

Dynamic Scheduling Under Uncertainty

2nd Summer School on Cognitive Robotics at MIT

Robert Morris, NASA Ames Research Center

July 23, 2018

Outline

- Review of Executive Functions
- Models of Uncertainty
- Dynamic Scheduling
- Dispatching Temporal Plans
- Conclusions

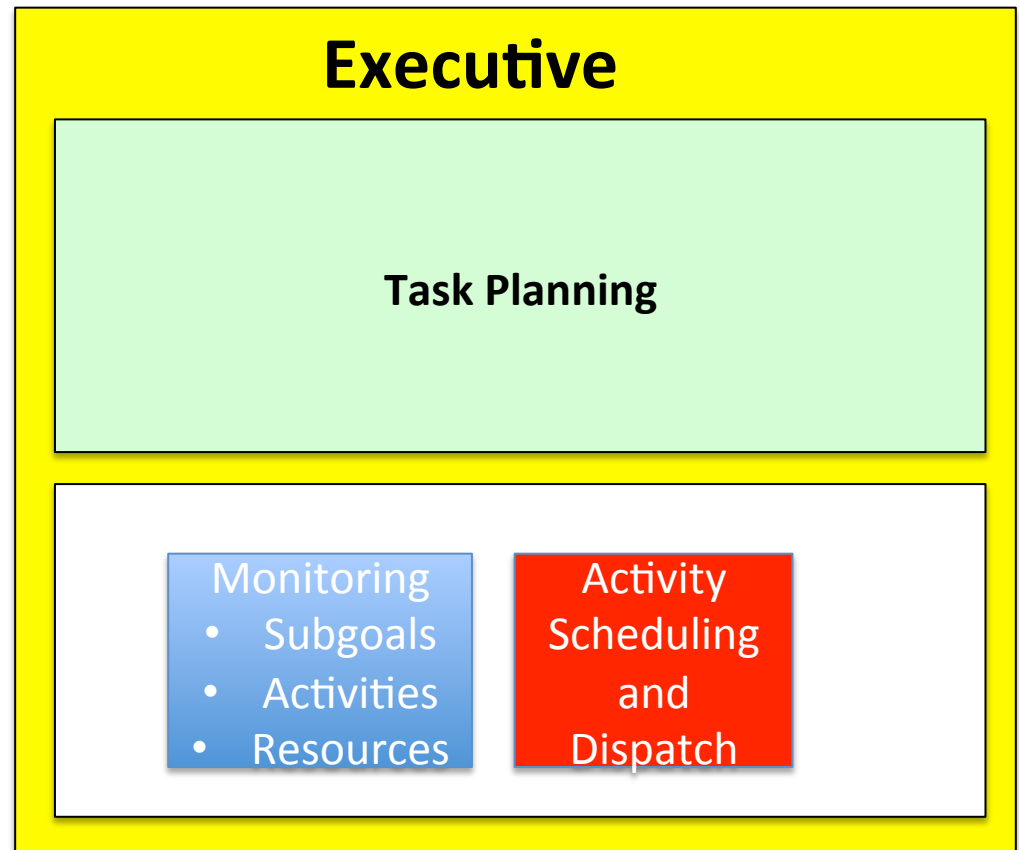


Review of Functions of Executive

- The Executive is the interface between the numerical behavioral control and the symbolic planning layers.
- It is responsible for translating abstract plans into low-level behaviors, invoking behaviors at the appropriate times, monitoring execution, and handling exceptions.
- In a control system with deliberative behaviors, the plan provides robust and effective directives to the executive on how to direct the system towards desired behaviors.
- A “model-based executive” enables goal-directed behavior. A user can specify a goal and the executive can determine the appropriate course of action to meet the goals.

Requirements of executive:

- Simple and Fast
- Robust to uncertainty in world (including the robot hardware)



Scheduling vs Planning



- Scheduling

- Decide **when** and **how** to perform a given set of actions

- Time constraints
- Resource constraints
- Objective functions

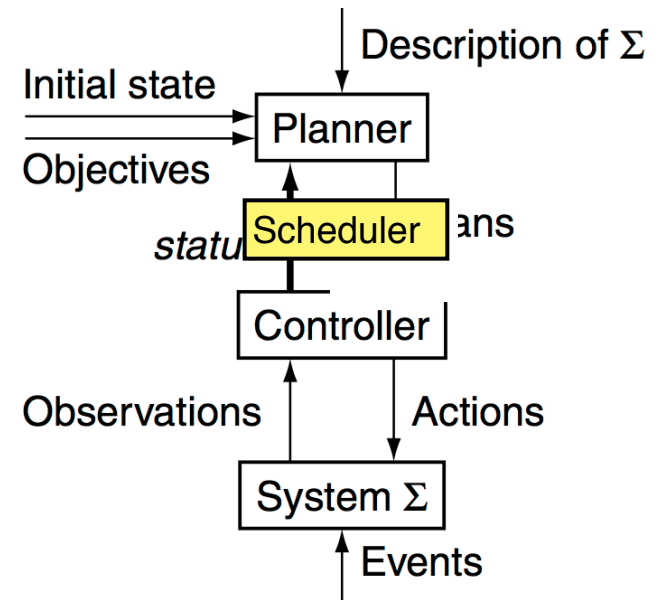
- Planning

- Decide **what** actions to use to achieve a set of objectives

- Performing planning and scheduling in sequence is difficult or impossible when planning for complex systems

- Concurrent actions
- Shared resources

- More recent approaches combine both.





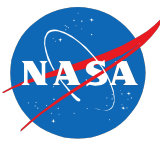
Plan Execution

- Simple Sequential Plan Execution:
 - loop
 - Take next action from plan
 - Send action to control (Dispatch)
 - Wait for action to finish (Monitor)
- Extensions to make plans more robust:
 - Time and resource management
 - Conditional dispatch
 - Looping



Time and Resource Executive

- Real world requires time and resource models
 - Durations, orderings and deadlines
 - Battery consumption
- Time & resource executive:
 - 1. Send time-enabled actions to control
 - Record the time of dispatch
 - 2. Propagate effects on the time of future actions
 - 3. Monitor effects of actions on resources
 - 4. Goto 1



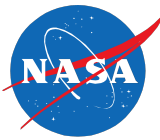
Conditional Executive

- Real world is uncertain
 - The effects of actions might be non-deterministic
 - Environment is dynamic and changing (full of exogenous events)
 - Environment often filled with unknowns (world model is incomplete).
- Solution: plans with conditional ‘branches’
- Executive must have a ‘condition monitor’ (state estimator) to decide what actions to dispatch



Other Extensions

- Dispatching more than a single plan
- Strategic vs. Tactical planning
 - Hierarchical planning
- Recursive planning
 - A plan might be interrupted by the opportunity for executing another plan
 - Once completed, the original plan is resumed

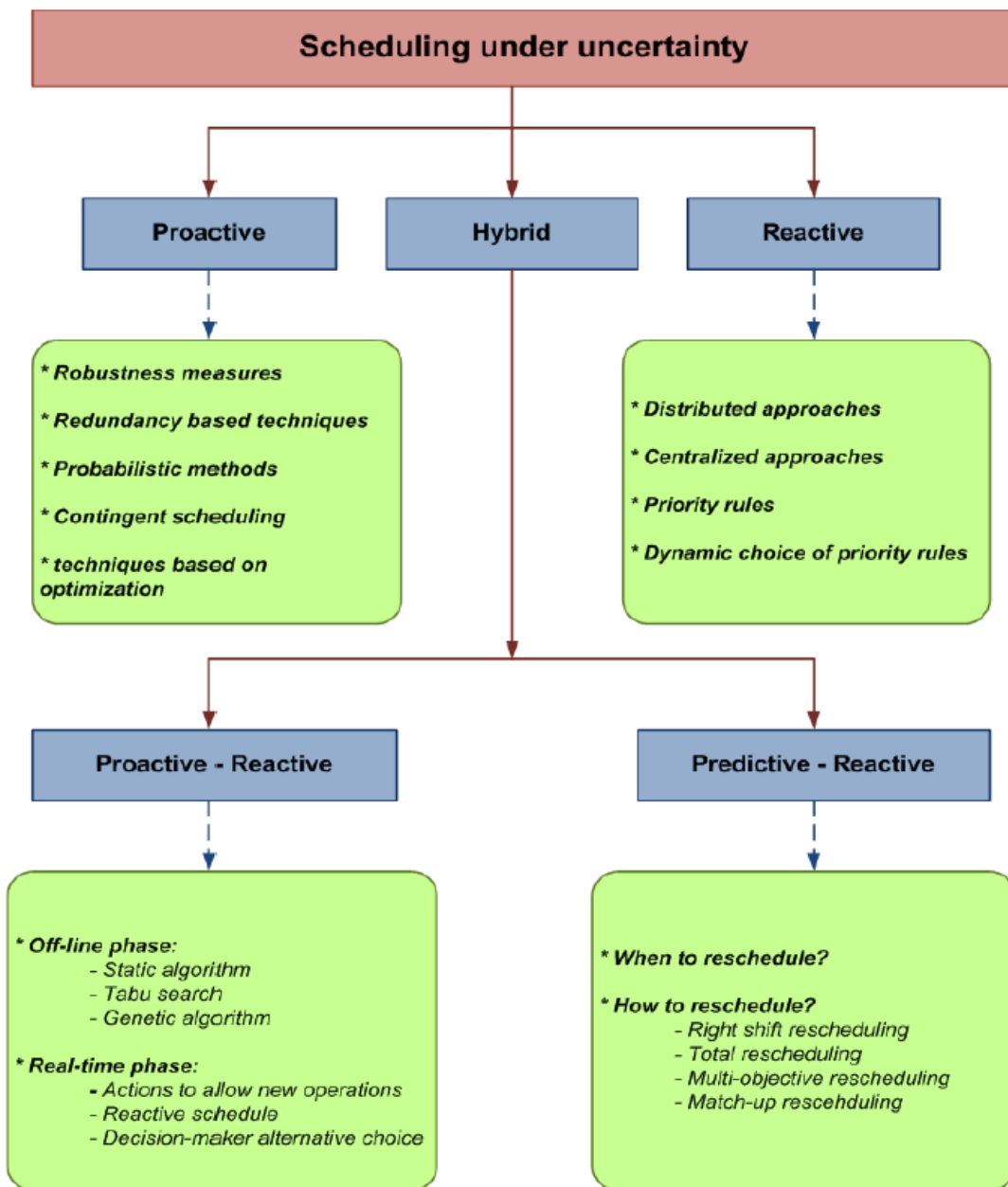


Uncertainty in Cognitive Robotics

- Uncertainty is Ubiquitous
 - Wheels slip (uncertainty in the effects of actions)
 - Sensors are affected by noise (knowledge about the current state is limited)
 - Objects move unpredictably (things happen in the world that are not under the control of the robot)
- Mobile robots need to be robust to these kinds of uncertainty.
 - They need to accomplish goals while remaining safe despite uncertainty
- Approaches to handling uncertainty
 - Reactive: recovering from the effect of uncertainty after it has occurred
 - Proactive: apply models to anticipate uncertainty and generate robust plans/schedules
 - Hybrid: combination of both
- Many kinds of uncertainty can be met by modeling and reasoning about it.
 - Non-deterministic models → uncertainty as a set of possible outcomes
 - Probabilistic models → uncertainty as conditional probability distribution over set of outcomes
- Uncertainty as a 'game against nature'.
 - Nature as an external 'decision maker' whose decisions set the values uncertain state parameters.



Approaches to scheduling under uncertainty

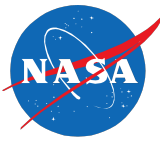


- **Proactive:** Model uncertainty and produce robust schedules (“ones that satisfy performance requirements predictively in an uncertain environment”).
- **Reactive:** Continually re-schedule based on changes to execution environment.
- **Hybrid Proactive-reactive:** Build a set of static schedules that make it easy to pass from one to another at run-time
- **Hybrid Predictive-reactive:** make a deterministic schedule off-line and adapt it during run-time.



Example: Slack-based Scheduling

- Main idea: provide each activity with extra time to execute so that some level of uncertainty can be absorbed without re-scheduling.
- How to add slack?
 - Before scheduling (temporal projection)
 - Statically as part of the problem definition (Time Window Slack, TWS)
 - Allows for reasoning about amount of slack available during scheduling.
 - Relative to where the activity is scheduled (Focused TWS)
 - Uses statistics to focus the slack on areas of the horizon that are in need of it.
 - Example: Focus on resources (machines) that have not been maintained in quite a while.



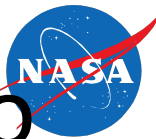
Dynamic Scheduling

- In classical scheduling, it is common to assume full information: all tasks and their constraints are known prior to any decision making.
- *Dynamic scheduling problems are characterized by a stream of tasks arriving stochastically over time.* Each job requires a combination of resources, sequentially and/or in parallel, for different processing times. The existence of any particular job and its corresponding characteristics are not known until its arrival.
- Dynamic scheduling problems consist of both challenging combinatorics, as found in classical scheduling problems, and stochastics due to uncertainty about the arrival times, resource requirements, and processing times of jobs.
- To solve a dynamic scheduling problem, the jobs must be assigned to appropriate re-sources and start times, respecting the resource and temporal constraints. As jobs arrive, there must be an online process to make decisions: it is not possible to solve the entire problem online.
- Often we settle for evaluating algorithm performance over a finite time horizon; we refer to such criteria as short-term criteria. In contrast, long-term objectives focus on system-level performance measures such as stability, which establishes whether the instantaneous number of jobs will remain finite over an infinite horizon for particular system parameters.

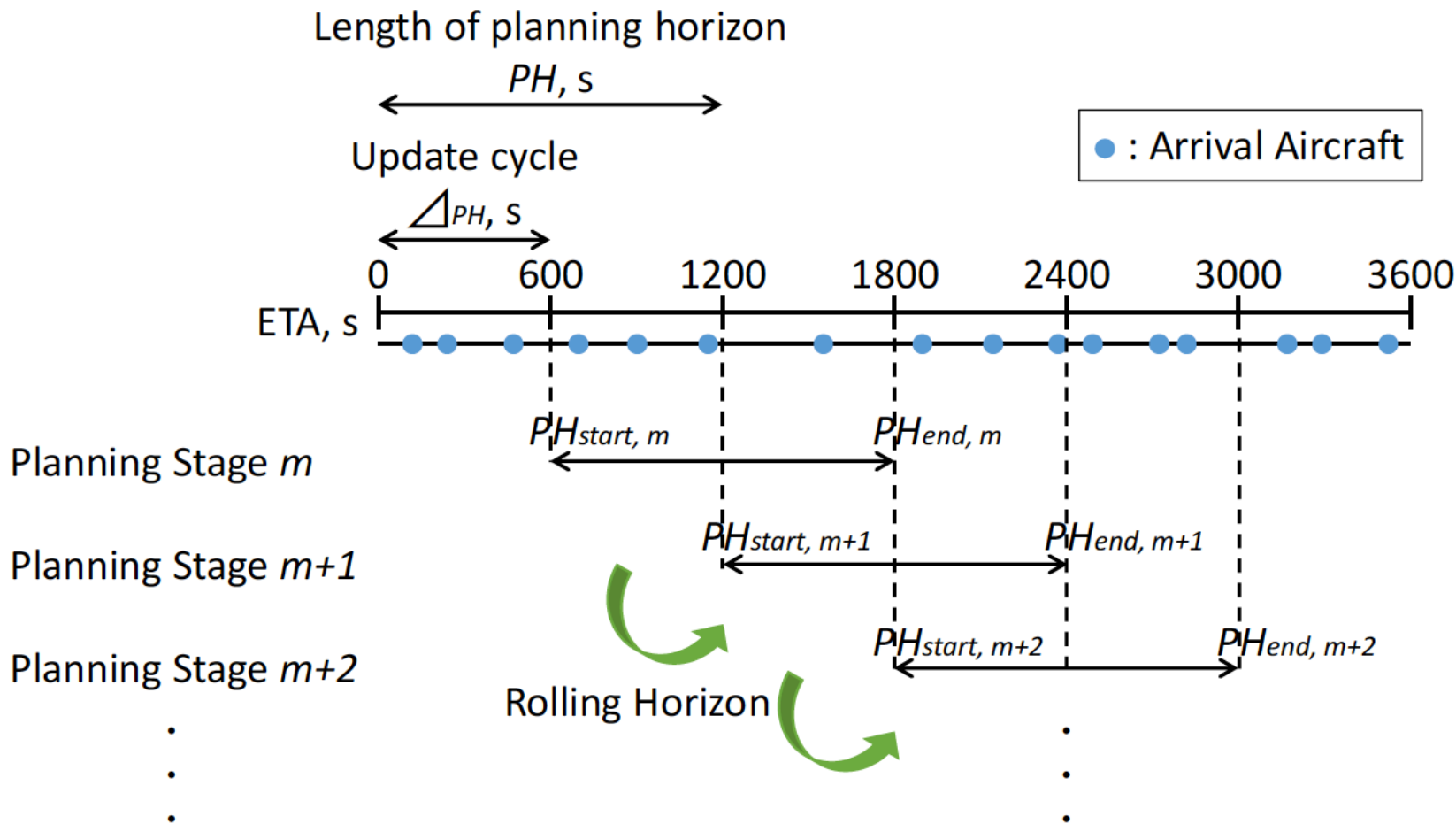


Approaches to Solving Dynamic Scheduling Problems

- Solving dynamic problems typically involves the periodic scheduling approach of solving a collection of linked static sub-problems.
- At a given time point, the static problem, consisting of the jobs currently present in the system, is solved to optimize some short-term objective function.
- That schedule is then executed (wholly or partially) until the creation of the next sub-problem is triggered. This viewpoint means that methods developed for static scheduling problems become directly applicable to dynamic ones.
- Such methods can effectively deal with complex combinatorics and can optimize the quality of schedules for each static sub-problem. However, they tend to overlook long-run performance and stochastic properties of the system.
- There is interest in understanding how long-term objectives can be achieved by solving a series of short-term, static scheduling problems.
 - Queueing theory has been used to characterize system stability (finite waiting lines)



