Simple Temporal Networks and some its Extensions

Version 162, 2019–12–04 14:43:39Z

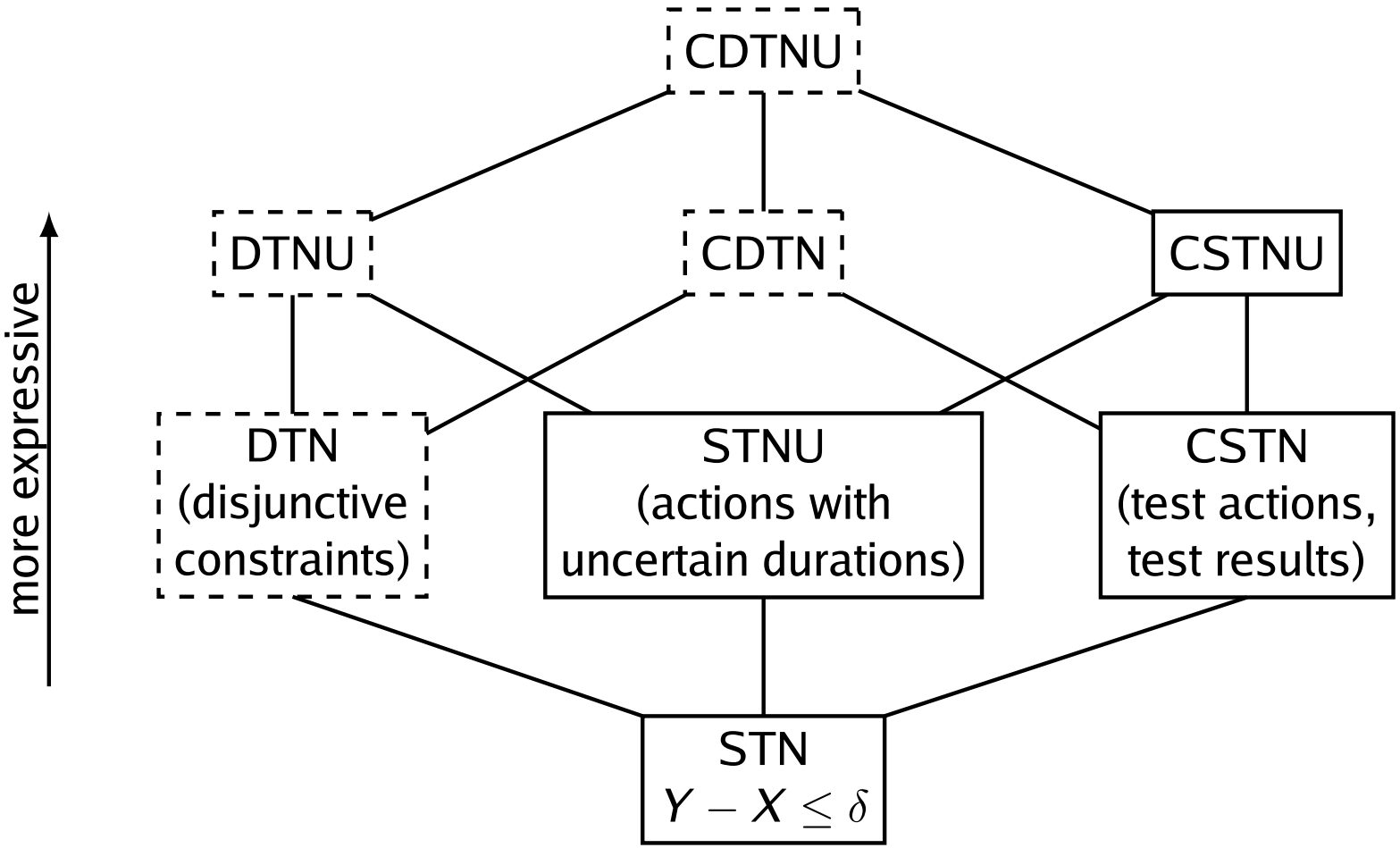
Roberto Posenato

Department of Computer Science, University of Verona

November 2019

Credits: this presentation is based on Luke Hunsberger and Roberto Posenato slides

Outline



- This lecture does not present dashed types.

Simple Temporal Network
Motivating Example

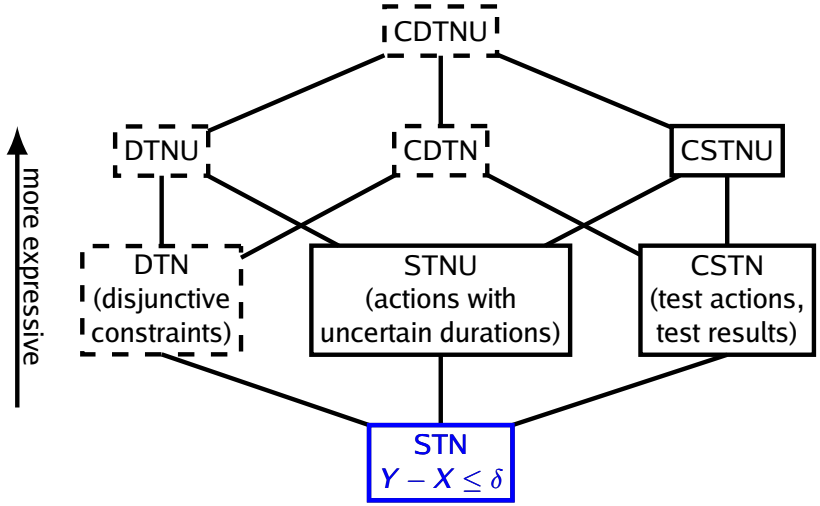
- Given a plan of a fly from New York to Rome,
- Determine if the plan is **consistent**,
- If it is consistent, determine a **temporal schedule**.

Fly from New York to Rome

- Leave New York after 4 p.m., June 8
- Return to New York before 10 p.m., June 18
- Away from New York no more than 7 days
- In Rome at least 5 days
- Return flight lasts no more than 7 hours

Simple Temporal Network (STN)*
Description

- Includes time–points and temporal constraints of only one kind.
- Flexible: Time–points may "float"; not "nailed down" until they are *executed*
- Efficient algorithms for determining consistency, managing real–time execution, and handling new constraints



* [Dechter et al., 1991]

Simple Temporal Network*
Definition

Definition 1 (Simple Temporal Network (STN))

A *Simple Temporal Network (STN)* is a pair, $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, where:

- $\mathcal{T}$ is a set of real–valued variables called time–points: $\{X_1, \ldots, X_n \mid X_i \in \mathbb{R}\}$; and
- $\mathcal{C}$ is a set of binary constraints, each of the form:

$$X_j - X_i \leq \delta$$

where $X_j, X_i \in \mathcal{T}$ and $\delta \in \mathbb{R}$.

- Time–points represent events.
- Each binary constraint represents the maximal temporal distance between the first time point and the second one.
- There is only one time unit.

*[Dechter et al., 1991]

Simple Temporal Network
The *Zero* Time–Point, Z

- It is useful to have one time–point, called Z, whose value is fixed at 0.
- Binary constraints involving Z are equivalent to unary constraints.

Example 1

$$X - Z \leq 7 \iff X \leq 7$$

$$Z - X \leq -3 \iff X \geq 3$$

- A *solution* to an STN $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ is a complete set of assignments to the time–points in $\mathcal{T}$:

$$\{X_1 = t_1,\ X_2 = t_2,\ \ldots,\ X_n = t_n\}$$

  that together satisfy all of the constraints in $\mathcal{C}$.
- An STN with at least one solution is *consistent.*
- STNs with identical solution sets are *equivalent*.

Simple Temporal Network
STN for Travel Example

$\mathcal{T} = \{Z, X_1, X_2, X_3, X_4\}$, where Z = Noon, June 8

$$\mathcal{C} = \begin{cases} Z - X_1 \leq -4 & X_1\text{=leave–time NYC (Lv NYC after 4 p.m., June 8)} \\ X_4 - Z \leq 250 & X_4\text{=return–time NYC (Ret NYC by 10 p.m., June 18)} \\ X_4 - X_1 \leq 168 & \text{(Gone no more than 7 days)} \\ X_2 - X_3 \leq -120 & X_2(X_3)\text{=arrive(leave)–time Rome (In Rome at least 5 days)} \\ X_4 - X_3 \leq 8 & \text{(Return flight less than 8 hrs)} \\ X_1 - X_2 \leq -7 & \text{(Flight requires at least 7 hrs)} \\ X_3 - X_4 \leq -7 & \text{(Return flight requires at least 7 hrs)} \end{cases}$$

Simple Temporal Network
Graph for an STN*

The *graph* for an STN, $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, is a graph, $\mathcal{G} = (\mathcal{T}, \mathcal{E})$, where:

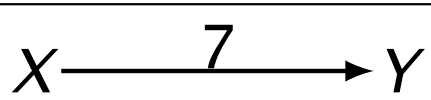
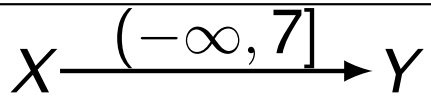
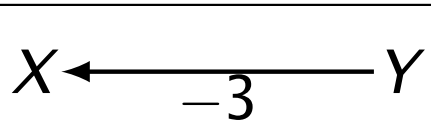
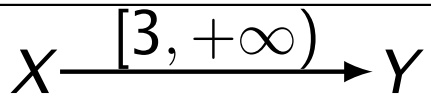
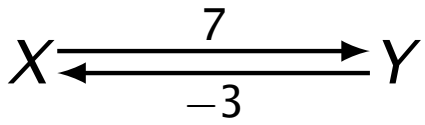
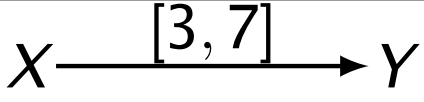
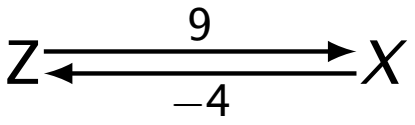
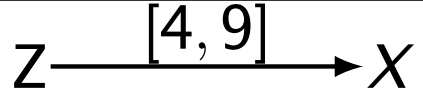
$$\text{Time–points in } \mathcal{S} \iff \text{nodes in } \mathcal{G}$$

$$\text{Constraints in } \mathcal{C} \iff \text{edges in } \mathcal{E}\text{:}$$

$$Y - X \leq \delta \iff X \xrightarrow{\delta} Y$$

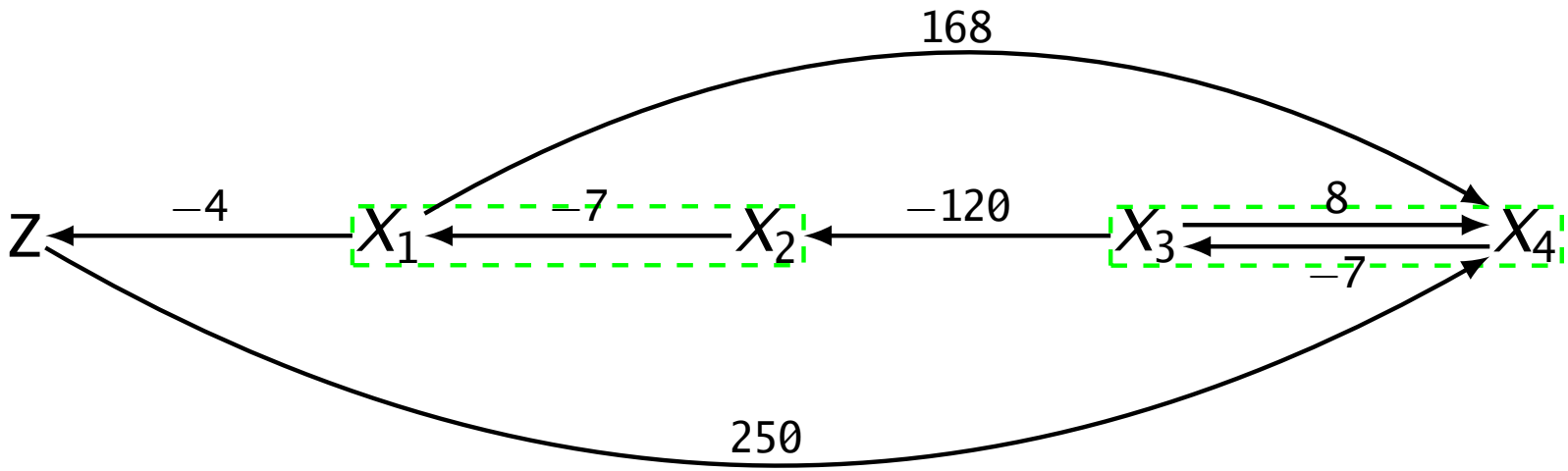
*[Dechter et al., 1991]

Simple Temporal Network
Graphical Representations

Constraint(s)	Edge(s)	Interval Notation
$Y - X \leq 7$		
$X - Y \leq -3$ ($\Leftrightarrow 3 \leq Y - X$)		
$3 \leq Y - X \leq 7$		
$4 \leq X \leq 9$		

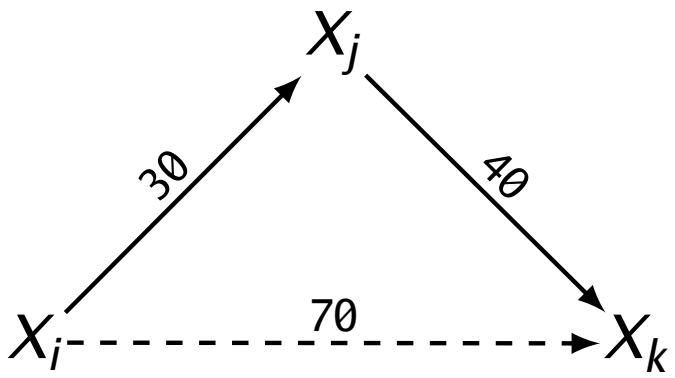
Simple Temporal Network
Graph for Airline Scenario

$$\begin{cases} Z - X_1 \leq -4, & X_4 - Z \leq 250 \\ X_4 - X_1 \leq 168, & X_2 - X_3 \leq -120 \\ X_4 - X_3 \leq 8, & X_1 - X_2 \leq -7 \\ X_3 - X_4 \leq -7 & \end{cases}$$



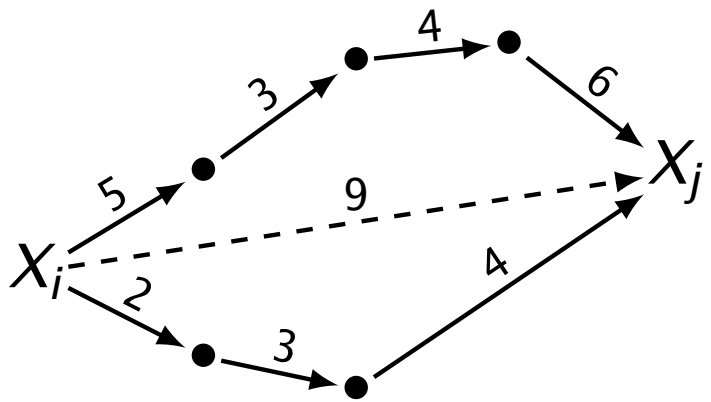
Explicit constraints combine (propagate) to form implicit constraints:

$$\begin{array}{rcl} X_j - X_i & \leq & 30 \\ X_k - X_j & \leq & 40 \\ \hline X_k - X_i & \leq & 70 \end{array}$$



Simple Temporal Network
Chains of Constraints as Paths

- Chains of constraints correspond to **paths** in the graph.
- **Stronger** constraints correspond to **shorter** paths.

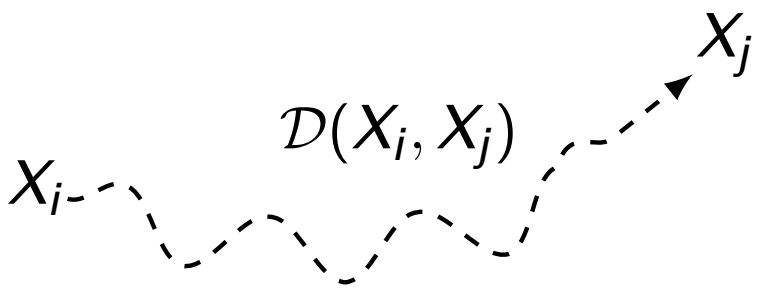


Simple Temporal Network
Distance Matrix*

Definition 2 (Distance Matrix)

The Distance Matrix for an STN, $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, is a matrix $\mathcal{D}$ defined by:

$\mathcal{D}(X_i, X_j) =$ Length of Shortest Path from X_i to X_j in the graph for $\mathcal{S}$



- The strongest implicit constraint on X_i and X_j in $\mathcal{S}$ is:

$$X_j - X_i \leq \mathcal{D}(X_i, X_j)$$

*[Dechter et al., 1991]

Simple Temporal Network
Distance Matrix

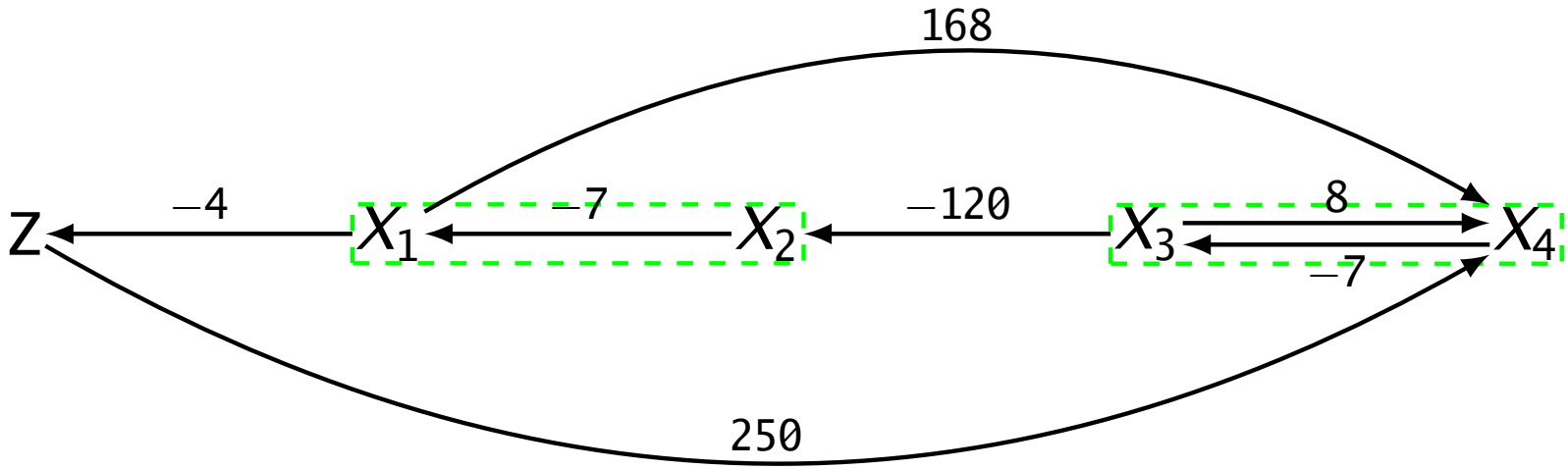
- The strongest implicit constraint on X_i and X_j in $\mathcal{S}$ is:

$$X_j - X_i \leq \mathcal{D}(X_i, X_j)$$

- $\mathcal{D}$ is the *All–Pairs, Shortest–Path* (APSP) Matrix for the STN's graph.*
  - Floyd–Warshall Algorithm
  - Johnson's Algorithm

*[Cormen et al., 2009]

Simple Temporal Network
Travel Scenario's Distance Matrix



$\mathcal{D}$	Z	X_1	X_2	X_3	X_4
Z	0	116	123	243	250
X_1	−4	0	41	161	168
X_2	−11	−7	0	154	161
X_3	−131	−127	−120	0	8
X_4	−138	−134	−127	−7	0

Gray cells contain the original values.

Simple Temporal Network
Fundamental Theorem of STNs*

For an STN $\mathcal{S}$, with graph $\mathcal{G}$, and distance matrix $\mathcal{D}$, the following are equivalent:

- $\mathcal{S}$ is consistent
- $\mathcal{D}$ has non–negative values on main diagonal
- $\mathcal{G}$ has no negative–length loops

Moreover, any consistent STN $\mathcal{S}$ is backtrack–free relative to the constraints in its distance matrix.

*[Dechter et al., 1991]

Simple Temporal Network
Finding a solution for an STN

- $\mathcal{D}$ has all necessary information.
- Time window for any X_i: $[-\mathcal{D}(X_i, Z), \mathcal{D}(Z, X_i)]$
- Two solutions are always given by free:
  1. Earlier execution–time solution:
     $X_1 = -\mathcal{D}(X_1, Z), X_2 = -\mathcal{D}(X_2, Z), \ldots, X_n = -\mathcal{D}(X_n, Z)$;
  2. Latest execution–time solution:
     $X_1 = \mathcal{D}(Z, X_1), X_2 = \mathcal{D}(Z, X_2), \ldots, X_n = \mathcal{D}(Z, X_n)$.
- Simple algorithm to find a different solution:
  - Pick any time–point that doesn't yet have a value;
  - Give it a value from its time–window;
  - Update $\mathcal{D}$; $\Leftarrow$ expensive ...
  - Repeat until all time–points have values.

