Massachusetts Institute of Technology

Robust Execution of Temporal Plans

Slide Contributions:
Andreas Hofmann
Julie Shah

Prof Brian Williams
16.410 / 16.413
October.6th, 2010

Model-based Embedded & Robotic Systems

CSAIL

Assignments

- Remember:
  Problem Set #5 due today, Wed, Oct. 6th, 2010.
  Problem Set #6 out today.
- Reading:
  - Today: Dechter, R., I. Meiri, J. Pearl, "Temporal Constraint Networks," Artificial Intelligence, 49, pp. 61-95,1991.
  - Wednesday: Logic *[AIMA] Ch. 7, 8*
- Exam:
  - Mid-Term - October 20th.

Executing Time Critical Missions

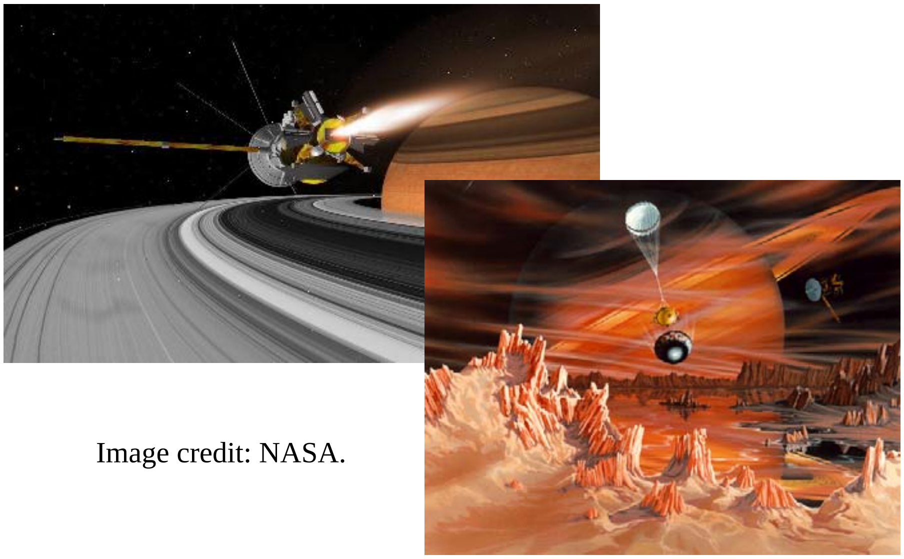


Image credit: NASA.

Team Coordination under Time Pressure

Images of scrub nurses and surgeons removed due to copyright restrictions.

An effective Scrub Nurse:

- works hand-to-hand, face-to-face with surgeon,
- assesses and anticipates needs of surgeon,
- provides assistance and tools in order of need,
- responds quickly to changing circumstances,
- responds quickly to surgeon's cues and requests.

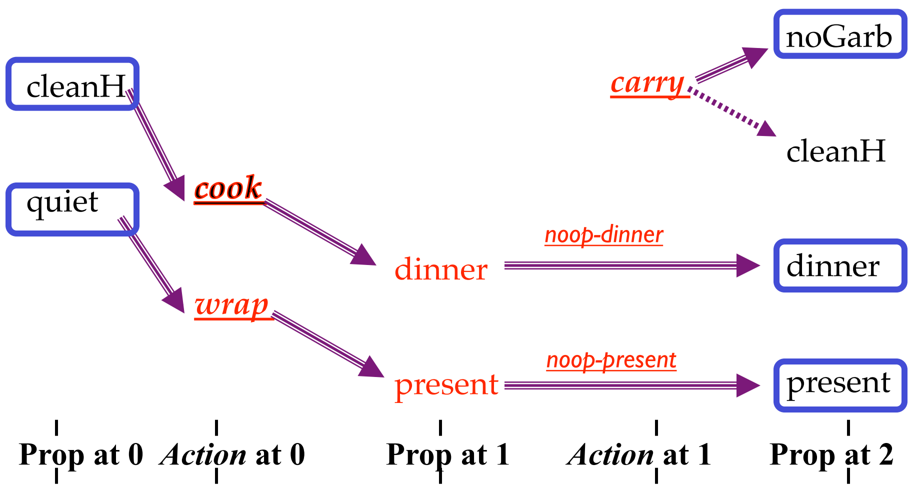
Human-Robot Teaming

Images of human-robot teaming (in surgical, space, and rescue settings) removed due to copyright restrictions.

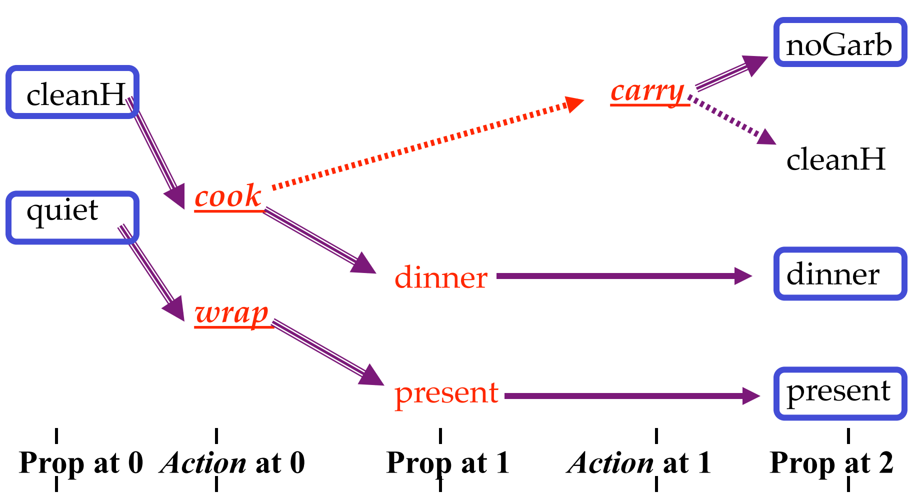
Robust Execution of Time-critical Tasks

- Executing Simple Plans
- Robust Execution
  - Describing Temporal Plans
  - Checking Temporal Plan Consistency
  - Scheduling Plans
  - Robust, Dynamic Scheduling