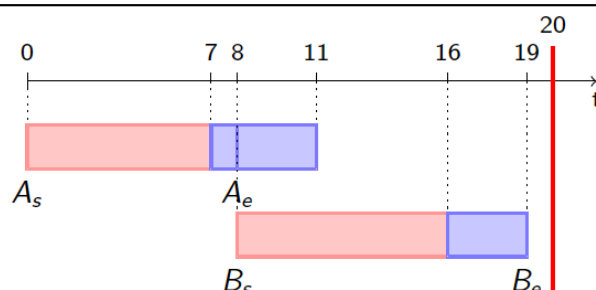
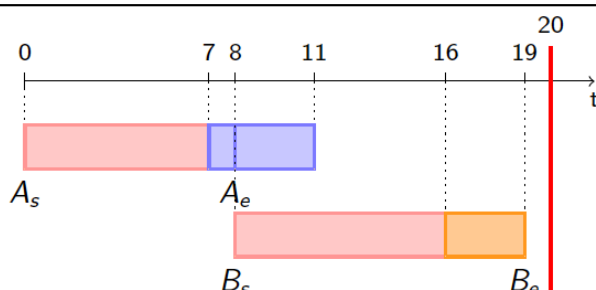
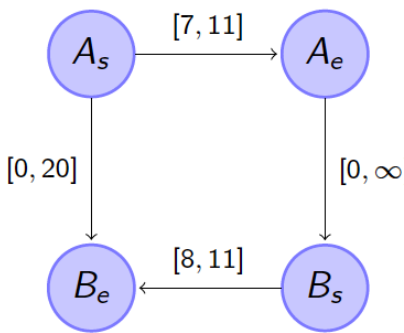
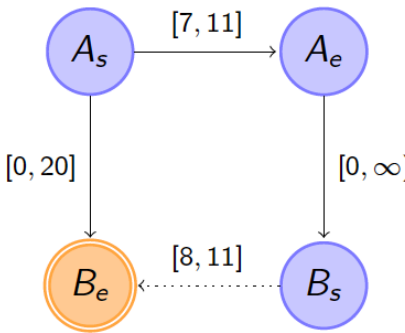
Example: Rolling Horizon Approach to Aircraft Runway Scheduling





Dynamically Scheduling Temporal Plans

- Plans are subject to temporal constraints on actions with duration
- Actions must be dispatched in a way that satisfies the constraints
- The executive might not always be able to control action duration

		Uncertainty Type	
		No Uncertainty	Uncertainty
Activities			
TimePoints			

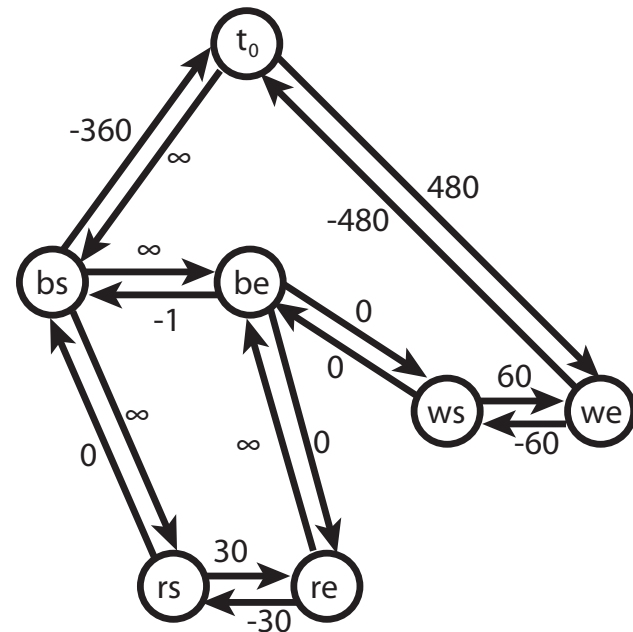
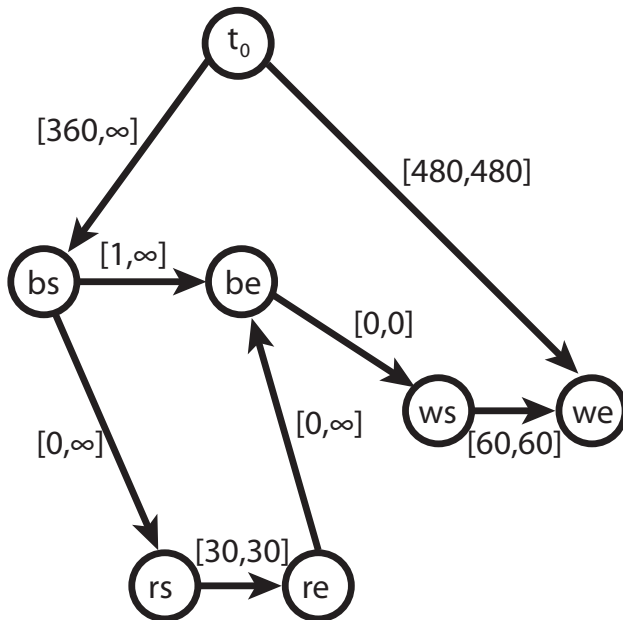


Simple Temporal Problems

- The basic temporal primitives are **time points**.
- Simple **temporal constraints** for instants t_i and t_j :
 - unary: $a_i \leq t_i \leq b_i$
 - binary: $a_{ij} \leq t_i - t_j \leq b_{ij}$,
where a_i, b_i, a_{ij}, b_{ij} are (real) constants
- **Simple Temporal Network (STN)**
 - constraints $r_{ij} = [a_{ij}, b_{ij}]$ are used
 - **operations**:
 - composition: $r_{ij} \bullet r_{jk} = [a_{ij} + a_{jk}, b_{ij} + b_{jk}]$
 - intersection: $r_{ij} \cap r'_{ij} = [\max\{a_{ij}, a'_{ij}\}, \min\{b_{ij}, b'_{ij}\}]$
 - **STN is consistent** if there is an assignment of values to time points (a **solution**) satisfying all the temporal constraints.
 - **Path consistency** is a complete technique making STN consistent (all inconsistent values are filtered out, one iteration is enough).
Another option is using all-pairs minimal distance **Floyd-Warshall algorithm**
- Two STNs are **equivalent** if they have the same solutions.

Distance graph

- Relations $a_{ij} \leq t_i - t_j \leq b_{ij}$ can be expressed as a pair of inequalities:
 - $t_i - t_j \leq b_{ij}$
 - $t_j - t_i \leq -a_{ij}$
 - Absolute times can be defined as a distance to a reference point t_0
- Solving a set of linear equalities is well-known.
- The inequalities can be given a graphical notation, called a **distance graph**.
 - Shortest path algorithms can be applied to do useful things.
 - Negative cycle in the distance graph means inconsistency.



Fundamental Theorem of STN



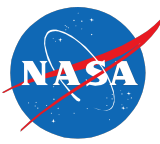
For an STN S with graph G and distance Matrix d , the following are equivalent:

- S is consistent
- d has non-negative values on main diagonal;
- G has no negative length loops

(Dechter, Meiri, Pearl, 1991)

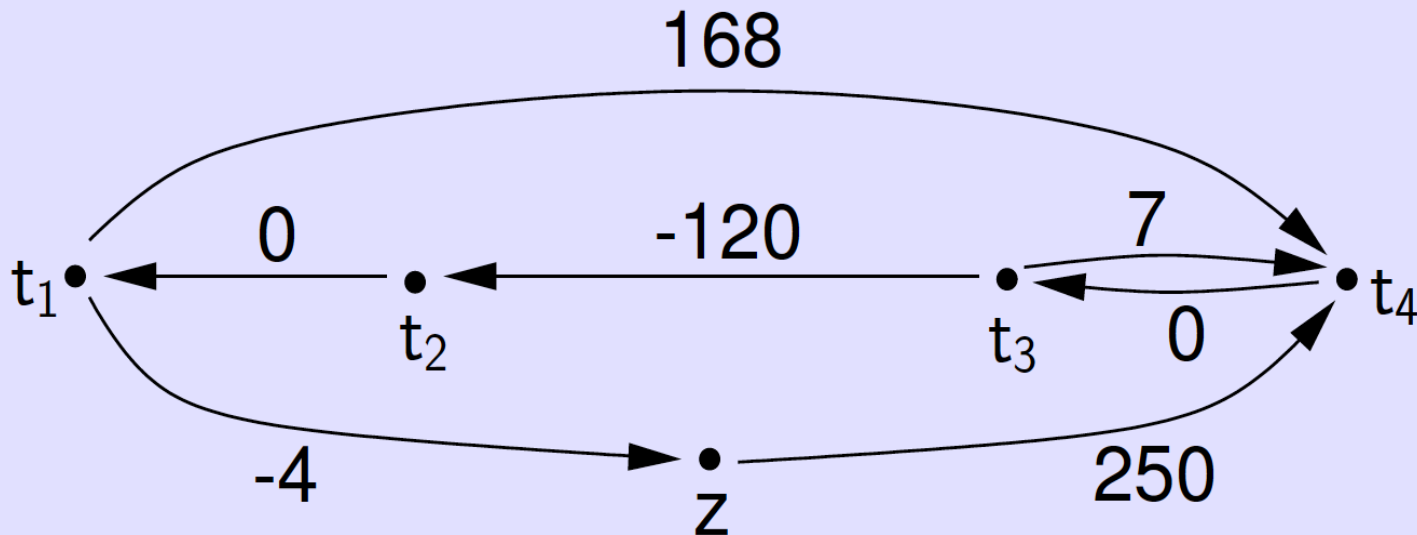
- **Floyd-Warshall algorithm $O(n^3)$**
 - finds minimal distances between all pairs of nodes
 - First, the temporal network is converted into a **distance graph**
 - there is an arc from i to j with distance b_{ij}
 - there is an arc from j to i with distance $-a_{ij}$.
 - STN is consistent iff there are no negative cycles in the graph
 - Same as saying $d(i,i) \geq 0$
 - d is called the distance matrix

```
Floyd-Warshall( $X, E$ )
  for each  $i$  and  $j$  in  $X$  do
    if  $(i, j) \in E$  then  $d(i, j) \leftarrow l_{ij}$  else  $d(i, j) \leftarrow \infty$ 
     $d(i, i) \leftarrow 0$ 
  for each  $i, j, k$  in  $X$  do
     $d(i, j) \leftarrow \min\{d(i, j), d(i, k) + d(k, j)\}$ 
end
```



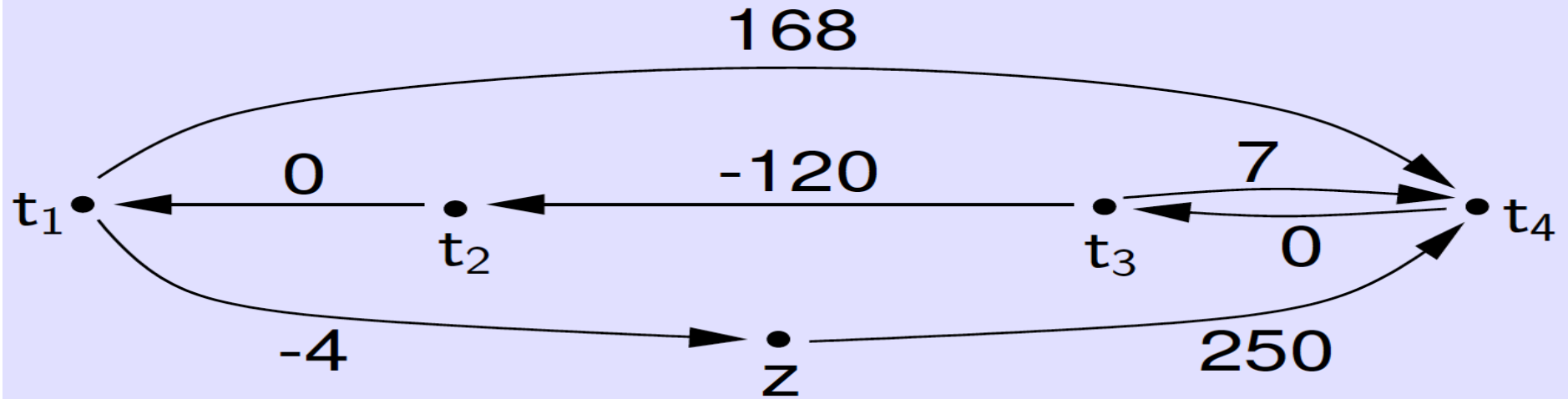
Set of Temporal Constraints and Corresponding Distance Graph

$$\left\{ \begin{array}{ll} Z - t_1 \leq -4, & t_4 - Z \leq 250 \\ t_4 - t_1 \leq 168, & t_2 - t_3 \leq -120 \\ t_4 - t_3 \leq 7, & t_1 - t_2 \leq 0 \\ t_3 - t_4 \leq 0 & \end{array} \right\}$$





STN and Minimal Distance Matrix



\mathcal{D}	z	t_1	t_2	t_3	t_4
z	0	130	130	250	250
t_1	-4	0	48	168	168
t_2	-4	0	0	168	168
t_3	-124	-120	-120	0	7
t_4	-124	-120	-120	0	0

Dynamic Updates and Incremental Consistency



Dynamic Updates:

- Update d incrementally by adding or deleting constraints
- Sometimes less expensive than computing d from scratch (Demetrescu and Italiano 2002)

Incremental consistency

- Verifying consistency after inserting/weakening constraints is less expensive than fully updating the distance matrix.
- Can verify consistency in $O(m + n \log n)$ time after inserting a new constraint. (Ramalingam et al. 1999)

