Graceful Operations in Link-State Routing Protocols

Francois Clad, Pascal Merindol, Stefano Vissicchio,
Jean-Jacques Pansiot and Pierre Francois

International Conference on Network Protocols
Göttingen, Germany, October 7-11, 2013

Université de Strasbourg
UCL Université catholique de Louvain
institute imdea networks

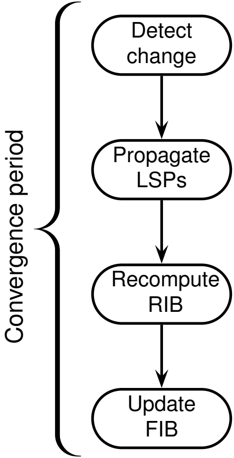
Some context

- Intra-domain routing in IP networks;
  - Link-state protocols (OSPF, IS-IS) possibly running MPLS with LDP;
- Frequent topological changes;
  - Maintenance operations on links or nodes;
  - Traffic engineering (weight modifications);

  ▷ Possible extension: unplanned changes;
- . . . and as many convergence periods;
  - Inconsistent transient state;
  - Possible traffic disruption.

Some context

- Intra-domain routing in IP networks;
  - Link-state protocols (OSPF, IS-IS) possibly running MPLS with LDP;
- Frequent topological changes;
  - Maintenance operations on links or nodes;
  - Traffic engineering (weight modifications);
  - ▷ Possible extension: unplanned changes;
- . . . and as many convergence periods;
  - Inconsistent transient state;
  - Possible traffic disruption.



1. Introduction
2. Transient loops
3. Node shut
4. Conclusion

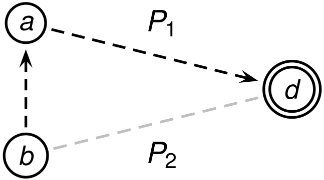
How do transient loops appear?

Routers' update order is **not controlled**!
(depends on *LSA flooding* and *RIB/FIB update* times)

Example:

- Initially, both *a* and *b* reach *d* through *a*;

Routes towards *d*:



$P_1 << P_2$

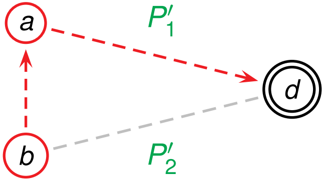
How do transient loops appear?

Routers' update order is **not controlled**!
(depends on *LSA flooding* and *RIB/FIB update* times)

Example:

- Initially, both *a* and *b* reach *d* through *a*;
- A change occurs on the network;
  Path through b more interesting, even for a;

Routes towards *d*:



Old: $P_1 << P_2$
New: $P'_1 >> P'_2$

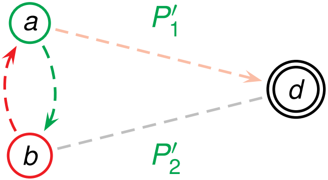
How do transient loops appear?

Routers' update order is **not controlled**!
(depends on *LSA flooding* and *RIB/FIB update* times)

Example:

- Initially, both *a* and *b* reach *d* through *a*;
- A change occurs on the network;
  Path through b more interesting, even for a;
- If *a* updates first and starts sending data towards *d* through *b*, while *b* still uses *a*;

Routes towards *d*:



Old: $P_1 << P_2$
New: $P'_1 >> P'_2$

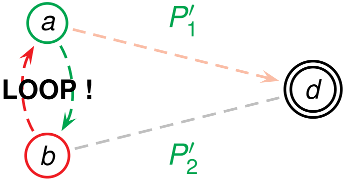
How do transient loops appear?

Routers' update order is **not controlled**!
(depends on *LSA flooding* and *RIB/FIB update* times)

Example:

- Initially, both *a* and *b* reach *d* through *a*;
- A change occurs on the network;
  Path through b more interesting, even for a;
- If *a* updates first and starts sending data towards *d* through *b*, while *b* still uses *a*;
- A **transient loop** appears on link (a, b);

Routes towards *d*:



Old: $P_1 << P_2$
New: $P'_1 >> P'_2$

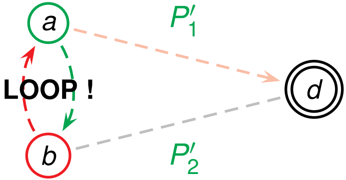
How do transient loops appear?

Routers' update order is **not controlled**!
(depends on *LSA flooding* and *RIB/FIB update* times)

Example:

- Initially, both *a* and *b* reach *d* through *a*;
- A change occurs on the network;
  Path through b more interesting, even for a;
- If *a* updates first and starts sending data towards *d* through *b*, while *b* still uses *a*;
- A **transient loop** appears on link (a, b);
  - ▷ Increased latency;
  - ▷ Packet losses.

Routes towards *d*:

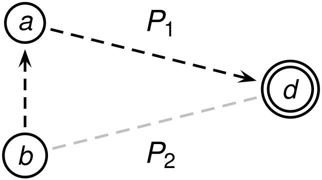


Old: $P_1 << P_2$
New: $P'_1 >> P'_2$

How to prevent them?

Force the routers to update in the *right* order.

- Initially, both *a* and *b* reach *d* through *a*;

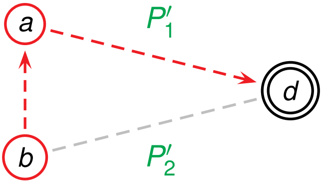


$$P_1 + w(b, a) < P_2$$

How to prevent them?

Force the routers to update in the *right* order.

- Initially, both *a* and *b* reach *d* through *a*;
- The same change occurs;

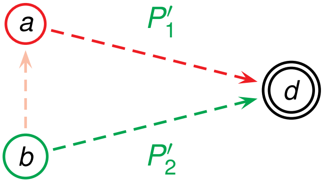


Old: $P_1 + w(b, a) < P_2$
New: $P'_1 > w(a, b) + P'_2$

How to prevent them?

Force the routers to update in the *right* order.

