_0 cannot be added to T. This implies that its highest ranking path P (consistent with T) has as next hop a node **w** that is not in T.
- P can be written as
  a) P : $(u_0,$ **w**$, \dots, u_1, v_1)T(v_1)$
  b) where u_1 is the last node in P not in T (u_1 = w is possible), and v_1 is in T
- Let $R_0 = (u_0, w, \dots, u_1)$, $Q_1 = (u_1, v_1)T(v_1)$, $P = R_0Q_1$
- From previous slide, $R_0Q_1 \in \mathcal{P}^{u0}$ implies $Q_1 \in \mathcal{P}^{u1}$

Stuck implies a dispute wheel (contd)

- Because P is the highest ranking consistent path at u_0, we have
  - $\lambda(Q_0) < \lambda(P) = \lambda(R_0 Q_1)$
- This completes one section of the dispute wheel
- Repeat the argument with u_1
  - note that
    - u_1 is not in T
    - Q_1 is permitted at u_1, and its next hop (v_1) is in T,
  - these are the same requirements we had on u_0.
- As we repeat the argument the wheel gets built.
- Since the number of nodes is finite, we eventually end up at u_0 again.

No dispute wheel → single solution

- If there is no dispute wheel, then there is a single solution.
- PLEASE READ THIS ON YOUR OWN ☺

No dispute wheel → Convergence

- Let α be a sequence of states such that
  a) For each i, $\alpha(i+1)$ is obtained from $\alpha(i)$ by executing one node
  b) For each i, $\alpha(i) \neq \alpha(i+1)$
- Let C be a cycle in α.
- A node u is *changing* in C if in two states in C it has different paths. Otherwise, u is *fixed.*
- Let F(C) be the set of fixed nodes in C
- If u $\notin$ F(C), let values(C, u) be the set of paths that u has taken in cycle C.

No dispute wheel → Convergence

First, a Lemma:

- Let
  a) u be a node that is <u>not fixed</u> in C,
  b) P be a path that is taken by u in C, (u changed its path from another path to P),
  c) and v is the first “fixed node” of P.
- Then, each non-fixed node w, w $\in$ P[u, v], takes the path P[w, 0] in some state in C.
- In particular, v stores P[v, 0] throughout C.
- READ THE PROOF OF THE LEMMA ON YOUR OWN

No dispute wheel → Convergence

Theorem: If there is a cycle C in an execution, then there exists a dispute wheel (no dispute wheel implies no cycles, and hence, convergence)

- Let u_0 be a node that
  a) is not fixed in C, and
  b) at some point in C, u_0 takes a path whose next hop w_0 is in F(C), i.e., u_0 takes a "direct" path to F(C).
  c) By the Lemma, u_0 must exist. Why?
- Let Q_0 be the direct path of u_0, i.e.,
  a) $Q_0 = (u_0, w_0)Q_0'$, where $w_0 \in F(C)$, Q_0' = path of w_0.
- Q_0 is unique and the lowest ranked in values(C, u_0). Why?