Scheduling Temporal Plans

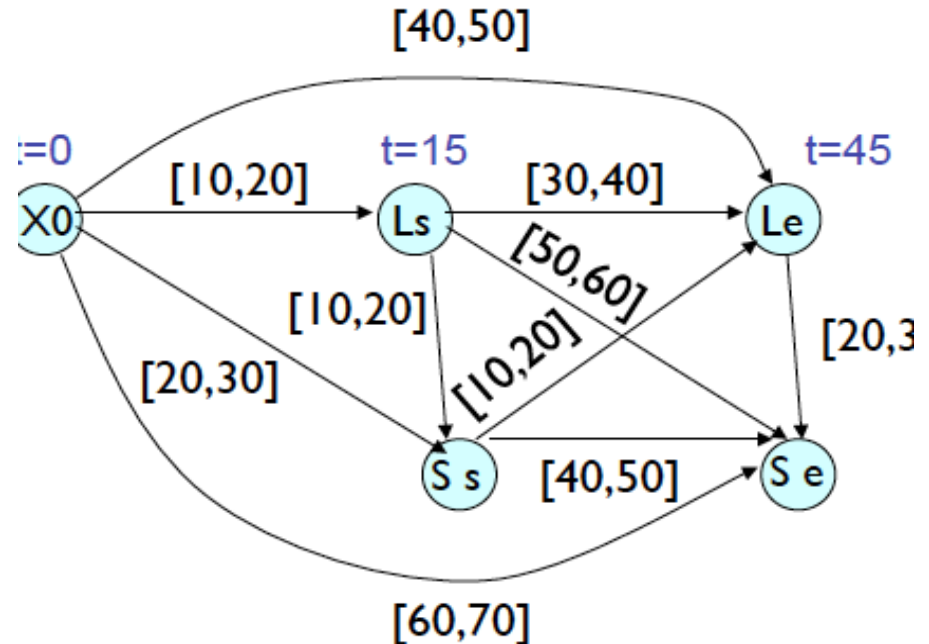
Option 1. Schedule Off Line

Describe Temporal Plan

Test for Consistency

Schedule Plan

Execute Plan



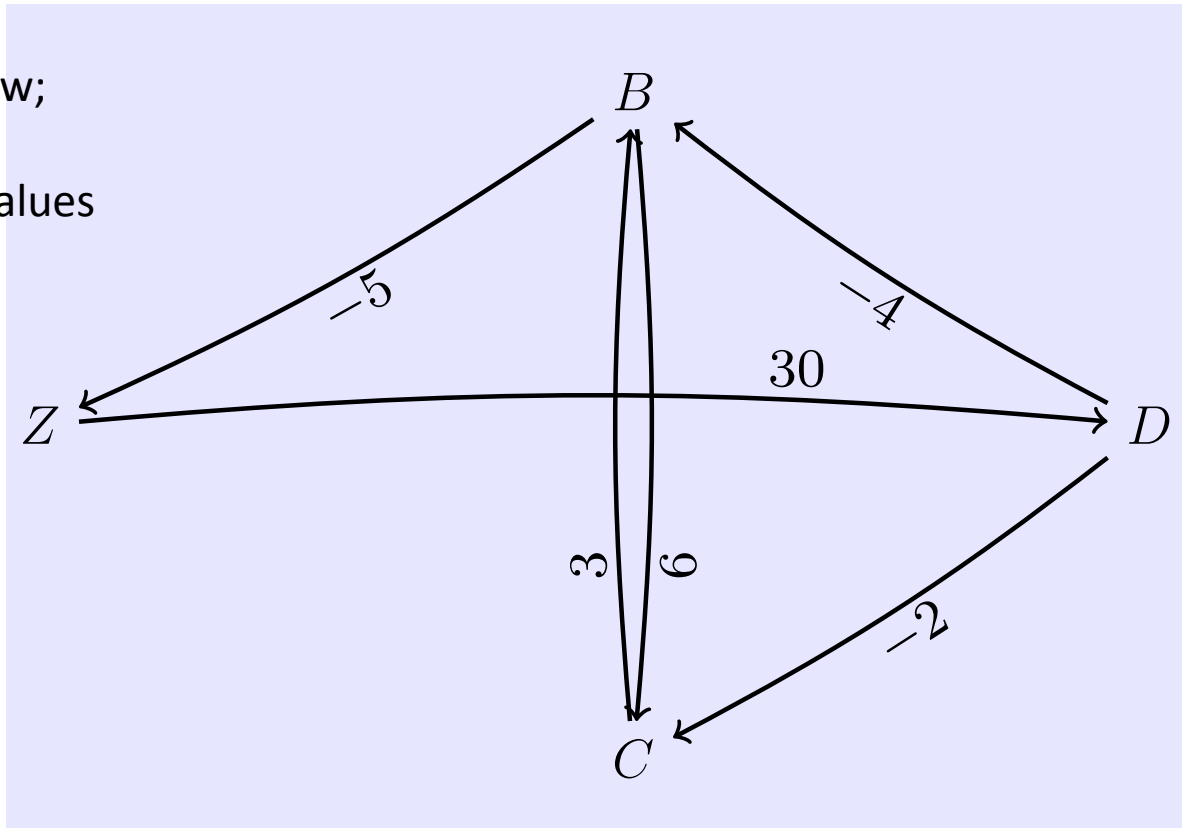
- Given an STN, a schedule is an assignment of times to all variables.
- To generate a schedule from a STN without search, first transform STN into a 'decomposable' STN, where all the implied constraints are revealed, using the all-pairs-shortest-path algorithm (Floyd-Warshall)
- Then incrementally assign times to variables (in any order) and propagate.



Scheduling Actions in the Plan)

- Pick any time-point that doesn't yet have a value;
- Give it a value from its time-window;
- Update D; ← Expensive
- Repeat until all time-points have values

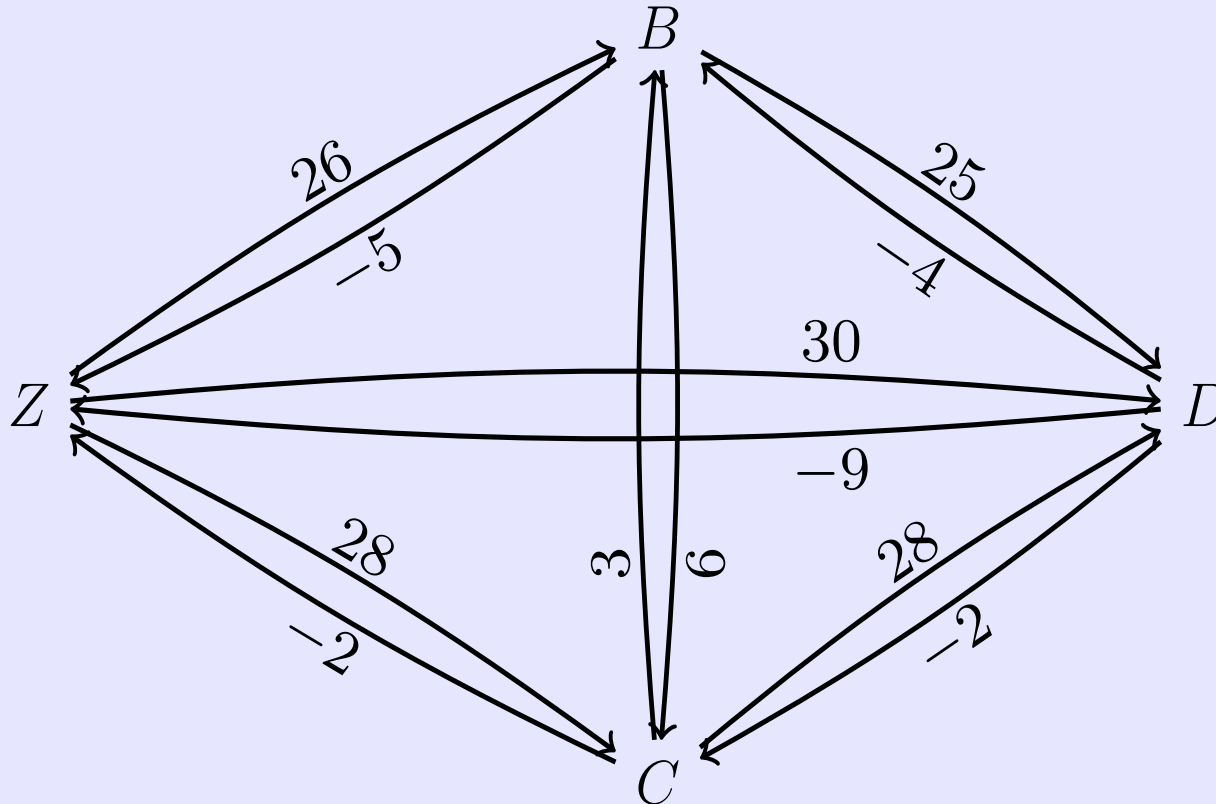
Example STN





Solving an Example STN

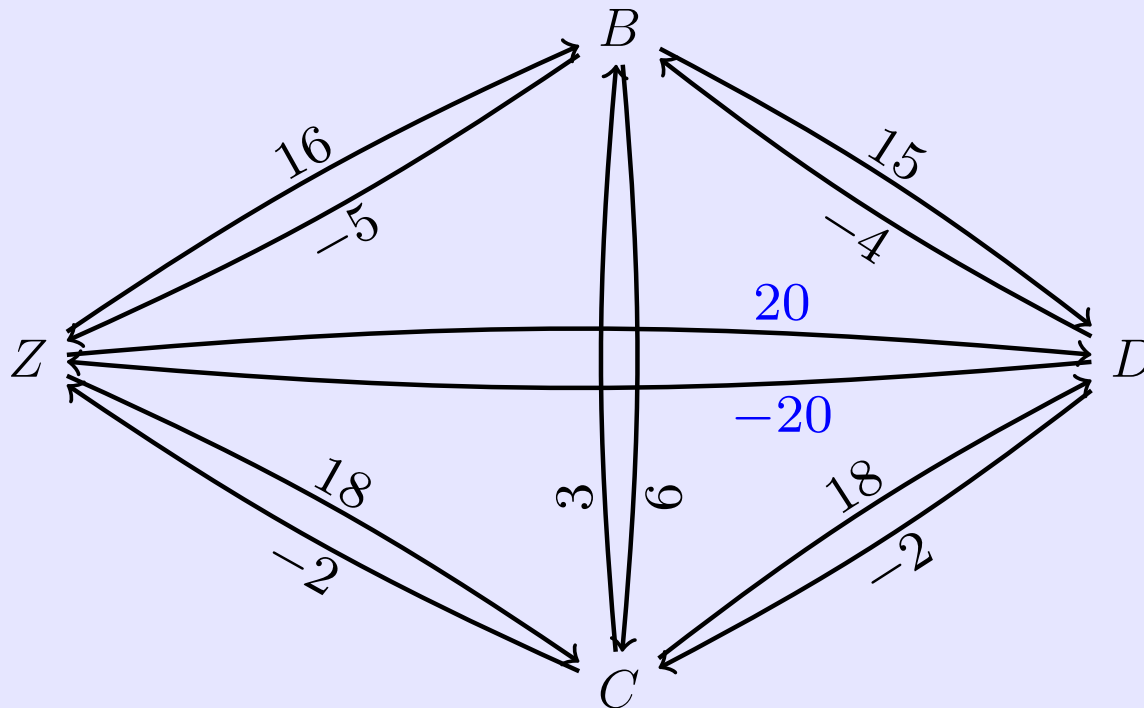
First, form APSP graph (equiv. compute \mathcal{D}).



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

“Solving” Sample STN (ctd.)

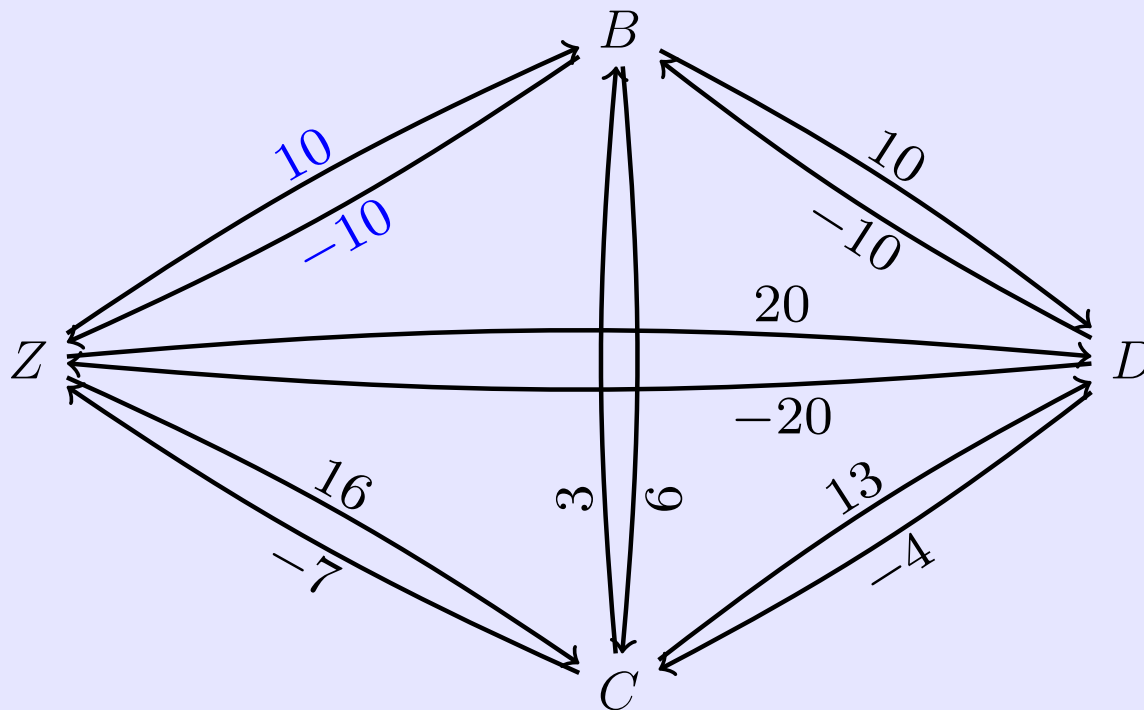
Next, select $D = 20$; and update APSP graph:



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

“Solving” Sample STN (ctd.)

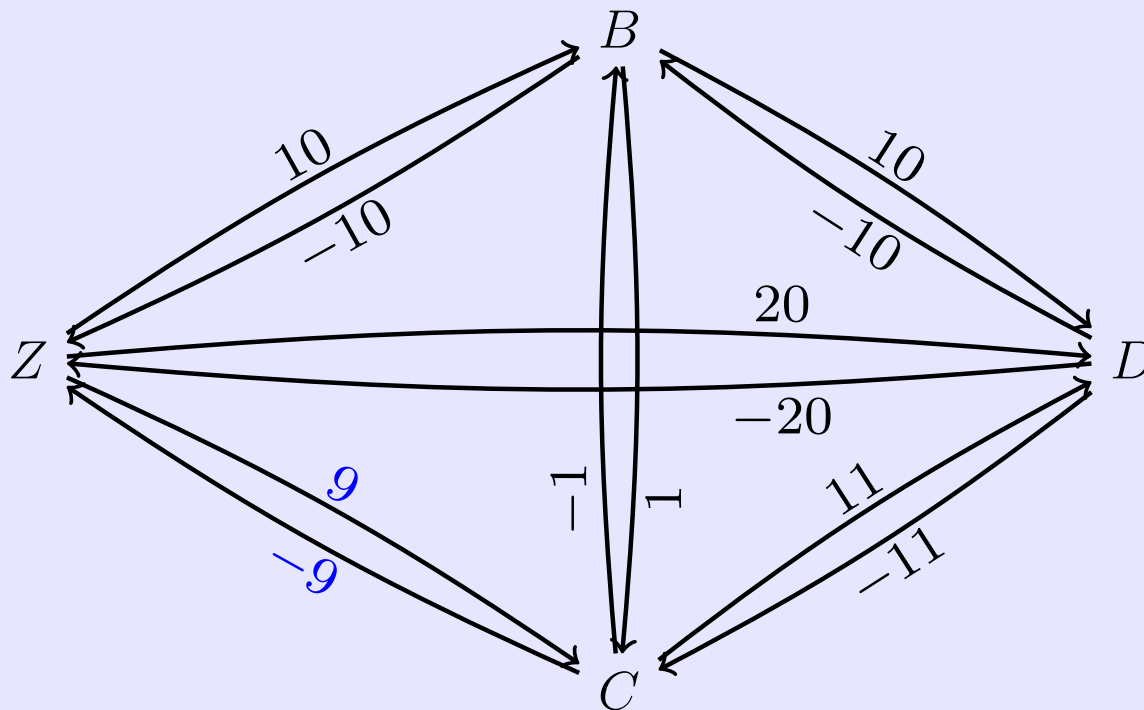
Next, select $B = 10$; and update APSP graph:



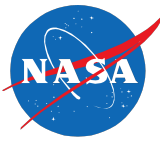
Remaining Time Windows: $C \in [7, 16]$

“Solving” Sample STN (ctd.)

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



Easy to verify that this is a solution.