- Initially, both *a* and *b* reach *d* through *a*;
- The same change occurs;
- Yet this time *b* updates first;

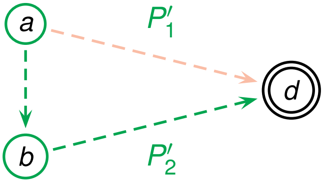


Old: $P_1 + w(b, a) < P_2$
New: $P'_1 > w(a, b) + P'_2$

How to prevent them?

Force the routers to update in the *right* order.

- Initially, both *a* and *b* reach *d* through *a*;
- The same change occurs;
- Yet this time *b* updates first;
- Then *a*, and no loop appears.



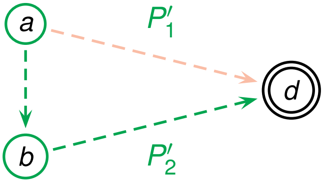
Old: $P_1 + w(b, a) < P_2$
New: $P'_1 > w(a, b) + P'_2$

How to prevent them?

Force the routers to update in the *right* order.

- Initially, both *a* and *b* reach *d* through *a*;
- The same change occurs;
- Yet this time *b* updates first;
- Then *a*, and no loop appears.

One goal, several approaches.



Old: $P_1 + w(b, a) < P_2$
New: $P'_1 > w(a, b) + P'_2$

Related works

- Ordered FIB *[INFOCOM'05, TON'07]*
  - Explicit router update ordering;
  - Relies on protocol extensions;
  - Non-incremental deployment;
- IGP migration *[SIGCOMM'12]*
  - Designed for network-wide migrations;
  - Requires to maintain two concurrent control planes;
  - Huge overhead for single link or node modifications;
- Metric increment - Link shut *[INFOCOM'07, TON'13]*
  - Progressive link weight update;
  - Suited for single link modifications;

Related works

- Ordered FIB *[INFOCOM'05, TON'07]*
  - Explicit router update ordering;
  - Relies on protocol extensions;
  - Non-incremental deployment;
- IGP migration *[SIGCOMM'12]*
  - Designed for network-wide migrations;
  - Requires to maintain two concurrent control planes;
  - Huge overhead for single link or node modifications;
- Metric increment - Link shut *[INFOCOM'07, TON'13]*
  - Progressive link weight update;
  - Suited for single link modifications;
  - Extension to node-wide modifications.

Progressive update

Basic idea

Split up the change into a sequence of **loop free** updates.

Objectives

Compute a sequence of intermediate updates, such that there is no transient loop between subsequent updates.

Challenge

- Sequences of minimal length (minimal operational impact);
- Efficient algorithm (embedded in router OS).

Illustration: path increment sequence

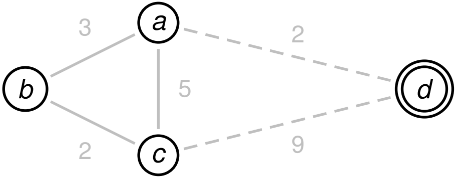


Illustration: path increment sequence

- Initially, *a*, *b* and *c* reach *d* through node *a*.

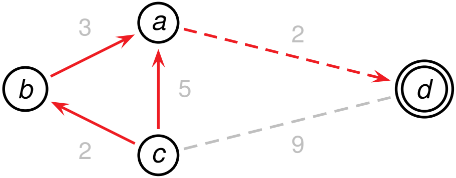


Illustration: path increment sequence

- Initially, a, b and c reach d through node a.
- If a change occur on path $P(a, d)$ increasing its cost to 50...

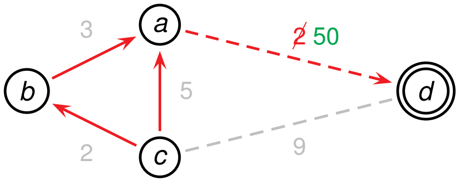


Illustration: path increment sequence

- Initially, *a*, *b* and *c* reach *d* through node *a*.
- If a change occur on path $P(a, d)$ increasing its cost to 50, all three nodes will go through *c* instead . . .

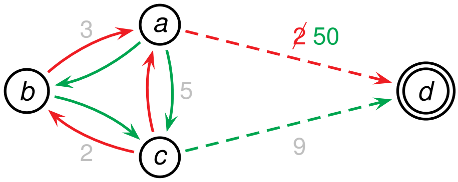


Illustration: path increment sequence

- Initially, a, b and c reach d through node a.
- If a change occur on path $P(a, d)$ increasing its cost to 50, all three nodes will go through c instead and transient loops may appear.

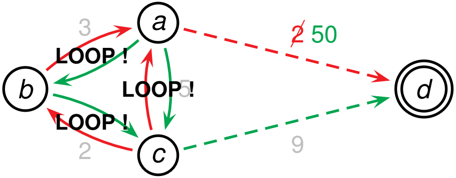


Illustration: path increment sequence

- Initially, a, b and c reach d through node a.
- If a change occur on path $P(a, d)$ increasing its cost to 50, all three nodes will go through c instead and transient loops may appear.

With incremental updates:

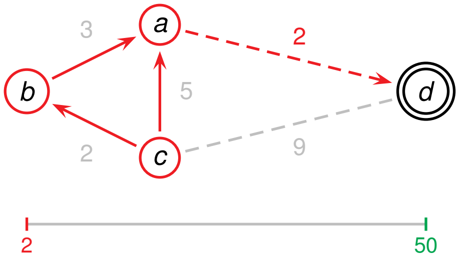


Illustration: path increment sequence

- Initially, *a*, *b* and *c* reach *d* through node *a*.
- If a change occur on path $P(a, d)$ increasing its cost to 50, all three nodes will go through *c* instead and transient loops may appear.

With incremental updates:

- Node *c* could update first;

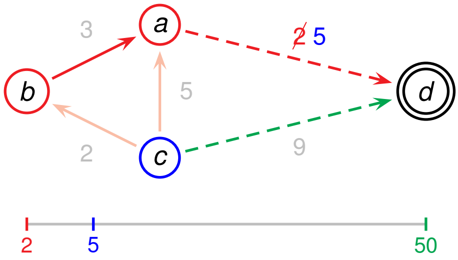


Illustration: path increment sequence

- Initially, *a*, *b* and *c* reach *d* through node *a*.
- If a change occur on path $P(a, d)$ increasing its cost to 50, all three nodes will go through *c* instead and transient loops may appear.