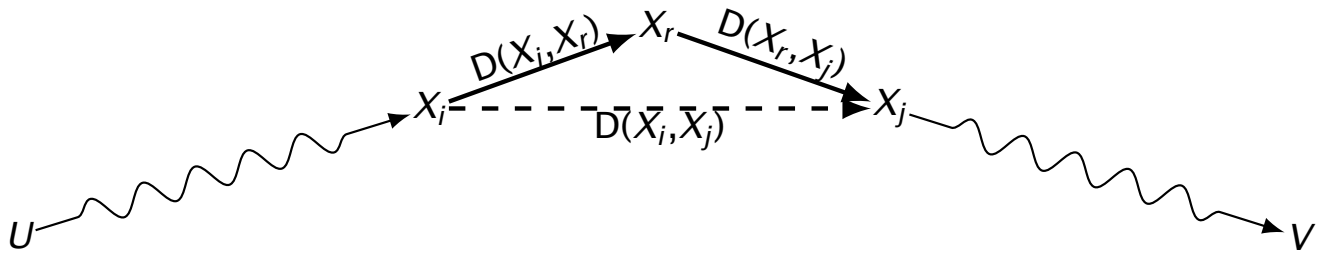
*[Dechter et al., 1991]

Simple Temporal Network
Computing $\mathcal{D}$ from Scratch*

- $\mathcal{D}$ is the *All–Pairs, Shortest–Paths* (APSP) Matrix for $\mathcal{G}$.
- If $\mathcal{S}$ has n time–points and m constraints:
  - Floyd–Warshall Algorithm: $O(n^3)$
  - Johnson's Algorithm: $O(n^2 \log n + mn)$
    - uses Bellman–Ford and Dijkstra

*[Cormen et al., 2009]

Floyd–Warshall Algorithm*



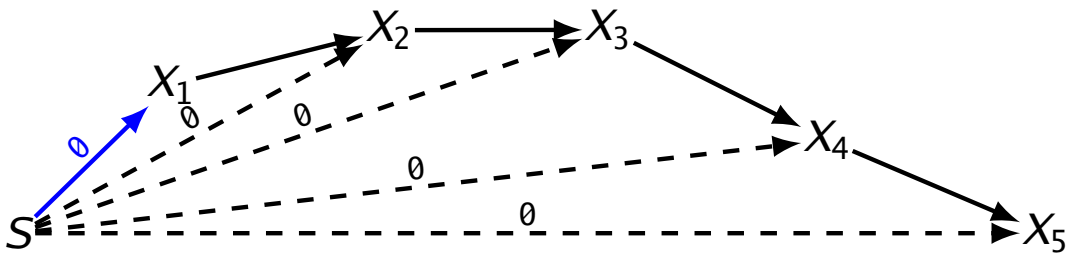
Algorithm 1: Flowd–Warshall(D)

Initialize D(_,_) using edge–weights;
for $r := 1$ *to* n **do**
  for $i := 1$ *to* n **do**
    for $j := 1$ *to* n **do**
      $D(X_i,X_j) := \min\{D(X_i,X_j), D(X_i,X_r) + D(X_r,X_j)\}$;

- The r^{th} cycle finds all minimal paths having r intermediate nodes at most;
- Time complexity $\Theta(n^3)$

*[Cormen et al., 2009]

Bellman–Ford Algorithm



Algorithm 2: Bellman–Ford()

foreach X_i **do**
  $d(X_i) := \infty$
$d(S) := 0$;
for $i := 1$ *to* $n - 1$ **do**
  foreach *edge* (X_i, δ, X_j) **do**
    $d(X_j) := \min\{d(X_j), d(X_i) + \delta\}$;
foreach *edge* (X_i, δ, X_j) **do**
  if $d(X_j) > d(X_i) + \delta$ **then return** *NO*;
return *{D(S,X$_1$),..., D(S,X$_n$)}*;

- Time complexity $O(nm)$

Dijkstra's SSSP Algorithm
Single–Source Shortest–Paths

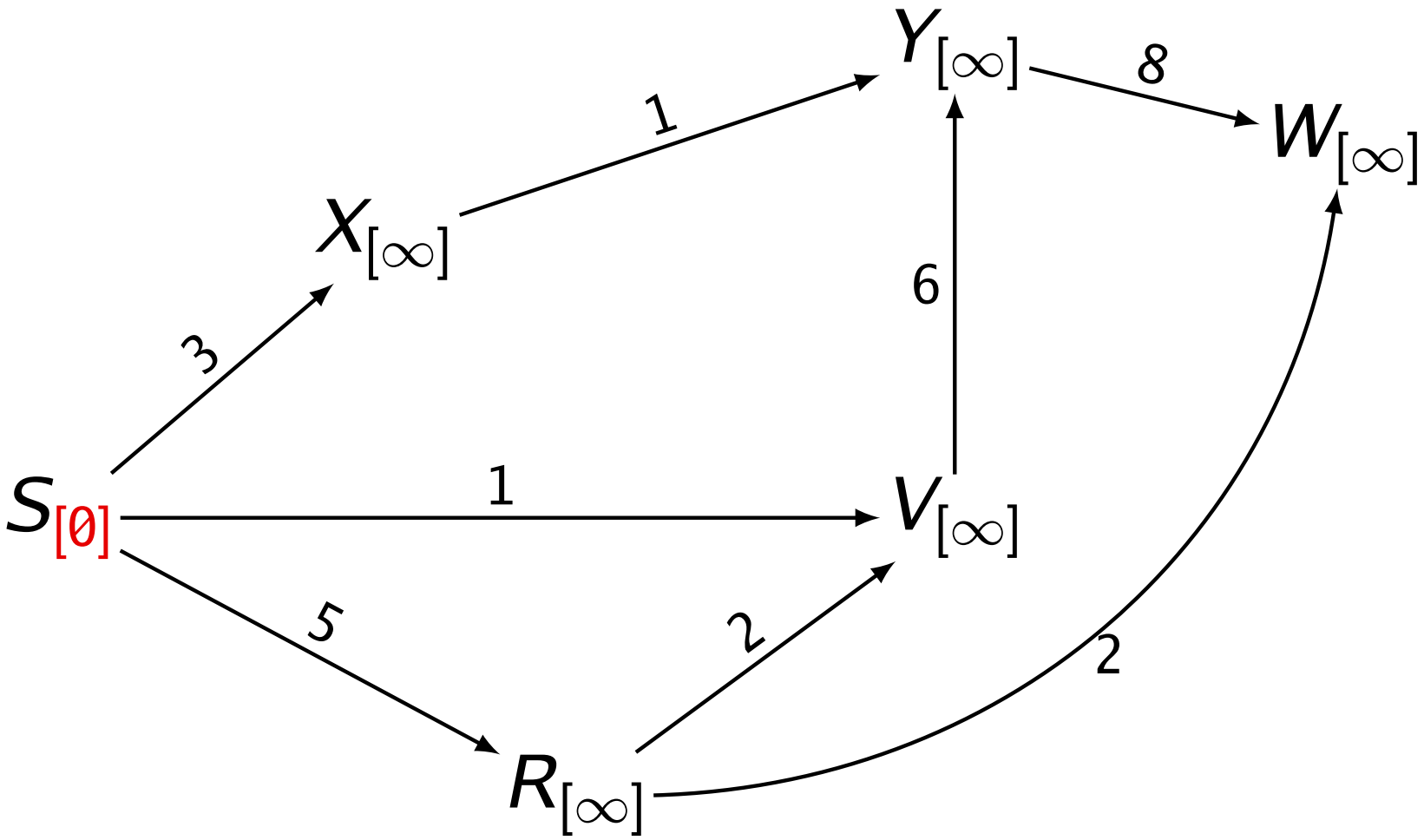
- Works on STN graphs with non–negative edges
- For a given source node S, computes $\mathcal{D}(S, X)$ for all X

Algorithm 3: Dijkstra()

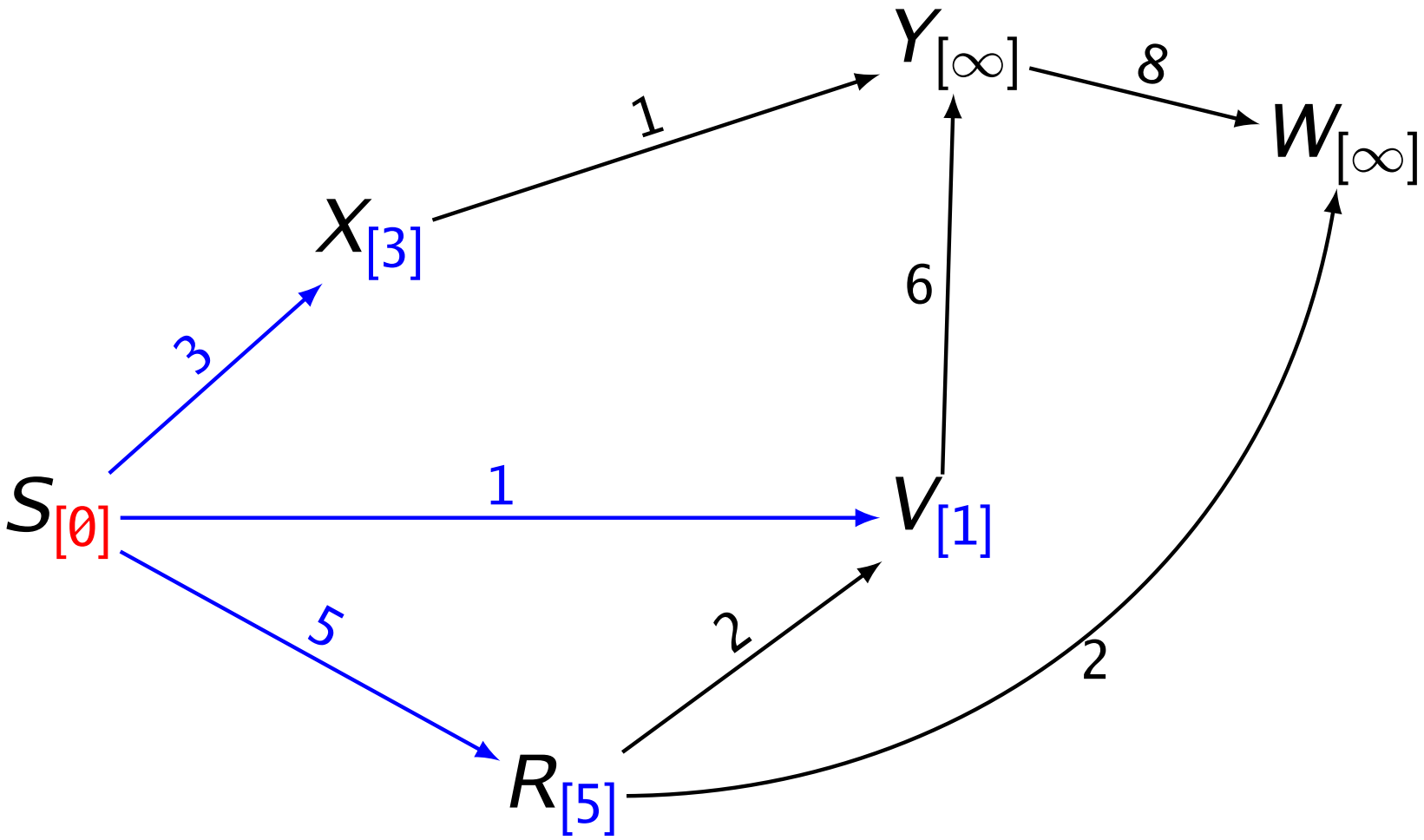
foreach X_i **do**
  $d(X_i) := \infty$
$d(S) := 0$;
$\mathcal{Q}$:= an empty priority queue;
Insert S into $\mathcal{Q}$ with priority 0;
while $\mathcal{Q} \neq \emptyset$ **do**
  X :=ExtractMinFrom($\mathcal{Q}$);
  foreach *edge* (X, δ, Y) **do**
    $d(Y) := \min\{d(Y), d(X) + \delta\}$;
return *{D(S,X$_1$),..., D(S,X$_n$)}*