Executing Temporal Plans

- Autonomous systems with a deliberative (planning) component combine planning with execution.
 - The subsystem responsible for carrying out a plan is called the *executive*.
 - When dispatching plans with flexible temporal constraints, there is a need for a separate *dispatcher*.
- Dispatcher notifies executive when an action **can** or **must** be executed.
 - Correctness: whatever executive does adheres to temporal constraints
 - Preserves flexibility: dispatcher never tells the executive that an action can't be performed at a certain time when it can.



Dispatching Temporal Plans

Option 1. Schedule Off Line

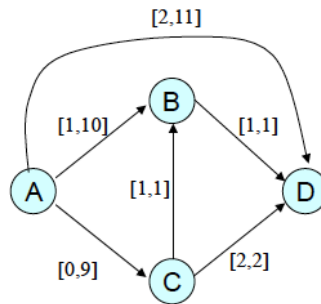
Option 2. Schedule On Line

Describe Temporal Plan

Test for Consistency

Schedule Plan

Execute Plan



Describe Temporal Plan

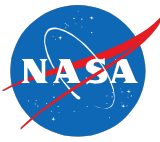
Test for Consistency

Reformulate Plan

Dynamic Execution

Off line

On line



Executing Temporal Plans

Option 1. Schedule Off Line

Describe Temporal
Plan

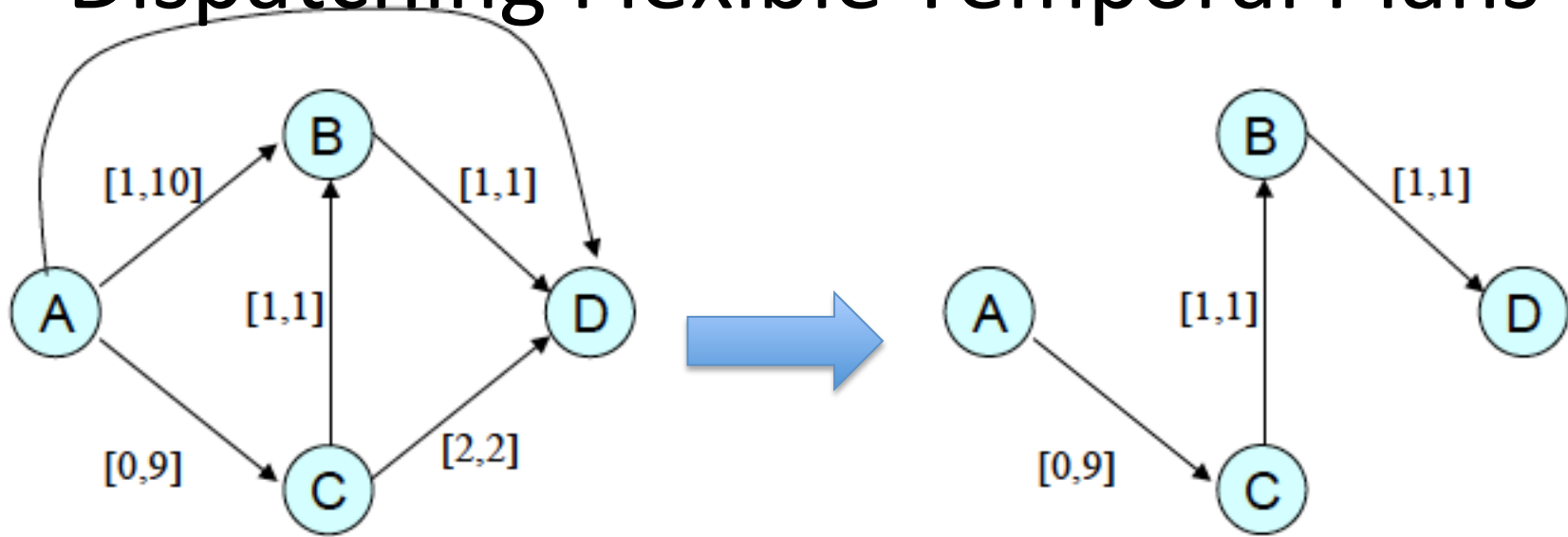
Test for Consistency

Schedule Plan

Execute Plan

- Problem: changes in task duration can cause plan failure if scheduling occurs off line.
 - Fixed schedule removes flexibility
- Solution: Execution adapts through dynamic scheduling.
 - Assign time to event at execution time.
 - Guarantee that all constraints will be satisfied.

Dispatching Flexible Temporal Plans



- Problem: execution latency while propagating effects of assigning times.
- Solution: generate equivalent STN with low latency through removal of **dominated** edges.
- This reformulation minimizes latency.

Reformulate Plan

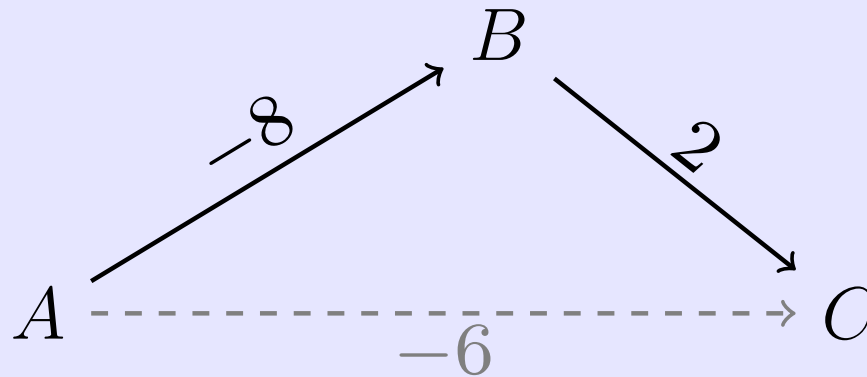
Off line

On line

Dynamic Execution

Remove Dominated Edges

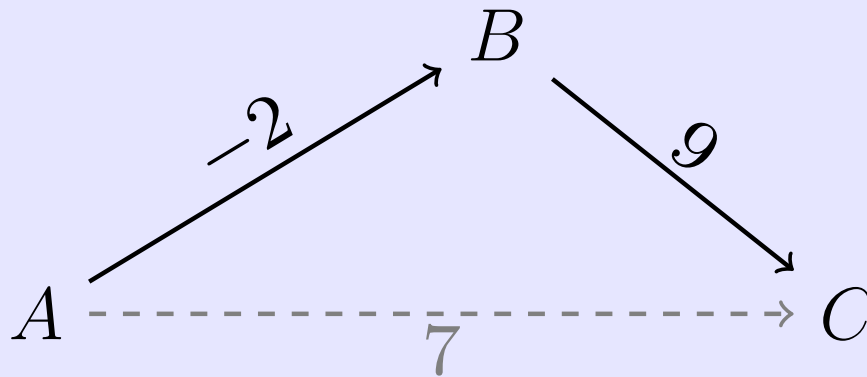
A **negative** edge AC is dominated by a **negative** edge AB if $\mathcal{D}(A, B) + \mathcal{D}(B, C) = \mathcal{D}(A, C)$:



Note: AB and AC have the *same source* node: A .

Remove Dominated Edges (ctd.)

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



Note: BC and AC have the *same destination* node: A .



Dynamic Execution of Flexible STNs

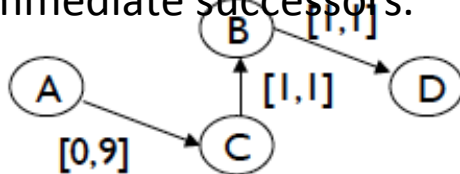
Option 2. Schedule On Line

- Execute an event when it is *enabled* and *active*.
- *Enabled* : predecessors of event have been scheduled.
- *Active* : Current time is within bounds of event.
- Algorithm: when event is enabled and active, assign time and propagate effects to immediate successors.

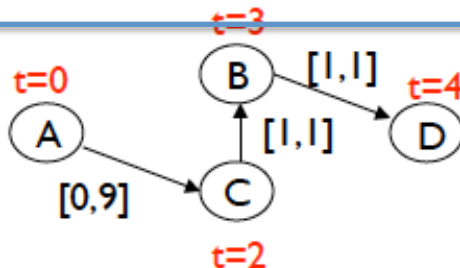
Describe Temporal Plan

Test for Consistency

Reformulate Plan



Off line

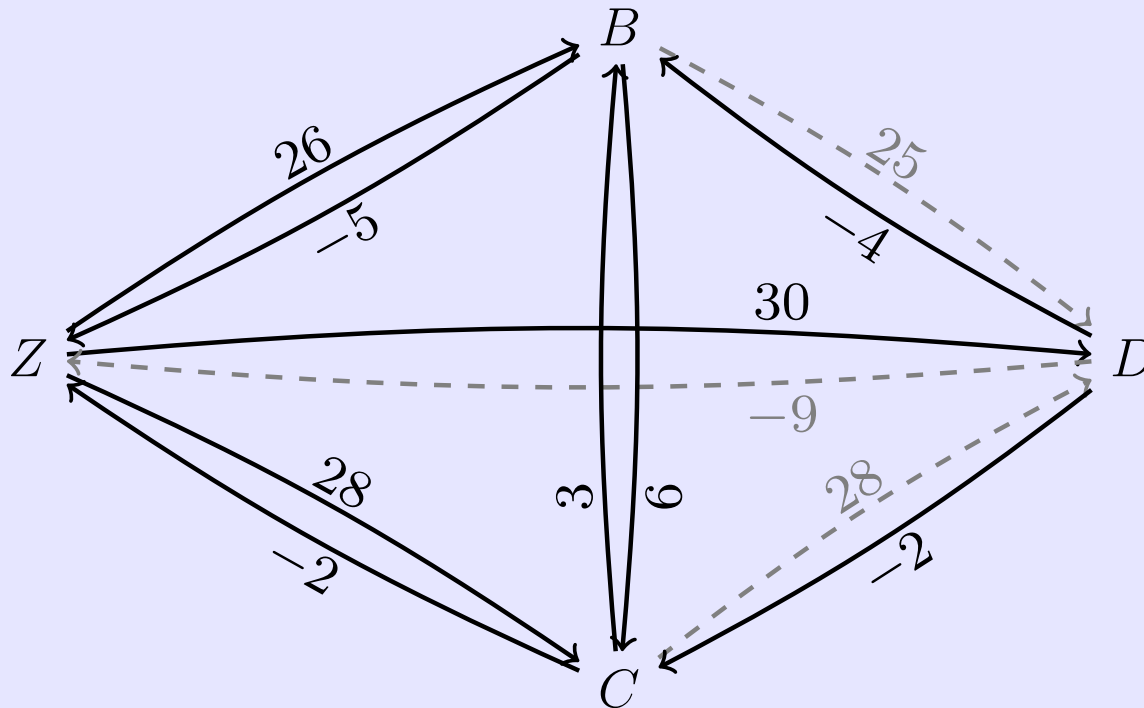


On line

Dynamic Execution

Making STN Dispatchable (ctd.)

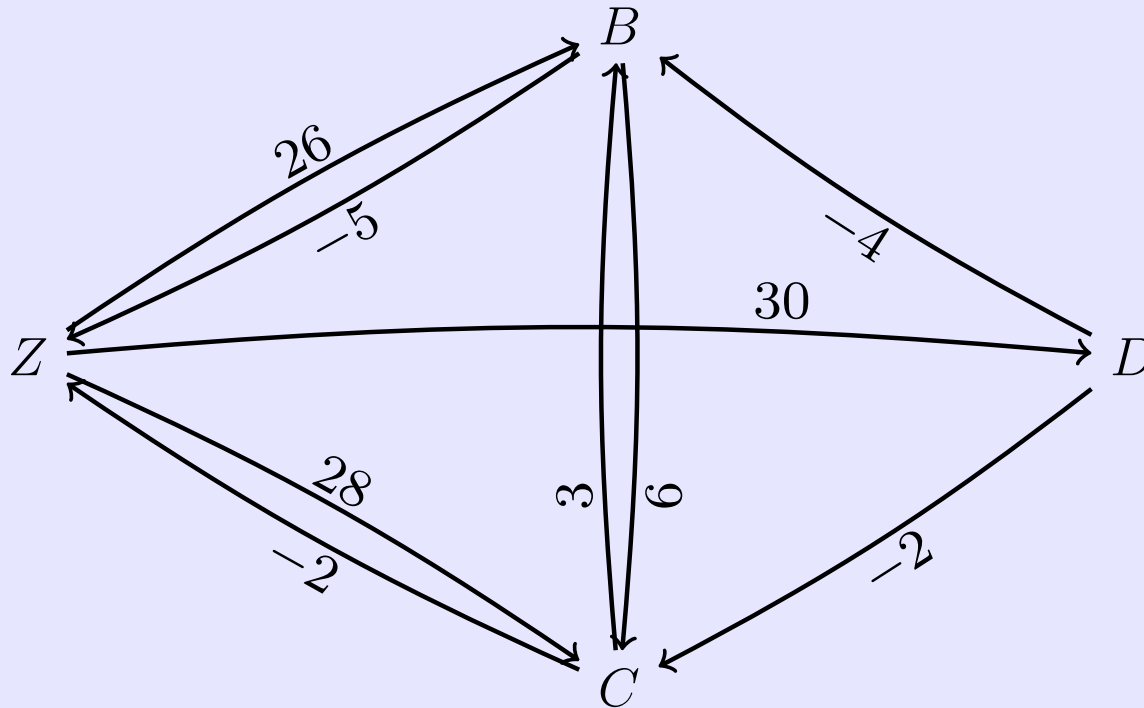
Remove “dominated” edges:*



*(Muscettola, Morris, and Tsamardinos 1998)

Dispatching the STN

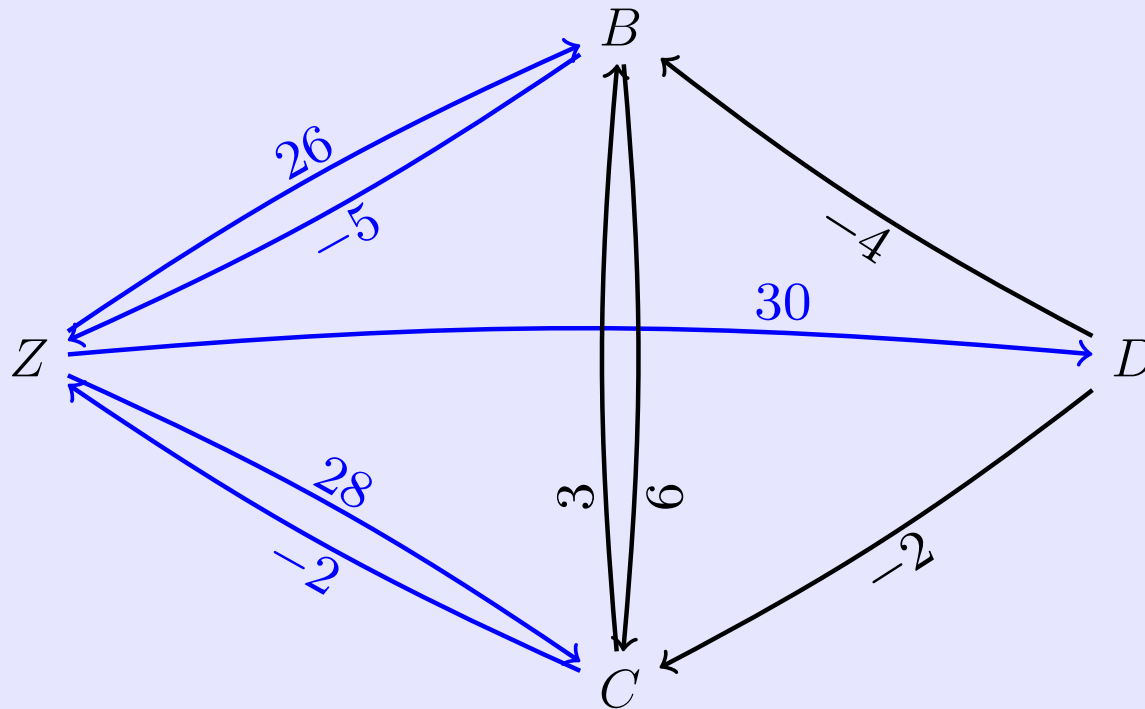
Initially: $t = 0$, $\mathcal{X} = \{\}$, $\mathbf{E} = \{Z\}$.



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

Dispatching the STN (ctd.)