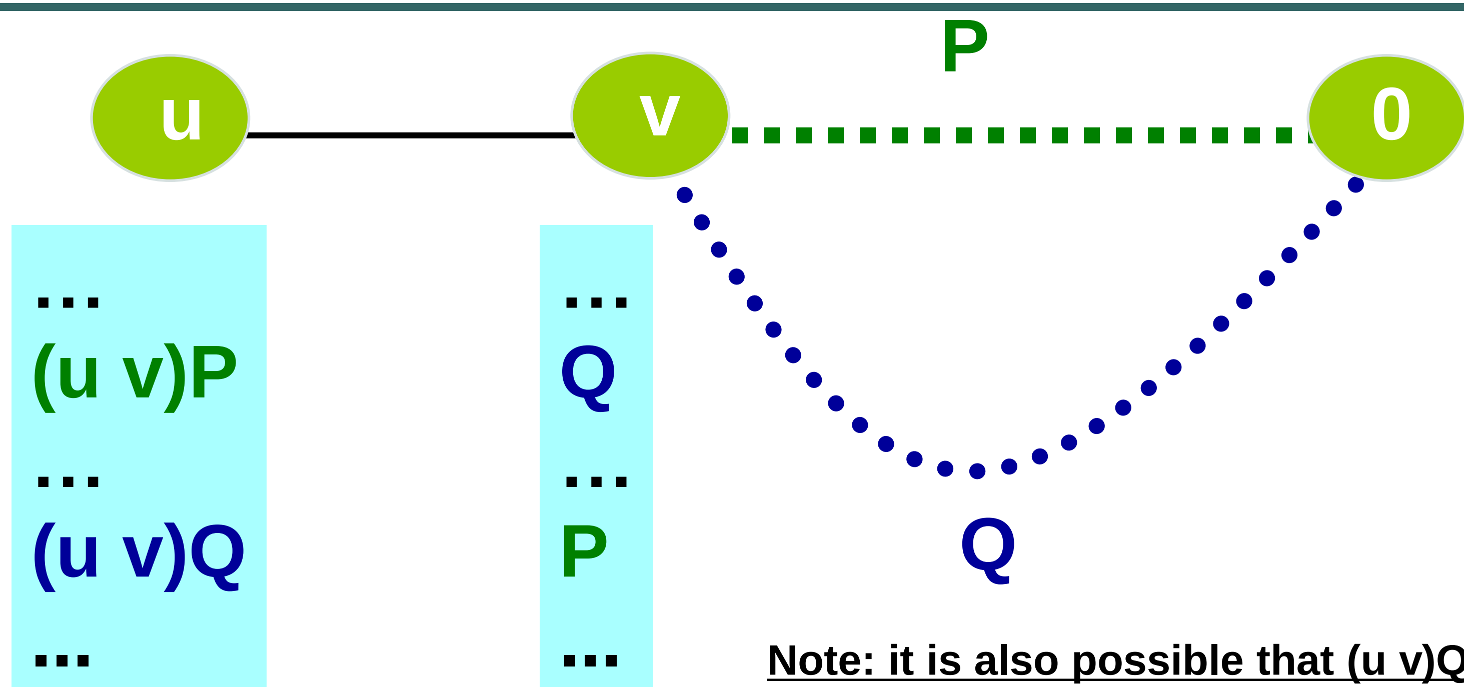
No dispute wheel → Convergence . . .

- Let Z be the highest ranked path in values(C, u_0)
- From the lemma, Z consists of a sequence of nodes that are not fixed, followed by a sequence of nodes that are fixed
  - $Z = (u_0, \ldots u_1, w_1)Z'$ where:
    - $u_0, \ldots, u_1$ are not fixed,
    - w_1 is fixed, and Z' is the path taken by w_1.
- Let $R_0 = (u_0, \ldots, u_1)$, and $Q_1 = (u_1, w_1)Z'$,
- We now have a spoke on the wheel
  - $\lambda(Q_0) < \lambda(Z) = \lambda(R_0\ Q_1)$
- Repeat for the next spoke using u_1 and Q_1.

Dispute Digraph

- The “nodes” in the graph correspond to paths in the SPP instance.
- It can be shown that a **cycle in the dispute graph** is the same as a **dispute wheel**.
- Hence, no dispute cycle iff no dispute wheel.
- There are two types of arcs (or edges) in the directed dispute graph:
  a) Conflict arcs
  b) Transmission arcs.