- Time complexity $O(m + n \log n)$ if using *Fibonacci Heap* for priority queue

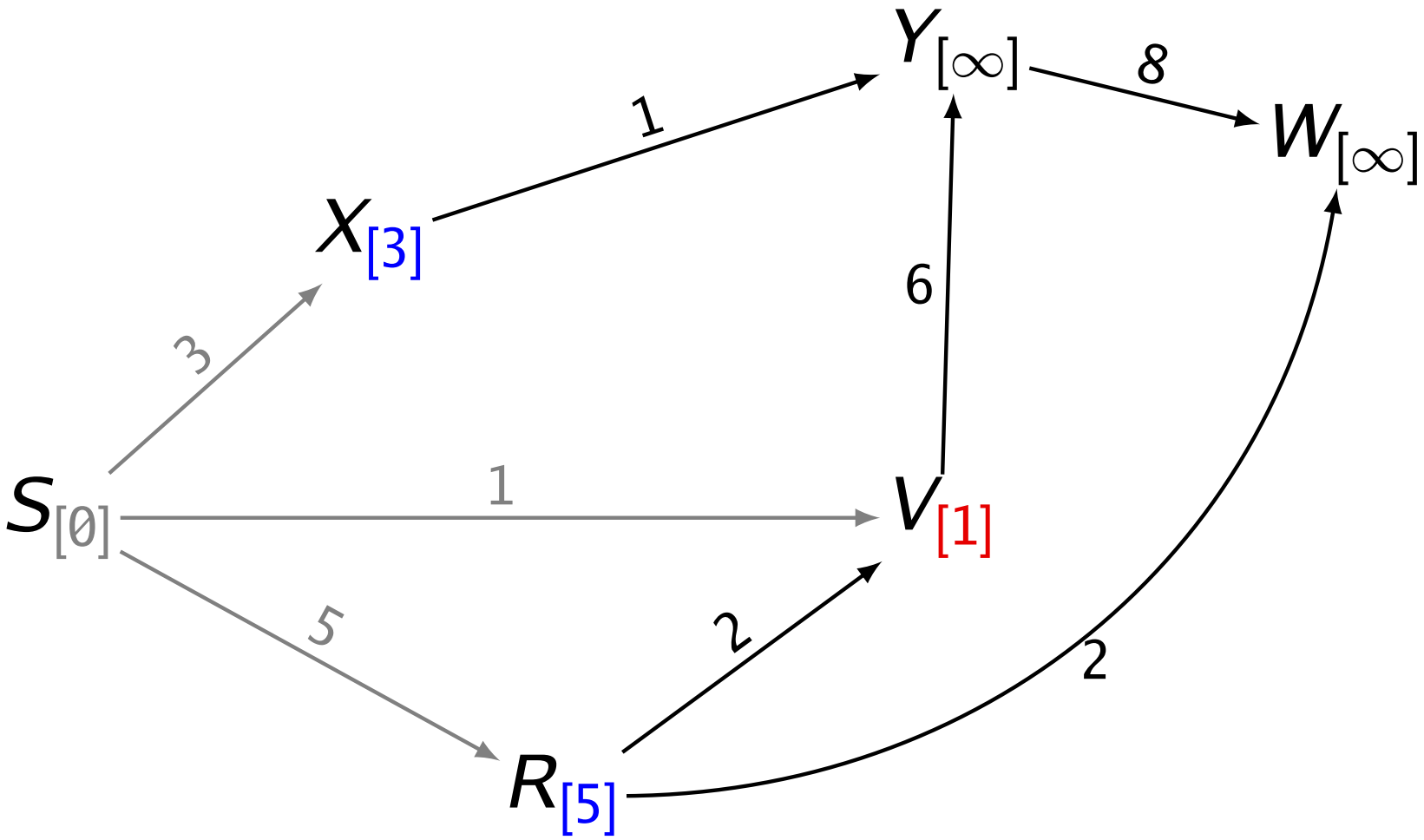
Dijkstra Example



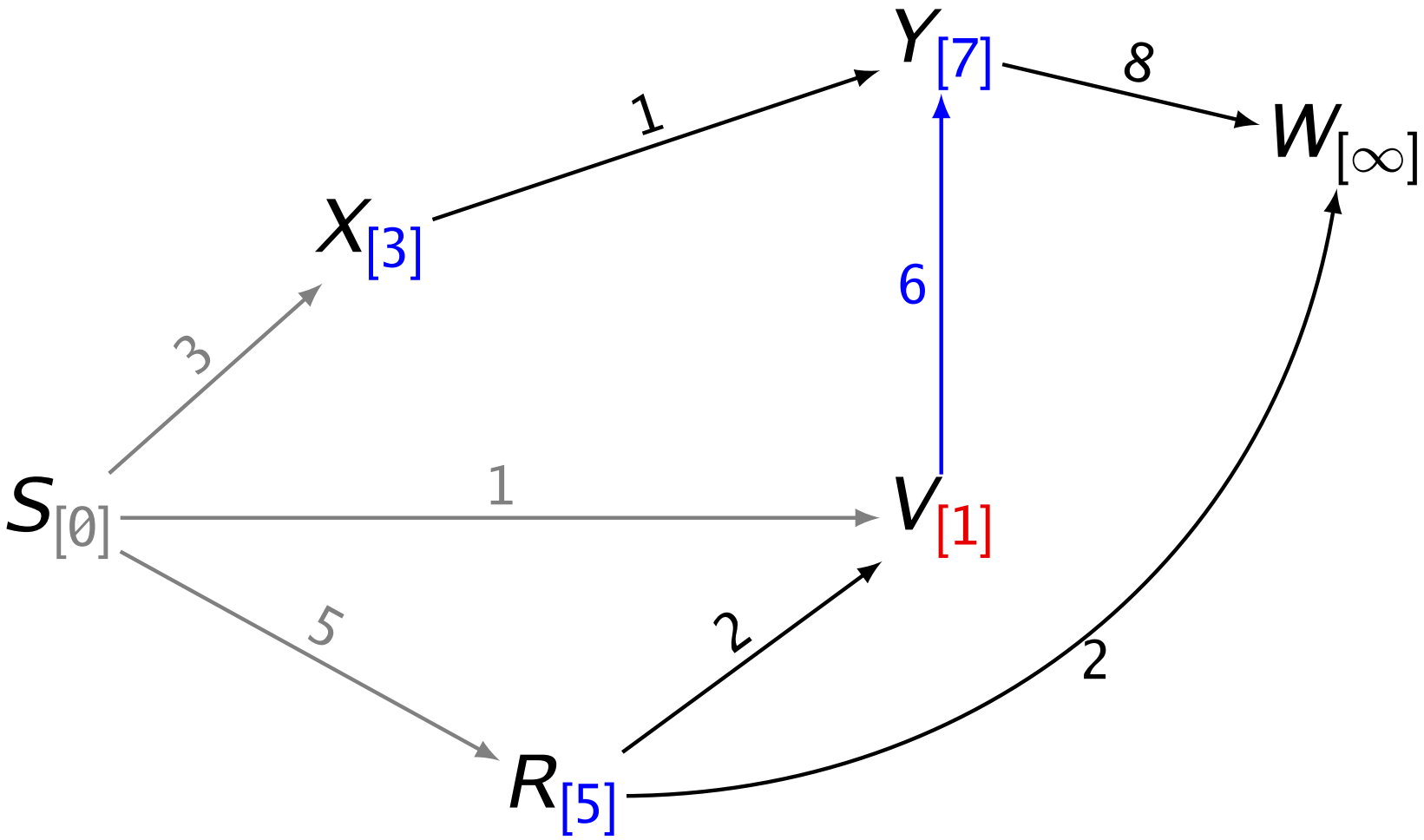
Dijkstra Example



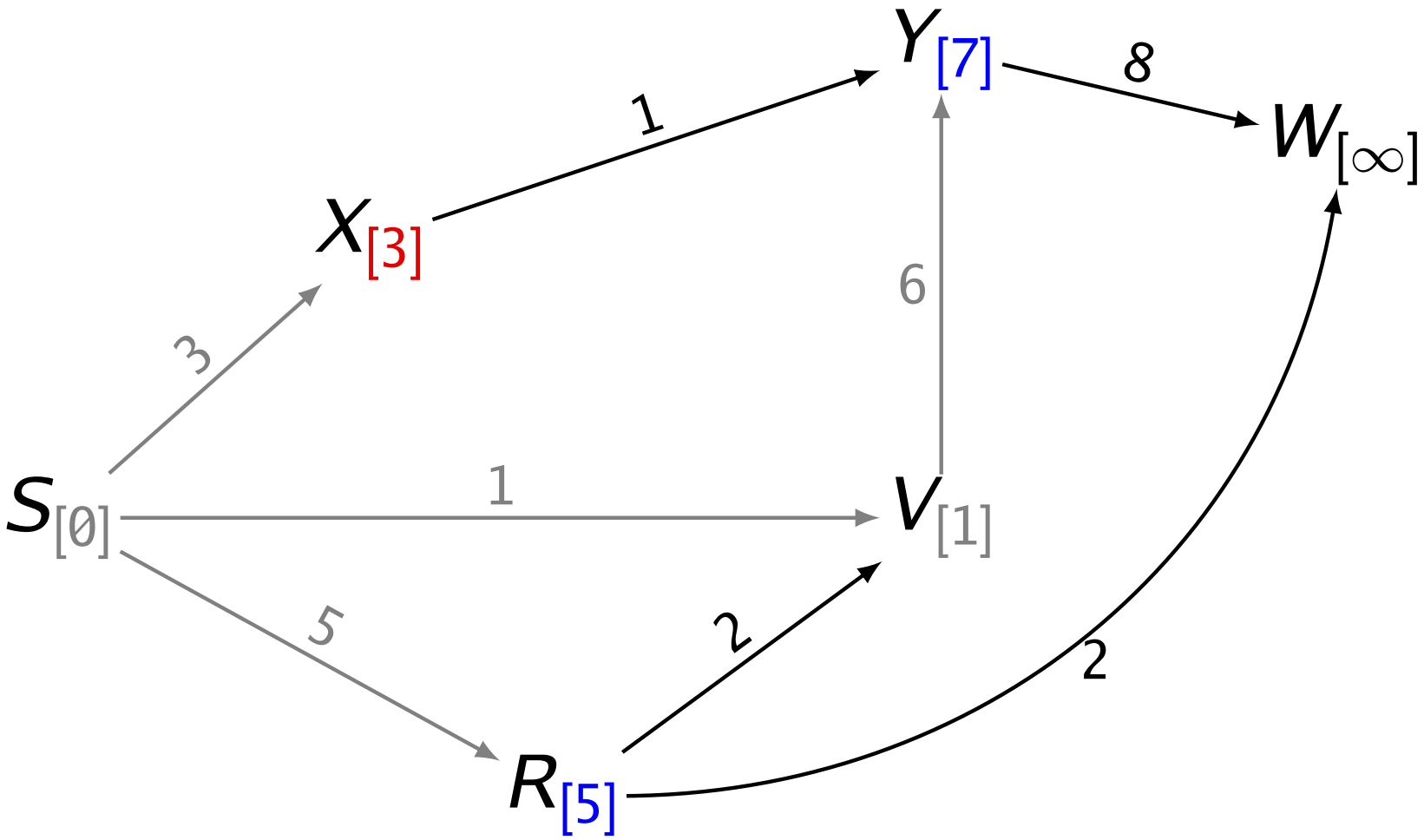
Dijkstra Example



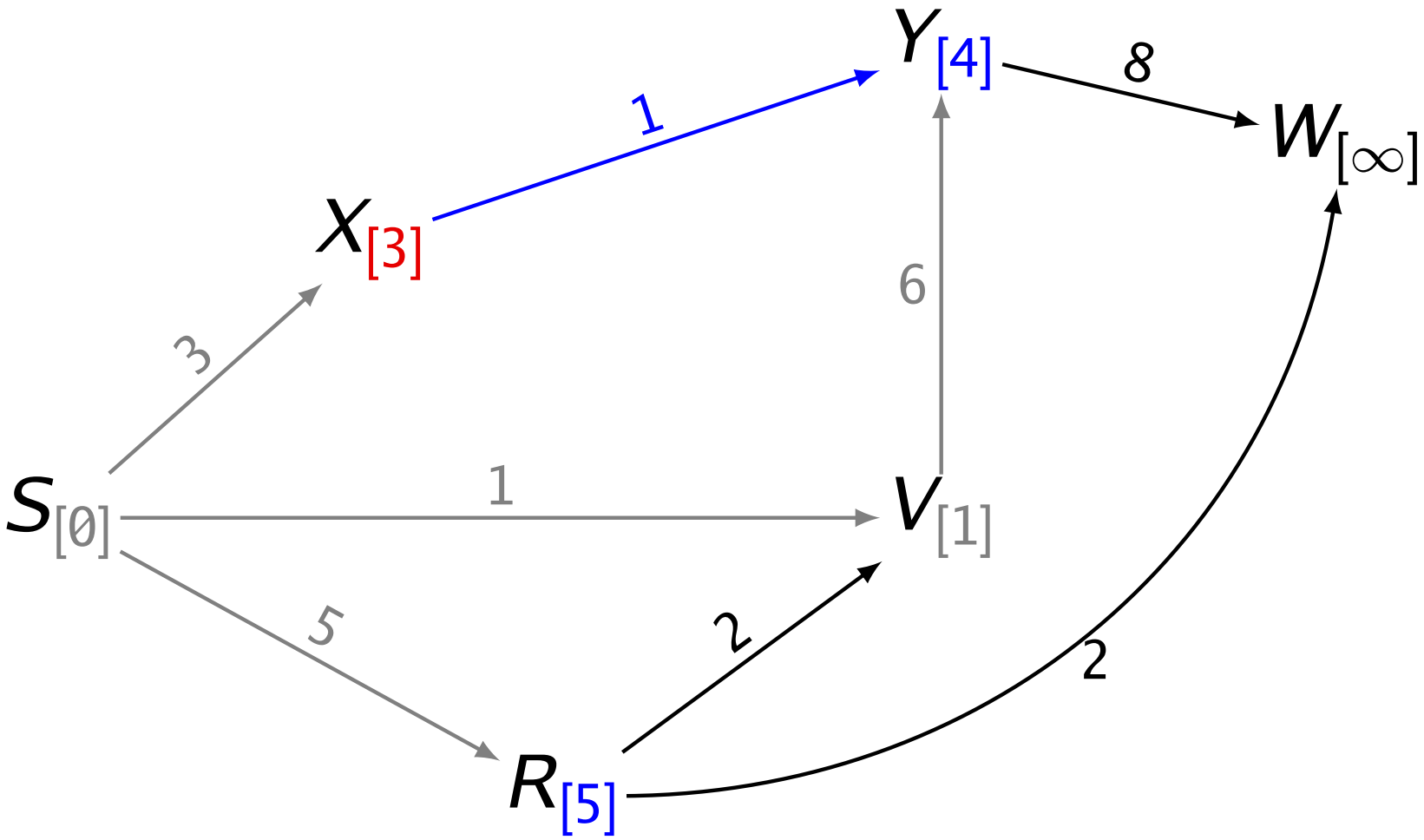
Dijkstra Example



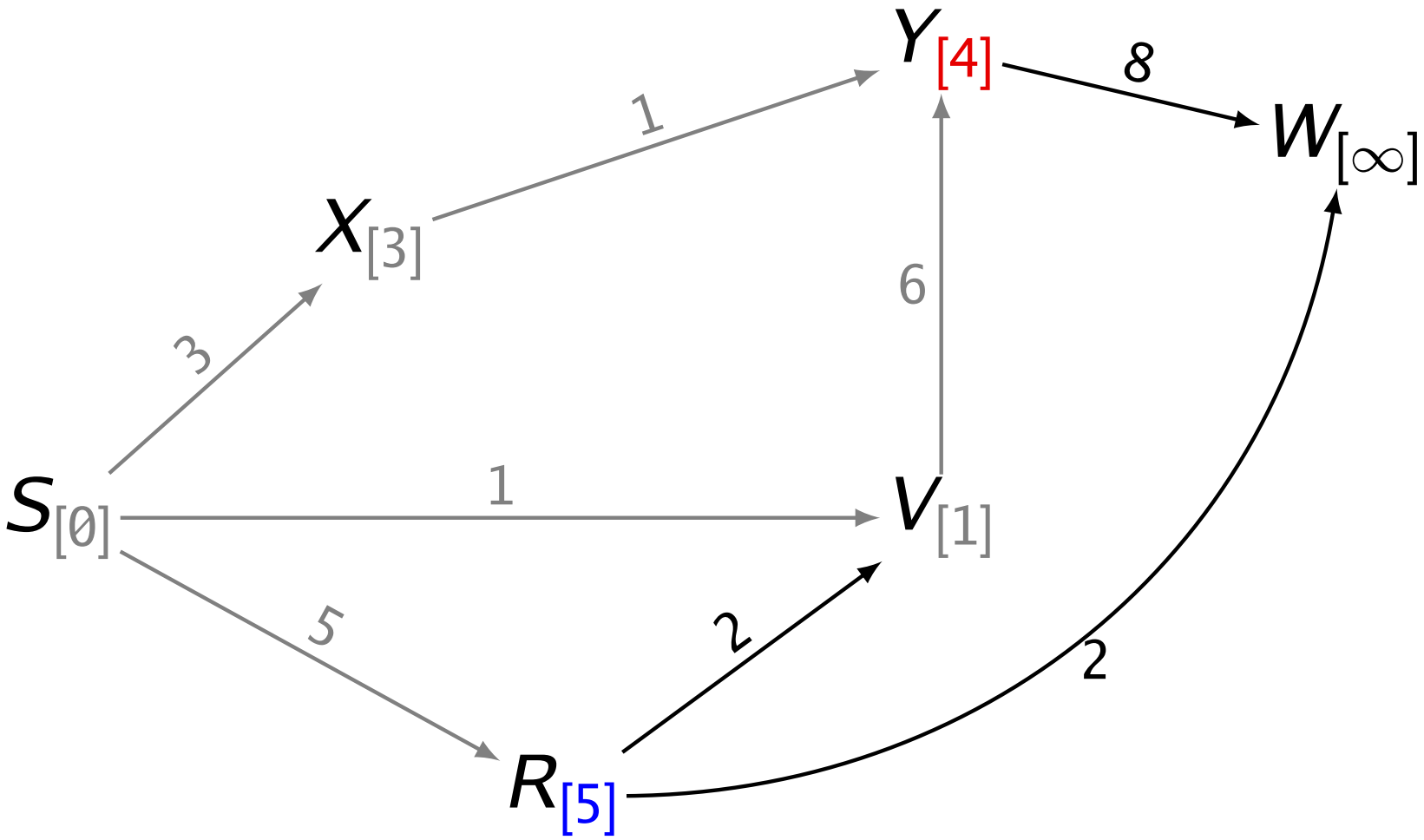
Dijkstra Example



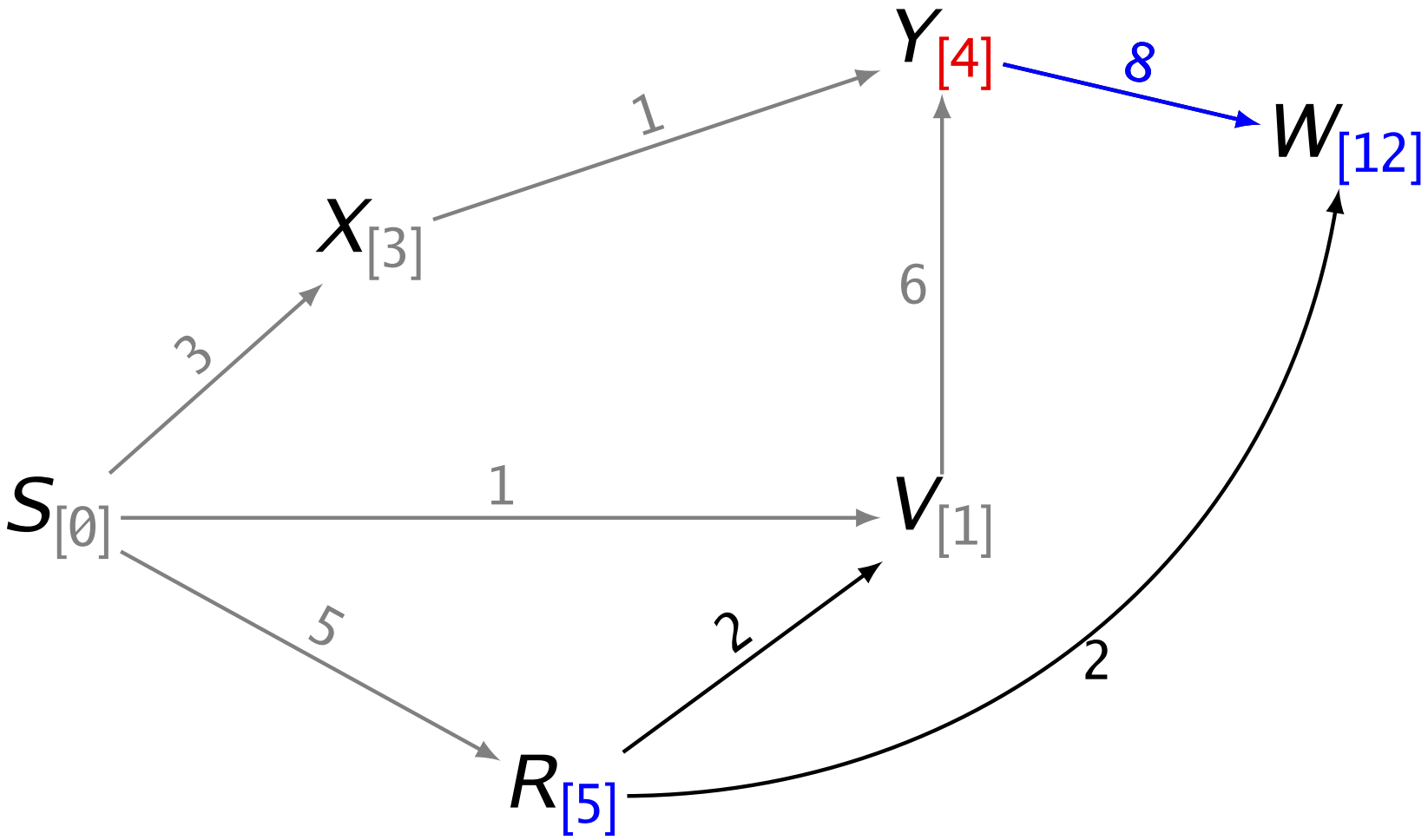
Dijkstra Example



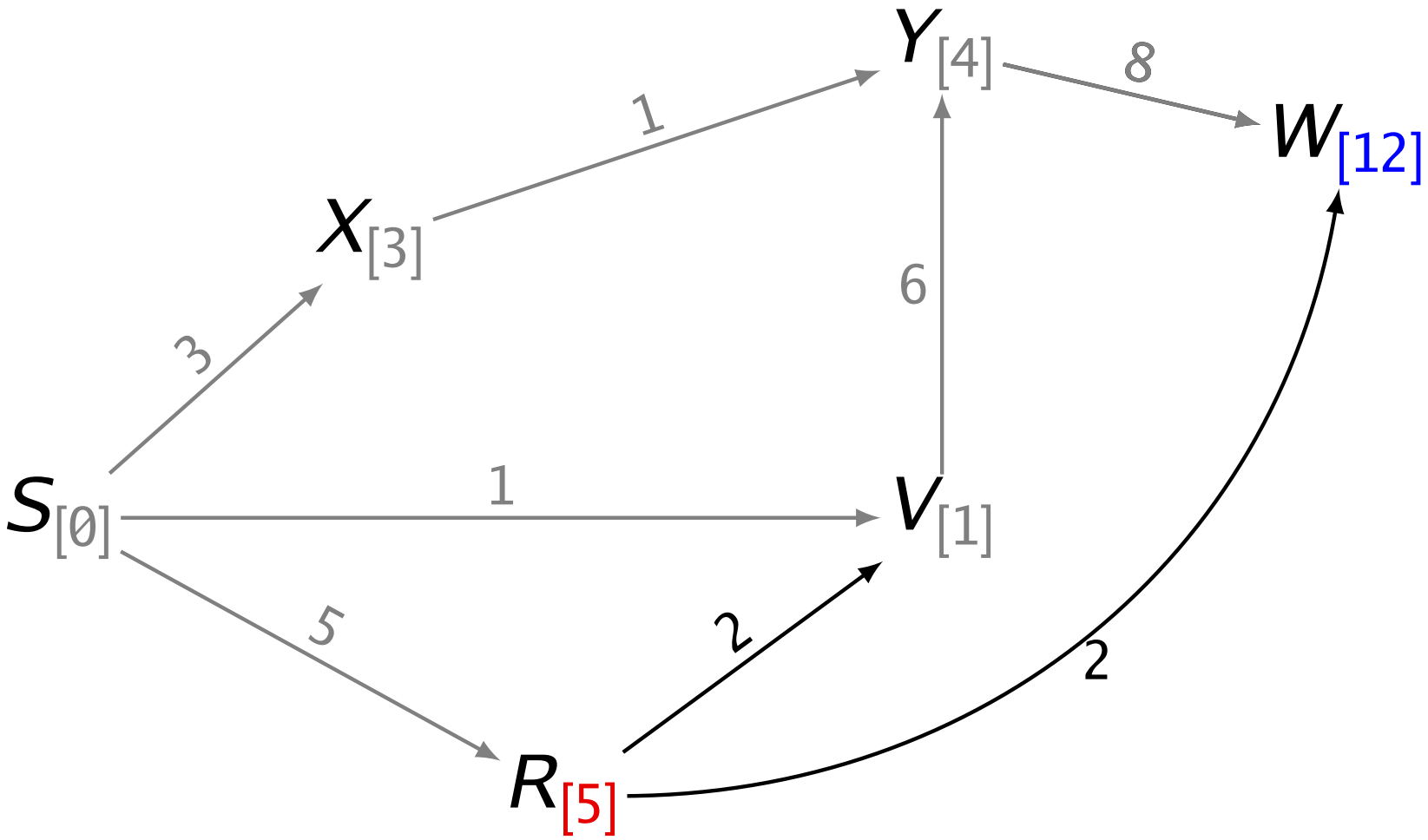
Dijkstra Example



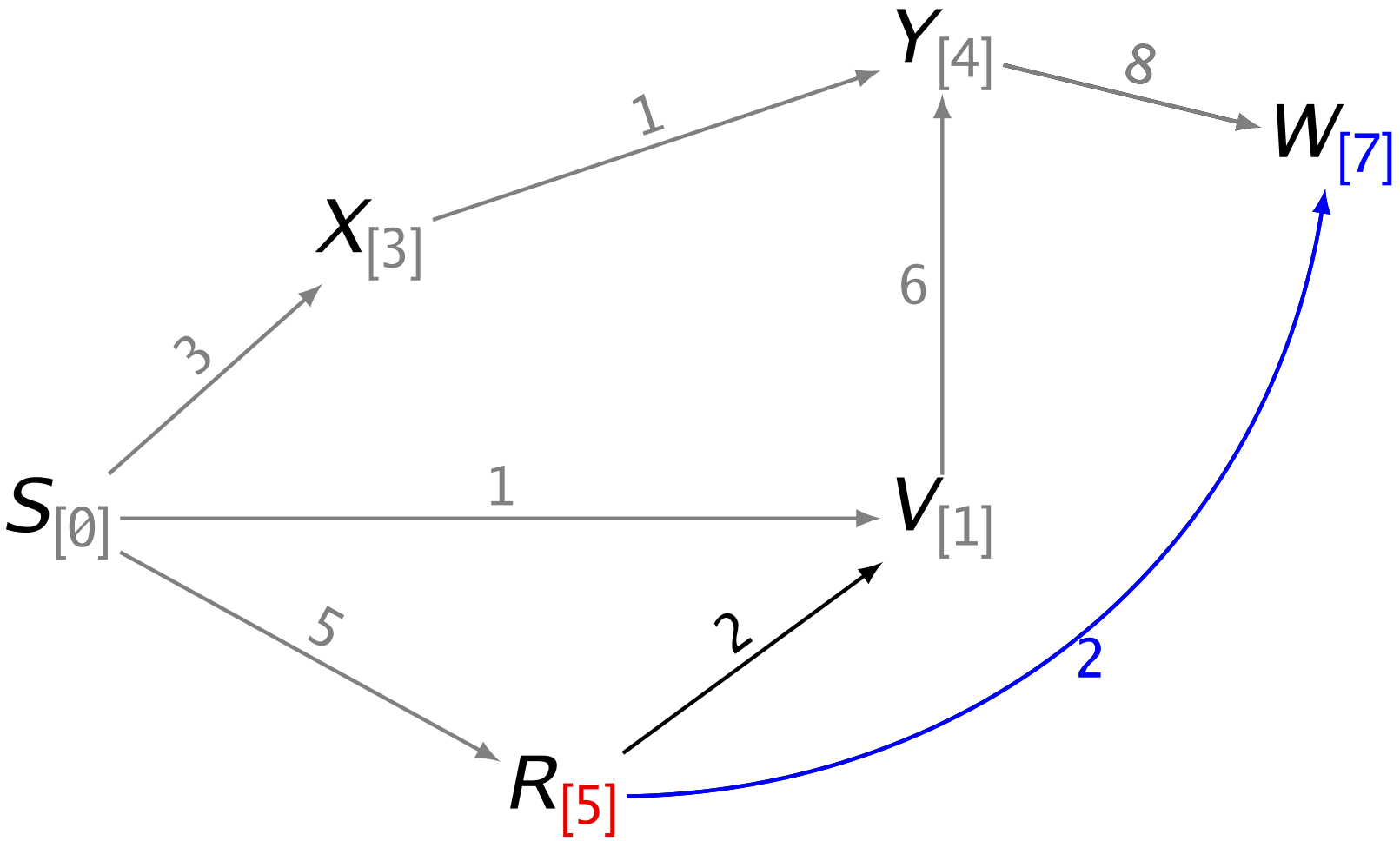
Dijkstra Example



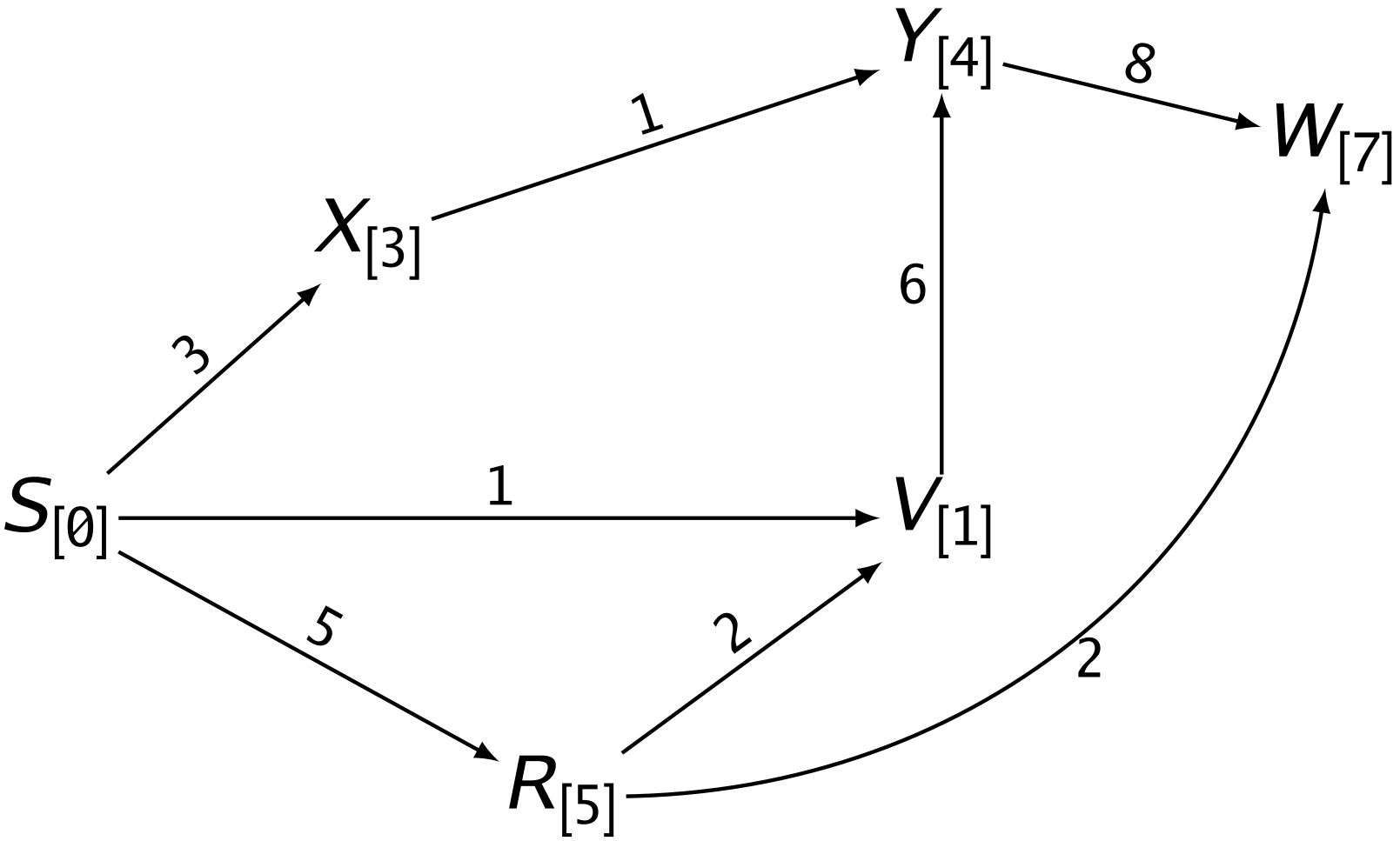
Dijkstra Example



Dijkstra Example



Dijkstra Example



Potential Functions & Re–weighted Graphs

- Let $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ be any consistent STN.
- Let $f : \mathcal{T} \to \mathbb{R}$ be any solution for $\mathcal{S}$.
  - Then $f(Y) - f(X) \leq \delta$ for each constraint $(Y - X \leq \delta) \in \mathcal{C}$
  - In other words, $0 \leq f(X) + \delta - f(Y)$
  - Let $\mathcal{C}' = \{(X, \delta', Y) \mid (X, \delta, Y) \in \mathcal{C}\}$, where $\delta' = f(X) + \delta - f(Y)$
  - Then $\mathcal{S}' = (\mathcal{T}, \mathcal{C}')$ has only non–negative edges.
  - Therefore, can use Dijkstra's SSSP algorithm on $\mathcal{S}'$