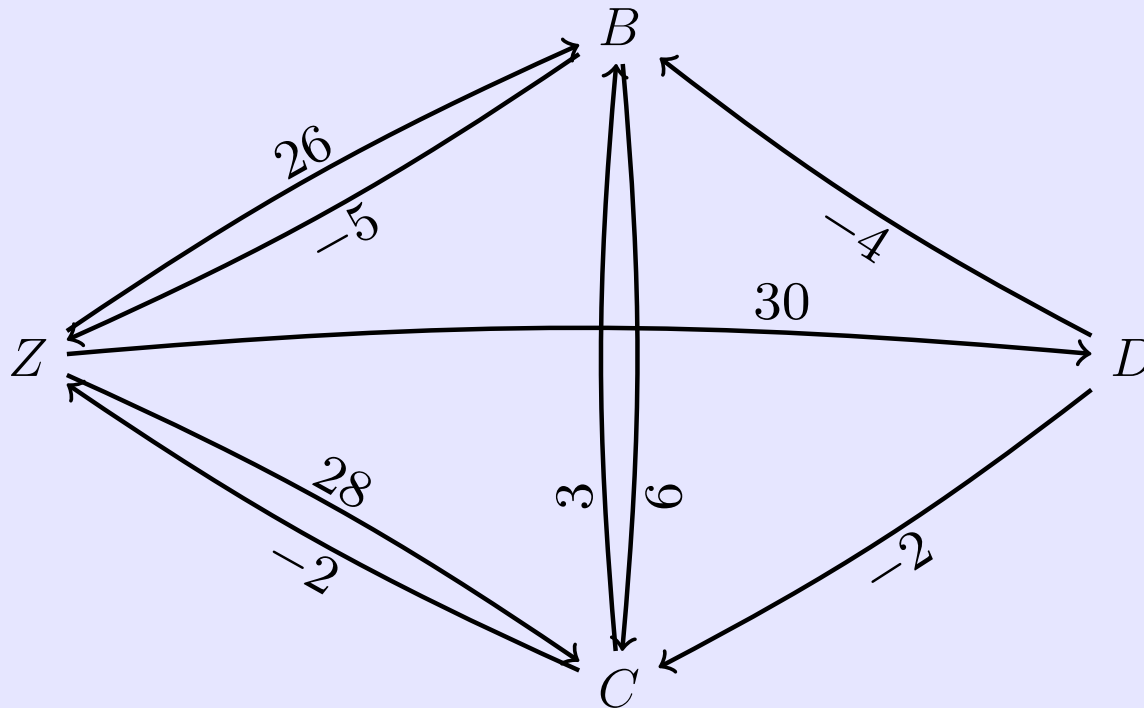
Propagate $Z = 0$ to neighbors;



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

Dispatching the STN (ctd.)

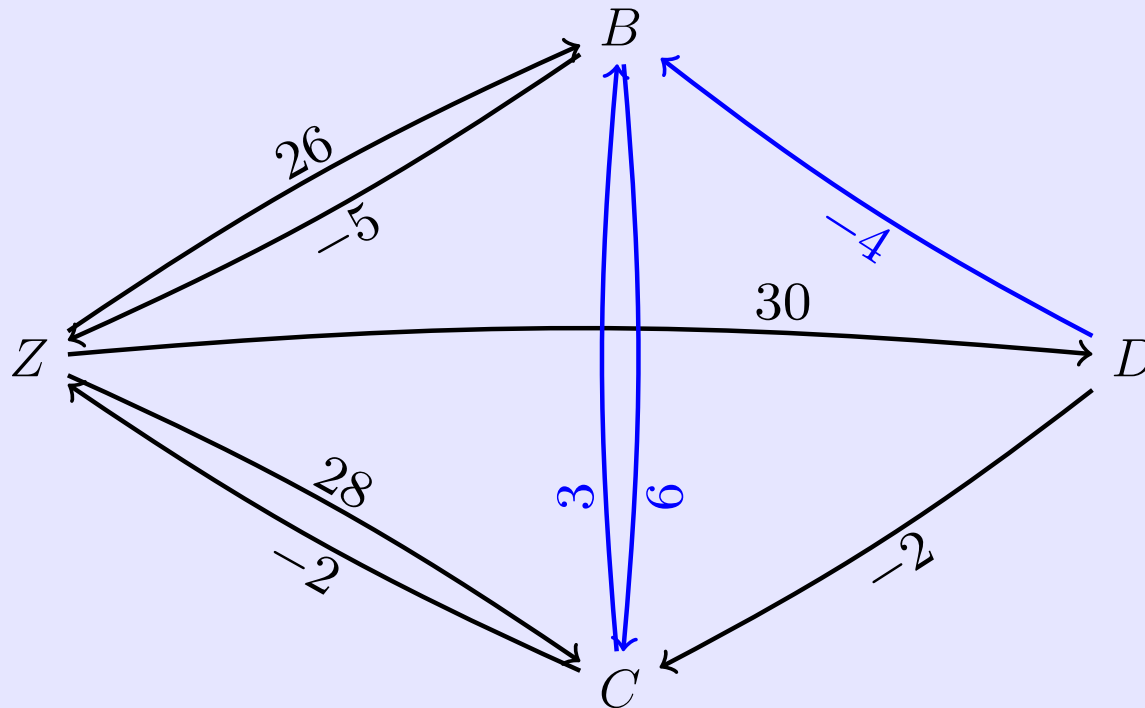
$\mathcal{X} = \{Z\}$, $\mathbf{E} = \{B, C\}$; Bounds: $B \in [5, 26]$, $C \in [2, 28]$.



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

Dispatching the STN (ctd.)

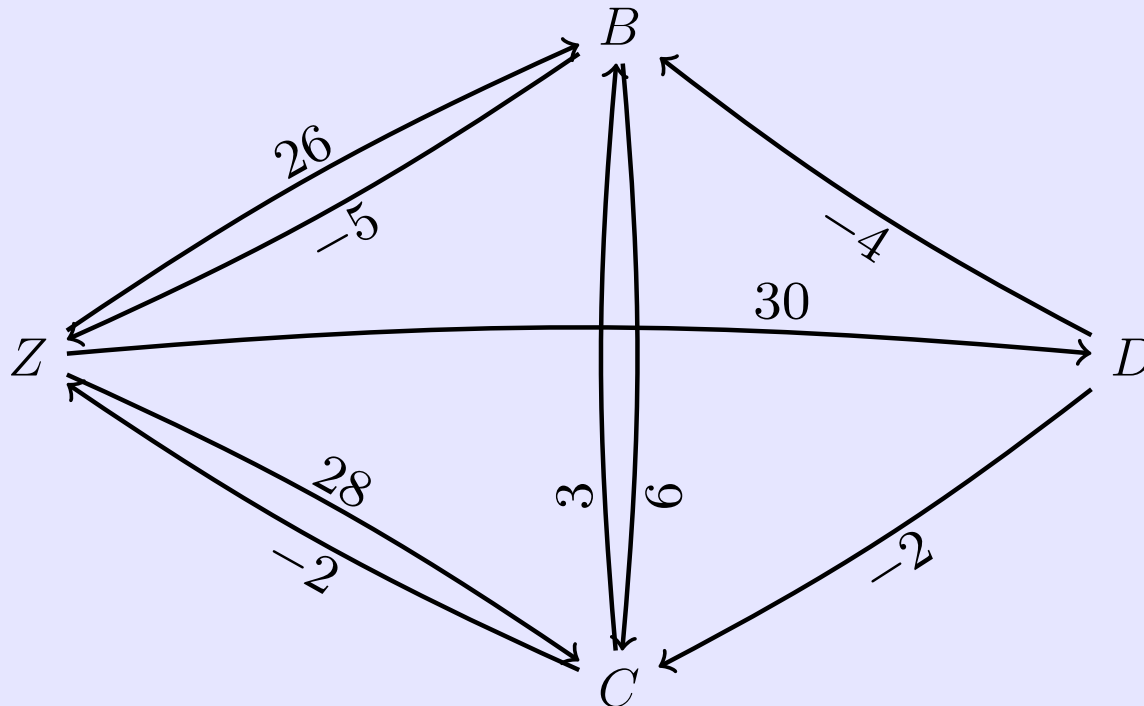
Propagate $B = 12$ to neighbors



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

Dispatching the STN (ctd.)

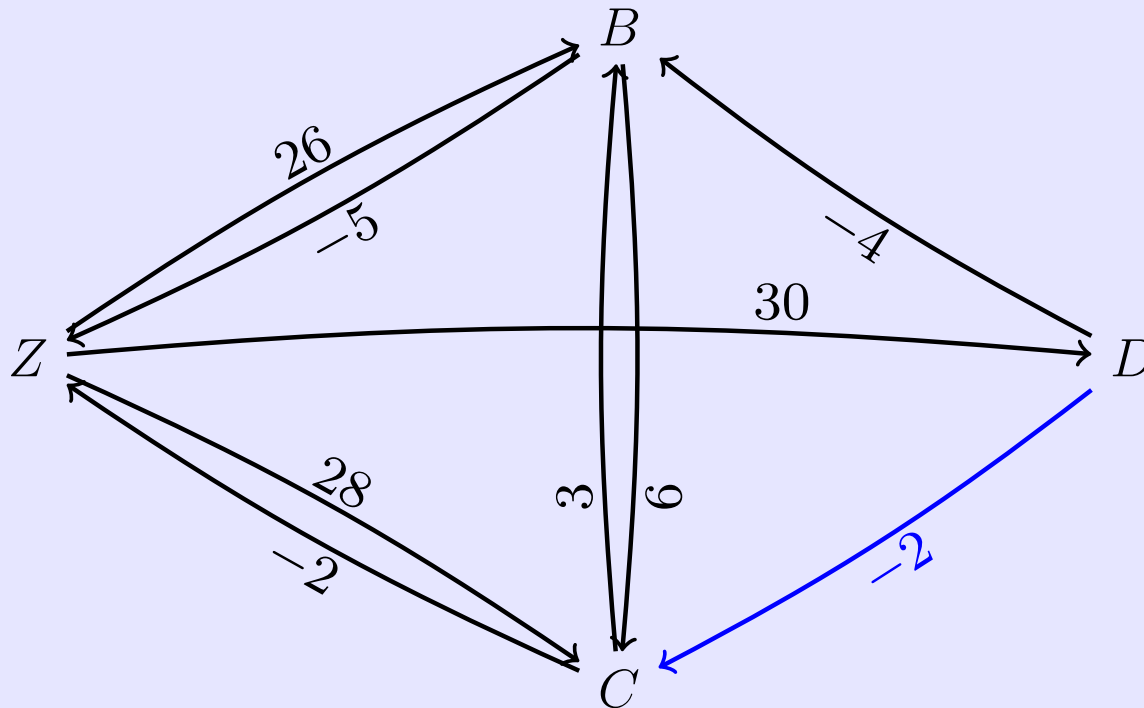
$\mathcal{X} = \{Z, B\}$, $t = 12$, $\mathbf{E} = \{C\}$, $C \in [12, 18]$, $D \in [16, 30]$



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

Dispatching the STN (ctd.)

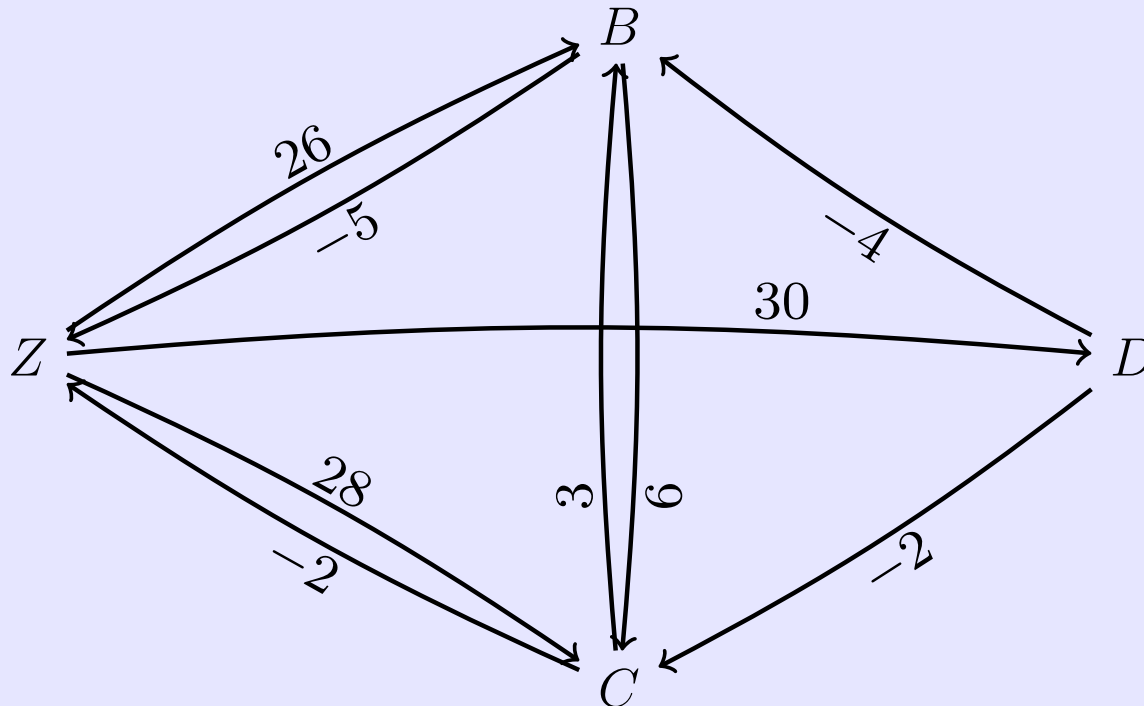
Propagate $C = 16$ to C 's only remaining neighbor, D .



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

Dispatching the STN (ctd.)

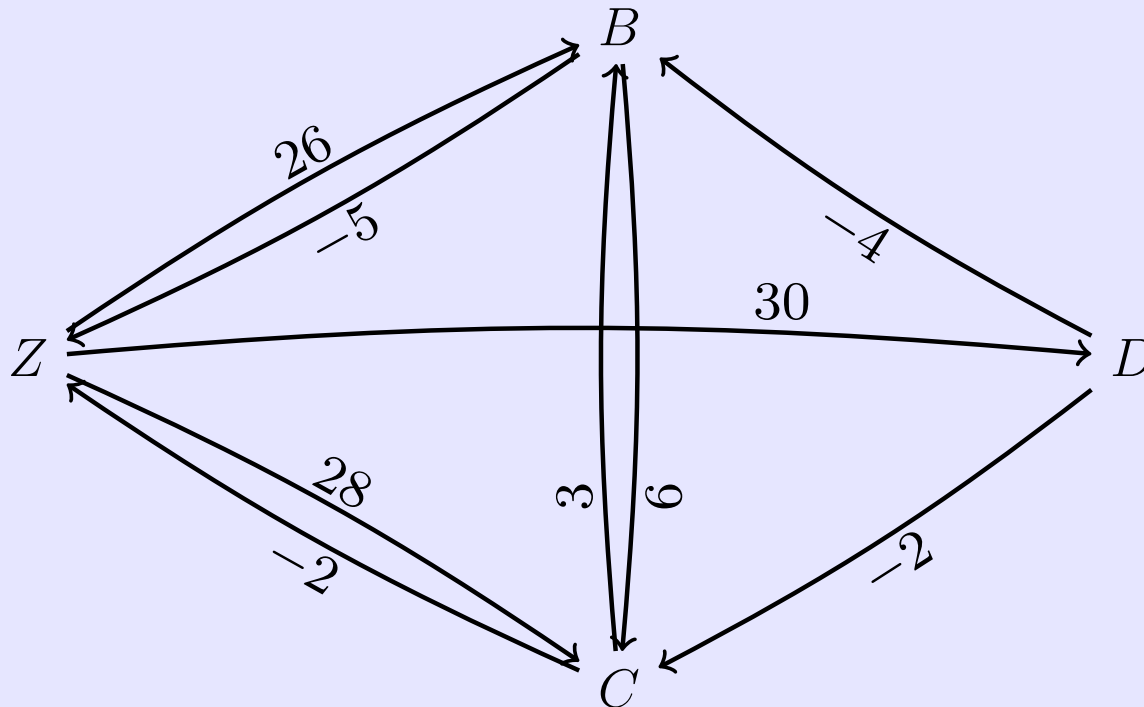
$\mathcal{X} = \{Z, B, C\}$, $t = 16$, $\mathbf{E} = \{D\}$, $D \in [18, 30]$



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

Dispatching the STN (ctd.)