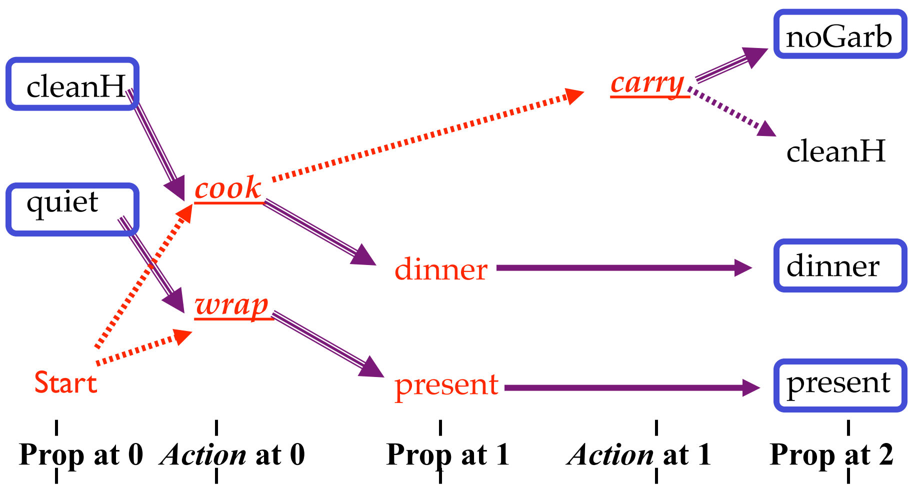
Total Order Plans <Actions[i] > Contain Irrelevant Commitments



Partial Order Plans <Actions, Orderings, Links> Remove Irrelevant Commitments



Partial Order Plan Execution



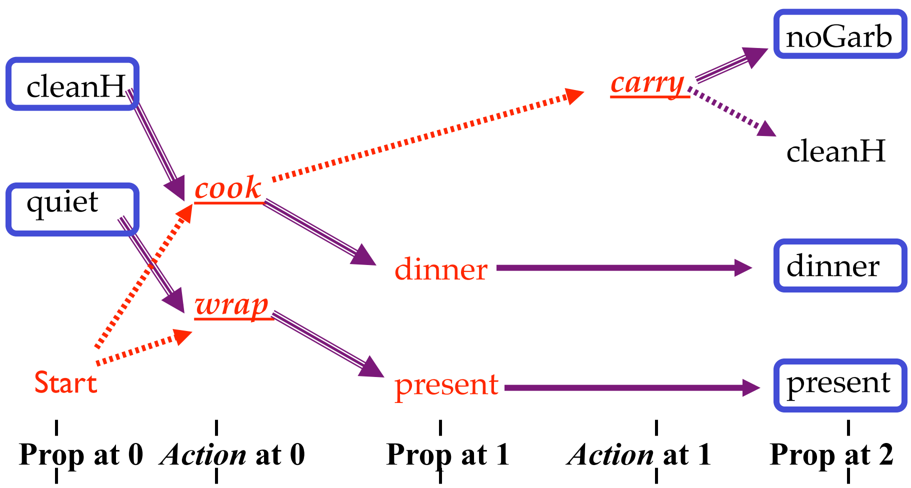
Partial Order Plan Execution

Initialize queue Ready, with action Start..
Mark all actions as "not executed."

Loop

- If Ready is empty, Then terminate.
- Dequeue action a from Ready and execute.
- When completed, mark a as executed.
- For each succeeding action b such that a < b or linked(a,b,p),
  - If every preceding action c is marked "executed," such that c < b or linked(c,b,p'),
  - Then queue b on Ready.

Plan Execution w Action Monitoring



Plan Execution w Action Monitoring

Initialize queue Ready, with action Start.
Mark all actions as "not executed."

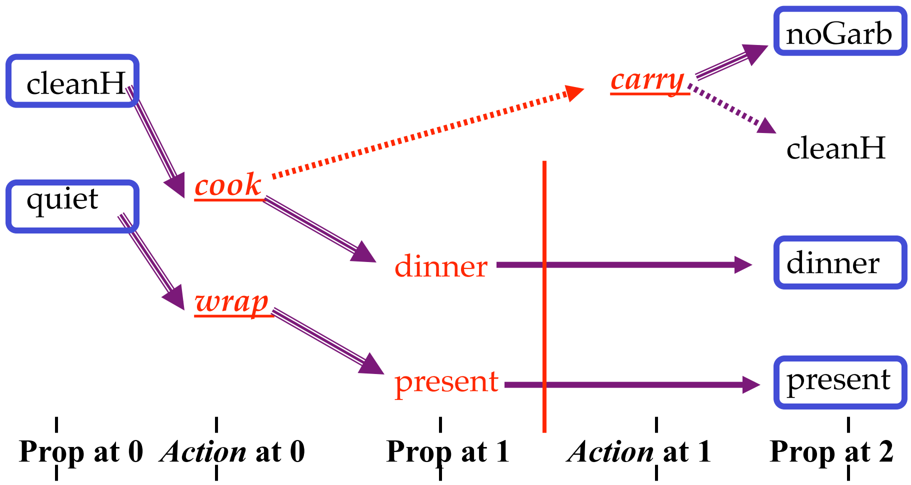
Loop

- If Ready is empty, Then terminate.
- Dequeue action a from Ready.
- If a's preconditions satisfied, then execute, else fail.
- When completed, mark a as executed.
- For each succeeding action b such that a < b or linked(a,b,p),
  - If every preceding action c is marked "executed," such that c < b or linked(c,b,p'),
  - Then queue b on Ready.

Execution Monitoring

- Check if any preconditions of unexecuted actions are violated.

⇒ Check if a causal link that crosses the current time is violated.



Plan Execution w Execution Monitoring

Initialize agenda Ready with action Start

Initialize agenda ActiveLinks to empty

Mark all actions as "not executed."

Loop

- If Ready is empty then terminate.
- For each link on ActiveLinks
  - If the proposition for link doesn't hold,
    Then return failure
- Dequeue action a from Ready
- If preconditions of action are satisfied
  - Then execute
  - Else return failure
- . . . (continued on next slide)

Plan Execution w Execution Monitoring (cont)

Loop

- … (continued from previous slide)
- Mark a as “executed.”
- For each action c such that linked(c,a,p).
  - dequeue <c,a,p> from ActiveLinks.
- For each action d such that linked(a,d,p).
  - queue <a,d,p> on ActiveLinks.
- For each action b such that a < b or linked(a,b,p).
  - If every action c has been executed, such that c < b or linked(c,b,p’)
  - Then queue b on Ready.