Dispute Arc

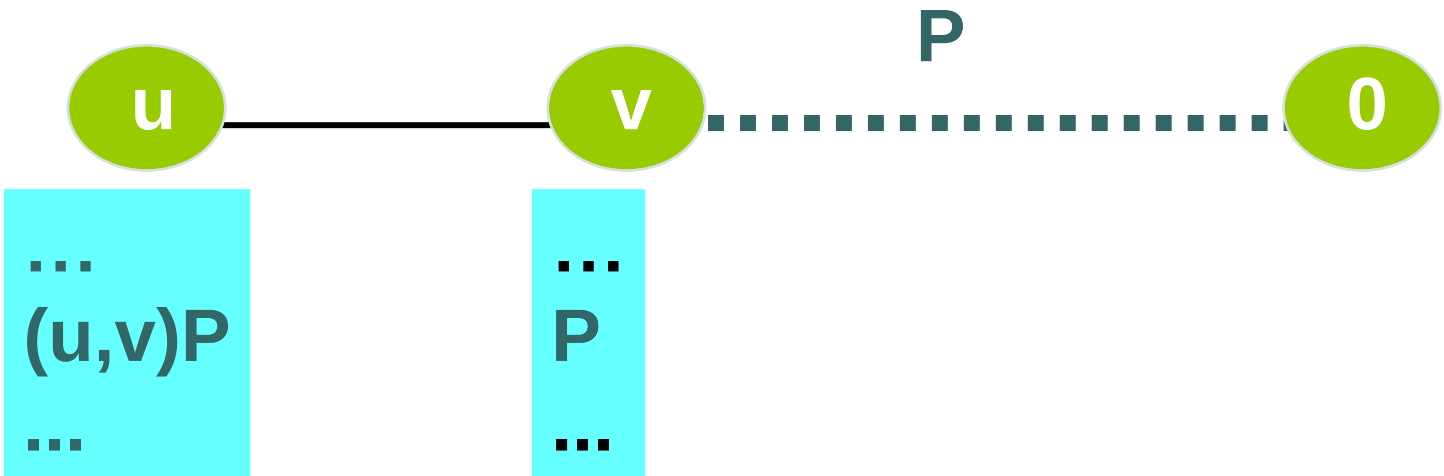


Note: it is also possible that (u v)Q is not allowed at u

Gives the *dispute arc*

$$\boxed{Q \longrightarrow (u\ v)P}$$

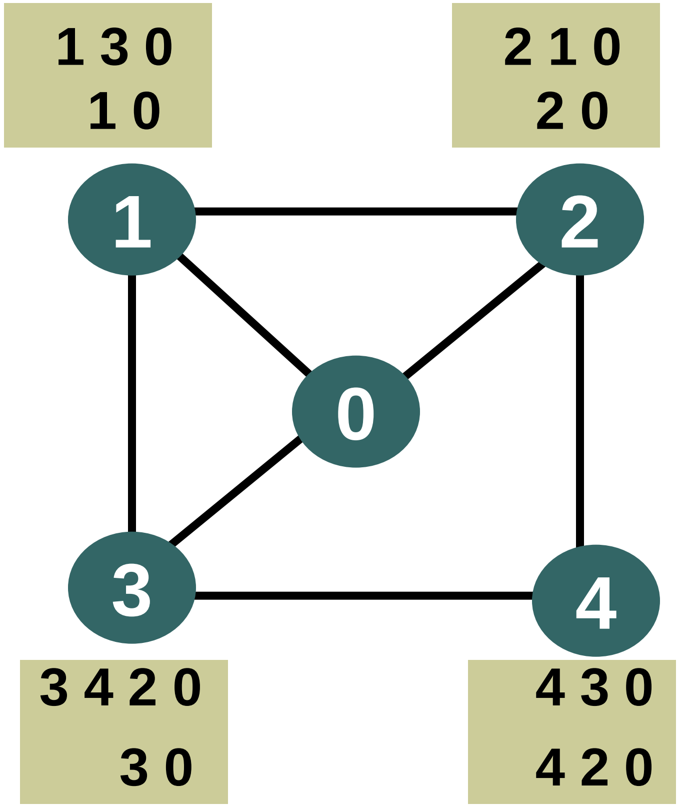
Transmission arc

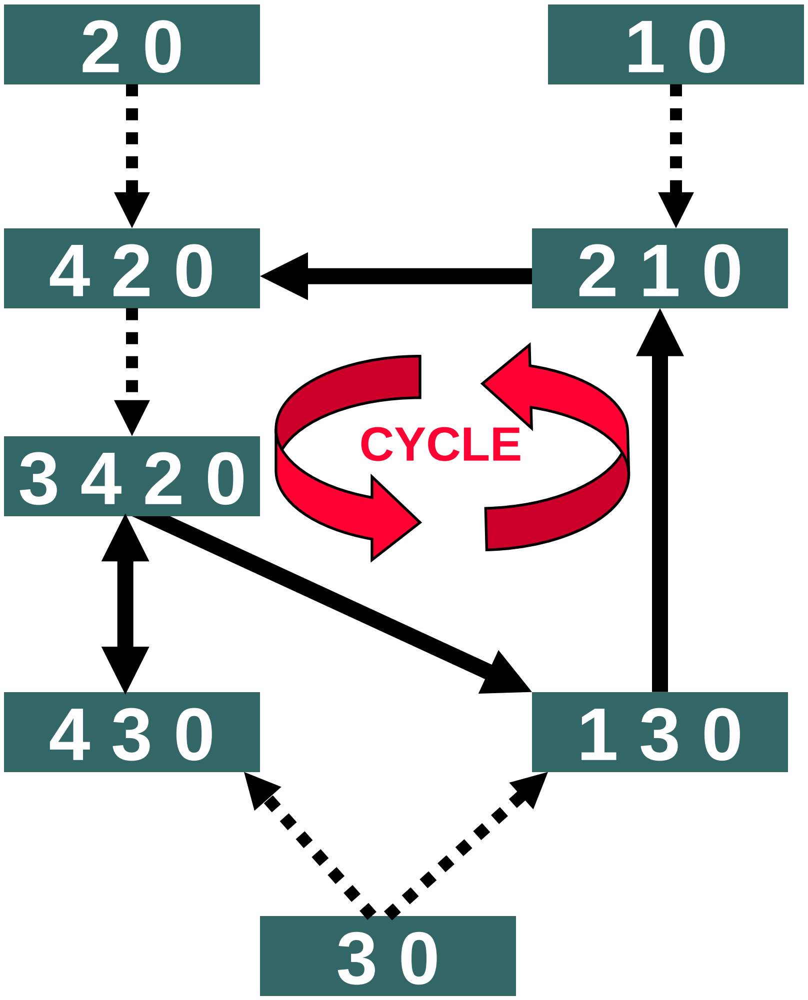


Gives the *transmission arc*

$$P \dashrightarrow (u,v)P$$

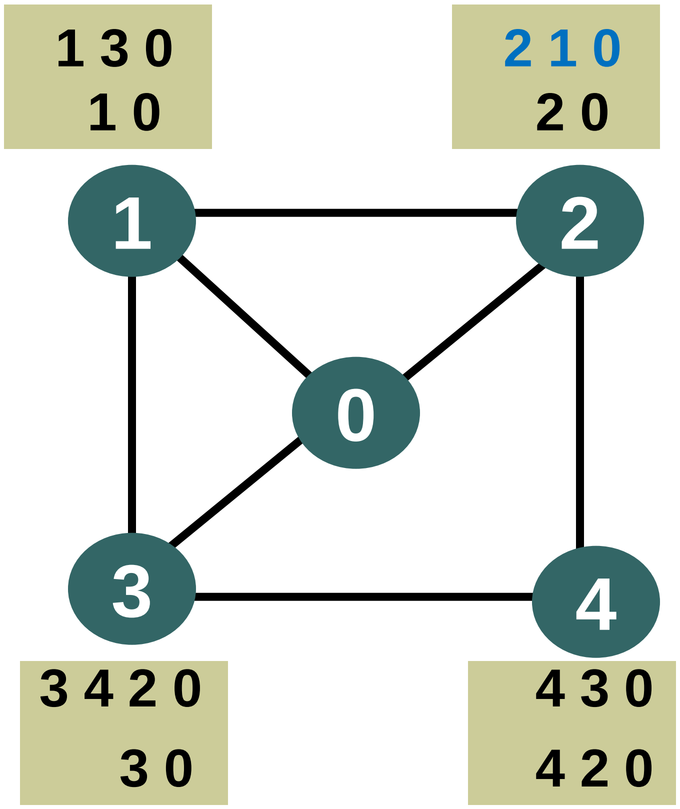
Dispute Digraph Example