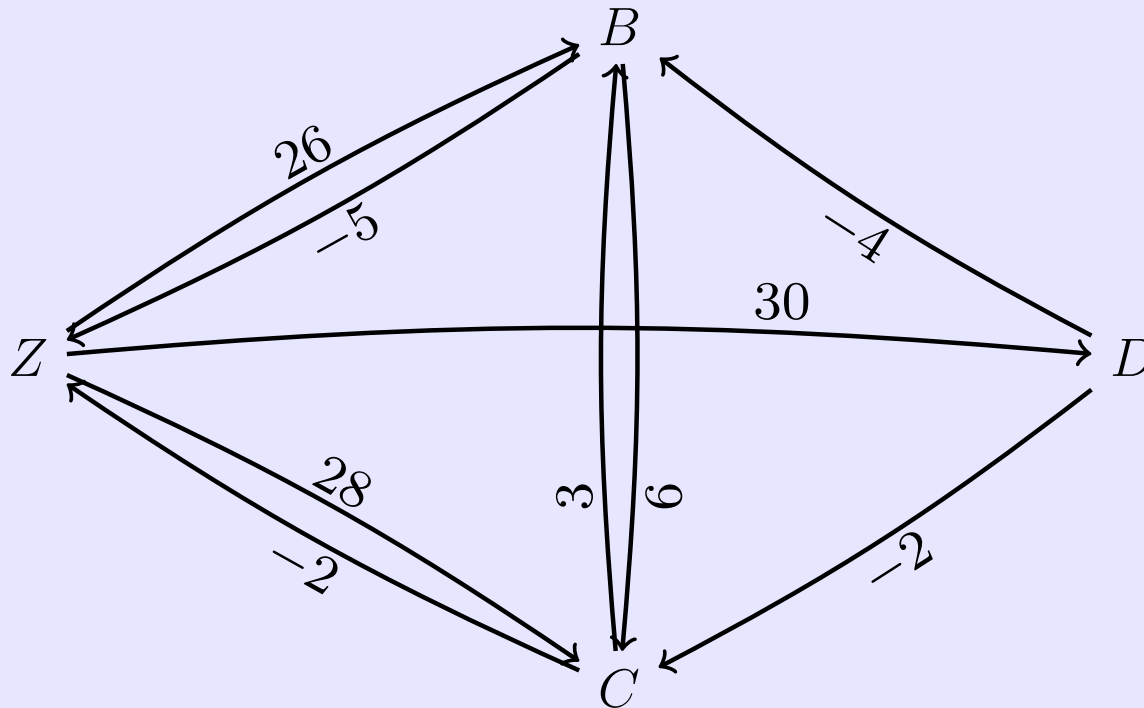
$$\mathcal{X} = \{Z, B, C, D\}, t = 25, \mathbf{E} = \{\}$$



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

Dispatching the STN (ctd.)

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



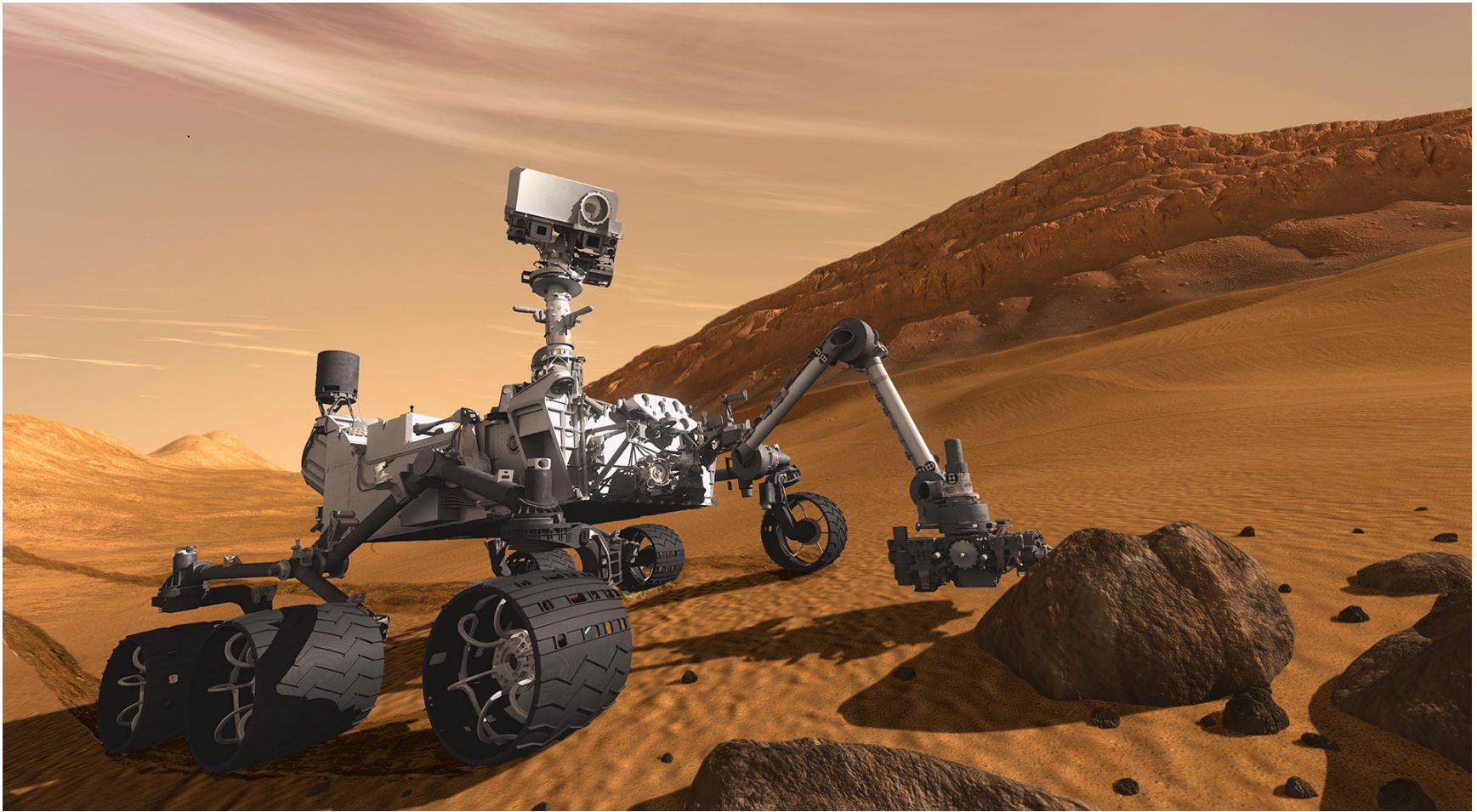
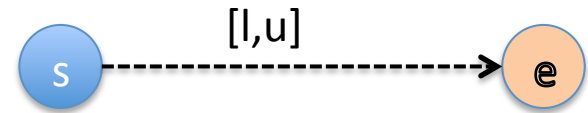


Summary

- 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.
- In order to represent richer families of choices, this basic model has been extended in different ways.
 - Uncertainty
 - Conditions
 - Disjunctions

Temporal Uncertainty: Motivation

- “Drill a hole 3 inches deep”
- Duration of drill action is unknown but within known bounds
- Such actions can be modeled as **contingent links** in network





Temporal Uncertainty

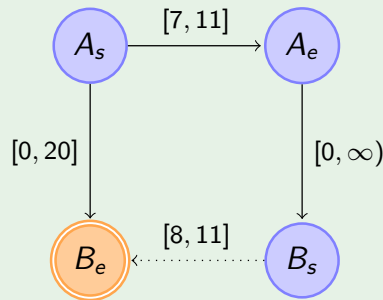
Uncertainty can be seen as a **game** between an *Executor* and the adversarial *Nature*.

Rules

- The *Executor* schedules a set of **Controllable Time Points** (X_c)
- The *Executor* must fulfill a set of temporal constraints called **Free Constraints** (C_f)
- The *Nature* tries to prevent the success of the executor scheduling a set of **Uncontrollable Time Points** (X_u)
- The *Nature* must fulfill a set of temporal constraints called **Contingent Constraints** (C_c)

STNU

Example



A_s, A_e, B_s are **Controllable Time Points** (X_c)

B_e is an **Uncontrollable Time Point** (X_u)

\longrightarrow represents **Free Constraints** (C_f)

$\cdots\rightarrow$ represents **Contingent Constraints** (C_c)

- A Simple Temporal Network with Uncertainty (STNU) [Vidal and Fargier 1999] is an extension of an STN that distinguishes between controllable and uncontrollable events.
- A directed graph, consisting of a set of nodes, representing timepoints, and a set of edges, called links, constraining the duration between the timepoints.
- The links represent two categories of constraints:
 - A free constraint specifies a constraint on the duration between two timepoints.
 - A contingent constraint models an uncontrollable process whose uncertain duration may last any duration between the specified lower and upper bounds.
- All contingent constraints terminate on a timepoint whose timing is controlled exogenously.
- All other timepoints are called requirement timepoints and are controlled by the scheduler.



Flavors of Controllability

Strong Controllability (No observation)

Find a *fixed schedule* for controllable time points that fulfills all the free constraints for every possible assignment to uncontrollable time points fulfilling contingent constraints.

Dynamic Controllability (Past observation)

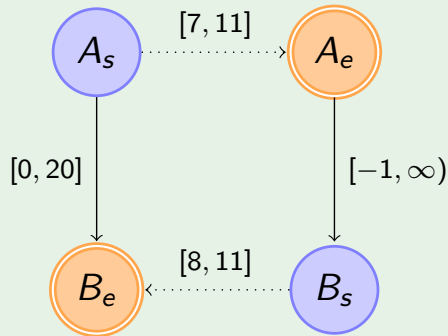
Find a *strategy*, that depends on past observations only, for scheduling controllable time points that fulfills all the free constraints for every possible assignment to uncontrollable time points fulfilling contingent constraints.

Weak Controllability (Full observation)

Find a *strategy* for scheduling controllable time points that fulfills all the free constraints for every possible assignment to uncontrollable time points fulfilling contingent constraints.

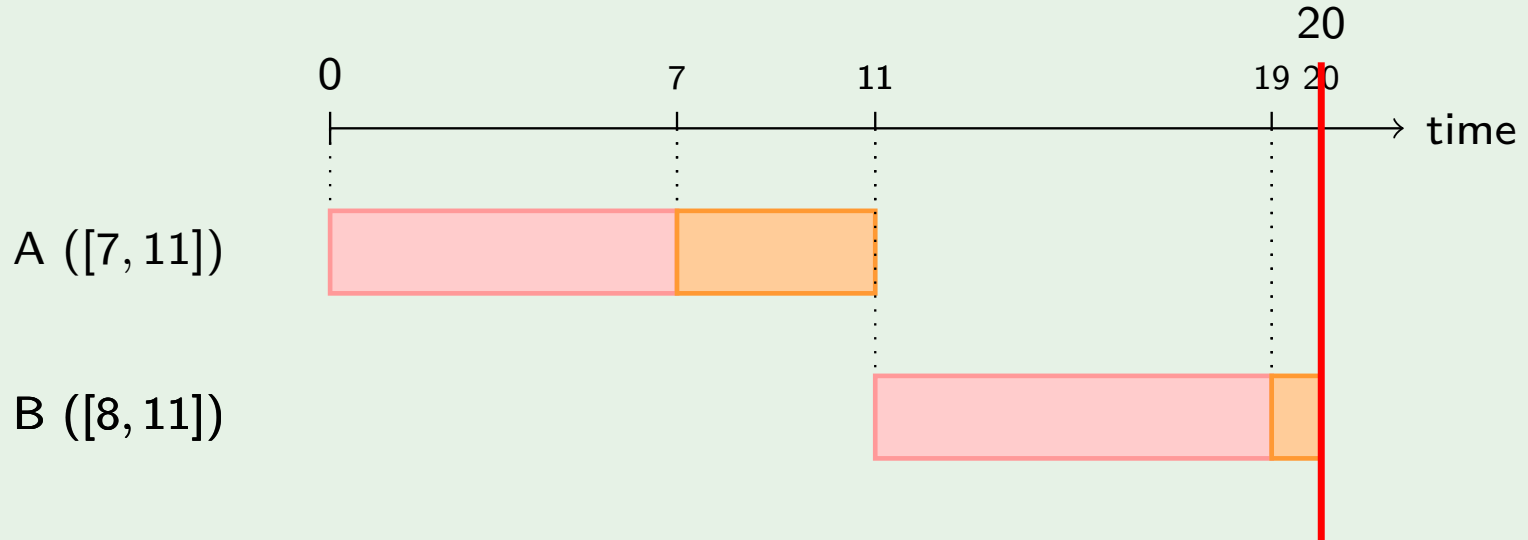
Schedules and Strategies Examples

Example



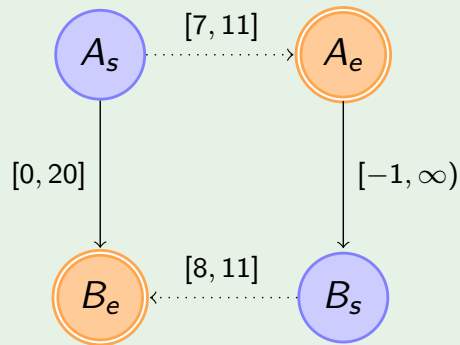
Fixed Schedule (Strong Controllability)

- $start(A)$ at 0
- $start(B)$ at 11



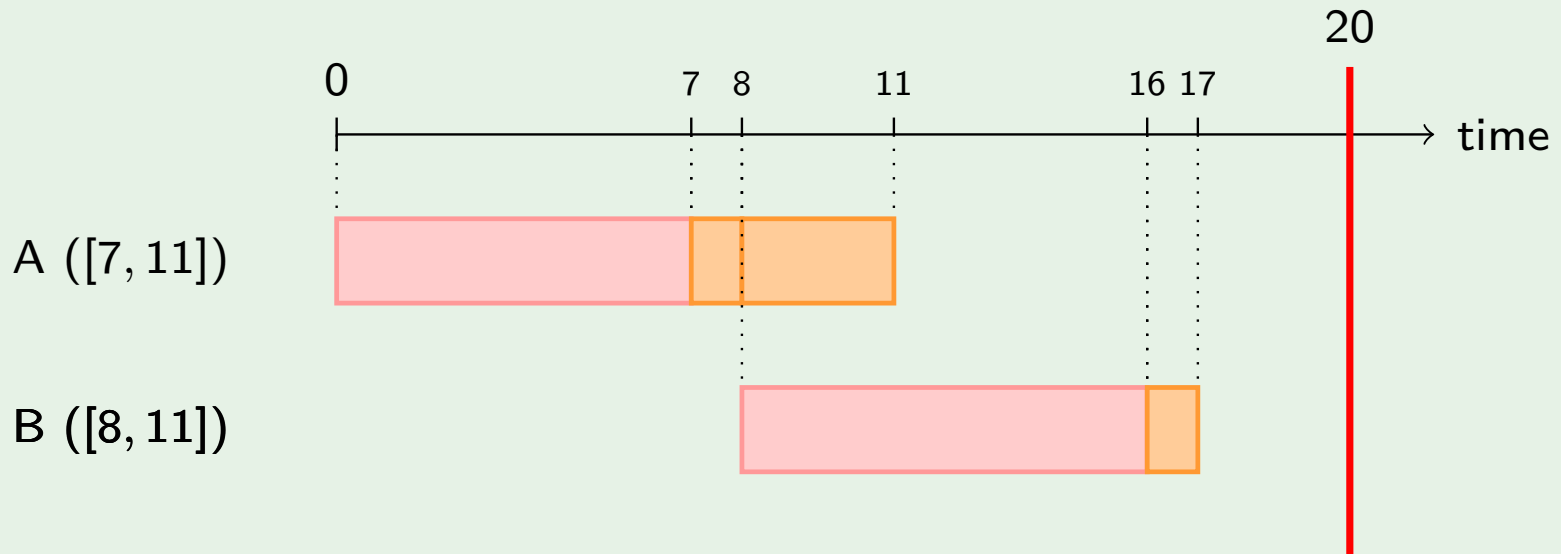
Dynamic Controllability

Example



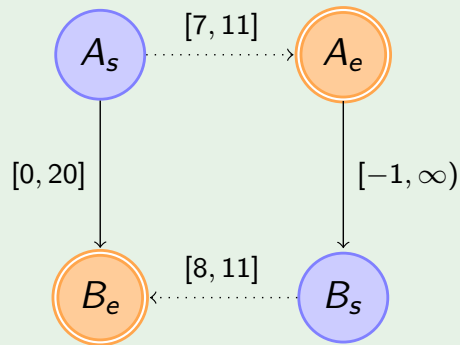
Dynamic Strategy (Dynamic Controllability)