With incremental updates:

- Node *c* could update first;
- Then *b*,

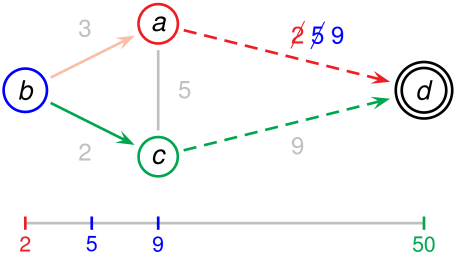


Illustration: path increment sequence

- Initially, *a*, *b* and *c* reach *d* through node *a*.
- If a change occur on path $P(a, d)$ increasing its cost to 50, all three nodes will go through *c* instead and transient loops may appear.

With incremental updates:

- Node *c* could update first;
- Then *b*, and *a*;

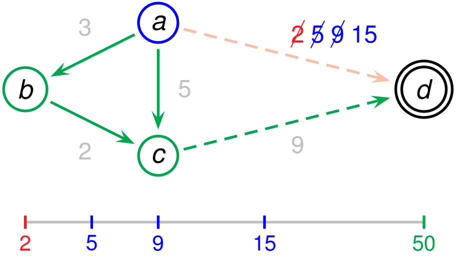


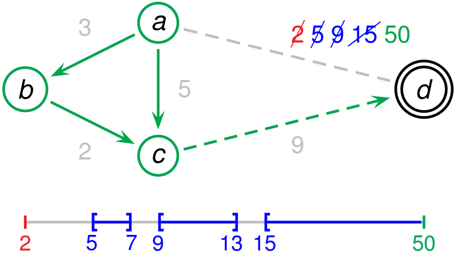
Illustration: path increment sequence

- Initially, *a*, *b* and *c* reach *d* through node *a*.
- If a change occur on path $P(a, d)$ increasing its cost to 50, all three nodes will go through *c* instead and transient loops may appear.

With incremental updates:

- Node *c* could update first;
- Then *b*, and *a*;

So that the transition to 50 will be loop free **for destination** *d*.



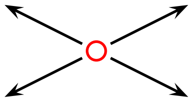
The Node Shutdown Problem

Objective: allow an operator to shut a router down without triggering transient inconsistencies.

The Node Shutdown Problem

Objective: allow an operator to shut a router down without triggering transient inconsistencies.

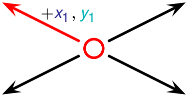
- Simple solution: shut down each link one by one
  - Number of intermediate steps proportional to the node degree



The Node Shutdown Problem

Objective: allow an operator to shut a router down without triggering transient inconsistencies.

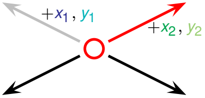
- Simple solution: shut down each link one by one
  - ▷ Number of intermediate steps proportional to the node degree



The Node Shutdown Problem

Objective: allow an operator to shut a router down without triggering transient inconsistencies.

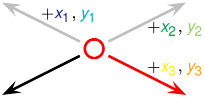
- Simple solution: shut down each link one by one
  - ▷ Number of intermediate steps proportional to the node degree



The Node Shutdown Problem

Objective: allow an operator to shut a router down without triggering transient inconsistencies.

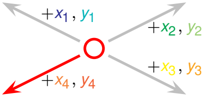
- Simple solution: shut down each link one by one
  - Number of intermediate steps proportional to the node degree



The Node Shutdown Problem

Objective: allow an operator to shut a router down without triggering transient inconsistencies.

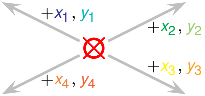
- Simple solution: shut down each link one by one
  - ▷ Number of intermediate steps proportional to the node degree



The Node Shutdown Problem

Objective: allow an operator to shut a router down without triggering transient inconsistencies.

- Simple solution: shut down each link one by one
  - Number of intermediate steps proportional to the node degree

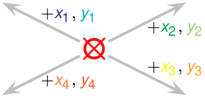


8 intermediate steps

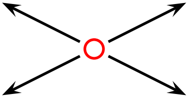
The Node Shutdown Problem

Objective: allow an operator to shut a router down without triggering transient inconsistencies.

- Simple solution: shut down each link one by one
  - ▷ Number of intermediate steps proportional to the node degree



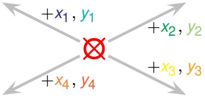
- Better solution: benefit from an existing OSPF / IS-IS feature
  - ▷ Simultaneous weight modifications



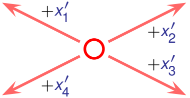
The Node Shutdown Problem

Objective: allow an operator to shut a router down without triggering transient inconsistencies.

- Simple solution: shut down each link one by one
  - ▷ Number of intermediate steps proportional to the node degree



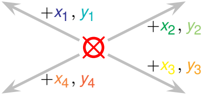
- Better solution: benefit from an existing OSPF / IS-IS feature
  - ▷ Simultaneous weight modifications



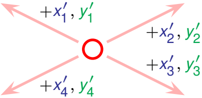
The Node Shutdown Problem

Objective: allow an operator to shut a router down without triggering transient inconsistencies.

- Simple solution: shut down each link one by one
  - Number of intermediate steps proportional to the node degree



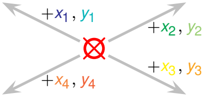
- Better solution: benefit from an existing OSPF / IS-IS feature
  - Simultaneous weight modifications



The Node Shutdown Problem

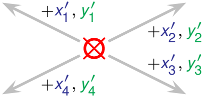
Objective: allow an operator to shut a router down without triggering transient inconsistencies.