Robust Execution of Time-critical Tasks

Model-based Embedded & Robotic Systems

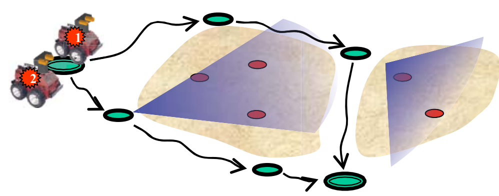
- Executing Simple Plans
- Robust Execution

Massachusetts Institute of Technology

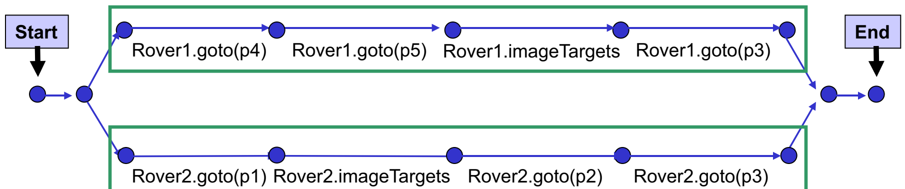
16

Executing Timed Programs and Plans Robustly

```
imageScienceTargets(Rover1, Rover2)
{Parallel
  {Sequence
     [5,10] Rover1.goto(p4);
     [5,10] Rover1.goto(p5);
     [2,5] Rover1.imageTargets();
     [5,10] Rover1.goto(p3);
  },
  {Sequence
     [5,10] Rover2.goto(p1);
     [5,10]Rover2.imageTargets();
     [2,5] Rover2.goto(p2);
     [5,10] Rover2.goto(p3);
  }
}
```



in RMPL [williams et al]



Agents adapt to temporal disturbances in a coordinated manner by scheduling the start of activities on the fly.

In general, categorize durations into controllable and uncontrollable (STNUs).

To Execute a Temporal Plan

Part I : Scheduling Off-line

1. Describe Temporal Plan
2. Test Consistency
3. Schedule Plan

offline

online

4. Execute Plan

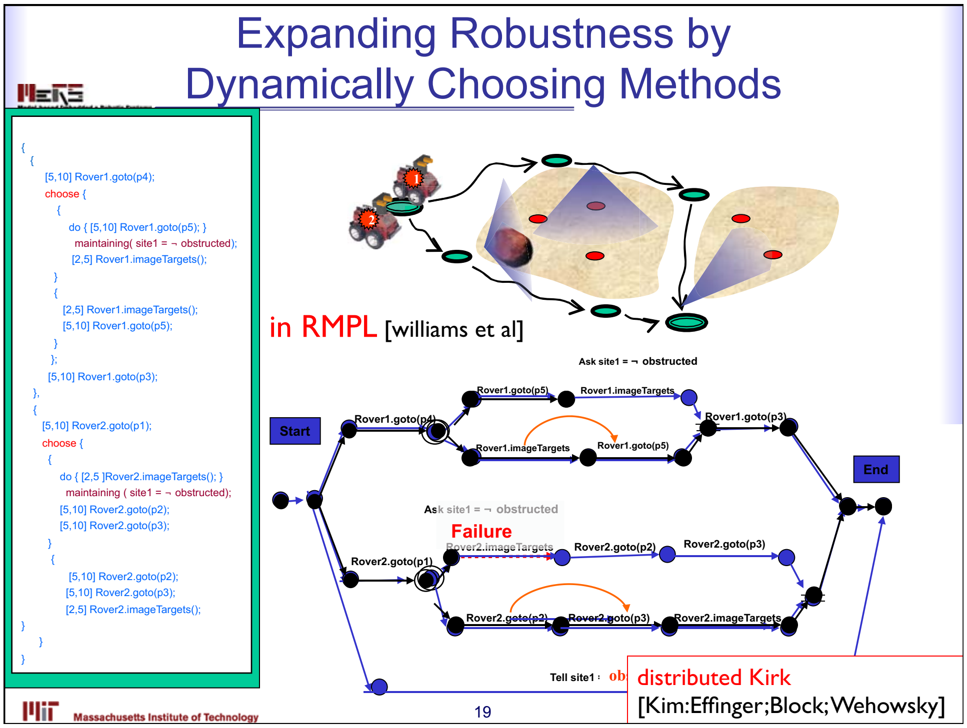
Part II: Scheduling Online

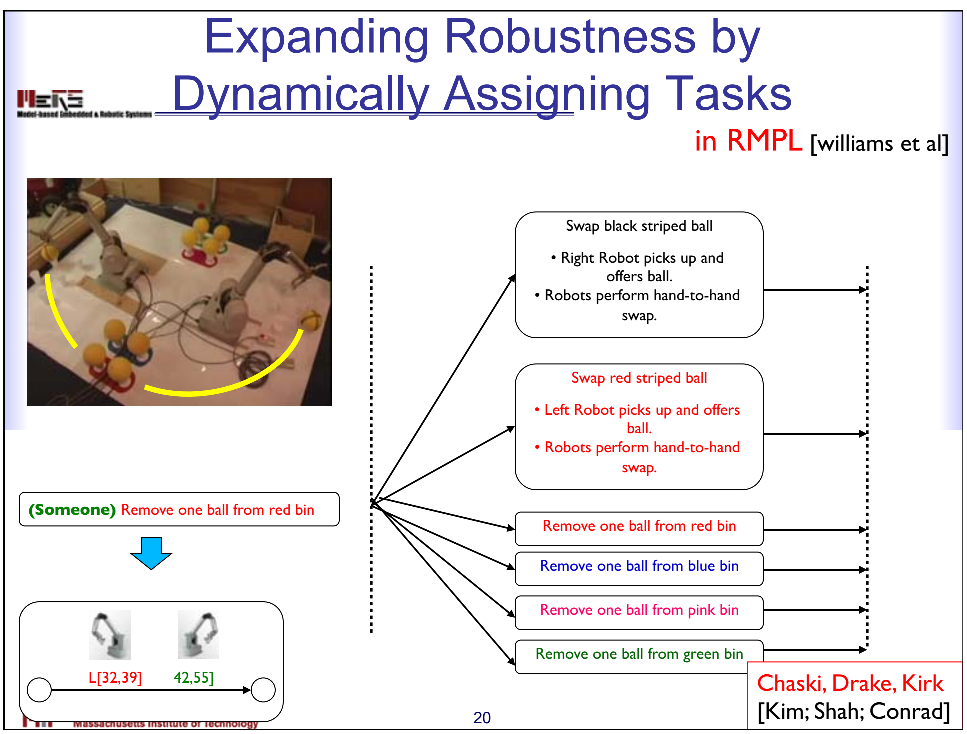
1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan