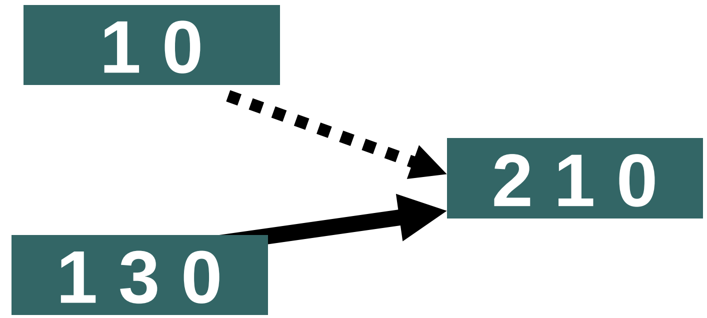




NAUGHTY GADGET

Dispute Digraph Example cont.





NAUGHTY GADGET

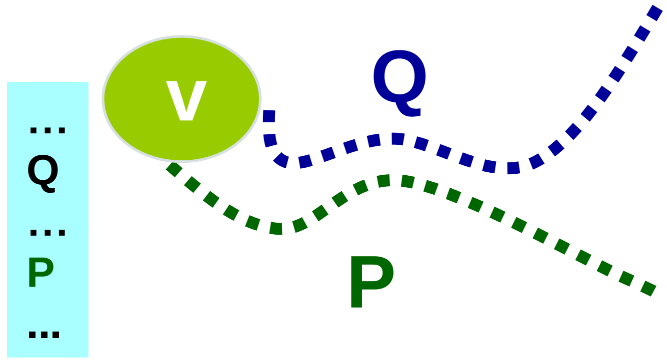
- Consider any node (say 2), think of it as u
- Take any path in u, say, 2 1 0 **(this is (u v)P)**
- The next node in the path (i.e. 1) is v
- If path 1 0 is in node 1 (it should be), then there is a **transmission arc** from 1 0 to 2 1 0.