- Simple solution: shut down each link one by one
  - ▷ Number of intermediate steps proportional to the node degree



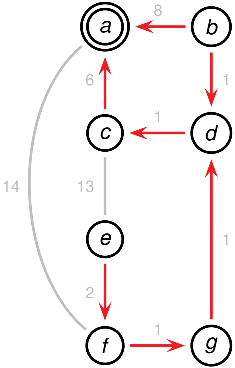
8 intermediate steps

- Better solution: benefit from an existing OSPF / IS-IS feature
  - ▷ Simultaneous weight modifications

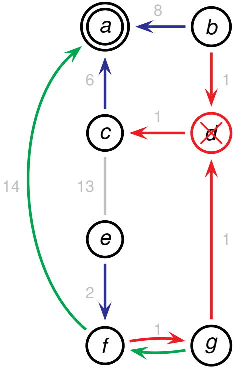


2 intermediate steps

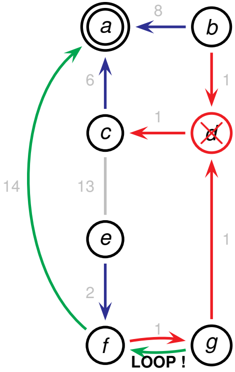
Towards Multi-Dimensional Increments



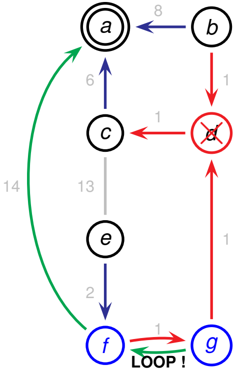
Towards Multi-Dimensional Increments



Towards Multi-Dimensional Increments



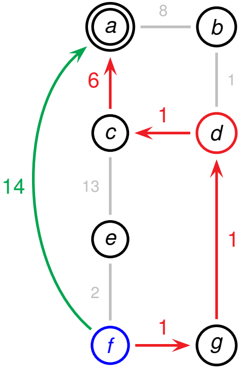
Towards Multi-Dimensional Increments



Vector of minimum increments such that a node x uses a new path, not through n, to reach d.

$$\Delta_d^n(x)[i] = C'(x,d) - C(x,i,d)$$

Towards Multi-Dimensional Increments

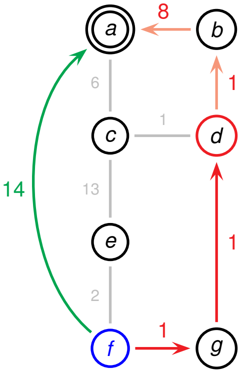


Vector of minimum increments such that a node x uses a new path, not through n, to reach d.

$$\Delta_d^n(x)[i] = C'(x,d) - C(x,i,d)$$

- $\Delta_a^d(f) = \begin{pmatrix} 14 - (1+1+1+6) \\ \end{pmatrix}$

Towards Multi-Dimensional Increments

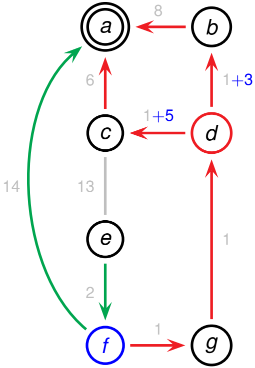


Vector of minimum increments such that a node x uses a new path, not through n, to reach d.

$$\Delta_d^n(x)[i] = C'(x,d) - C(x,i,d)$$

- $\Delta_a^d(f) = \begin{pmatrix} 14 - (1+1+1+6) \\ 14 - (1+1+1+8) \end{pmatrix}$

Towards Multi-Dimensional Increments

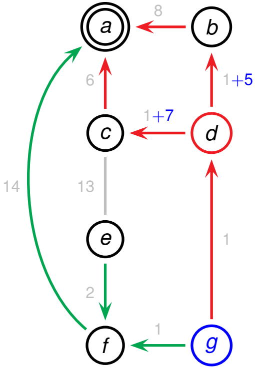


Vector of minimum increments such that a node x uses a new path, not through n, to reach d.

$$\Delta_d^n(x)[i] = C'(x,d) - C(x,i,d)$$

- $\Delta_a^d(f) = \begin{pmatrix} 14 - (1+1+1+6) \\ 14 - (1+1+1+8) \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \end{pmatrix}$

Towards Multi-Dimensional Increments

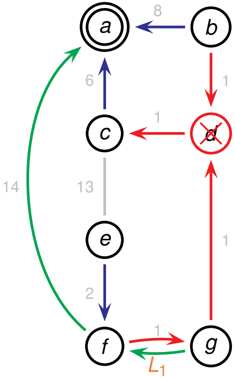


Vector of minimum increments such that a node x uses a new path, not through n, to reach d.

$$\Delta_d^n(x)[i] = C'(x,d) - C(x,i,d)$$

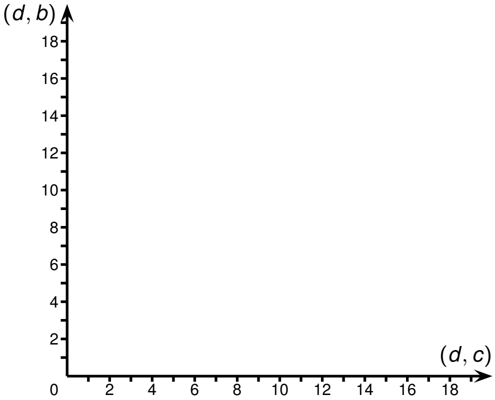
- $\Delta_a^d(f) = \begin{pmatrix} 14 - (1+1+1+6) \\ 14 - (1+1+1+8) \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \end{pmatrix}$
- $\Delta_a^d(g) = \begin{pmatrix} 15 - 8 \\ 15 - 10 \end{pmatrix} = \begin{pmatrix} 7 \\ 5 \end{pmatrix}$

Modeling Loops as Vectorial Constraints



$$\Delta_a^d(f) = \begin{pmatrix} 5 \\ 3 \end{pmatrix}$$

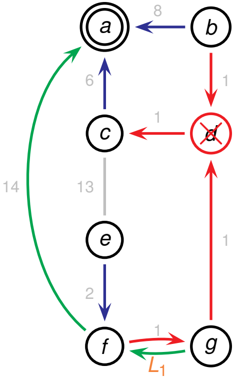
$$\Delta_a^d(g) = \begin{pmatrix} 7 \\ 5 \end{pmatrix}$$



Constraint c associated to a given a loop L.