offline

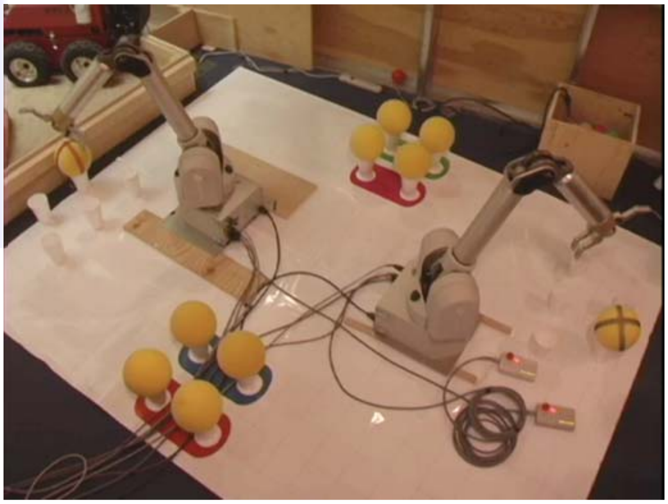
online

4. Dynamically Execute Plan


Expanding Robustness by
Dynamically Choosing Methods
{
{
[5,10] Rover1.goto(p4);
choose {
{
do { [5,10] Rover1.goto(p5); }
maintaining(site1 = ¬ obstructed);
[2,5] Rover1.imageTargets();
}
{
[2,5] Rover1.imageTargets();
[5,10] Rover1.goto(p5);
}
};
[5,10] Rover1.goto(p3);
},
{
[5,10] Rover2.goto(p1);
choose {
{
do { [2,5]Rover2.imageTargets(); }
maintaining (site1 = ¬ obstructed);
[5,10] Rover2.goto(p2);
[5,10] Rover2.goto(p3);
}
{
[5,10] Rover2.goto(p2);
[5,10] Rover2.goto(p3);
[2,5] Rover2.imageTargets();
}
}
}
in RMPL [williams et al]
Ask site1 = ¬ obstructed
Rover1.goto(p5)
Rover1.imageTargets
Start
Rover1.goto(p4)
Rover1.goto(p3)
Rover1.imageTargets
Rover1.goto(p5)
End
Ask site1 = ¬ obstructed
Failure
Rover2.imageTargets
Rover2.goto(p2)
Rover2.goto(p3)
Rover2.goto(p1)
Rover2.goto(p2)
Rover2.goto(p3)
Rover2.imageTargets
Tell site1 :
distributed Kirk
[Kim:Effinger;Block;Wehowsky]
Massachusetts Institute of Technology
19


Expanding Robustness by
Dynamically Assigning Tasks
Model-based Embedded & Robotic Systems
in RMPL [williams et al]
(Someone) Remove one ball from red bin
L[32,39]
42,55]
Swap black striped ball
• Right Robot picks up and offers ball.
• Robots perform hand-to-hand swap.
Swap red striped ball
• Left Robot picks up and offers ball.
• Robots perform hand-to-hand swap.
Remove one ball from red bin
Remove one ball from blue bin
Remove one ball from pink bin
Remove one ball from green bin
Chaski, Drake, Kirk
[Kim; Shah; Conrad]
20

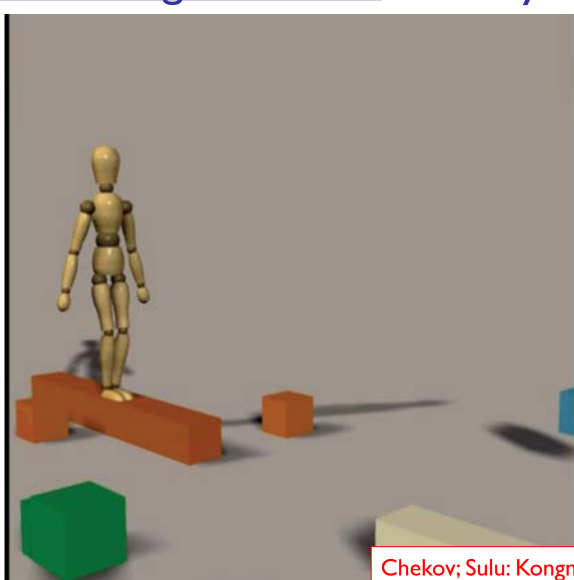
Expanding Robustness by Dynamically Assigning Tasks



- Off-nominal
- Partner adapts in response to teammate's failure.

Massachusetts Institute of Technology 21

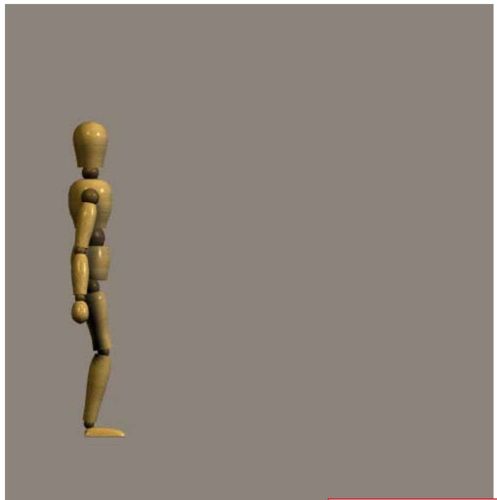
Expanding Robustness by Coordinating Underactuated Systems



Chekov; Sulu: Kongming
[Hofmann; Leaute; Blackmore; Ono; Li]

Massachusetts Institute of Technology 22

Expanding Robustness by Coordinating Underactuated Systems



Chekov; Sulu: Kongming
[Hofmann; Leaute; Blackmore; Ono; Li]

Robust Execution of Time-critical Tasks

- Executing Simple Plans
- Robust Execution
  - Describing Temporal Plans
  - Checking Temporal Plan Consistency
  - Scheduling Plans
  - Robust, Dynamic Scheduling