- If there is a path in 1 with higher rank than 1 0, (yes there is, its 1 3 0), and 2 1 3 0 is ranked lower at 2 than 2 1 0 (it is lower ranked, since it is not even allowed at 2) then there is a **conflict arc** between 1 3 0 and 2 1 0.

An Application

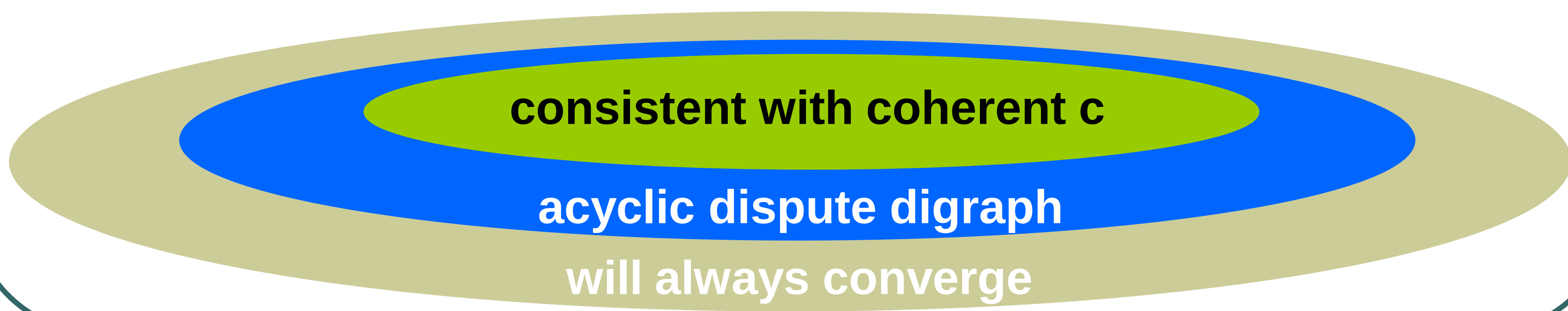
c is a cost function on edges in SPP.

c is *coherent* if all cycles have positive cost (> 0).