Johnson's Algorithm*

Algorithm 4: Johnson()

Input: An STN, $\mathcal{S} = (\mathcal{T}, \mathcal{C})$
Output: $\mathcal{S}$ minimized if consistent, NO otherwise
Run Bellman–Ford SSSP with *new* source node S;
if *Bellman–Ford returns NO* **then return** *NO*;
$f(X) := D(S, X)$ for each $X \in \mathcal{T}$; // It is a solution for $\mathcal{S}$
foreach $(X, \delta, Y \in \mathcal{C})$ **do**
  $\delta' := f(X) + \delta - f(Y)$; // $\delta' \geq 0$
$\mathcal{S}' := (\mathcal{T}, \mathcal{C}')$; // Re–weighted graph based on δ'
foreach $X \in \mathcal{T}$ **do**
  Run Dijkstra on $\mathcal{S}'$ with X as source node; // Computes $\mathcal{D}'(X, Y)$ for all $Y \in \mathcal{T}$.
foreach $X, Y \in \mathcal{T}$ **do**
  $\mathcal{D}(X, Y) = -f(X) + \mathcal{D}'(X, Y) + f(Y)$; // Reverse the re–weighting to obtain $\mathcal{D}$ for $\mathcal{S}$

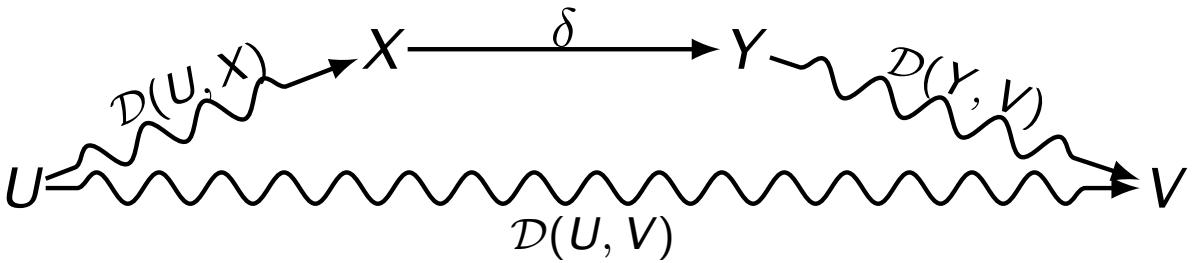
Time Complexity: $O(mn) + n * O(m + n \log n) = O(mn + n^2 \log n)$

*[Cormen et al., 2009]

Simple Temporal Network
Incrementally updating $\mathcal{D}$

- Given a consistent STN $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ with distance matrix $\mathcal{D}$.
- Insert new constraint $(Y - X \leq \delta)$.
- How to update $\mathcal{D}$?
  - Re–compute from scratch: $O(mn + n^2 \log n)$.
  - "Naïve" algorithm: $O(n^2)$: For each $U, V \in \mathcal{T}$,

$$\mathcal{D}(U, V) := \min\{\mathcal{D}(U, V),\ \mathcal{D}(U, X) + \delta + \mathcal{D}(Y, V)\}$$



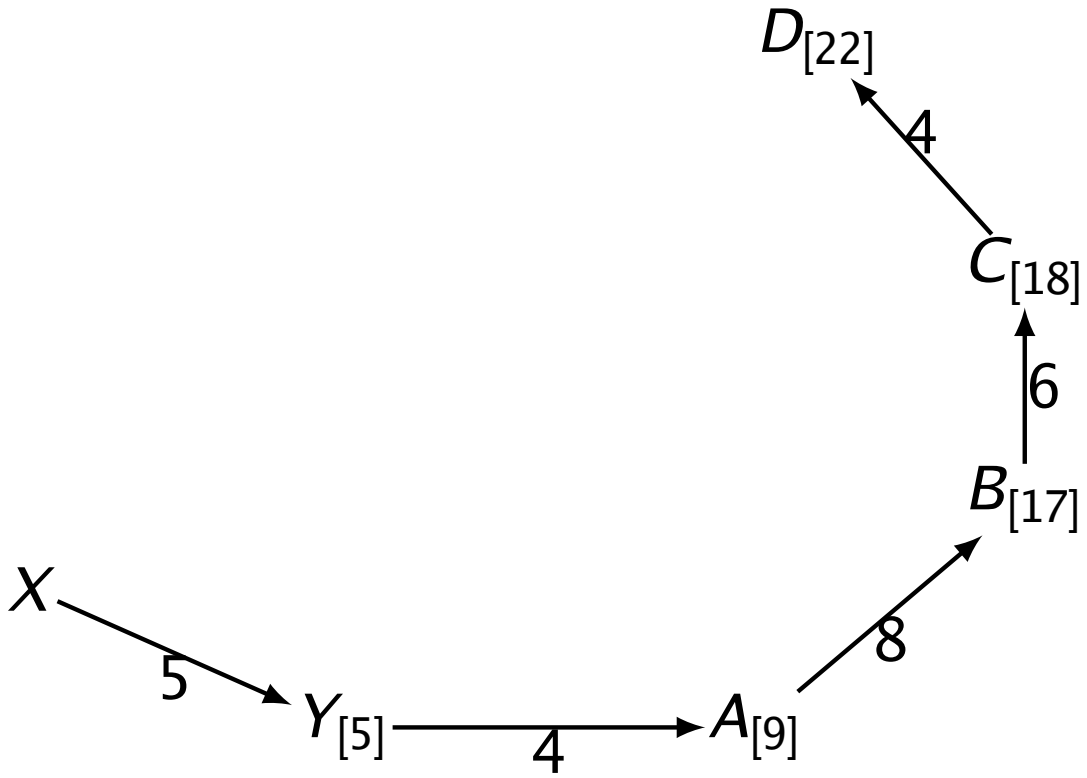
Simple Temporal Networks
Incremental Update Algorithm

- Propagate updates to $\mathcal{D}$ along edges in graph
- Only propagate along *tight* edges; // $(Y - X \leq \delta)$ is tight iff $\mathcal{D}(X, Y) = \delta$
- Phase I: propagate forward
- Phase II: propagate backward
- Checks no more than $b \cdot \Delta$ cells of $\mathcal{D}$, where:
  - $\Delta =$ number of cells needing updating;
  - $b =$ max number of edges incident to any node.

* [Even and Gazit, 1985, Ramalingam and Reps, 1996, Rohnert, 1985]

Incremental Update Algorithm

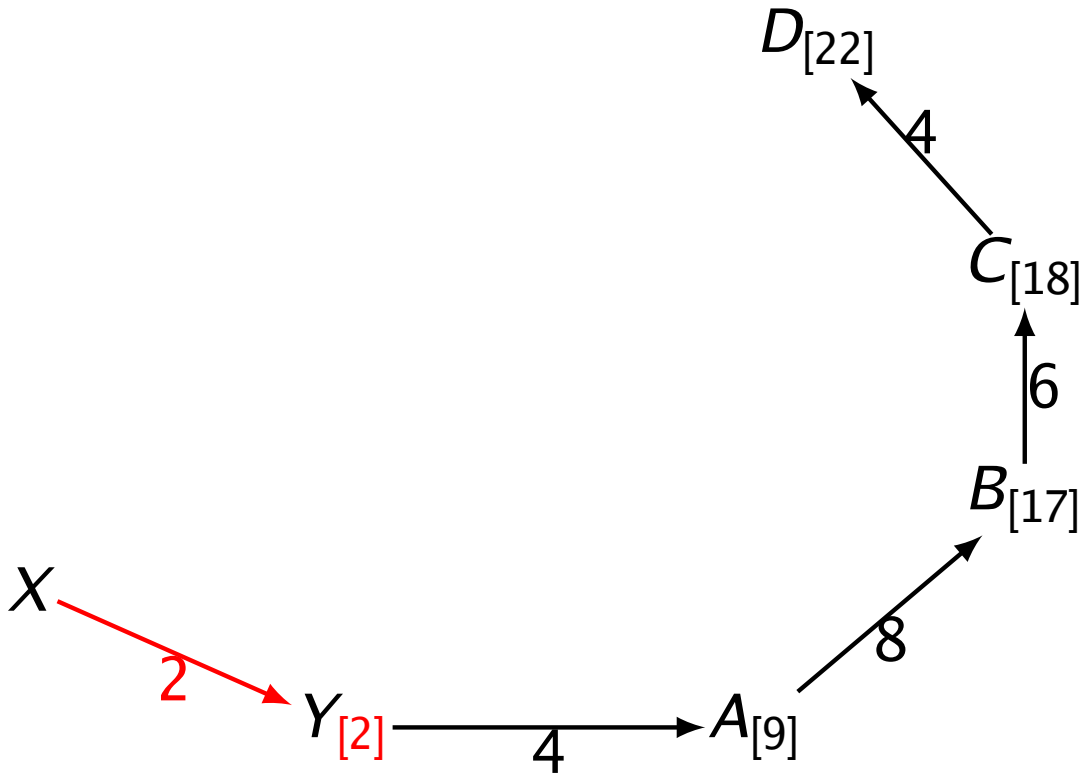
Propagate forward from XY as long as new values obtained



Numbers in brackets are current values of $\mathcal{D}(X, \_)$.

Incremental Update Algorithm

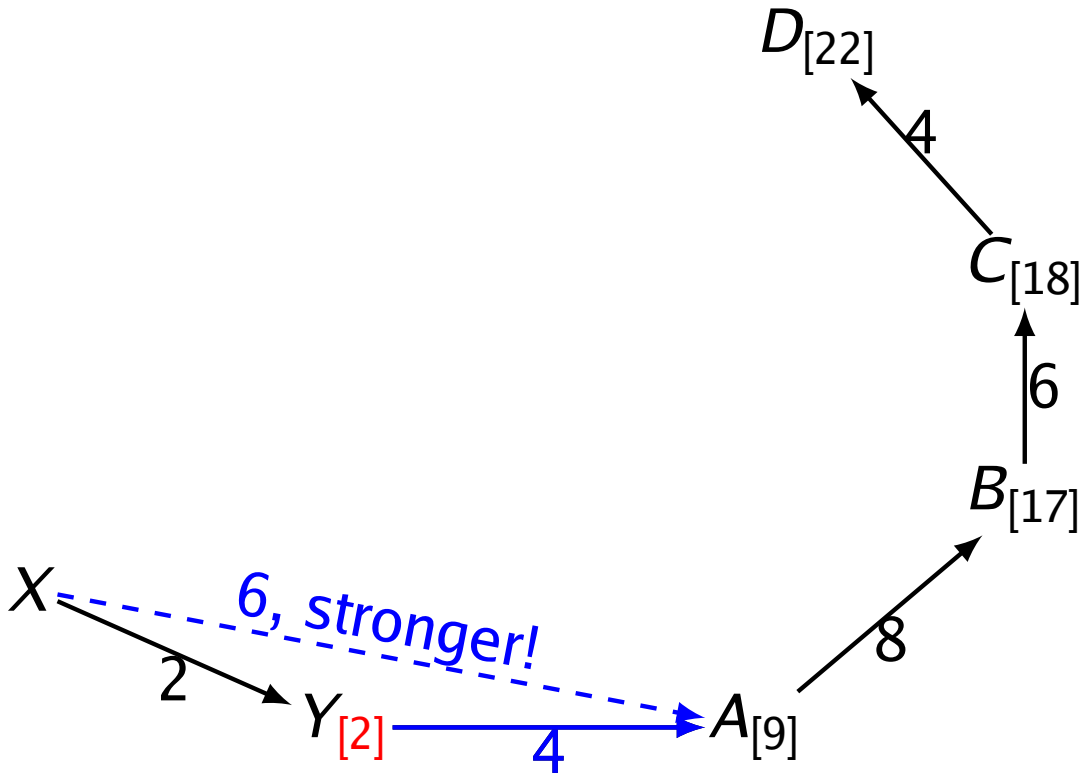
Propagate forward from XY as long as new values obtained



Numbers in brackets are current values of $\mathcal{D}(X, \_)$.

Incremental Update Algorithm

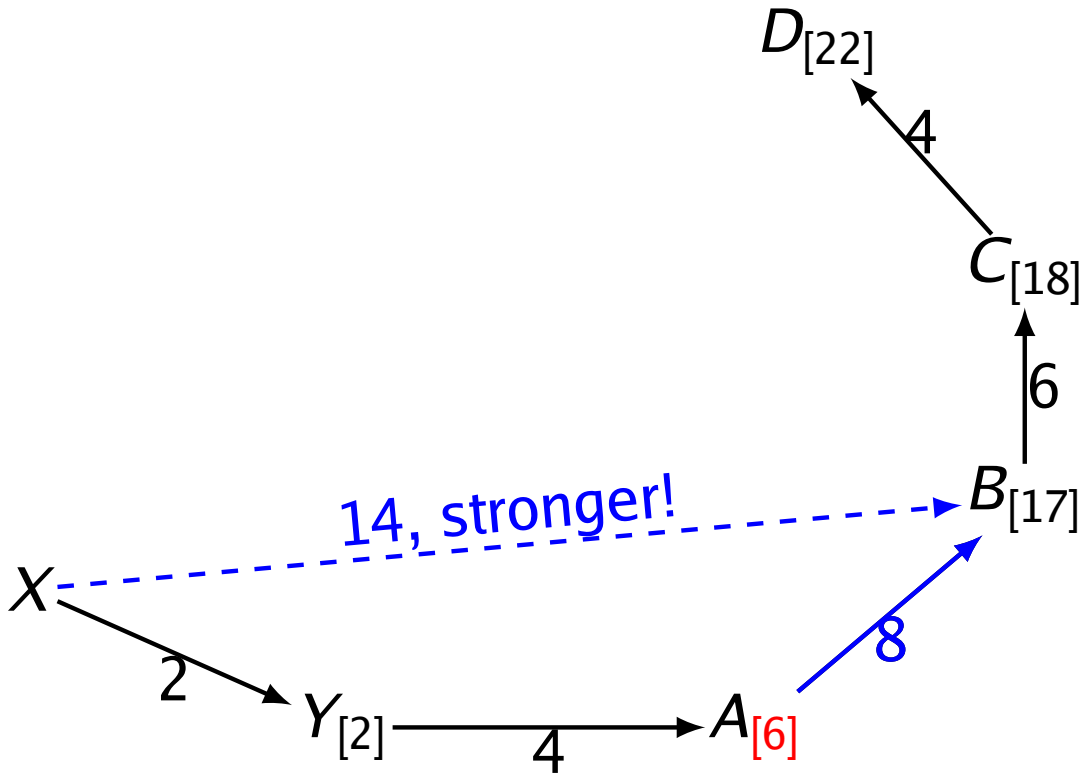
Propagate forward from XY as long as new values obtained



Numbers in brackets are current values of $\mathcal{D}(X, \_)$.

Incremental Update Algorithm

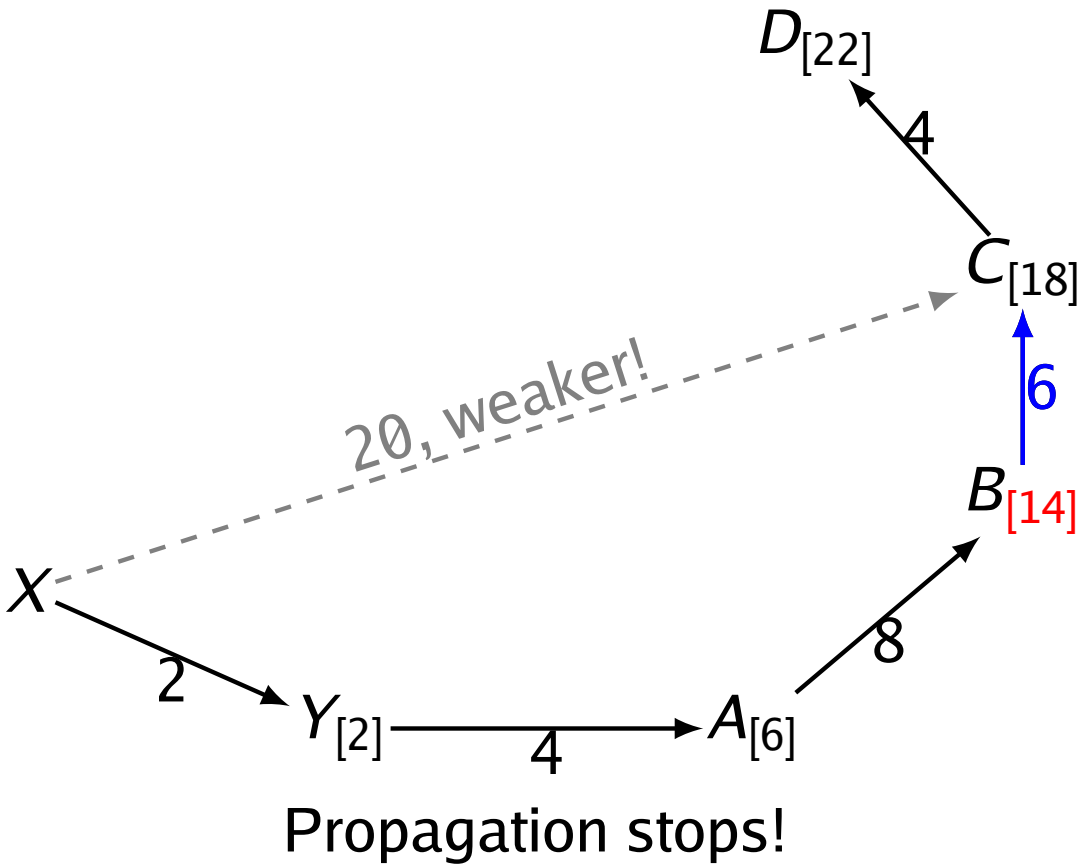
Propagate forward from XY as long as new values obtained



Numbers in brackets are current values of $\mathcal{D}(X, \_)$.