To Execute a Temporal Plan

Part I : Schedule Off-line

1. Describe Temporal Plan
2. Test Consistency
3. Schedule Plan

offline

online

4. Execute Plan

Part II: Schedule Online

1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan

offline

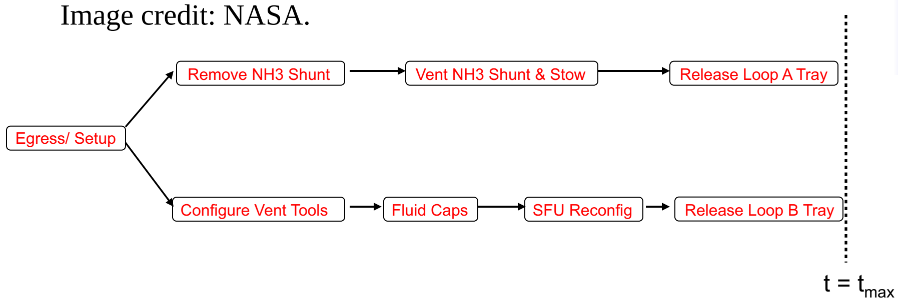
online

4. Dynamically Execute Plan

Describing Temporal Plans

Image credit: NASA.

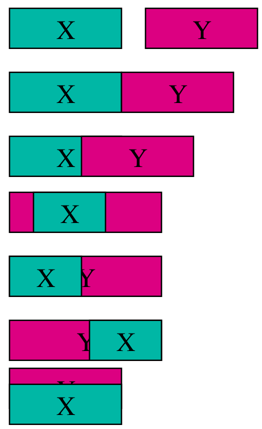
- Activities to perform
- Relationships among activities



Describing Temporal Plans

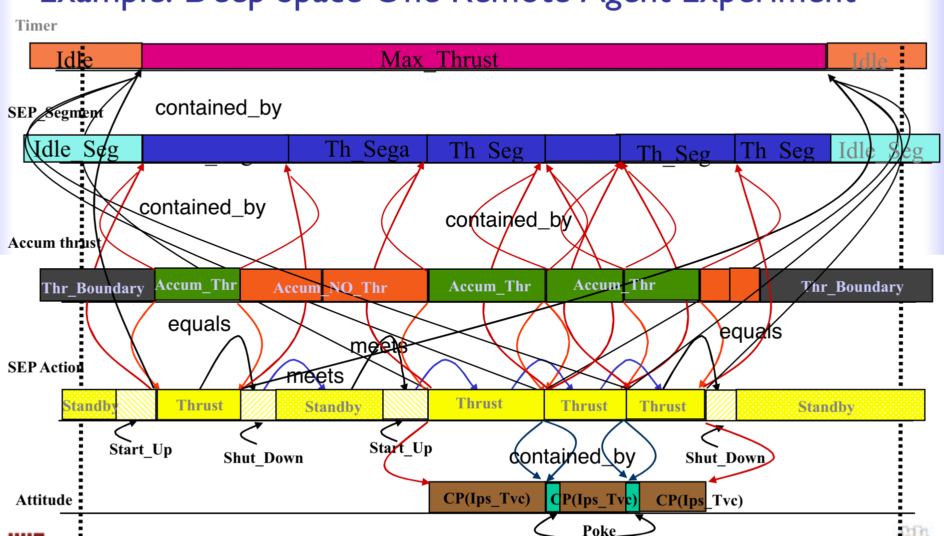
Qualitative Temporal Relationships (Allen 83)

X before Y	Y after X
X meets Y	Y met-by X
X overlaps Y	Y overlapped-by X
X during Y	Y contains X
X starts Y	Y started-by X
X finishes Y	Y finished-by X
X equals Y	Y equals X
X disjoint Y	



Describing Temporal Plans

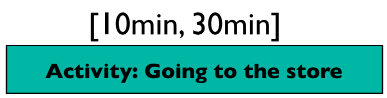
Example: Deep Space One Remote Agent Experiment



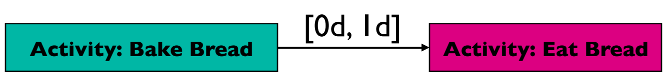
Describing Temporal Plans

Adding Metric Information

- Going to the store takes at least 10 min and at most 30 min.

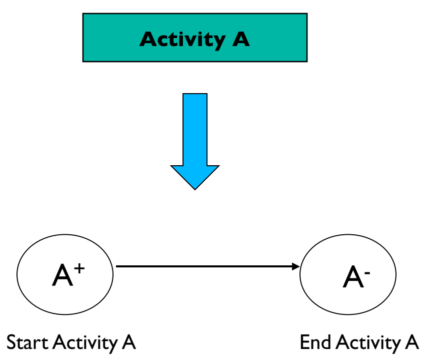


- Bread should be eaten within one day of baking.



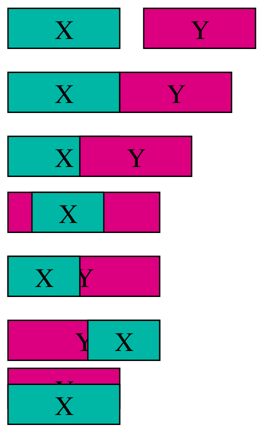
Describing Temporal Plans

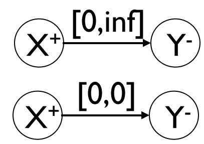
Simplify by reducing interval relations to relations on timepoints.



Describing Temporal Plans

Qualitative Temporal Relationships as timepoint inequalities

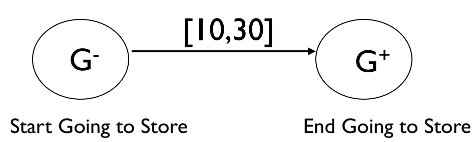
X before Y		$X^+ < Y^-$
X meets Y		$X^+ = Y^-$
X overlaps Y		$Y^- < X^+$ and $X^- < Y^+$
X during Y		$Y^- < X^-$ and $X^+ < Y^+$
X starts Y		$X^- = Y^-$ and $X^+ < Y^+$
X finishes Y		$X^- < Y^-$ and $X^+ = Y^+$
X equals Y		$X^- = Y^-$ and $X^+ = Y^+$
X disjoint Y		$X^+ < Y^-$ or $Y^+ < X^-$



Describing Temporal Plans

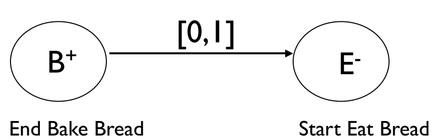
Encode metric Information by generalizing inequalities to interval constraints.