An SPP specification is *consistent with c* if



implies $c(P) \geq c(Q)$



Dynamic execution model

- I WILL NOT COVER IT, AND IT WILL NOT BE IN THE EXAM
- However, feel free to read it for the fun of it ☺

Dynamic Execution Model

- For our purposes (to make our life easier) we will consider a “shared memory model” as opposed to a message passing model.
- Assume that each node can “read” the state of its neighbor (i.e. if $(u,v) \in E$, then u can read $\pi(v)$ and v can read $\pi(u)$).
- Execution is simple
  - a) Do forever:
    - If there is a node u such that $\pi(u) \neq best(\pi,u)$, then
      - a) $\pi(u) := best(\pi,u)$
  - b) We assume a “fair” execution that does not ignore some nodes.

A Dynamic Solution

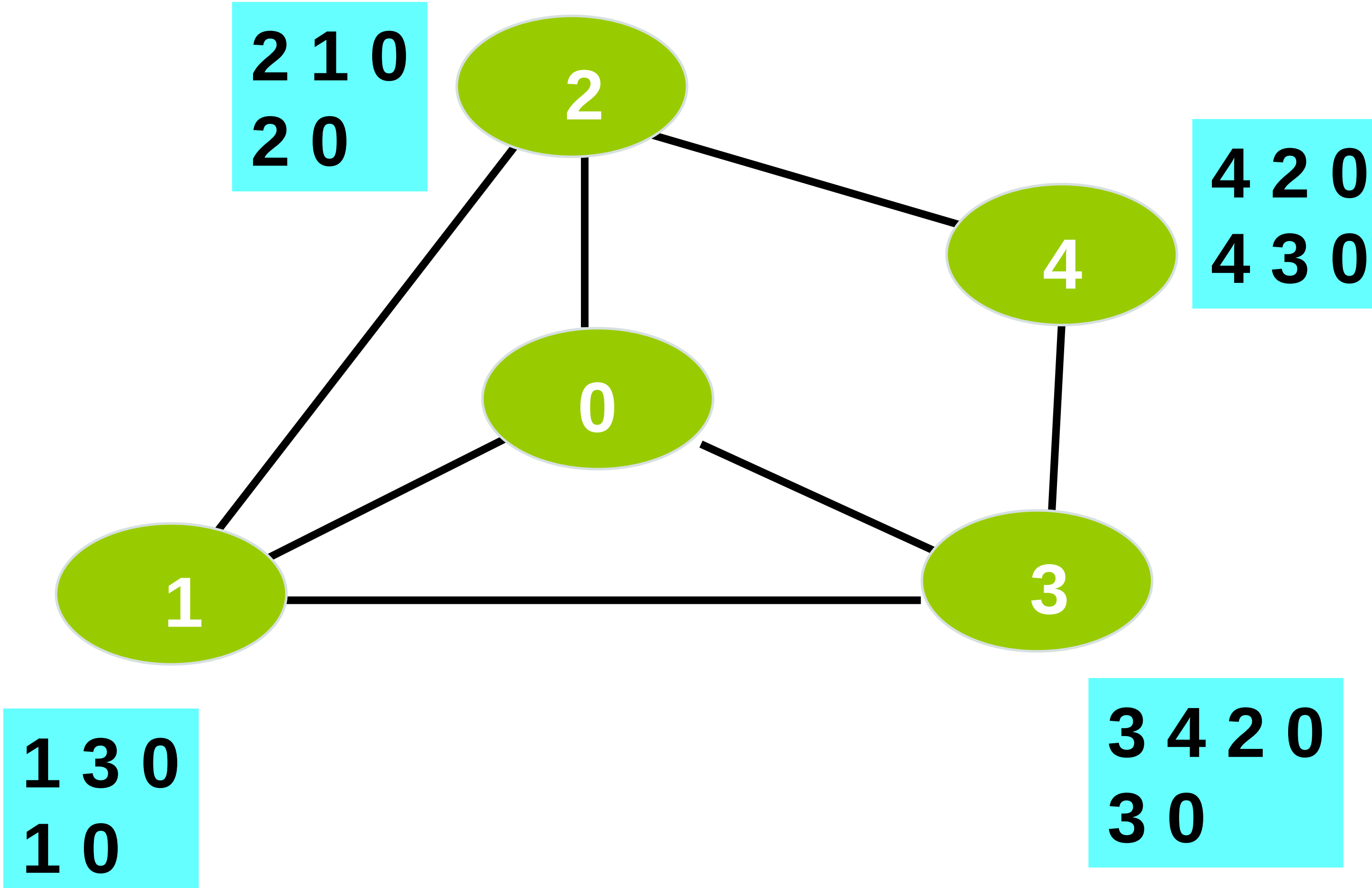
- Extend SPP with a history attribute of each route
  a) A route's history contains a path in the dispute digraph that "explains" how the route was obtained,
- Thus, if a route history contains a dispute cycle then a policy dispute was realized during the execution,
  a) i.e. the dispute graph has this cycle.
- If a route's history contains a cycle, then suppress it …. (we will see how later)
- Nodes read their neighbor's current path and the associated path history of the path.