- $start(A)$ at 0
- $start(B)$ at A_e



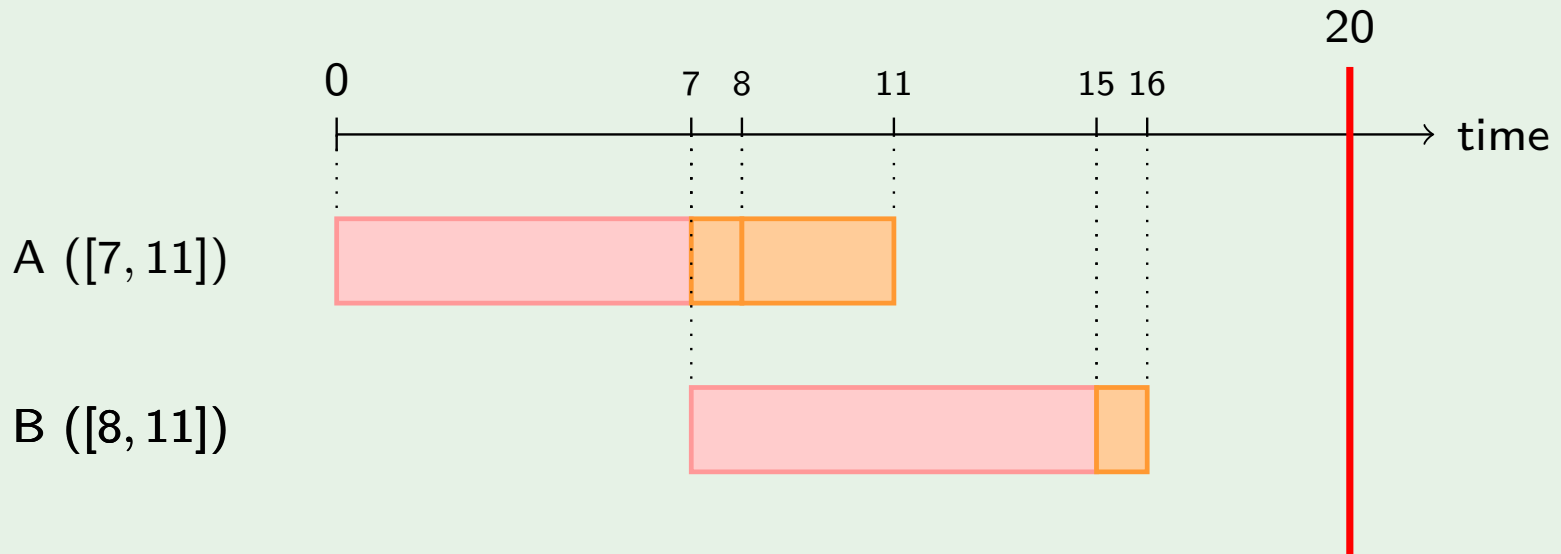
Weak Controllability

Example



Clairvoyant Strategy (Weak Controllability)

- $start(A)$ at 0
- $start(B)$ at $A_e - 1$



STNU Graph

- Nodes and Edges as in an STN graph

$$Y - X \in [3, 7] \quad \iff \quad X \begin{array}{c} \xrightarrow{7} \\ \xleftarrow{-3} \end{array} Y$$

- Contingent Links \iff Labeled Edges*

$$C - A \in [3, 7] \quad \iff \quad A \begin{array}{c} \xrightarrow{c : 3} \\ \xleftarrow{C : -7} \end{array} C$$

Labeled edges represent **uncontrollable possibilities**.

* (Morris and Muscettola 2005)



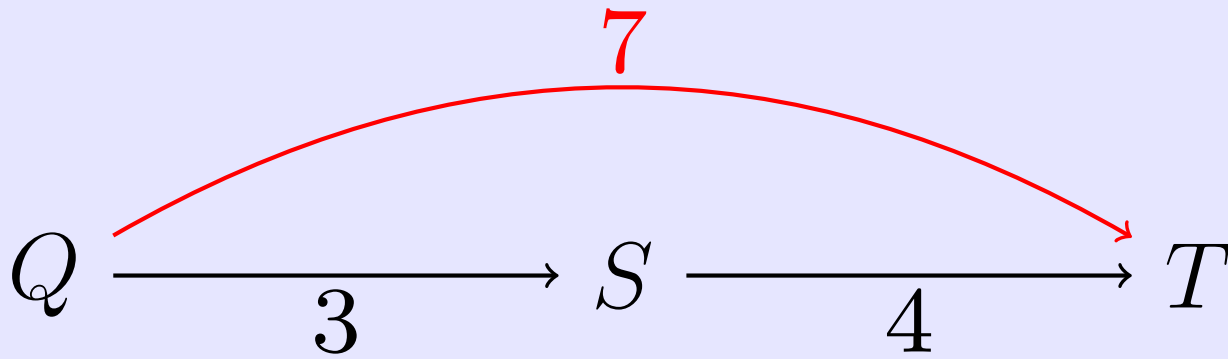
Edge-Generation Rules

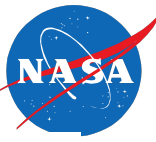
- *No Case* Rule
- *Upper-Case* Rule
- *Lower-Case* Rule
- *Cross-Case* Rule
- *Label-Removal* Rule

(Morris and Muscettola 2005)

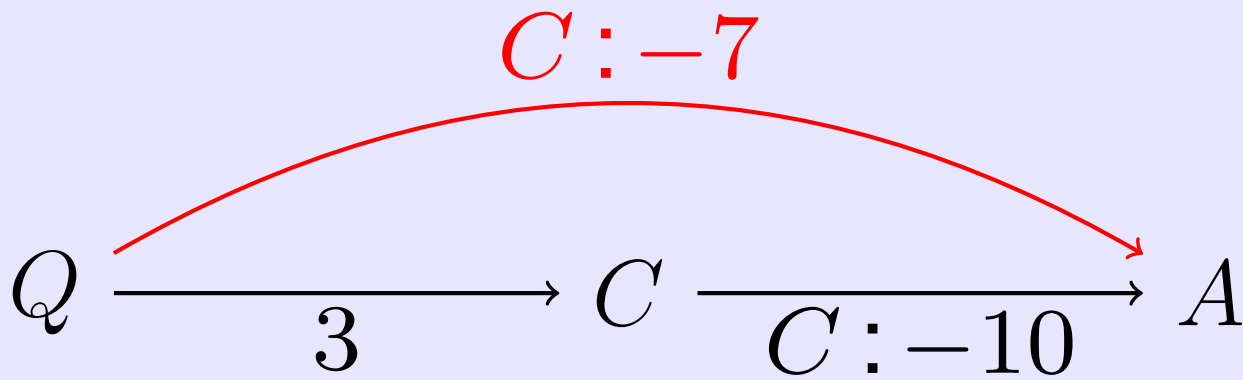


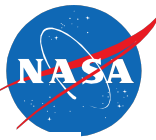
The No-Case Rule



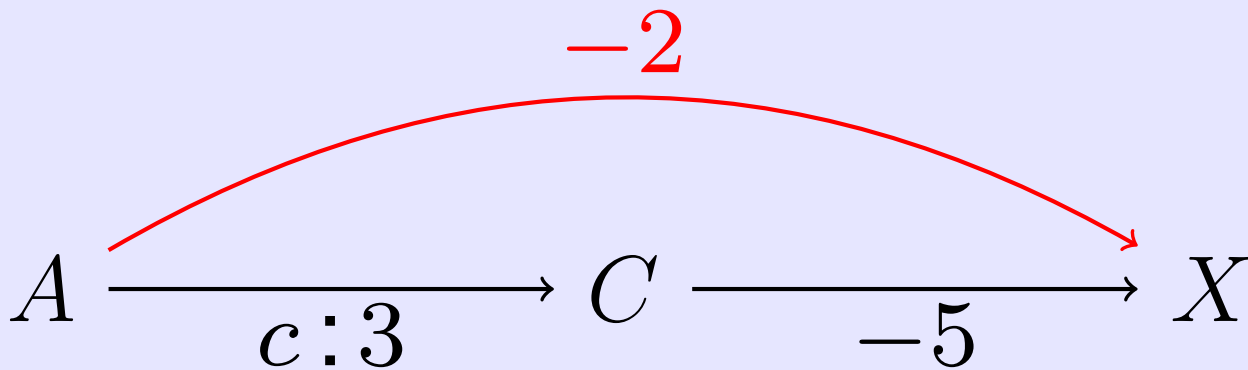


The Upper-Case Rule





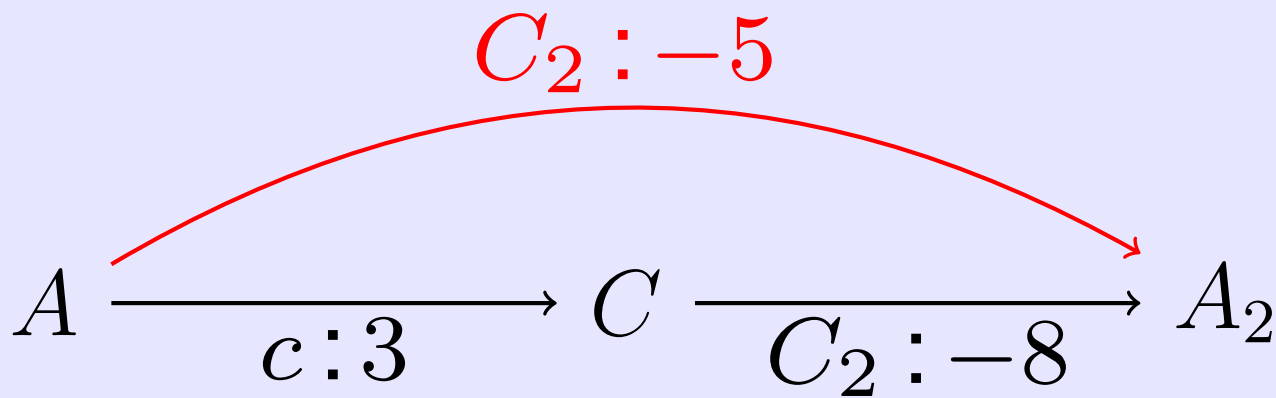
The Lower-Case Rule



(Applies since $-5 \leq 0$)



The Cross-Case Rule



(Applies since $-8 \leq 0$ and $C \neq C_2$)



The Label-Removal Rule

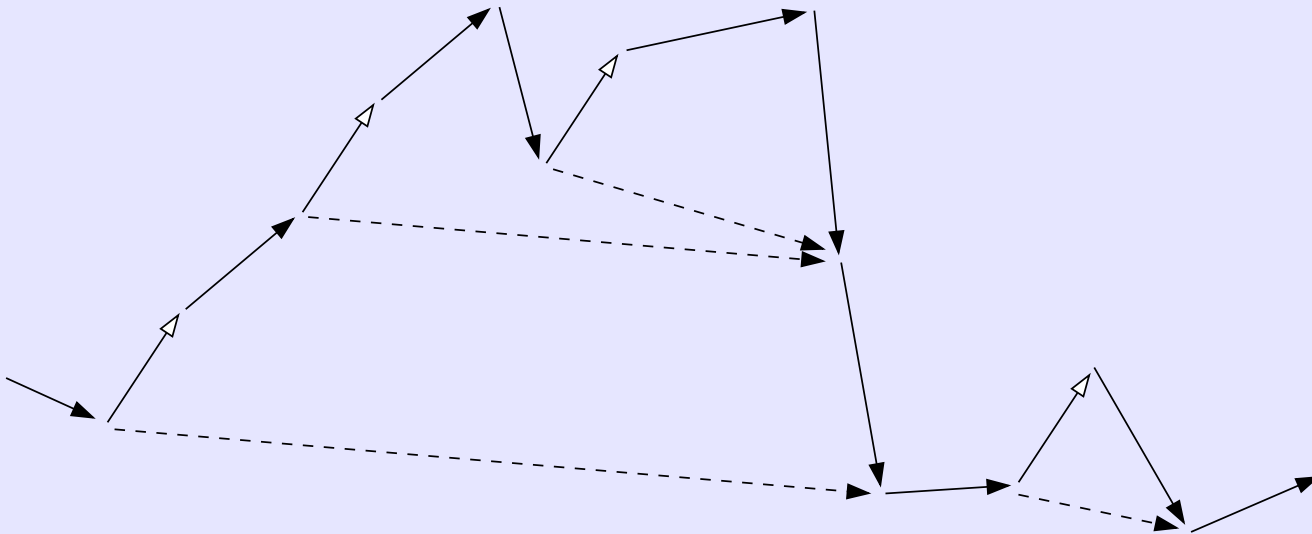
$$X \xrightarrow[C:-1]{} A \xrightarrow{c:3} C$$

The diagram shows a sequence of two transitions. The first transition is from state X to state A , labeled with $C:-1$ below the arrow. A red curved arrow points from X to A with the label -1 above it. The second transition is from state A to state C , labeled with $c:3$ below the arrow.

(Applies since $1 \leq 3$)

Semi-Reducibility

A path is *semi-reducible* if it can be transformed into a path with no *lower-case* edges.





Fundamental Theorem of STNUs

For an STNU \mathcal{S} , with graph \mathcal{G} , and APSSRP matrix \mathcal{D}^* , the following are equivalent:

- \mathcal{S} is dynamically controllable
- \mathcal{G} has no *semi-reducible* negative loops
- \mathcal{D}^* has non-negative values on its main diagonal

(Morris and Muscettola 2005; Morris 2006; Hunsberger 2010; 2013b)



Compiling and Dispatching STNUs

- An STNU must be checked to determine if a dynamically controllable execution strategy exists, and if so compile it into a dispatchable form.
- Once the plan is dispatchable, the dispatcher schedules timepoints through local propagation of timebounds [Muscettola 1998, Morris et al. 2000].
- STNUs can be translated into *labeled distance graphs* and tested for consistency, like STNs.
 - An STNU is consistent only if its associated distance graph contains no negative cycles.
- However, consistency is not sufficient to guarantee dynamic controllability.
- The dynamic controllability (DC) algorithm [Morris et al. 2001] reformulates the distance graph to ensure that each uncontrollable duration, is free to finish any time in the interval $[l_i, u_i]$, as specified by the contingent constraint, C_i .



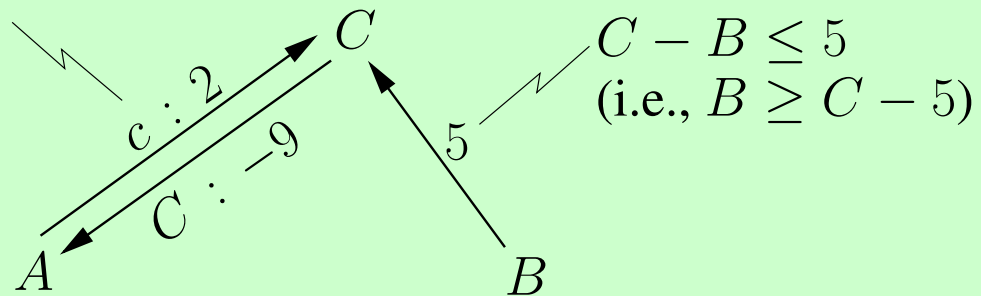
DC Algorithm Steps

1. Using Floyd-Warshall, compute the distance graph to uncover the implicit constraints.
 - If the exposed constraints imply strictly tighter bounds on an uncontrollable duration, then that uncontrollable duration is squeezed [Morris et al. 2001] and the plan is not dynamically controllable.
 - An STNU is pseudo-controllable [Morris et al. 2001] if it is both temporally consistent and none of its uncontrollable durations are squeezed.
 - If the scheduler makes the wrong decisions during plan execution, then uncontrollable durations make be squeezed then.
2. To avoid squeezing uncontrollable durations during scheduling, the DC algorithm adds constraints to the plan.
 - The constraints take the form of simple temporal constraints and conditional constraints (or “wait” constraints) and are applied according to the precede, unordered, and unconditional unordered reduction rules applied to triangular graphs.
3. The conditional constraints can be regressed to deduce new conditional constraints.