- Going to the store takes at least 10 min and at most 30 min.



$10 \leq [G^+ - G^-] \leq 30$

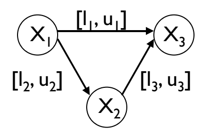
- Bread should be eaten within one day of baking.



$0 \leq [E^- - B^+] \leq 1$

Temporal Relations Described as an STP

- Simple Temporal Problem (STP)
  - variables $X_1, \ldots X_n$, representing time points with real-valued domains,
  - binary constraints of the form:



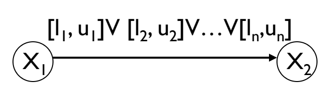
$$(X_k - X_i) \in [a_{ik}, b_{ik}].$$

Sufficient to represent:
- all Allen relations but 1…
- simple metric constraints

Can't represent:
- Disjoint activities

Temporal Relations Described as a TCSP

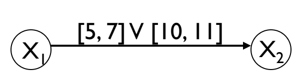
- Temporal Constraint Satisfaction Problem (TCSP)
  - Extends STP by allowing multiple intervals for each binary constraints:



$$(X_k - X_i) \in P(\{[a_{ik}, b_{ik}] \mid a_{ik} \le b_{ik}\}).$$

Supports:

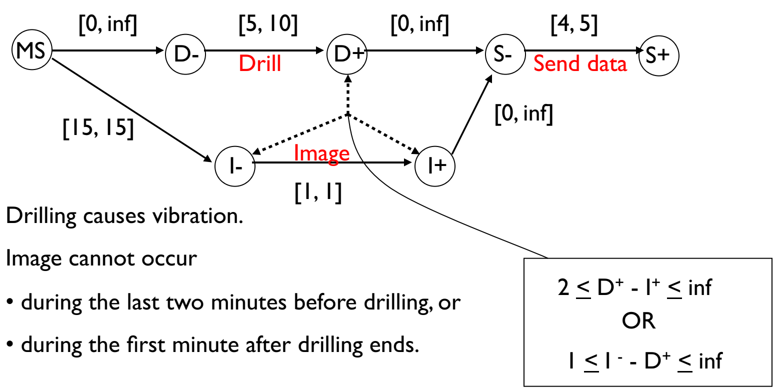
- Multiple time windows for accomplishing an activity.
- Different methods of accomplishing an activity.



Temporal Relations Described as a DTP

- Disjunctive Temporal Problem (DTP)
  - Extends TCSP by allowing non-binary constraints.

Activities of Mars Rover: Drill (D) , Image (I), Send Data (S)



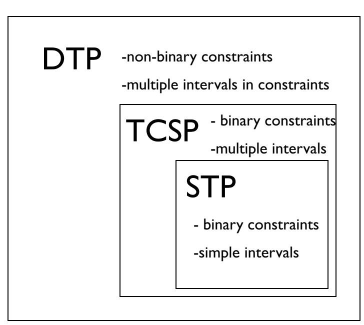
Drilling causes vibration.

Image cannot occur

- during the last two minutes before drilling, or
- during the first minute after drilling ends.

A Temporal Plan Described as a DTP

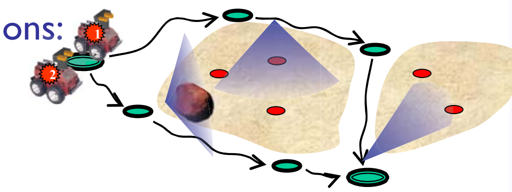
- Disjunctive Temporal Problem (DTP)
  - extends a TCSP by allowing non-binary constraints.



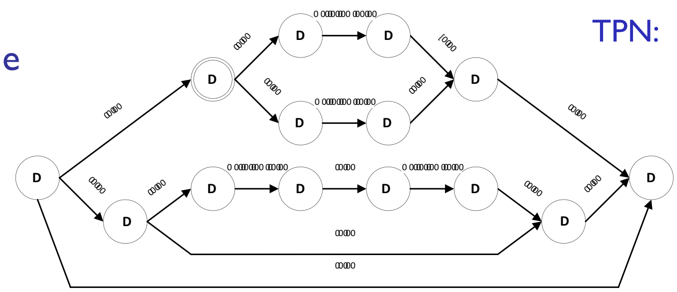
Temporal Plan Networks and Conditional STPs

RMPL - Nested Compositions:

- Activity
- Sequence
- Parallel
- Choice
- With Time



TPN:



To Execute a Temporal Plan

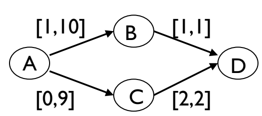
Part I : Schedule Off-line

1. Describe Temporal Plan
2. Test Consistency
3. Schedule Plan

offline

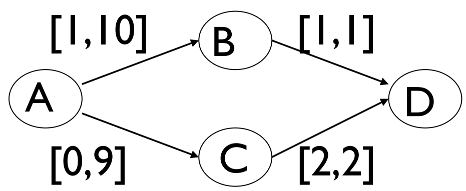
online

4. Execute Plan



Consistency of an STP

Input: An STP <X, C> where $C_j = \langle X_k, X_i, l_j, u_j \rangle$



Output: True iff there exists an X satisfying C

Map STP to Equivalent Distance Graph

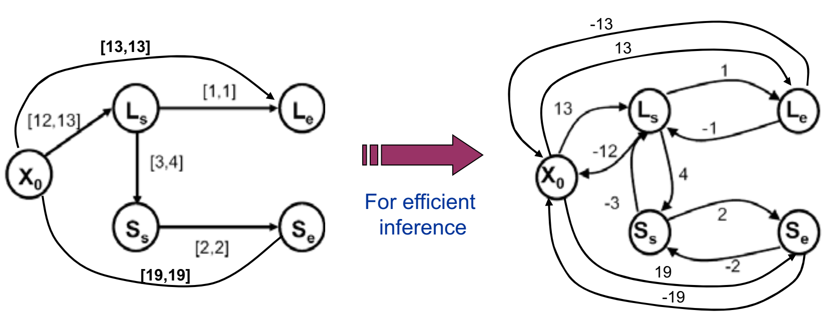
Idea: Map STN to distance (weighted) graph and check for negative cycles.