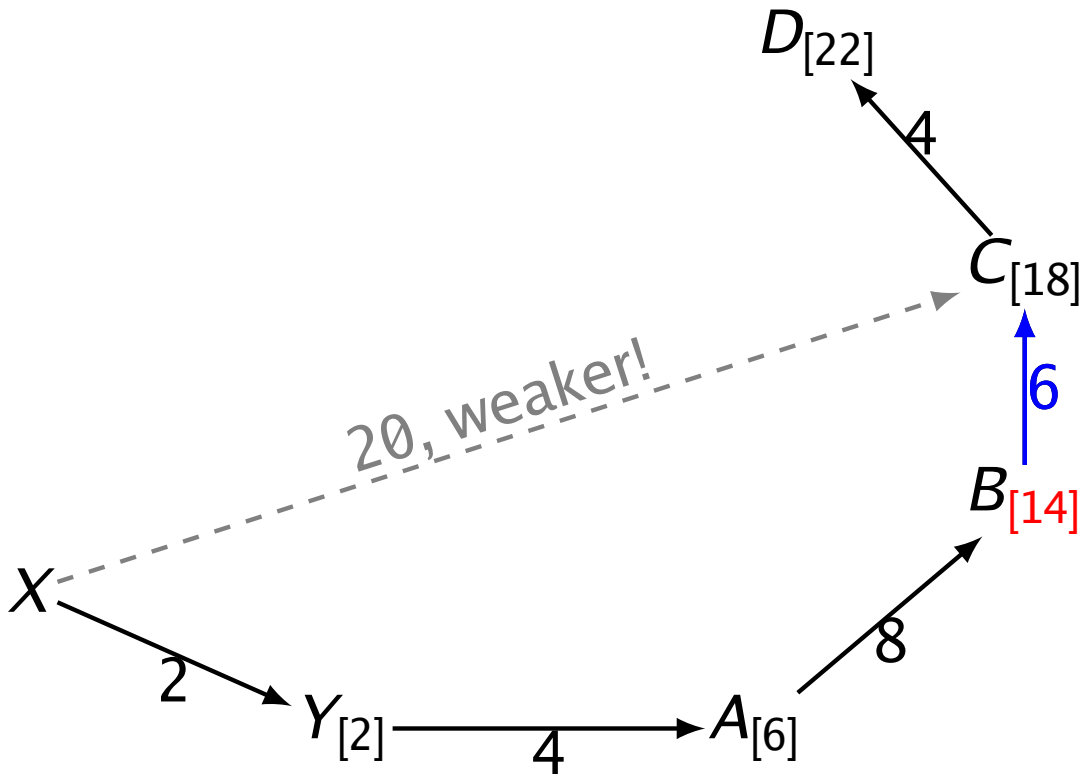
Incremental Update Algorithm

Propagate forward from XY as long as new values obtained



Incremental Update Algorithm

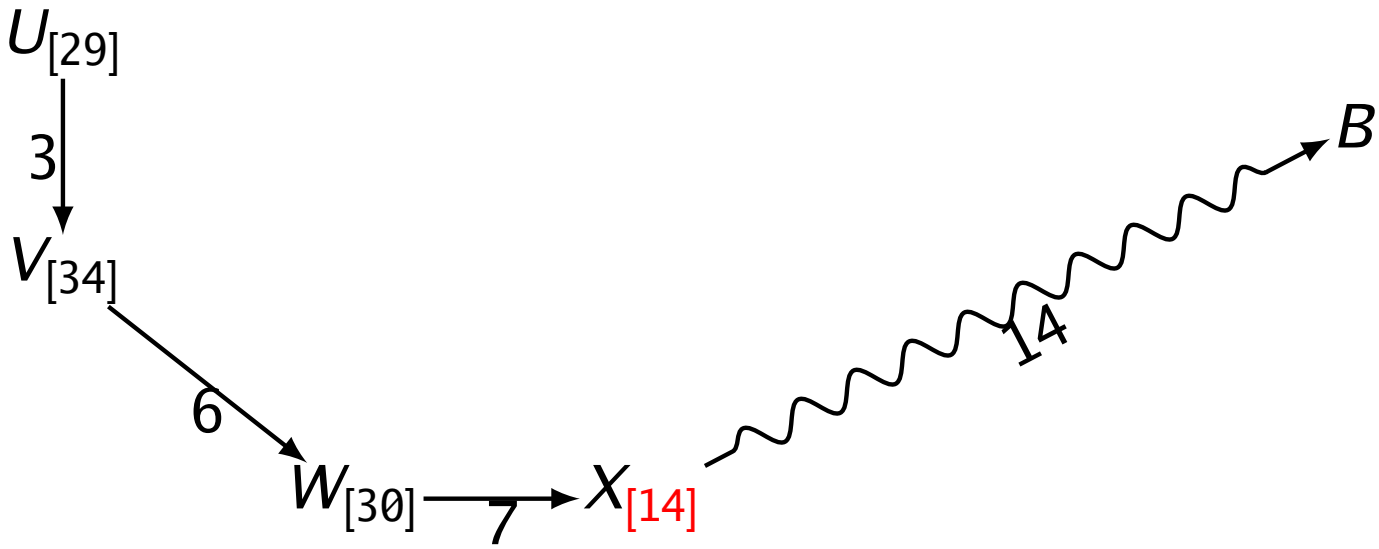
Propagate forward from XY as long as new values obtained



Forward propagation done along this path.

Incremental Update Algorithm

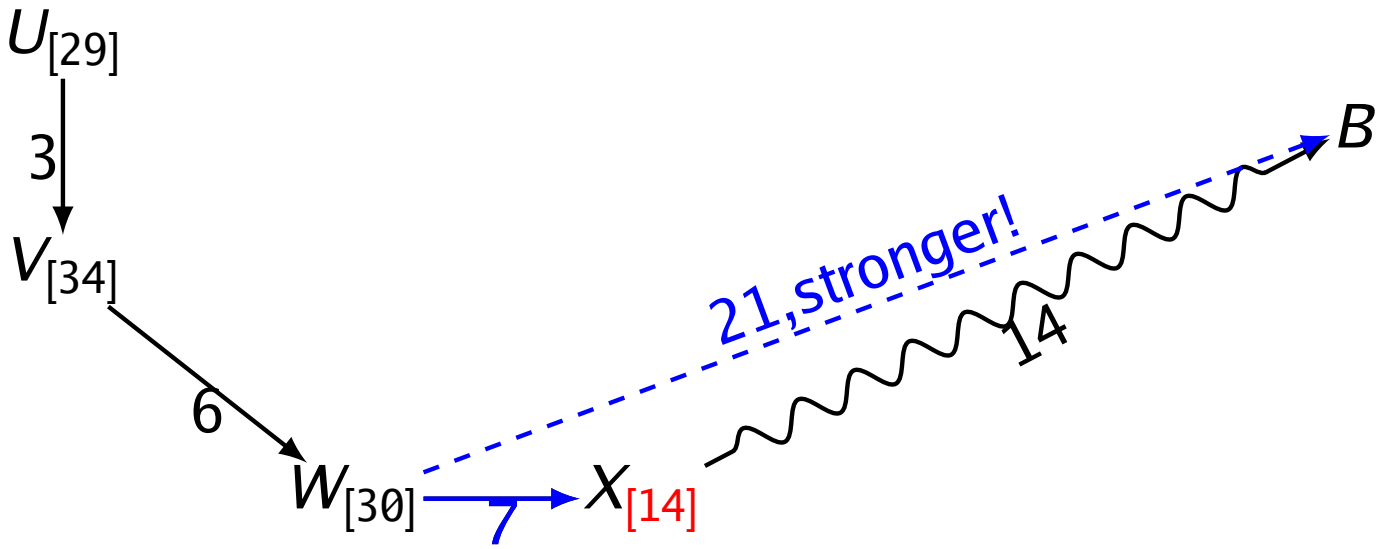
For each entry $\mathcal{D}(X, \_)$ that was updated, propagate backward from X



Numbers in brackets are $\mathcal{D}(\_, B)$ values.

Incremental Update Algorithm

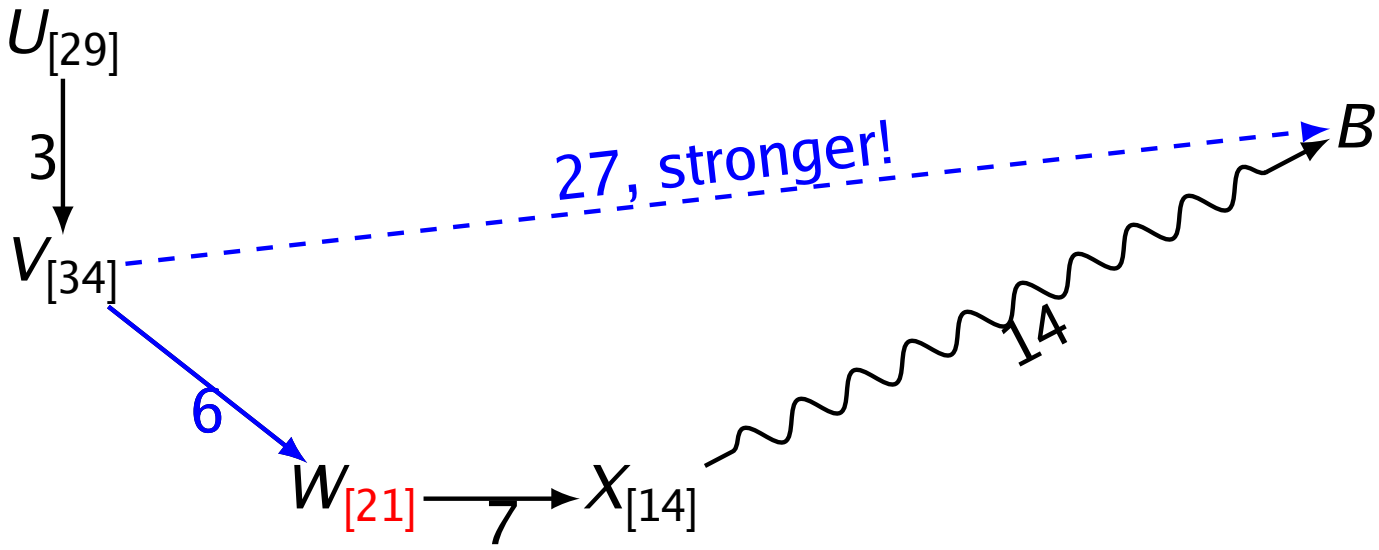
For each entry $\mathcal{D}(X, \_)$ that was updated, propagate backward from X



Numbers in brackets are $\mathcal{D}(\_, B)$ values.

Incremental Update Algorithm

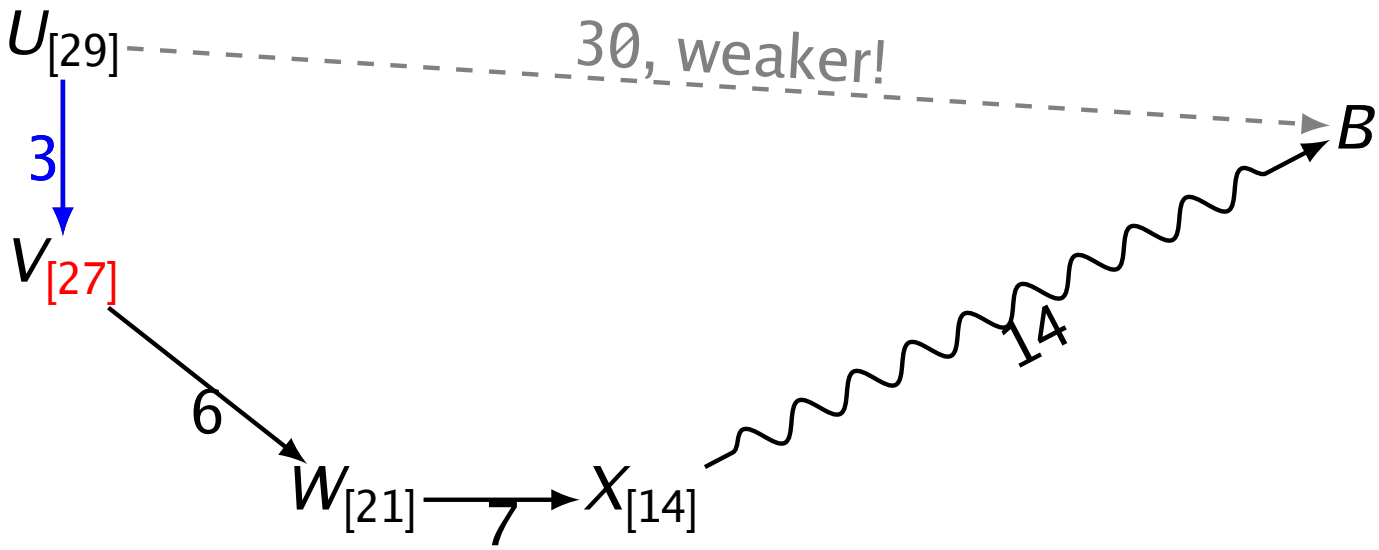
For each entry $\mathcal{D}(X, \_)$ that was updated, propagate backward from X



Numbers in brackets are $\mathcal{D}(\_, B)$ values.

Incremental Update Algorithm

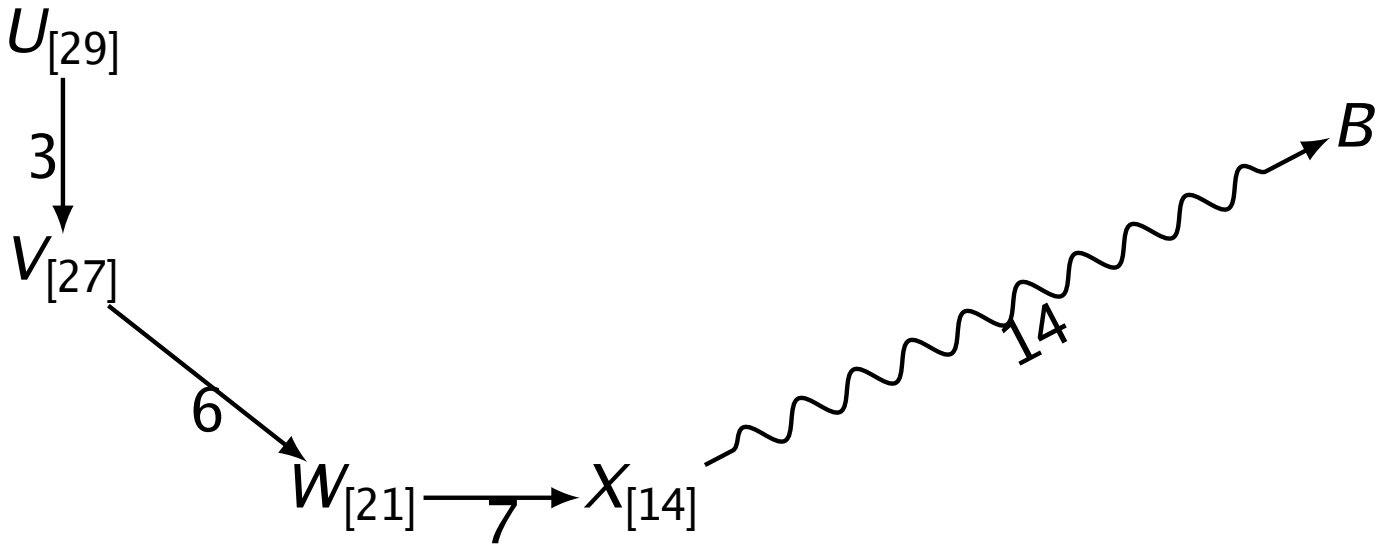
For each entry $\mathcal{D}(X, \_)$ that was updated, propagate backward from X



No further updates along this path.

Incremental Update Algorithm

For each entry $\mathcal{D}(X, \_)$ that was updated, propagate backward from X



No further updates along this path.

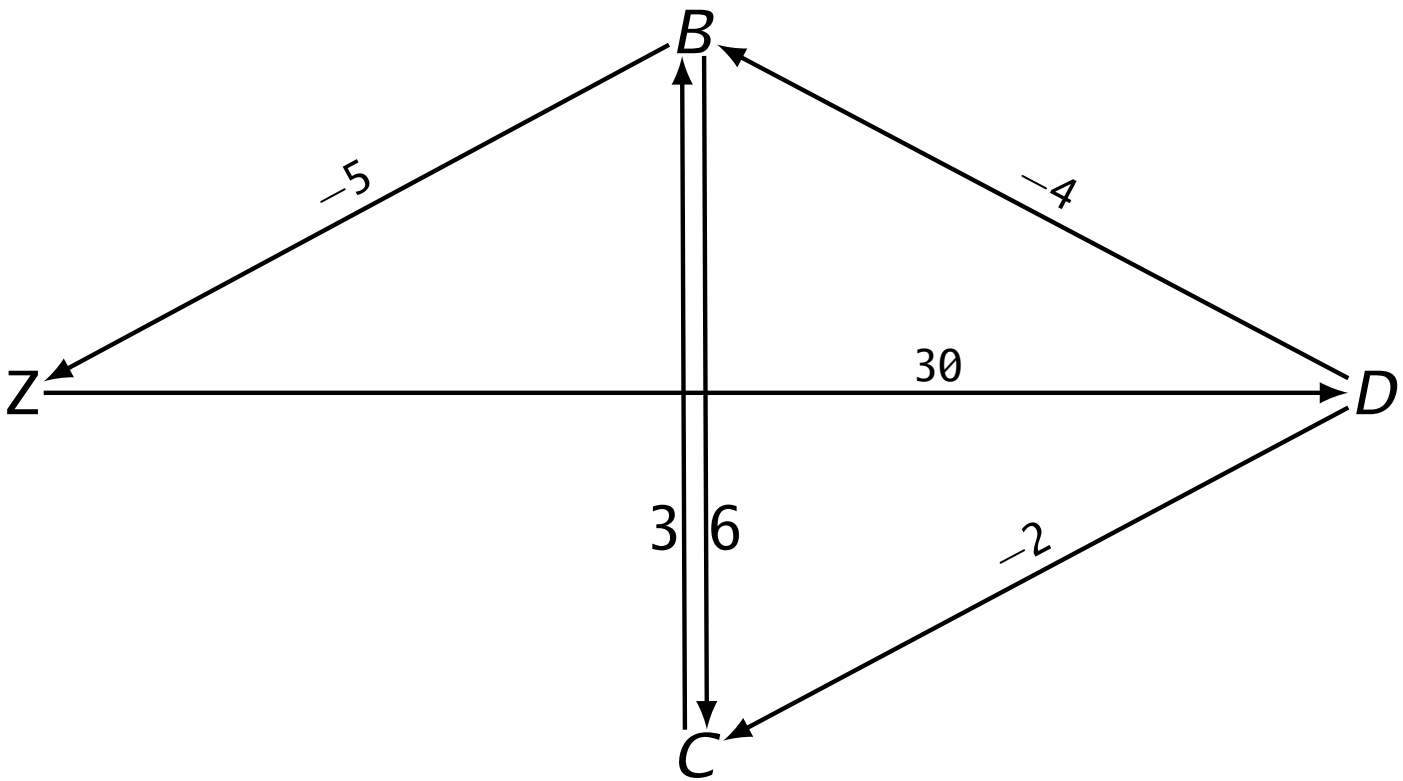
Simple Temporal Network
Incremental Consistency

Verifying consistency of an STN after inserting, weakening or deleting a constraint is less expensive than fully updating the distance matrix.*

- Algorithm maintains/updates a solution to the STN.
- After inserting a new constraint (or strengthening an existing one), can verify consistency in $O(m + n \log n)$ time.
- After deleting or weakening a constraint, only need constant time, because the same solution will work for the modified STN.