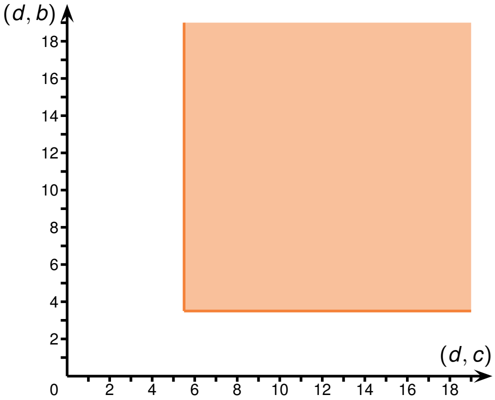
$$c := (\min_{\forall x \in L}(\Delta(x)), \max_{\forall x \in L}(\Delta(x)))$$

Modeling Loops as Vectorial Constraints



$$\Delta_a^d(f) = \begin{pmatrix}5\\3\end{pmatrix}$$

$$\Delta_a^d(g) = \begin{pmatrix}7\\5\end{pmatrix}$$

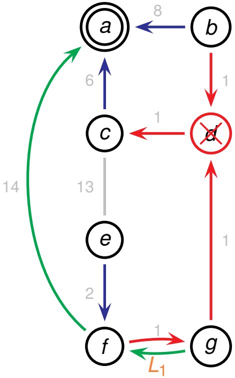


Constraint c associated to a given a loop L.

$$c := (\min_{\forall x \in L}(\Delta(x)), \max_{\forall x \in L}(\Delta(x)))$$

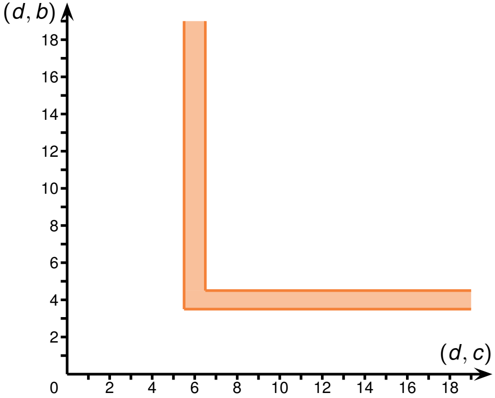
$$c_1 = \left(\begin{pmatrix}5\\3\end{pmatrix}\right)$$

Modeling Loops as Vectorial Constraints



$$\Delta_a^d(f) = \begin{pmatrix} 5 \\ 3 \end{pmatrix}$$

$$\Delta_a^d(g) = \begin{pmatrix} 7 \\ 5 \end{pmatrix}$$

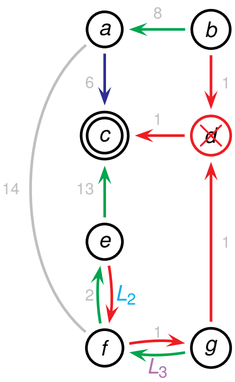


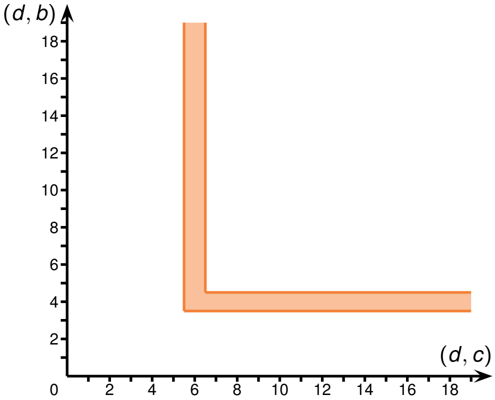
Constraint c associated to a given a loop L.

$$c := (\min_{\forall x \in L}(\Delta(x)), \max_{\forall x \in L}(\Delta(x)))$$

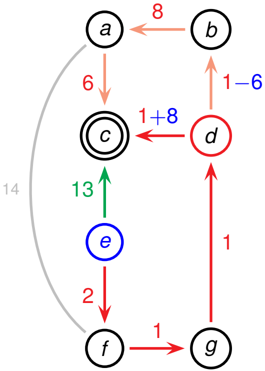
$$c_1 = \left(\begin{pmatrix} 5 \\ 3 \end{pmatrix}, \begin{pmatrix} 7 \\ 5 \end{pmatrix} \right)$$

Modeling Loops as Vectorial Constraints (2)

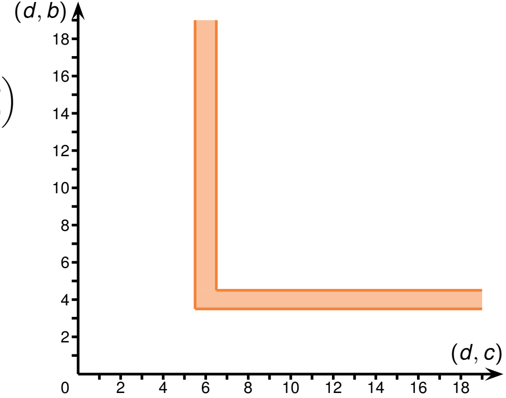




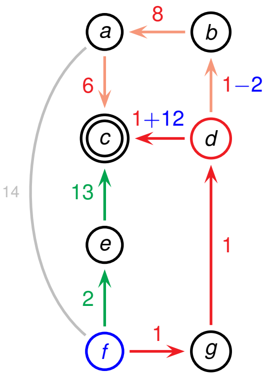
Modeling Loops as Vectorial Constraints (2)



$$\Delta_2^3(e) = \begin{pmatrix} 8 \\ 0 \end{pmatrix}$$

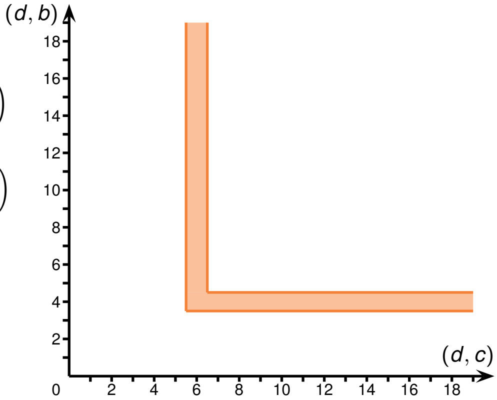


Modeling Loops as Vectorial Constraints (2)