Updating the history

- If your new path is better than your previous path (higher ranked)
  a) Add your NEW path (with a + sign) to the history of your NEW neighbor.

- If your new path is worse (lower ranked) than your previous path
  a) Add your OLD path (with a – sign) to the (updated) history of your OLD neighbor
     - Note: you choose a new lower ranked path because the old path you had is no longer available.
     - Thus, the old neighbor changed paths (and of course also its history)

BAD GADGET



path	event history for path
1 : (1 0)	e
2 : (2 0)	e
3 : (3 4 2 0)	e
4 : (4 2 0)	e

execute 2

1 : (1 0)	e
2 : (2 1 0)	(+ 2 1 0)
3 : (3 4 2 0)	e
4 : (4 2 0)	e

execute 4

1 : (1 0)	e
2 : (2 1 0)	(+ 2 1 0)
3 : (3 4 2 0)	e
4 : ( )	(- 4 2 0) (+ 2 1 0)

path | **event history for path**

execute 3

1 : (1 0)	e
2 : (2 1 0)	(+ 2 1 0)
3 : (3 0)	(- 3 4 2 0) (- 4 2 0) (+ 2 1 0)
4 : ( )	(- 4 2 0) (+ 2 1 0)

execute 1

1 : (1 3 0)	(+ 1 3 0) (- 3 4 2 0) (- 4 2 0) (+ 2 1 0)
2 : (2 1 0)	(+ 2 1 0)
3 : (3 0)	(- 3 4 2 0) (- 4 2 0) (+ 2 1 0)
4 : ( )	(- 4 2 0) (+ 2 1 0)

TIME

A CYCLE!

execute 2

1 : (1 3 0)	(+ 1 3 0) (- 3 4 2 0) (- 4 2 0) (+ 2 1 0)
2 : (2 0)	(- 2 1 0) (+ 1 3 0) (- 3 4 2 0) (- 4 2 0) (+ 2 1 0)
3 : (3 0)	(- 3 4 2 0) (- 4 2 0) (+ 2 1 0)
4 : ( )	(- 4 2 0) (+ 2 1 0)

What's going on ?

Dynamic cycles of event history correspond exactly to static cycles in the dispute digraph.