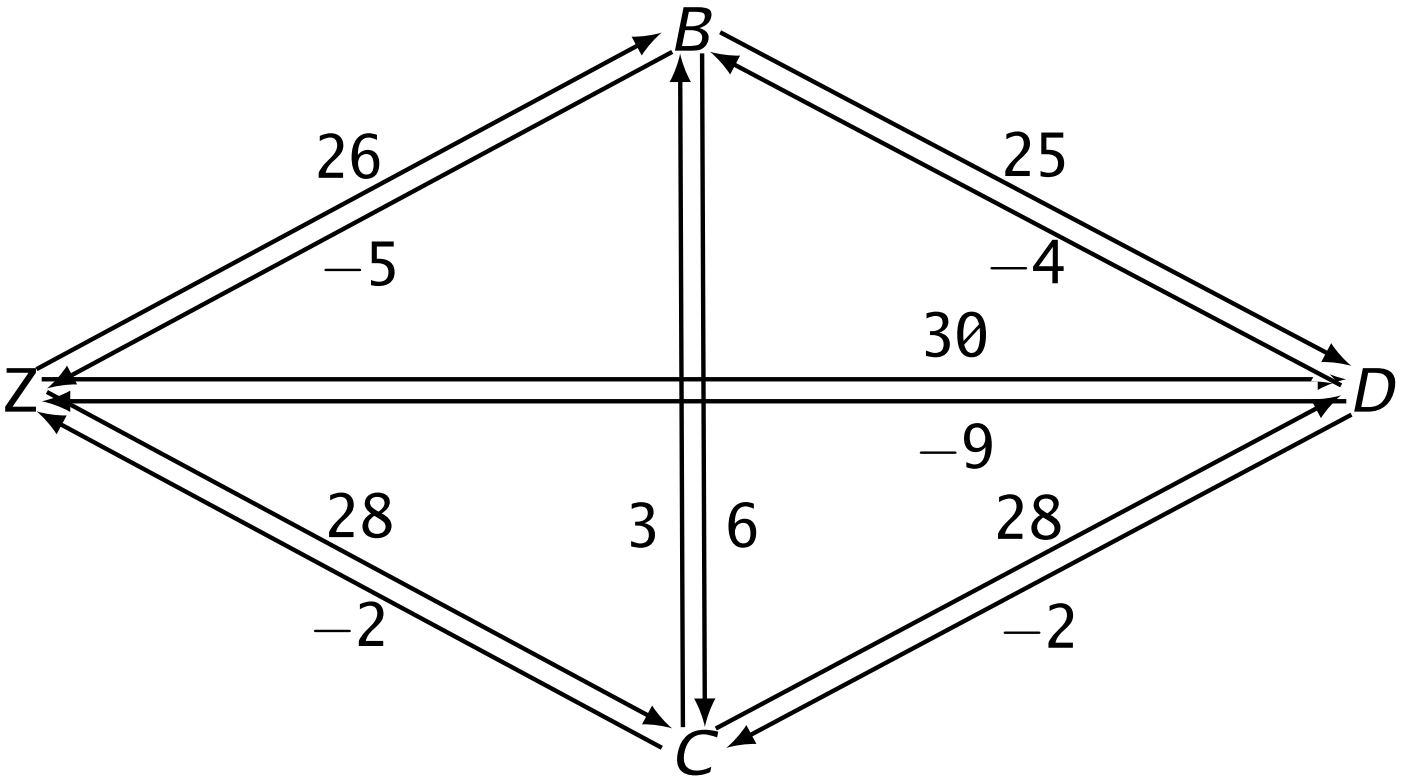
* [Ramalingam et al., 1999]

Simple Temporal Network
Sample STN



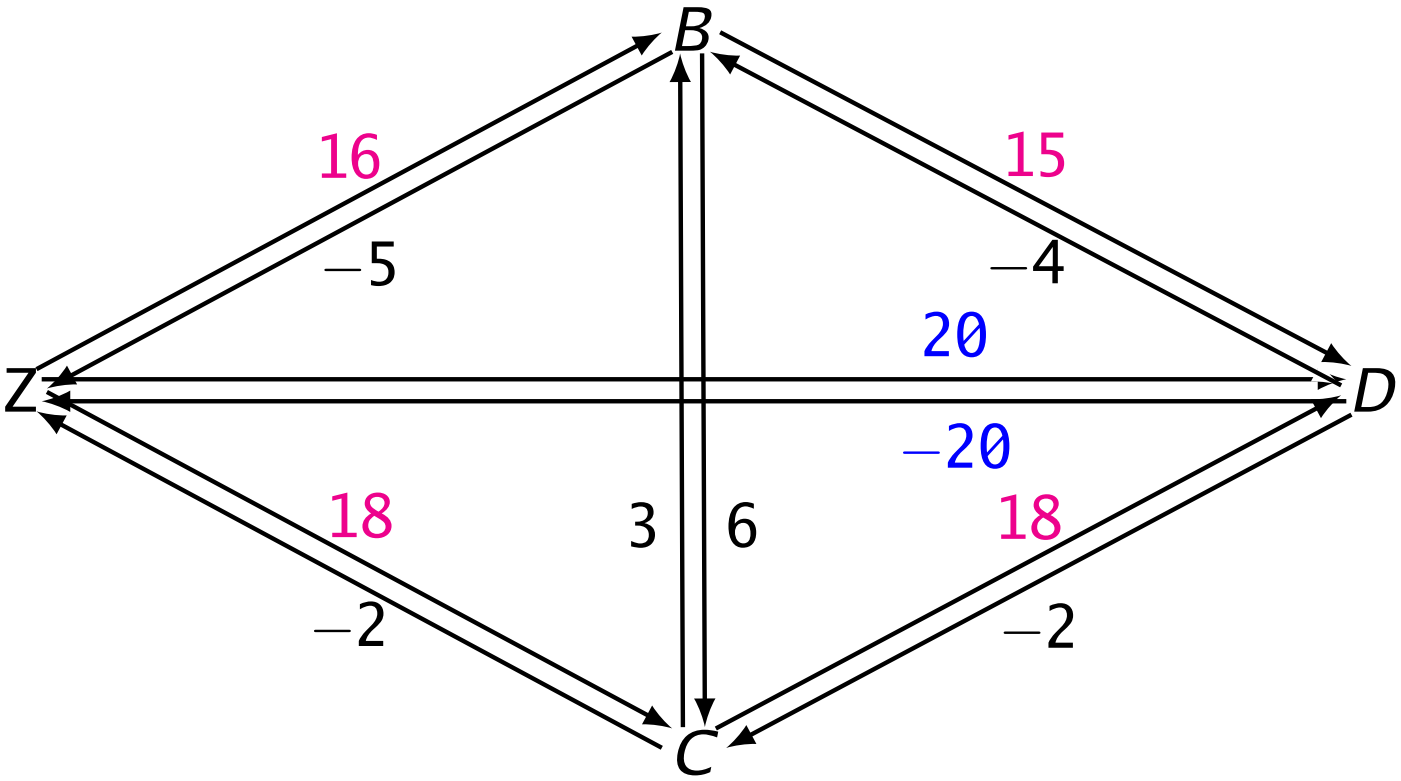
Simple Temporal Network
"Solving" Sample STN

First, form APSP graph (equiv. compute $\mathcal{D}$).



Time Windows: $B \in [5, 26]$, $C \in [2, 28]$, $D \in [9, 30]$

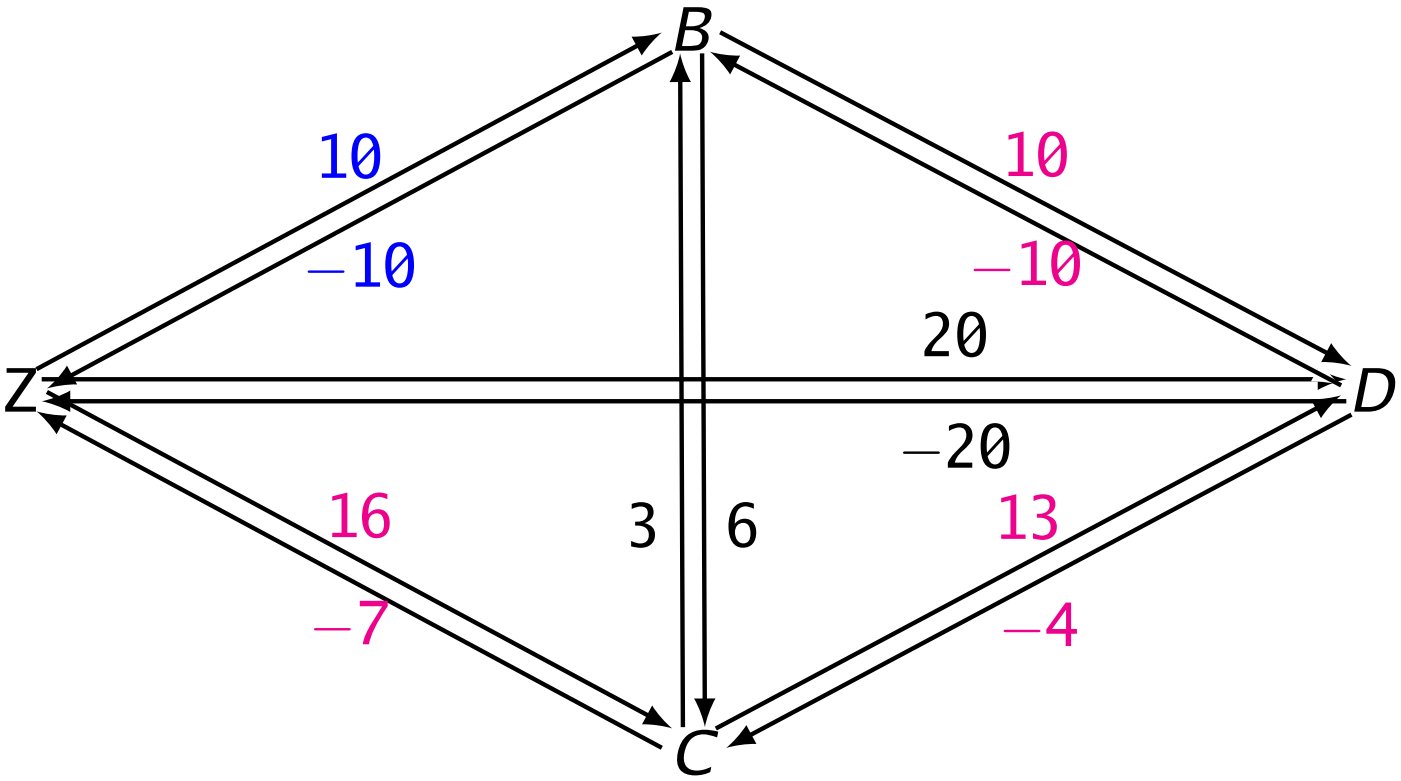
Next, select $D = 20$; and update APSP graph:



Remaining Time Windows: $B \in [5, 16]$, $C \in [2, 18]$

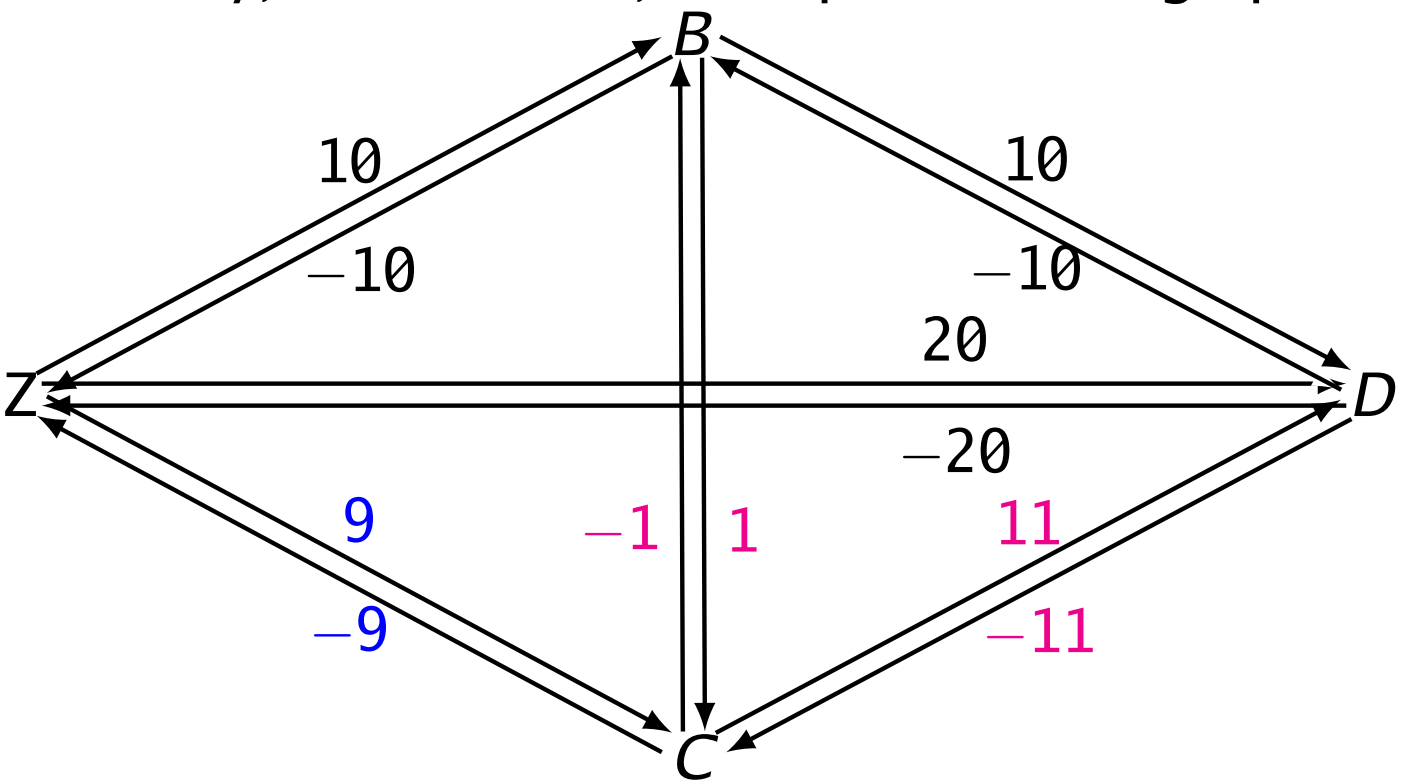
Simple Temporal Network
"Solving" Sample STN

Next, select $B = 10$; and update APSP graph:



Remaining Time Windows: $C \in [7, 16]$

Finally, select $C = 9$; and update APSP graph:



Easy to verify that this is a solution.

Simple Temporal Network
Problems with "Solving" an STN

- May need to go back in time:
  Pick $D = 20$, then after updating, pick $B = 10$
  (i.e., no relationship to real–time execution)
- Expensive to update $\mathcal{D}$

Simple Temporal Network
Executing an STN in real time

- Only executed enabled time–points: those having no negative edges to unexecuted time–points.
- Focus updating on entries involving Z: reduces cost to linear time per update, $O(n^2)$ overall.*
- Alternatively, prior to execution, transform STN into dispatchable form in $O(n^2 \log n + nm)$ time; then during execution, only need to propagate bounds to *neighboring* time–points.†

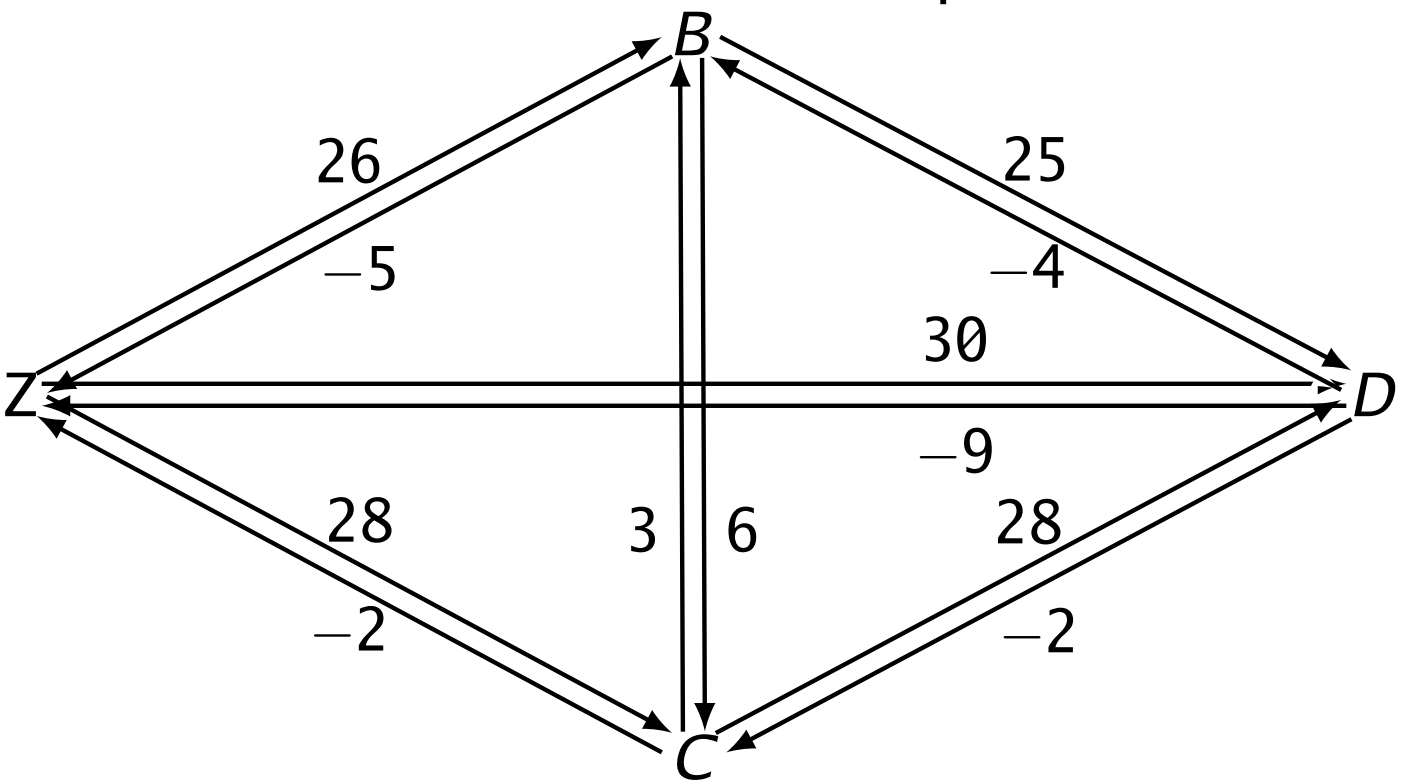
*[Hunsberger, 2008]; †[Muscettola et al., 1998], †[Tsamardinos et al., 1998]

An STN $\mathcal{S}$ is dispatchable if the following algorithm necessarily successfully executes $\mathcal{S}$:

Algorithm 5: ExecuteDispatchableNetwork($\mathcal{G} = (\mathcal{T}, \mathcal{E})$)

$t := 0$; // current time
$\mathcal{X} := \{\}$; // executed nodes
$\mathbf{E} := \{Z\}$; // currently enabled nodes
while $\mathcal{X} \neq \mathcal{T}$ **do**
  Pick any $X \in \mathbf{E}$ such that t is in X's time window;
  $X := t$, $\mathcal{X} := \mathcal{X} \cup \{X\}$;
  Propagate $t \leq X \leq t$ to X's immediate neighbors;
  $\mathbf{E} := \mathbf{E} \cup \{$all time–points Y s.t. all non–positive edges emanating from Y have a destination in $\mathcal{X}\}$;
  Wait until t has advanced to some time in $[\min\{lb(W) \mid W \in \mathbf{E}\}, \min\{ub(W) \mid W \in \mathbf{E}\}]$;

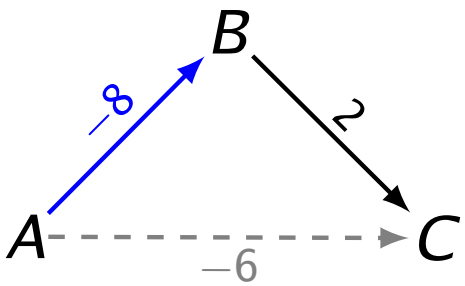
Start with APSP Graph:



Then remove *dominated* edges . . .

Simple Temporal Network
Remove Dominated Edges

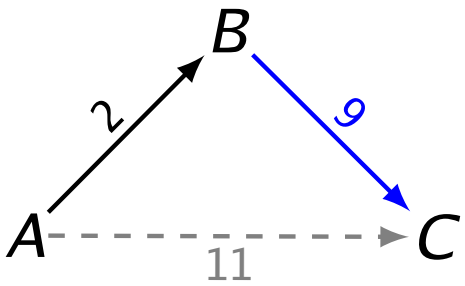
A negative edge AC is dominated by a negative edge AB if $\mathcal{D}(A, B) + \mathcal{D}(B, C) = \mathcal{D}(A, C)$:



- AB and AC have the same source node: A.
- During the execution, it is not necessary to propagate along dominated edges like AC.

Simple Temporal Network
Remove Dominated Edges (ctd.)

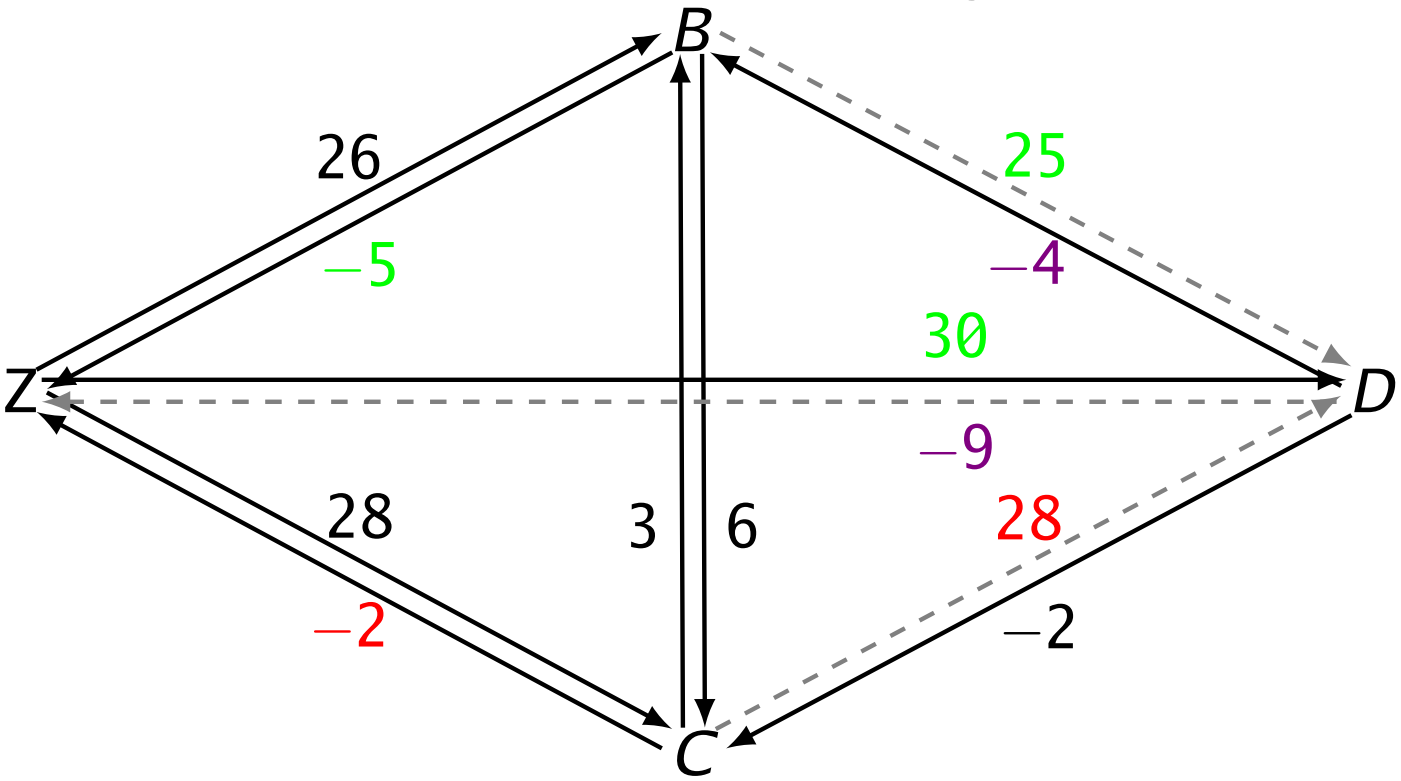
A non–negative edge AC is dominated by a non–negative edge BC if $\mathcal{D}(A, B) + \mathcal{D}(B, C) = \mathcal{D}(A, C)$:



- BC and AC have the same destination node: C.
- During the execution, it is not necessary to propagate along dominated edges like AC.