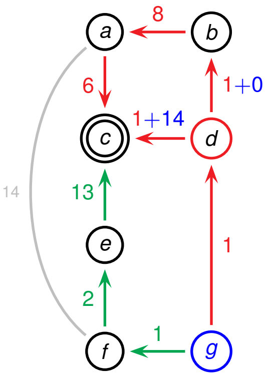


$$\Delta_2^3(e) = \begin{pmatrix} 8 \\ 0 \end{pmatrix}$$

$$\Delta_2^3(f) = \begin{pmatrix} 12 \\ 0 \end{pmatrix}$$



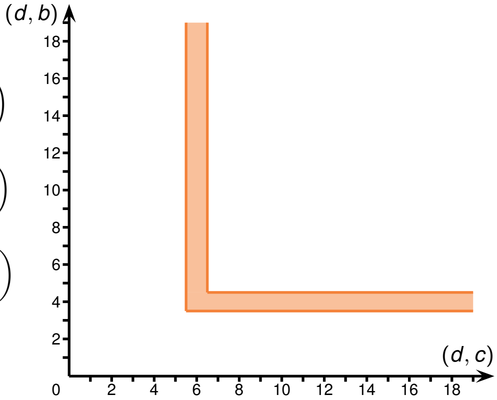
Modeling Loops as Vectorial Constraints (2)



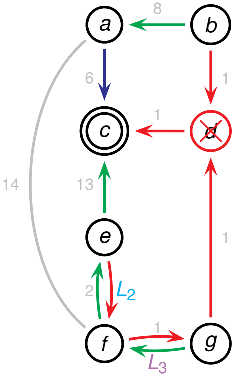
$$\Delta_2^3(e) = \begin{pmatrix} 8 \\ 0 \end{pmatrix}$$

$$\Delta_2^3(f) = \begin{pmatrix} 12 \\ 0 \end{pmatrix}$$

$$\Delta_2^3(g) = \begin{pmatrix} 14 \\ 0 \end{pmatrix}$$



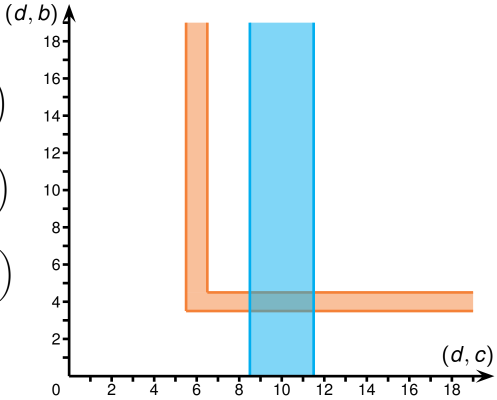
Modeling Loops as Vectorial Constraints (2)



$$\Delta_2^3(e) = \begin{pmatrix} 8 \\ 0 \end{pmatrix}$$

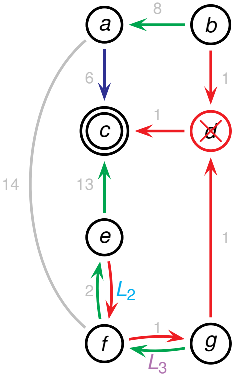
$$\Delta_2^3(f) = \begin{pmatrix} 12 \\ 0 \end{pmatrix}$$

$$\Delta_2^3(g) = \begin{pmatrix} 14 \\ 0 \end{pmatrix}$$



$$c_2 = \left(\begin{pmatrix} 8 \\ 0 \end{pmatrix}, \begin{pmatrix} 12 \\ 0 \end{pmatrix} \right)$$

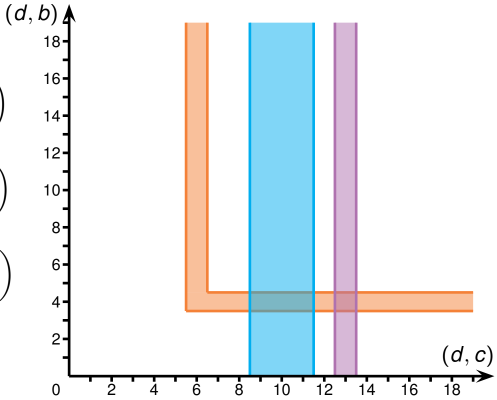
Modeling Loops as Vectorial Constraints (2)



$$\Delta_2^3(e) = \begin{pmatrix} 8 \\ 0 \end{pmatrix}$$

$$\Delta_2^3(f) = \begin{pmatrix} 12 \\ 0 \end{pmatrix}$$

$$\Delta_2^3(g) = \begin{pmatrix} 14 \\ 0 \end{pmatrix}$$

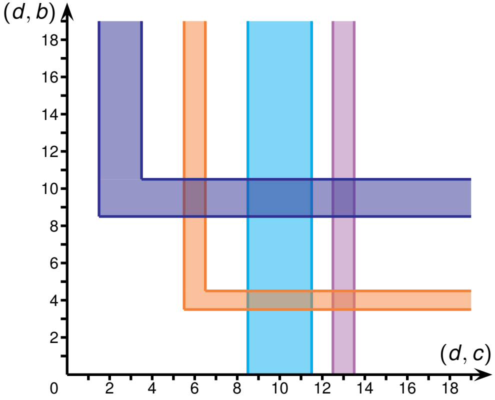


$$c_2 = \left(\begin{pmatrix} 8 \\ 0 \end{pmatrix}, \begin{pmatrix} 12 \\ 0 \end{pmatrix} \right)$$

$$c_3 = \left(\begin{pmatrix} 12 \\ 0 \end{pmatrix}, \begin{pmatrix} 14 \\ 0 \end{pmatrix} \right)$$

Modeling Loops as Vectorial Constraints (3)

$$c_4 = \left(\begin{pmatrix} 1 \\ 8 \end{pmatrix}, \begin{pmatrix} 4 \\ 11 \end{pmatrix} \right)$$

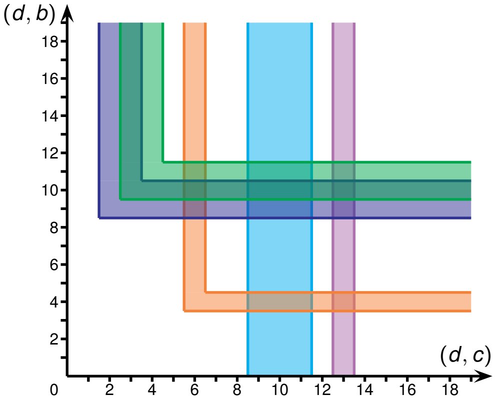


An update sequence s avoids a loop L if and only if s contains at least one vector meeting the corresponding constraint.