STNU Example

Contingent Link: $(A, 2, 9, C)$

$$C - A \in [2, 9]$$

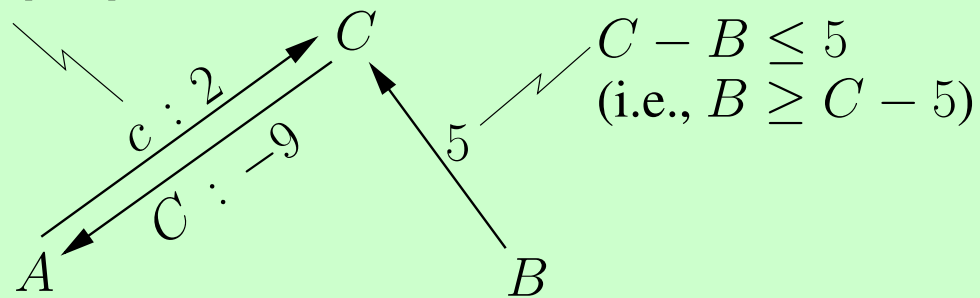


If $A = 0$, when is it safe to execute B ?

STNU Example

Contingent Link: $(A, 2, 9, C)$

$$C - A \in [2, 9]$$

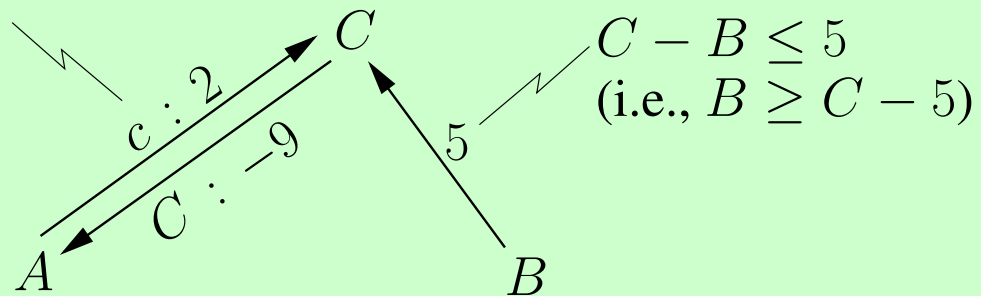


If $A = 0$ and $B = 2$, then problem if $C > 7$.

STNU Example

Contingent Link: $(A, 2, 9, C)$

$$C - A \in [2, 9]$$

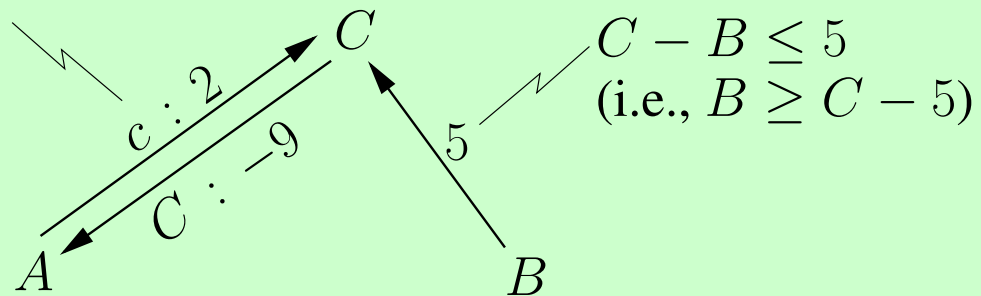


If $A = 0$ and $B \geq 4$, then no problems!

STNU Example

Contingent Link: $(A, 2, 9, C)$

$$C - A \in [2, 9]$$

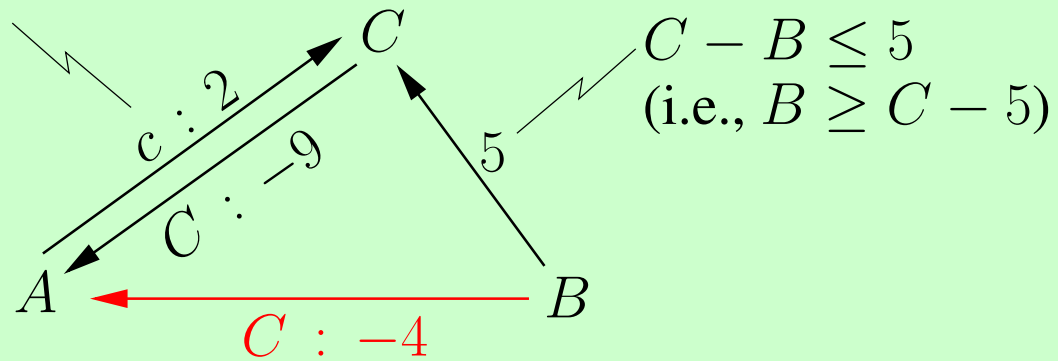


If $A = 0$ and $C = 3$, then $B > 3$ no problem!

STNU Example

Contingent Link: $(A, 2, 9, C)$

$$C - A \in [2, 9]$$



Strategy: As long as C unexecuted,
 B must wait at least 4 after A .



Dynamic Controllability (DC)

An STNU is *dynamically controllable* (DC) if:

there exists a *dynamic strategy* ...

for executing the *non-contingent* time-points ...

such that *all* of the constraints will be satisfied ...

no matter how the contingent durations turn out.

⇒ A dynamic strategy can *react* to contingent executions.



Real-Time Execution Decisions*

The semantics for dynamic controllability can be stated in terms of *Real-Time Execution Decisions* (RTEDs):

- **WAIT:**
Wait for some activated contingent link to complete.
- (t, χ) :
If nothing happens before time $t \in \mathbb{R}$, then execute the (non-contingent) time-points in χ at time t .

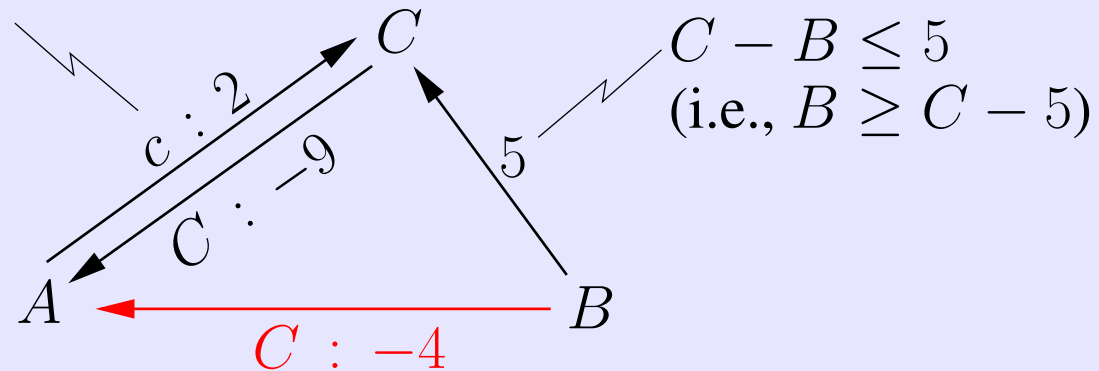
* (Hunsberger 2009)



RTED Example

Contingent Link: $(A, 2, 9, C)$

$$C - A \in [2, 9]$$



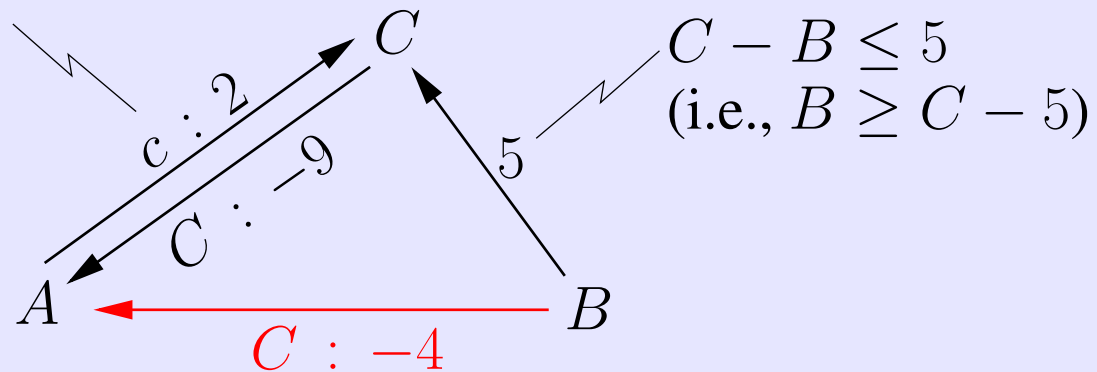
Initial Decision: $(4, \{B\})$

(If nothing happens before time 4, execute B at 4.)

RTED Example (ctd.)

Contingent Link: $(A, 2, 9, C)$

$$C - A \in [2, 9]$$



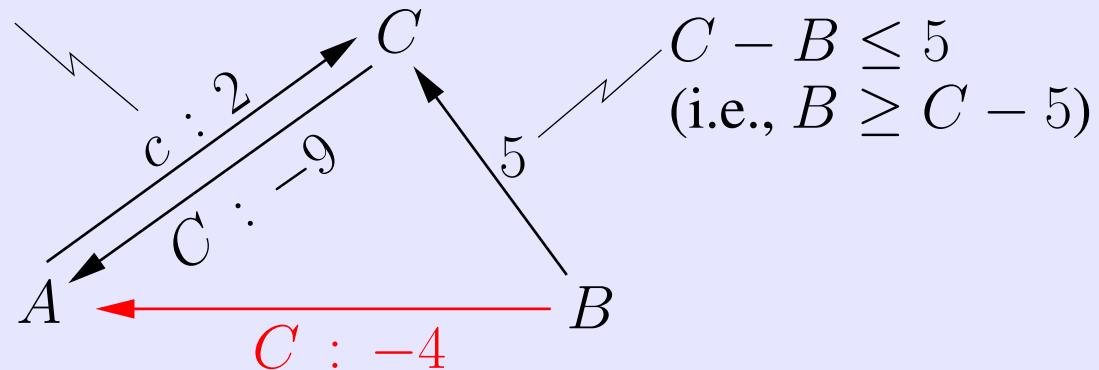
Possible Outcome: C executes at time 2.

Next decision: $(3, \{B\})$

RTED Example (ctd.)

Contingent Link: $(A, 2, 9, C)$

$$C - A \in [2, 9]$$



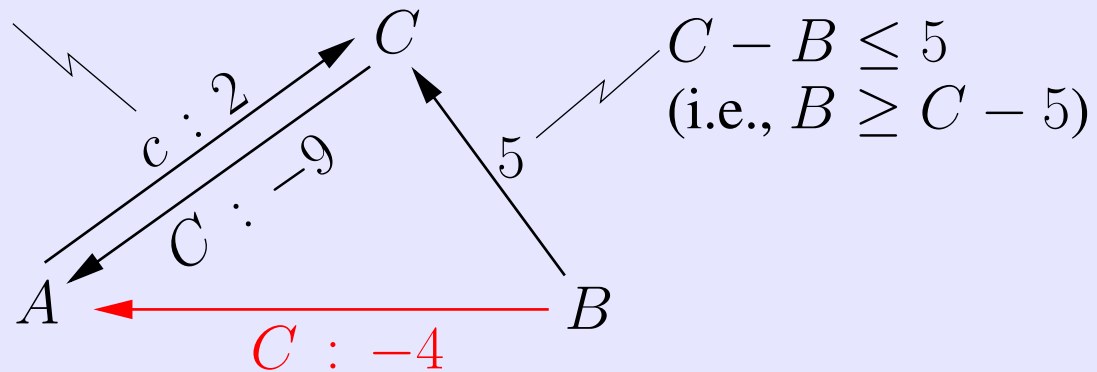
Initial Decision: $(4, \{B\})$

(If nothing happens before time 4, execute B at 4.)

RTED Example (ctd.)

Contingent Link: $(A, 2, 9, C)$

$$C - A \in [2, 9]$$



Possible Outcome: C does not execute yet;
so B is executed at 4

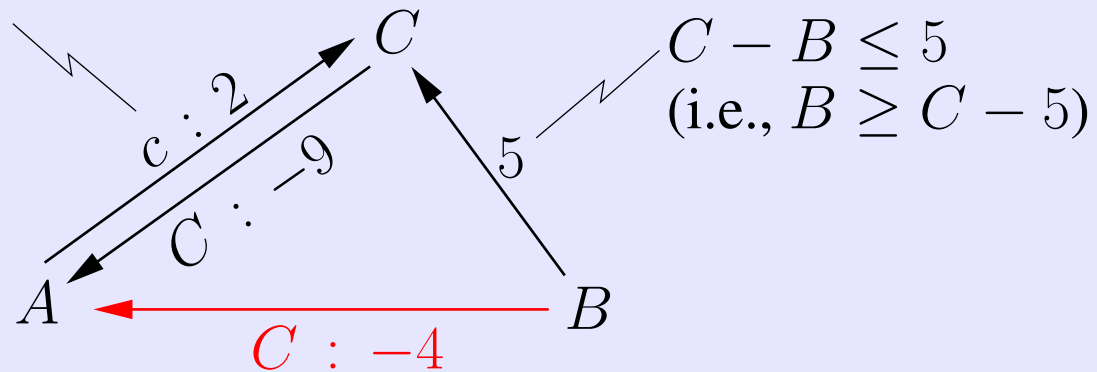
Next decision: WAIT (for C to execute)



RTED Example (ctd.)

Contingent Link: $(A, 2, 9, C)$

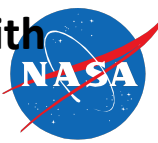
$$C - A \in [2, 9]$$



Possible Outcome: C does not execute yet;
so B is executed at 4

Next decision: WAIT (for C to execute)

Patrick Conrad and Brian Williams. "Drake: An Efficient Executive for Temporal Plans with Choice." Journal of Artificial Intelligence Research 42 (2011) 607-659



David Kortenkamp and Reid Simmons, "Robotic Systems Architectures and Programming". Chapter 8 of Robotics Systems Handbook.

Tarek Chaari, Sondes Chaabane, Nassima Aissani and Damien Trentesaux. "Scheduling under uncertainty: survey and research directions." International Conference on Advanced Logistics and Transport 2014.

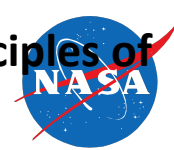
Andrew J. Davenport and Christophe Gefflot and J. Christopher Beck. "Slack-Based Techniques for Robust Schedules" Proceedings of the Sixth European Conference on Planning 2014

Tsamardinos, I., Muscettola, N., Morris, P. 1998 "Fast transformation of temporal plans for efficient execution." Proc. AAAI-98.

Laura Hiatt and Reid Simmons. "Pre-positioning Assets to Increase Execution Efficiency" ICRA 2007.

Julie Shah, John Stedl, Brian Williams, and Paul Robertson. "A Fast Incremental Algorithm for Maintaining Dispatchability of Partially Controllable Plans Executing Reactive, Model-based Programs Through Graph-based Temporal Planning" IJCAI 2001.

Tsamardinos, I. "Flexible dispatch of disjunctive plans.) Proceedings of the 6th European Conference on Planning 2001

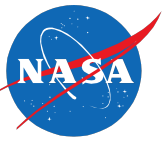


Muscettola, Morris et al., “Reformulating Temporal Plans For Efficient Execution” Principles of Knowledge Representation and Reasoning, 1998

Ono, et. al, “Probabilistic Planning for Continuous Dynamic Systems under Bounded Risk.” JAIR 2013.

Jing Cui “Models of Robustness for Temporal Planning and Scheduling with Dynamic Controllability”, 2015.

Daria Terekov, et. al. “Integrating Queueing Theory and Scheduling for Dynamic Scheduling Problems” Journal of Artificial Intelligence Research 50 (2014) 535-572



Backup



Planning under Uncertainty

- Solution to a Planning problem: a sequence of actions that transforms an initial state to one that realizes a set of goals, possibly optimizing a cost function along the way.
 - Uncertainty forces us to reconsider this definition.