- Map upper bound to outgoing, non-negative arc.
- Map lower bound to incoming, negative arc.

$l \le X_j - X_i \le u$

$X_j - X_i \le u$

$X_i - X_j \le -l$

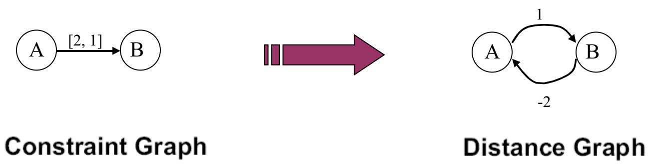


Constraint Graph

Distance Graph

STP Consistency

- Example of inconsistent constraint:



– An STP is consistent iff its distance graph has no negative cycles.

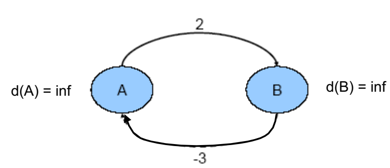
– Detect by computing shortest path from one node to all other nodes.

  - Single Source Shortest Path (SSSP)

STP Consistency: Generic Labeling Algorithm

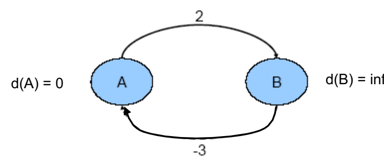
Detect negative cycles by computing the shortest-path
from a single node to all other nodes (Single Source Shortest Path).

1. For all nodes *s* in graph G
2. d(s) = inf
3. $d(s_{start}) = 0$
4. while some arc(i,j) is violating,
5. d(j) = d(i) + c(i,j)



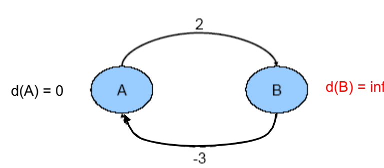
STP Consistency: Generic Labeling Algorithm

1. For all nodes *s* in graph G
2. d(s) = inf
3. $d(s_{start}) = 0$
4. while some arc(i,j) is violating,
5. d(j) = d(i) + c(i,j)



STP Consistency: Generic Labeling Algorithm

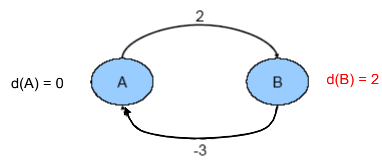
1. For all nodes *s* in graph G
2. d(s) = inf
3. $d(s_{start}) = 0$
4. while some arc(i,j) is violating,
5. d(j) = d(i) + c(i,j)



arc(i,j) is violating if,
$d(j) > d(i) + c(i,j)$

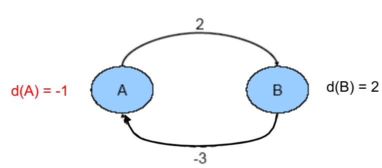
STP Consistency: Generic Labeling Algorithm

1. For all nodes *s* in graph G
2. d(s) = inf
3. $d(s_{start}) = 0$
4. while some arc(i,j) is violating,
5. d(j) = d(i) + c(i,j)



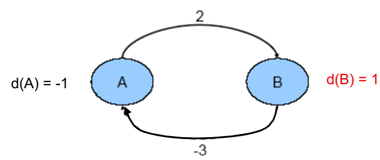
STP Consistency: Generic Labeling Algorithm

1. For all nodes *s* in graph G
2. d(s) = inf
3. $d(s_{start}) = 0$
4. while some arc(i,j) is violating,
5. d(j) = d(i) + c(i,j)



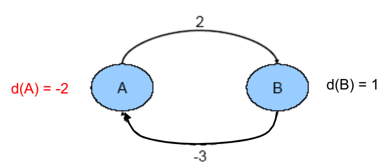
STP Consistency: Generic Labeling Algorithm

1. For all nodes *s* in graph G
2. d(s) = inf
3. $d(s_{start}) = 0$
4. while some arc(i,j) is violating,
5. d(j) = d(i) + c(i,j)



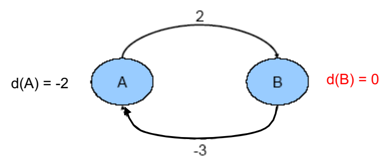
STP Consistency: Generic Labeling Algorithm

1. For all nodes *s* in graph G
2. d(s) = inf
3. $d(s_{start}) = 0$
4. while some arc(i,j) is violating,
5. d(j) = d(i) + c(i,j)



STP Consistency: Generic Labeling Algorithm

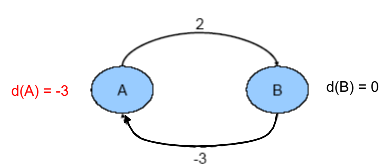
1. For all nodes *s* in graph G
2. d(s) = inf
3. $d(s_{start}) = 0$
4. while some arc(i,j) is violating,
5. d(j) = d(i) + c(i,j)



Massachusetts Institute of Technology 49

STP Consistency: Generic Labeling Algorithm

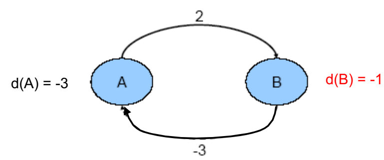
1. For all nodes *s* in graph G
2. d(s) = inf
3. $d(s_{start}) = 0$
4. while some arc(i,j) is violating,
5. d(j) = d(i) + c(i,j)



Massachusetts Institute of Technology 50

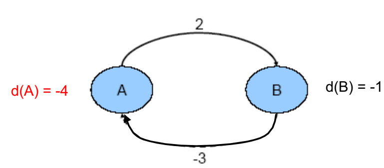
STP Consistency: Generic Labeling Algorithm

1. For all nodes *s* in graph G
2. d(s) = inf
3. $d(s_{start}) = 0$
4. while some arc(i,j) is violating,
5. d(j) = d(i) + c(i,j)



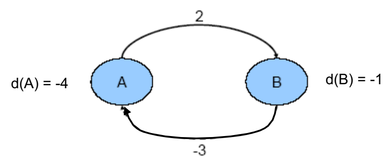
STP Consistency: Generic Labeling Algorithm

1. For all nodes *s* in graph G
2. d(s) = inf
3. $d(s_{start}) = 0$
4. while some arc(i,j) is violating,
5. d(j) = d(i) + c(i,j)



STP Consistency: Generic Labeling Algorithm

1. For all nodes *s* in graph G
2. d(s) = inf
3. $d(s_{start}) = 0$
4. while some arc(i,j) is violating,
5. d(j) = d(i) + c(i,j)

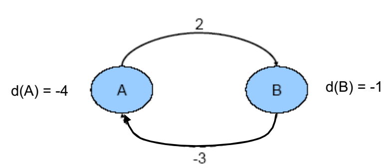


How do we detect inconsistency?