Modeling Loops as Vectorial Constraints (3)

$$c_4 = \left(\begin{pmatrix} 1 \\ 8 \end{pmatrix}, \begin{pmatrix} 4 \\ 11 \end{pmatrix} \right)$$

$$c_5 = \left(\begin{pmatrix} 2 \\ 9 \end{pmatrix}, \begin{pmatrix} 5 \\ 12 \end{pmatrix} \right)$$

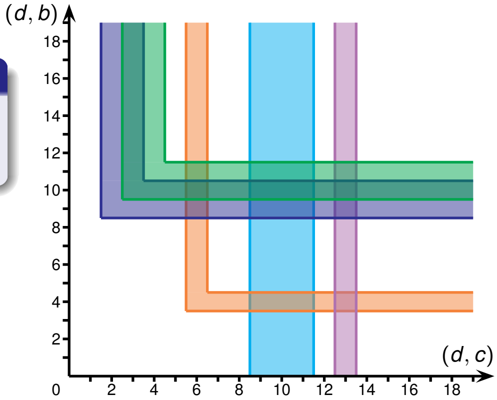


An update sequence s avoids a loop L if and only if s contains at least one vector meeting the corresponding constraint.

Defining Safe Weight Increment Sequences

Greedy Backward Algorithm (GBA)

At each step, retrieve the maximum value on each index among the lower bounds of the remaining constraints.

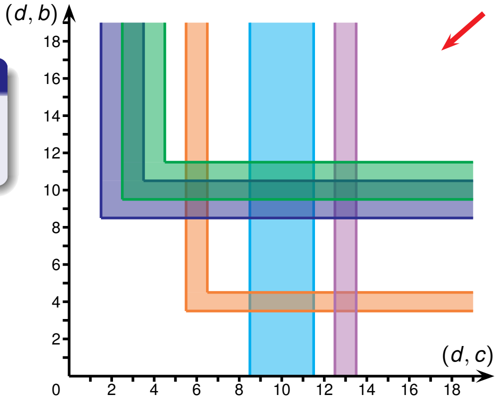


Defining Safe Weight Increment Sequences

Greedy Backward Algorithm (GBA)

At each step, retrieve the maximum value on each index among the lower bounds of the remaining constraints.

Sequence:

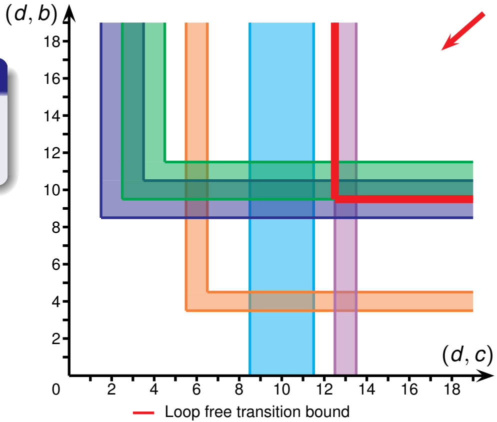


Defining Safe Weight Increment Sequences

Greedy Backward Algorithm (GBA)

At each step, retrieve the maximum value on each index among the lower bounds of the remaining constraints.

Sequence:



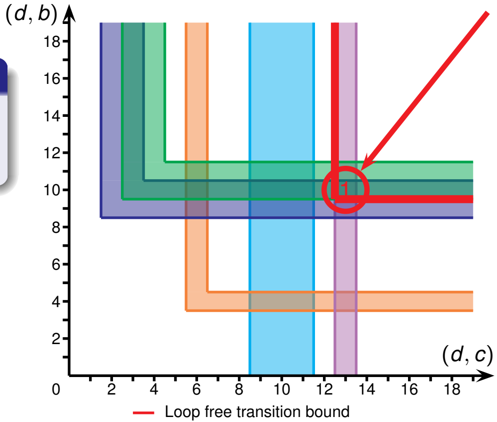
Defining Safe Weight Increment Sequences

Greedy Backward Algorithm (GBA)

At each step, retrieve the maximum value on each index among the lower bounds of the remaining constraints.

Sequence:

1. C_3, C_4, C_5



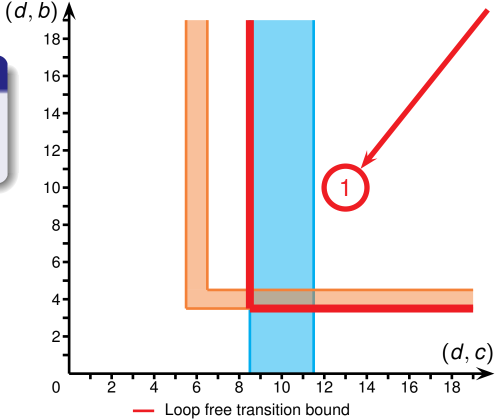
Defining Safe Weight Increment Sequences

Greedy Backward Algorithm (GBA)

At each step, retrieve the maximum value on each index among the lower bounds of the remaining constraints.

Sequence:

1. C_3, C_4, C_5



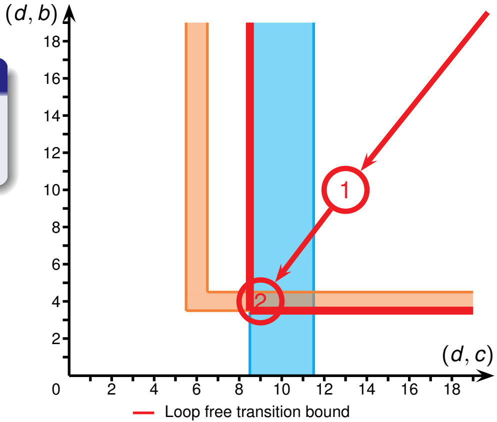
Defining Safe Weight Increment Sequences

Greedy Backward Algorithm (GBA)

At each step, retrieve the maximum value on each index among the lower bounds of the remaining constraints.