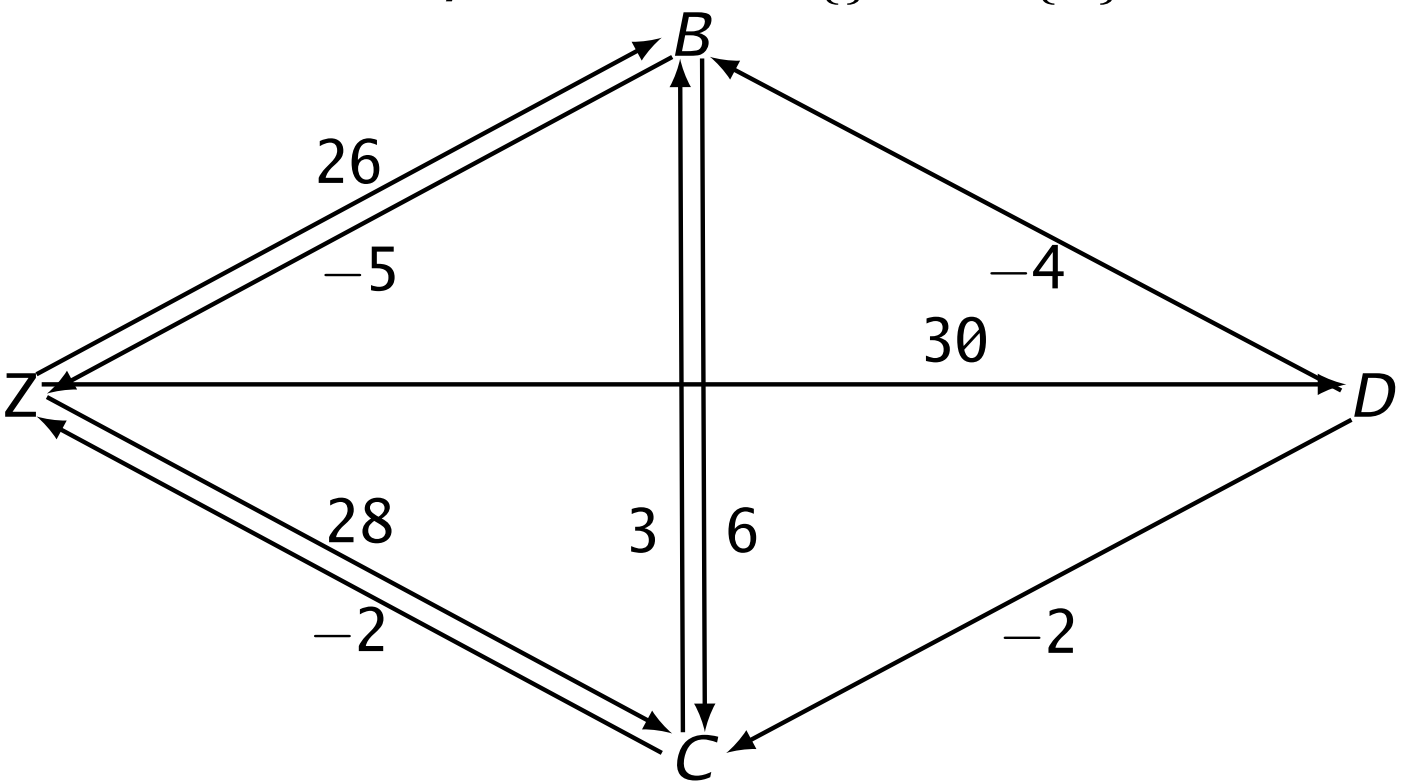
Simple Temporal Network
Making STN Dispatchable (ctd.)

Remove “dominated” edges:*



*[Muscettola et al., 1998]

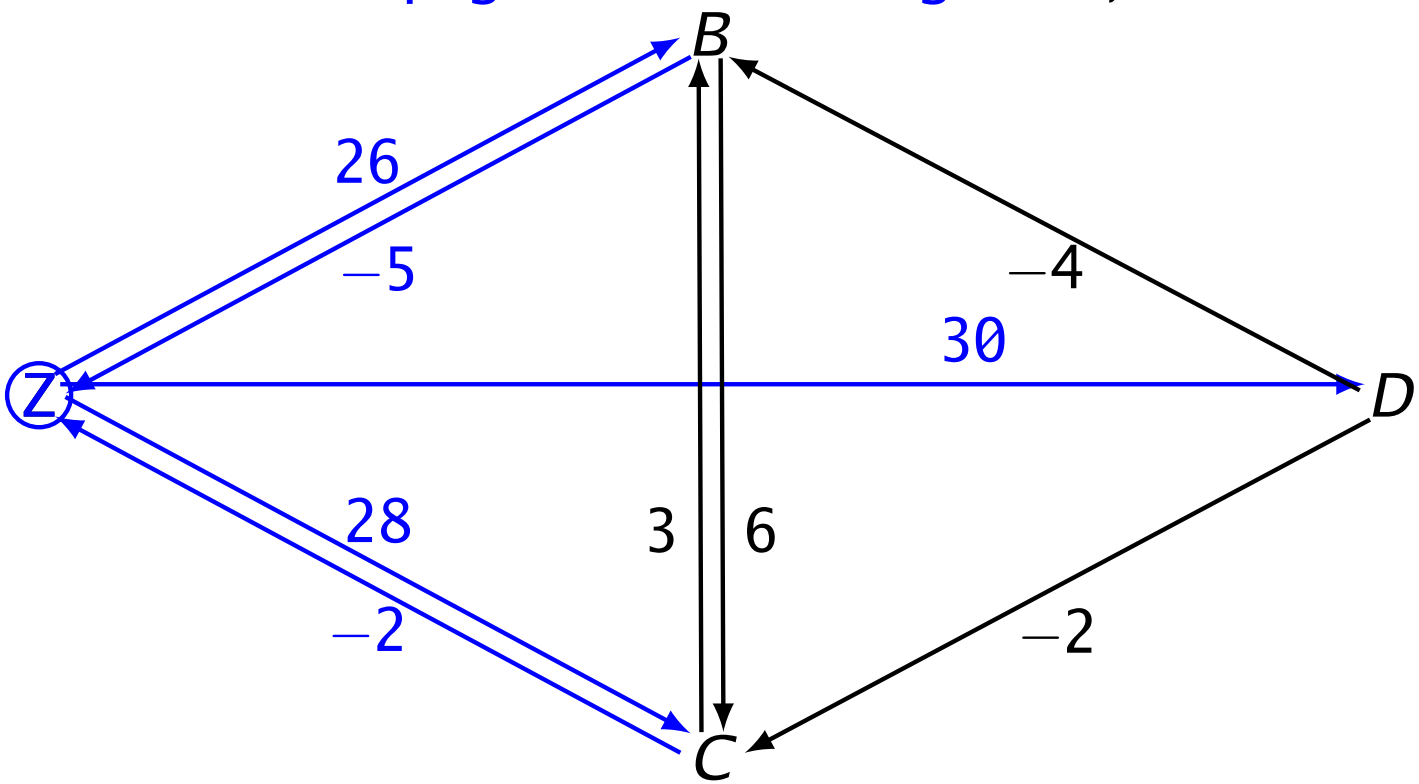
Initially: $t = 0$, $\mathcal{X} = \{\}$, $\mathbf{E} = \{\mathsf{Z}\}$.



Pick Z from $\mathbf{E}$. Set $\mathsf{Z} = 0$.

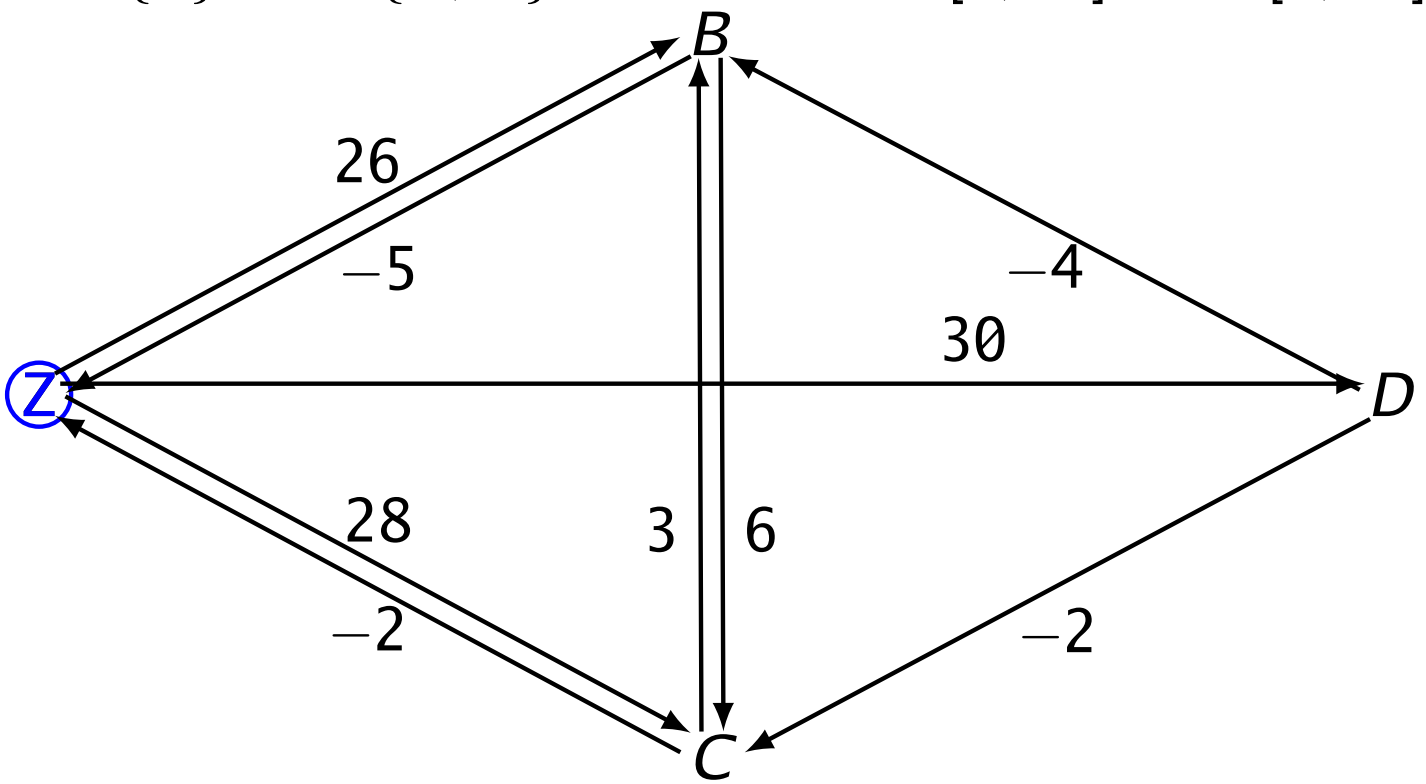
Simple Temporal Network
Dispatching the STN (ctd.)

Propagate $Z = 0$ to neighbors;



$\mathcal{X} = \{Z\}$, $\mathbf{E} = \{B, C\}$; $B \in [5, 26]$, $C \in [2, 28]$, $D \in [0, 30]$.

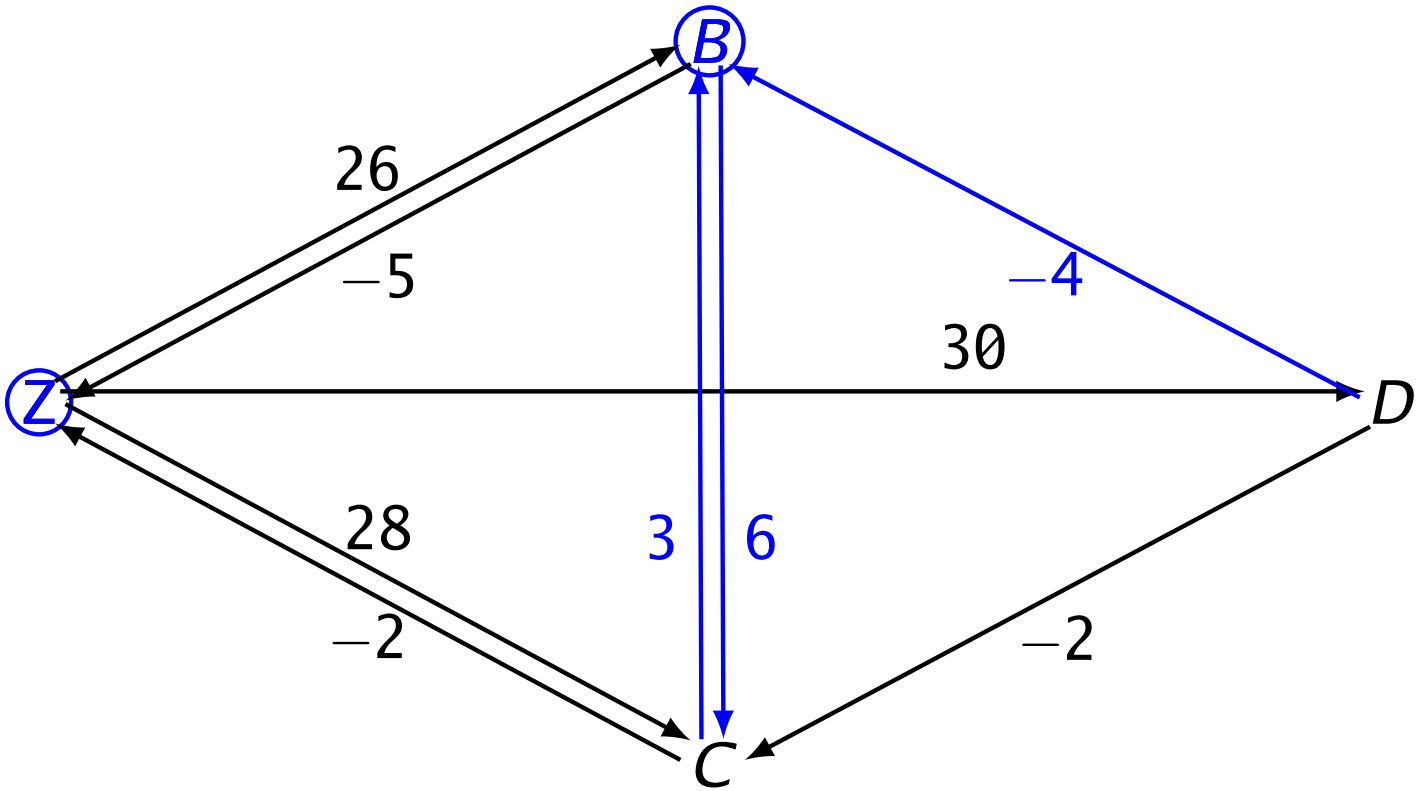
$\mathcal{X} = \{Z\}$, $\mathbf{E} = \{B, C\}$; Bounds: $B \in [5, 26]$, $C \in [2, 28]$.



Let t advance to 12; Pick B from $\mathbf{E}$; Set $B = 12$.

Simple Temporal Network
Dispatching the STN (ctd.)

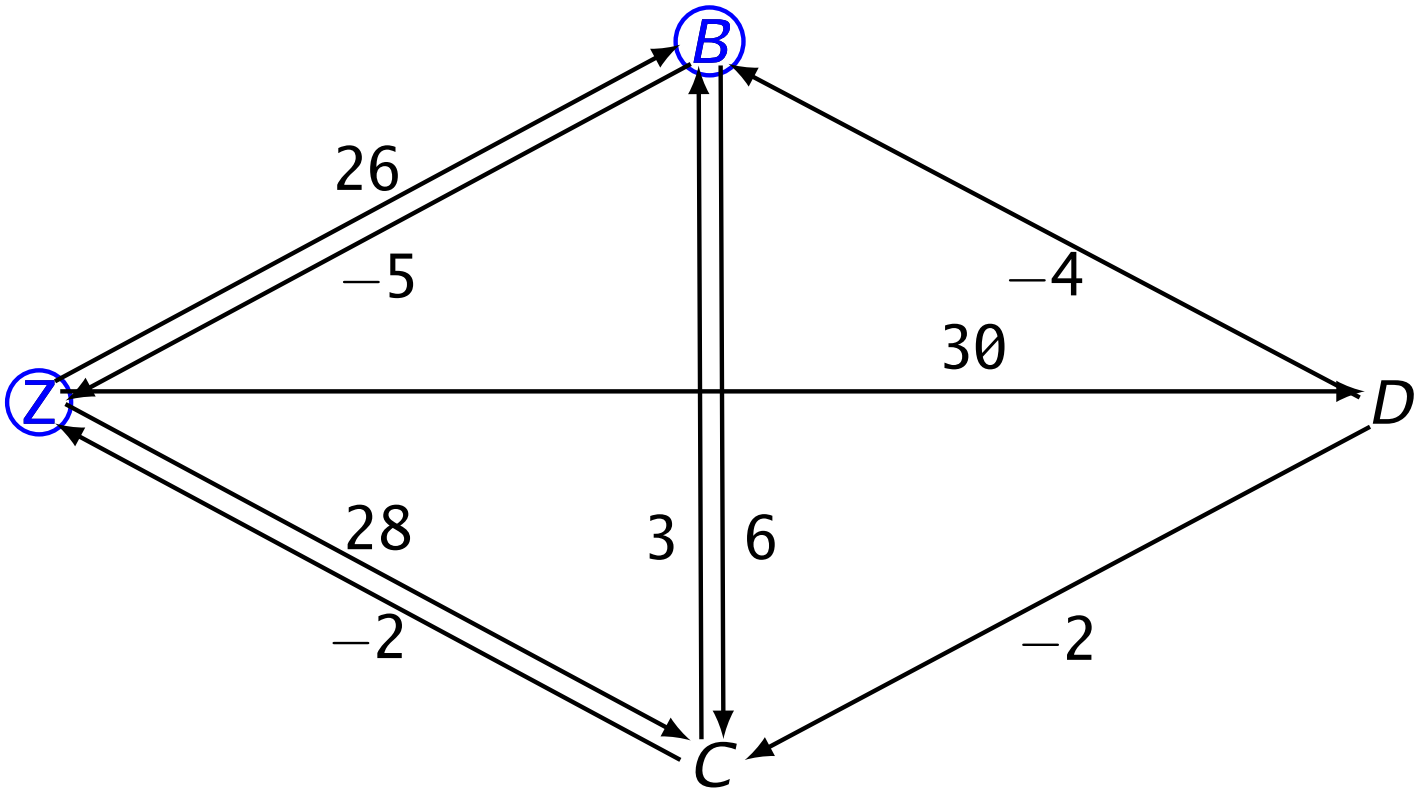
Propagate $B = 12$ to neighbors



$\mathcal{X} = \{Z, B\}$, $t = 12$, $\mathbf{E} = \{C\}$, $C \in [12, 18]$, $D \in [16, 30]$

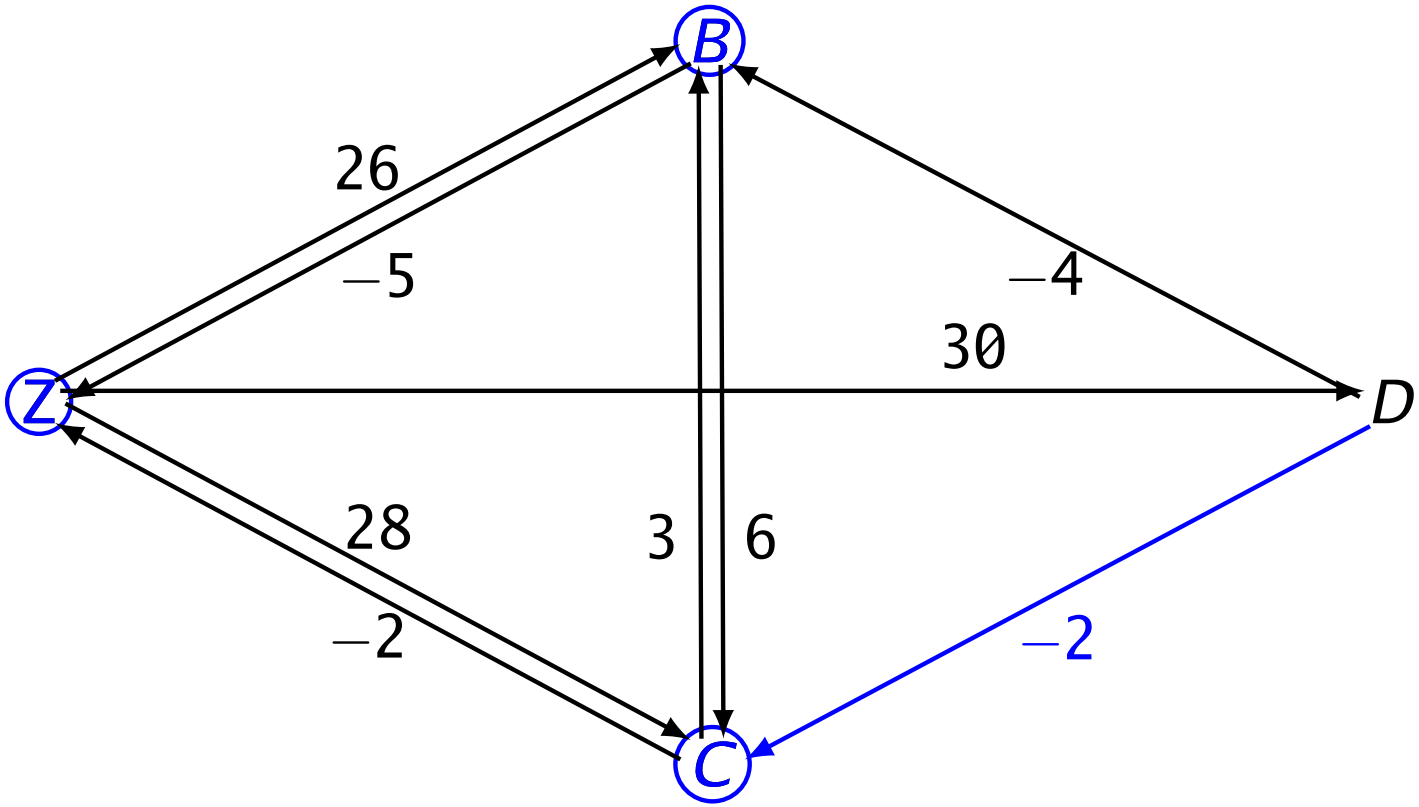
Simple Temporal Network
Dispatching the STN (ctd.)