Sequence:

1. C_3, C_4, C_5
2. C_1, C_2



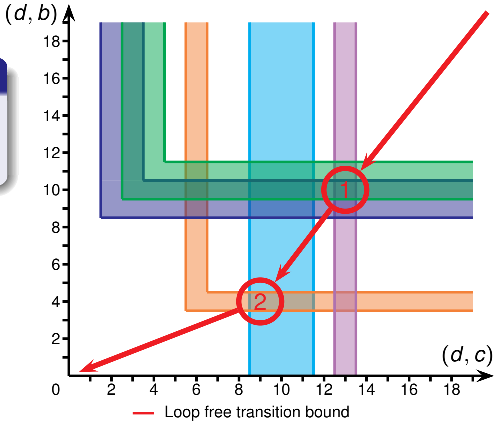
Defining Safe Weight Increment Sequences

Greedy Backward Algorithm (GBA)

At each step, retrieve the maximum value on each index among the lower bounds of the remaining constraints.

Sequence:

1. C_3, C_4, C_5
2. C_1, C_2



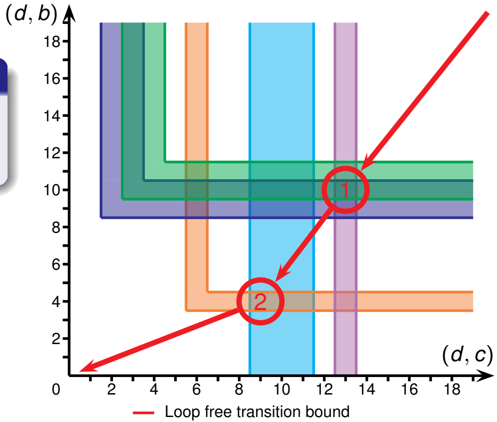
Defining Safe Weight Increment Sequences

Greedy Backward Algorithm (GBA)

At each step, retrieve the maximum value on each index among the lower bounds of the remaining constraints.

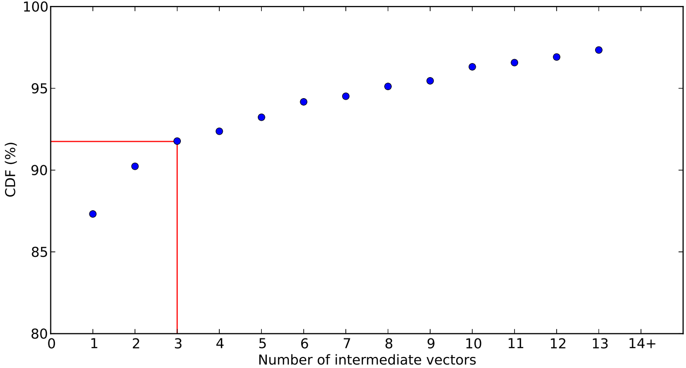
Sequence:

1. C_3, C_4, C_5
2. C_1, C_2



Given a set of loop-constraints, GBA computes a minimal sequence of intermediate increments preventing convergence loops.

Sequence Lengths on a Real ISP Network



- Graph with more than 1000 nodes and 4000 links;
- 90% of the nodes requiring at most 3 intermediate steps.

Conclusion

- Minimal solution;
- Time efficient algorithm;
- Generic approach;

Conclusion

- ✓ Minimal solution;
  - ▷ Shortest possible sequences;
- Time efficient algorithm;
- Generic approach;

Conclusion

- ✓ Minimal solution;
  - ▷ Shortest possible sequences;
- ✓ Time efficient algorithm;
  - ▷ Polynomial complexity;
- Generic approach;

Conclusion

- ✓ Minimal solution;
  - ▷ Shortest possible sequences;
- ✓ Time efficient algorithm;
  - ▷ Polynomial complexity;
- ✓ Generic approach;
  - ▷ Covers the single link problem.

Conclusion

- ✓ Minimal solution;
  - ▷ Shortest possible sequences;
- ✓ Time efficient algorithm;
  - ▷ Polynomial complexity;
- ✓ Generic approach;
  - ▷ Covers the single link problem.

Future works

- ▷ Implementation in Quagga;
- ▷ Evaluation in a real network.

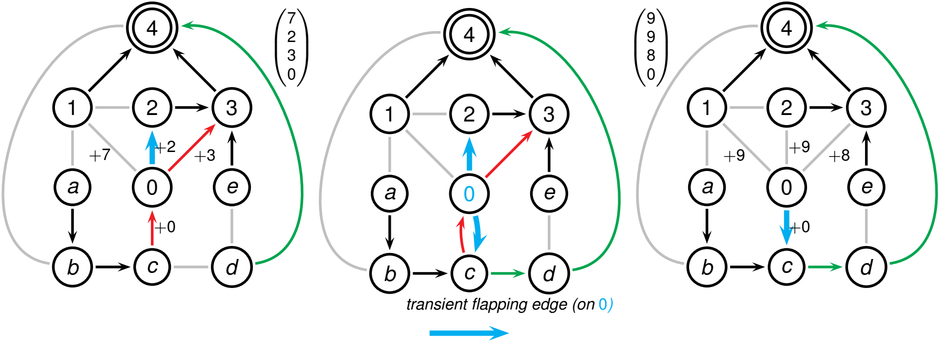
Thank you for your attention.

5 Appendix

Transient loop induced by route flapping

$\rightarrow$ $RSPDAG_1(4)$
Intermediate routing state towards 4 considering the first vector

$\rightarrow$ $RSPDAG_2(4)$
Intermediate routing state towards 4 considering the second vector



$S_{GBA} = \begin{pmatrix}7\\2\\3\\0\end{pmatrix}, \begin{pmatrix}9\\9\\8\\0\end{pmatrix}$

$S_{FF1} = \begin{pmatrix}3\\2\\3\\0\end{pmatrix}, \begin{pmatrix}7\\4\\5\\0\end{pmatrix}, \begin{pmatrix}9\\9\\8\\0\end{pmatrix}$ $S_{FF2} = \begin{pmatrix}7\\2\\3\\0\end{pmatrix}, \begin{pmatrix}9\\9\\8\\3\end{pmatrix}$