$\mathcal{X} = \{Z, B\}$, $t = 12$, $\mathbf{E} = \{C\}$, $C \in [12, 18]$, $D \in [16, 30]$



Let t advance to 16, pick C from $\mathbf{E}$, set $C = 16$.

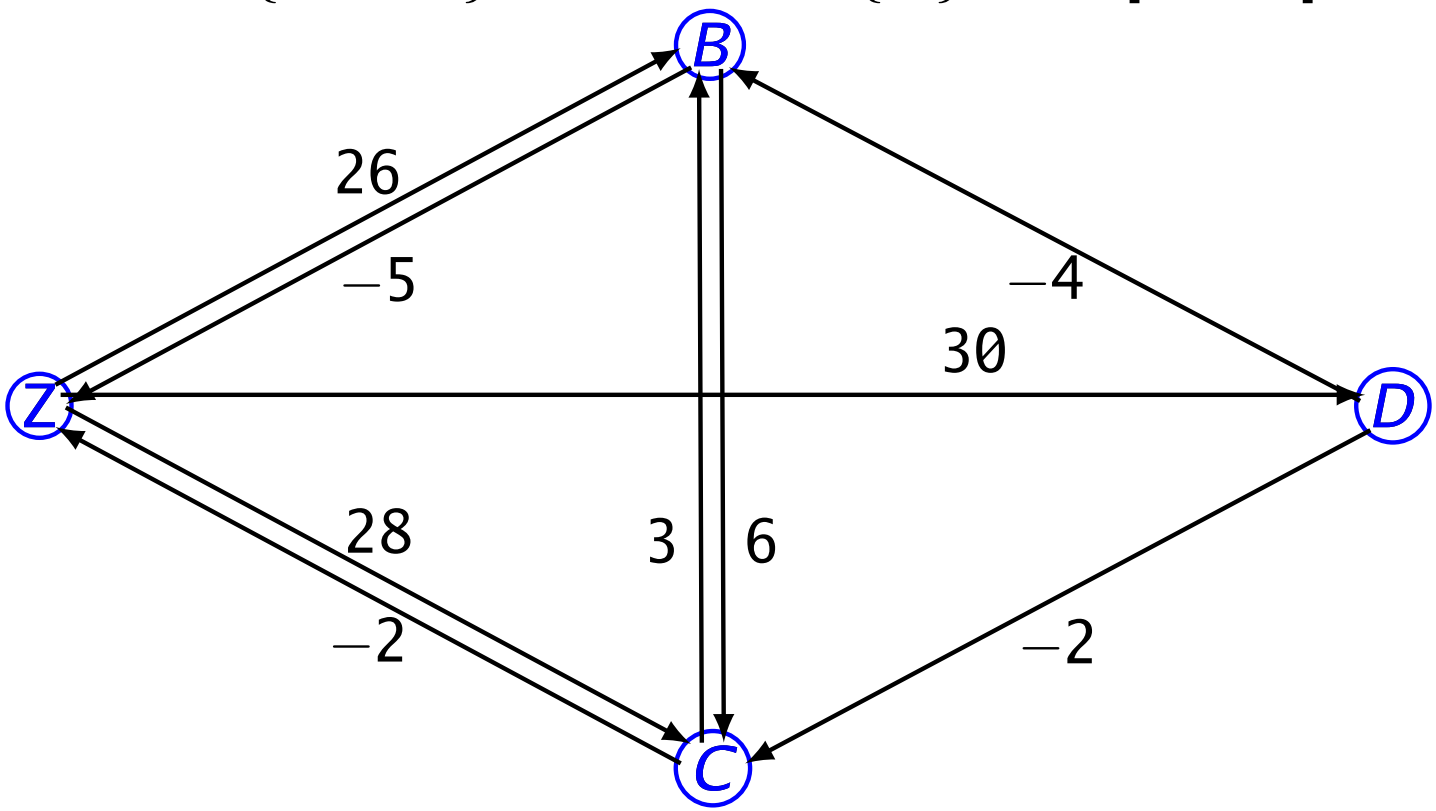
Propagate $C = 16$ to C's only remaining neighbor, D.



$\mathcal{X} = \{Z, B, C\}$, $t = 16$, $\mathbf{E} = \{D\}$, $D \in [18, 30]$

Simple Temporal Network
Dispatching the STN (ctd.)

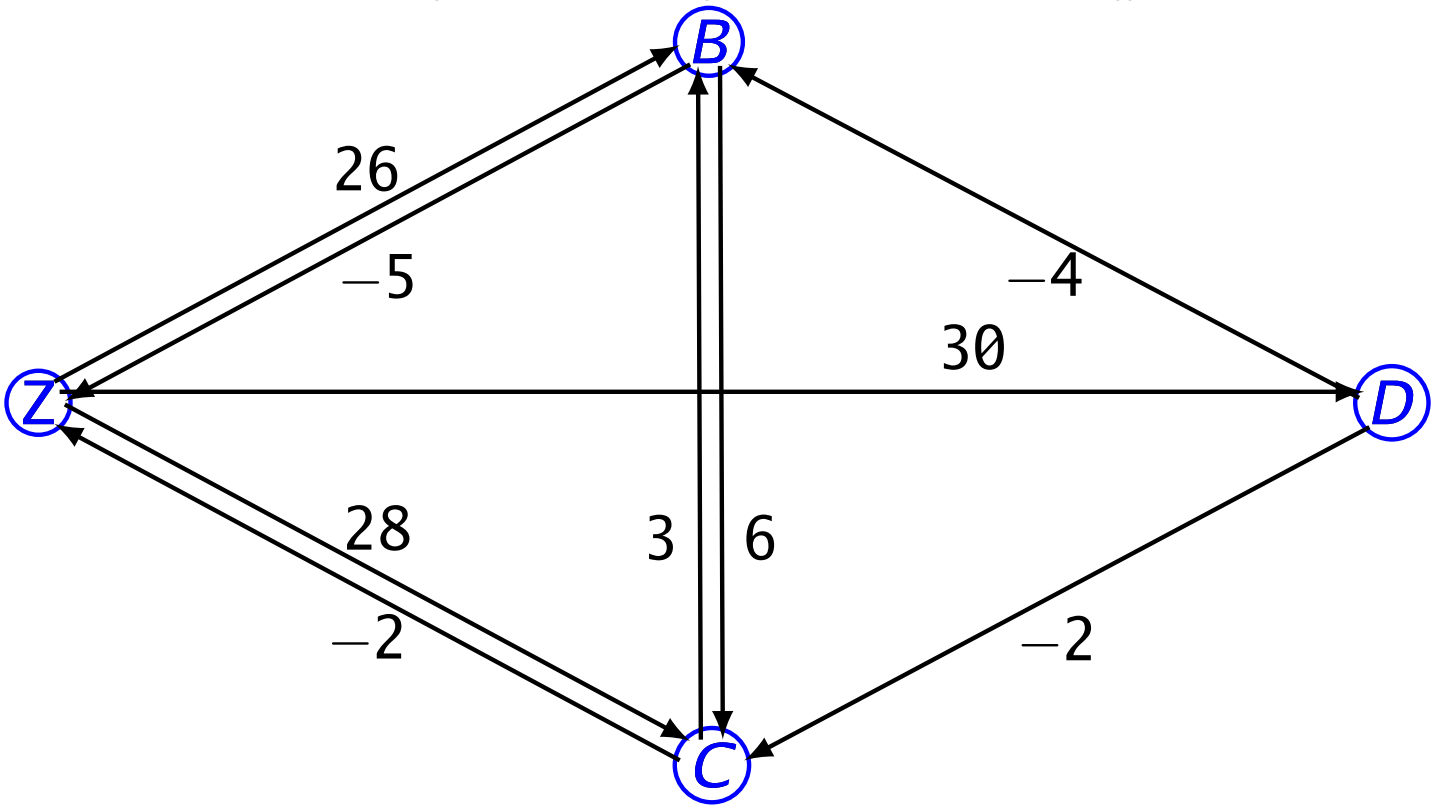
$\mathcal{X} = \{Z, B, C\}$, $t = 16$, $\mathbf{E} = \{D\}$, $D \in [18, 30]$



Let t advance to 25, pick D from $\mathbf{E}$, set $D = 25$.

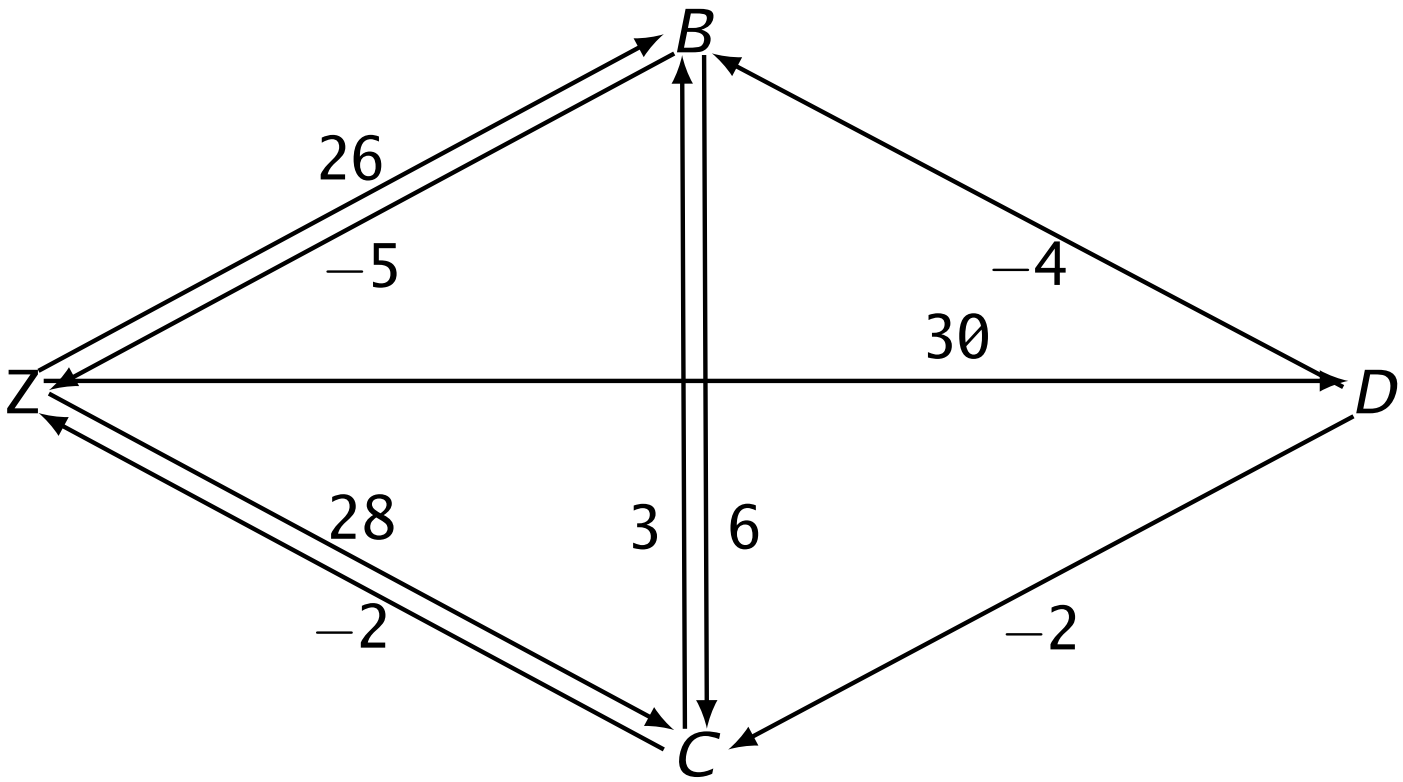
Simple Temporal Network
Dispatching the STN (ctd.)

$\mathcal{X} = \{Z, B, C, D\}$, $t = 25$, $\mathbf{E} = \{\}$



Solution: $Z = 0, B = 12, C = 16, D = 25$.

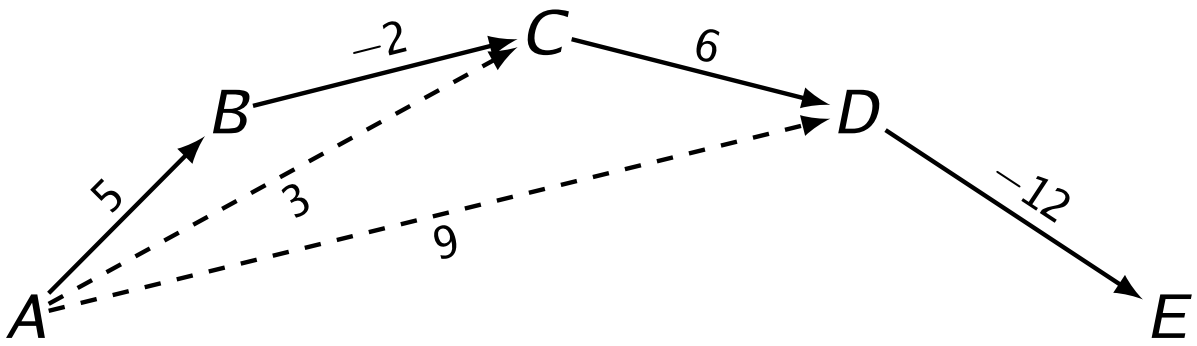
Easy to check that $Z = 0, C = 20, B = 23, D = 28$ can also be generated by the dispatcher.



Simple Temporal Network
New View of Dispatchability*

(1) A path $\mathcal{P}$ has the prefix/postfix (PP) property if:

- Every proper prefix of $\mathcal{P}$ has non–negative length, and
- Every proper postfix of $\mathcal{P}$ has negative length.

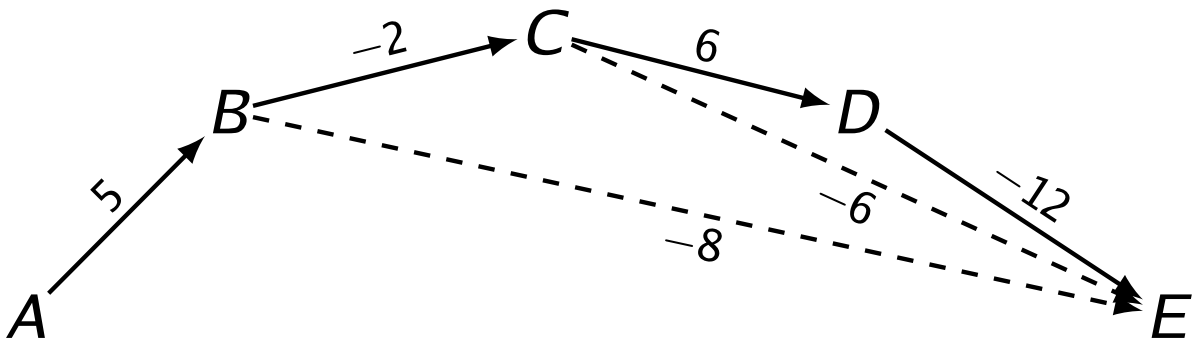


* [Morris, 2014]

Simple Temporal Network
New View of Dispatchability*

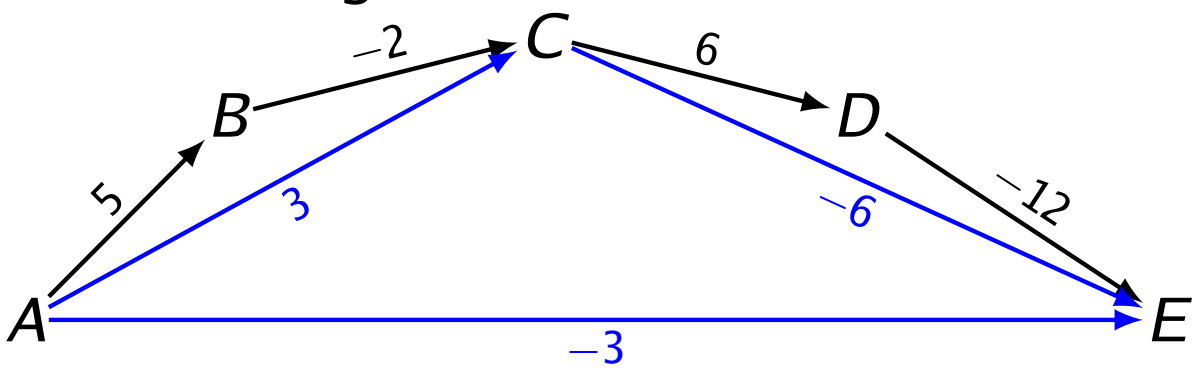
(1) A path $\mathcal{P}$ has the prefix/postfix (PP) property if:

- Every proper prefix of $\mathcal{P}$ has non–negative length, and
- Every proper postfix of $\mathcal{P}$ has negative length.



* [Morris, 2014]

(2) An STN is PP–complete if for each shortest path from any X to any Y that has the prefix/postfix property, there is an edge from X to Y with the same length.



(3) **A consistent and PP–complete STN is dispatchable.**

* [Morris, 2014]

Further graphical analyses of the dispatchability of STNs has been presented recently [Morris, 2016].

Additional Research on STNs

- Temporal Decoupling Problem (TDP) [Hunsberger, 2002, Jr. and Durfee, 2013, Mountakis et al., 2017]
- APSP algorithms on *chordal graphs* [Xu and Choueiry, 2003]
- Enforcing partial path consistency [Planken, 2008]
- Incorporating STNs in multi–agent auctions [Hunsberger and Grosz, 2000]
- For further info:
  `http://www.cs.vassar.edu/~hunsberg/__papers__/`
  (See pages 22–24, 27–28, 32, 53–70, 76–79 of 2005 AAMAS tutorial.)

STN Summary

- STNs have been used to provide flexible planning and scheduling systems for more than a decade.
- Efficient algorithms for checking consistency, incrementally updating the APSP matrix, and managing execution in real time for maximum flexibility.
- However, STNs cannot represent uncertainty (e.g., actions with uncertain durations) or conditional constraints (e.g., only do X if test result is negative).

References I


Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009).
Introduction to Algorithms.
The MIT Press, 3rd edition.


Dechter, R., Meiri, I., and Pearl, J. (1991).
Temporal Constraint Networks.
Artificial Intelligence, 49(1–3):61–95.


Even, S. and Gazit, H. (1985).
Updating Distances in Dynamic Graphs.
Methods of Operations Research, 49:371–387.


Hunsberger, L. (2002).
Group Decision Making and Temporal Reasoning.
PhD thesis, Harvard University.
Available as Harvard Technical Report TR–05–02.


Hunsberger, L. (2008).
A practical temporal constraint management system for real–time applications.
In *European Conf. on Artificial Intelligence (ECAI–2008)*, pages 553–557.

References II


Hunsberger, L. and Grosz, B. J. (2000).
A combinatorial auction for collaborative planning.
In *4th Int. Conf. on Multi–Agent Systems (ICMAS–2000)*.


Jr., J. C. B. and Durfee, E. H. (2013).
Decoupling the multiagent disjunctive temporal problem.
In *27th AAAI Conf. on Artificial Intelligence*.


Morris, P. (2014).
Dynamic controllability and dispatchability relationships.
In *Integration of AI and OR Techniques in Constraint Programming*, volume 8451 of *LNCS*, pages 464–479.


Morris, P. (2016).
The mathematics of dispatchability revisited.
In *26th Int. Conf. on Automated Planning and Scheduling (ICAPS–2016)*, pages 244–252.


Mountakis, K. S., Klos, T., and Witteveen, C. (2017).
Dynamic temporal decoupling.
In *14th Int. Conf. Integration of AI and OR Techniques in Constraint Programming*, volume 10335 of *Lecture Notes in Computer Science (LNCS)*, pages 328–343.

References III


Muscettola, N., Morris, P. H., and Tsamardinos, I. (1998).
Reformulating Temporal Plans for Efficient Execution.
In *6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR–1998)*, pages 444–452.


Planken, L. (2008).
Incrementally Solving the STP by Enforcing Partial Path Consistency.
In *27th PlanSIG Workshop*, pages 87–94.


Ramalingam, G. and Reps, T. (1996).
On the Computational Complexity of Dynamic Graph Problems.
Theoretical Computer Science, 158:233–277.


Ramalingam, G., Song, J., Joskowicz, L., and Miller, R. E. (1999).
Solving Systems of Difference Constraints Incrementally.
Algorithmica, 23(3):261–275.


Rohnert, H. (1985).
A Dynamization of the All Pairs Least Cost Path Problem.
In Mehlhorn, K., editor, *2nd Symp. of Theoretical Aspects of Computer Science (STACS–1985)*, volume 182 of *Lecture Notes in Computer Science (LNCS)*, pages 279–286.

References IV

Tsamardinos, I., Muscettola, N., and Morris, P. (1998).
Fast Transformation of Temporal Plans for Efficient Execution.
In *15th National Conf. on Artificial Intelligence (AAAI–1998)*, pages 254–261.

Xu, L. and Choueiry, B. Y. (2003).
A new efficient algorithm for solving the simple temporal problem.
In *10th Int. Symp. on Temporal Representation and Reasoning and 4th Int. Conf. on Temporal Logic (TIME–ICTL–2003)*, pages 210–220.