1. One way: Check for any d-value to drop below –nC

STP Consistency: FIFO Labeling Algorithm

1. For all nodes *s* in graph G
2. d(s) = inf
3. $d(s_{start}) = 0$
4. while some arc(i,j) is violating,
5. d(j) = d(i) + c(i,j)



- Maintain queue of updated nodes.
- For each node on queue, check for outgoing arcs that may be potentially violating.

To Execute a Temporal Plan

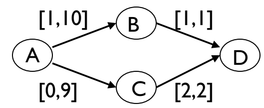
Part 1: Schedule Off-line

1. Describe Temporal Plan
2. Test Consistency
3. Schedule Plan

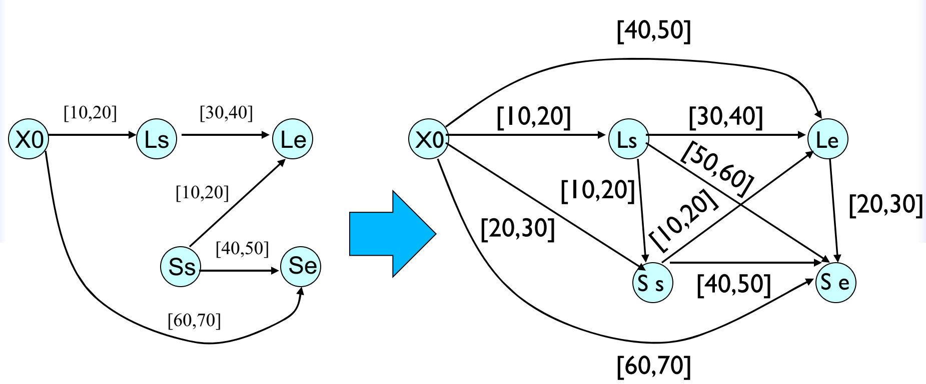
offline

online

4. Execute Plan

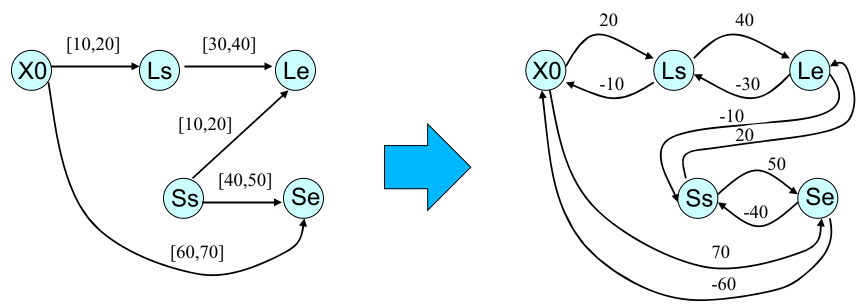


Scheduling



- Idea: Expose Implicit Constraints in STP

Scheduling with All Pairs Shortest Path Graph



- Idea: Expose Implicit Constraints in STP
- Compute All-Pairs-Shortest-Path (APSP) of d-graph (Floyd-Warshall).

Distance Graph G_d implies Constraints

- Path constraint: $i_0 = i$, $i_1 = \ldots$, $i_k = j$

$$X_j - X_i \leq \sum_{j=1}^{k} u_{i_{j-1}, i_j}$$

→ Conjoined path constraints result in the shortest path as bound:

$$X_j - X_i \leq d_{ij}$$

where d_{ij} is the shortest path from i to j

All Pairs Shortest Path

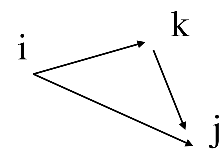
Floyd-Warshall (alternatively Johnson)

1. for i := 1 to n do $d_{ii} \leftarrow 0$;
2. for i, j := 1 to n do $d_{ij} \leftarrow w(i,j)$;

Initialize distances

3. for k := 1 to n do
4. for i, j := 1 to n do
5. $d_{ij} \leftarrow \min\{d_{ij}, d_{ik} + d_{kj}\}$;

Take minimum distance over all triangles



Complexity $O(n^3)$

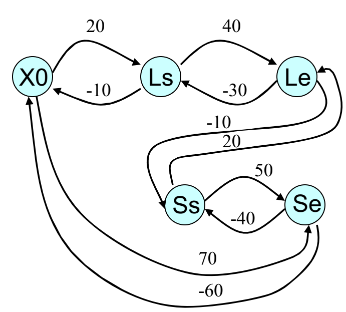
APSP

$i = X_0, k = L_s, j = L_e$

	X_0	Ls	Le	Ss	Se
X_0	0	20	~~inf~~ 60	inf	70
Ls	-10	0	40	inf	inf
Le	inf	-30	0	-10	inf
Ss	inf	inf	20	0	50
Se	-60	inf	inf	40	0

Initial d-graph

3. for k := 1 to n do
4. for i, j := 1 to n do
5. d_{ij} $\min\{d_{ij}, d_{ik} + d_{kj}\}$;

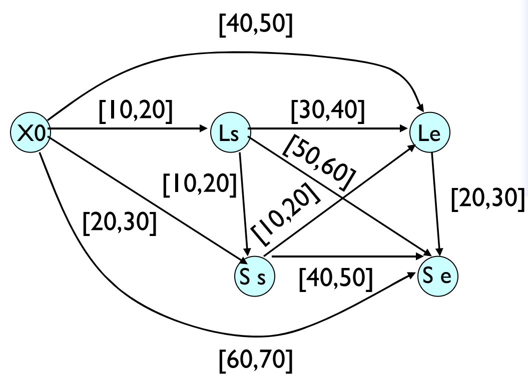


$S_{latest} = (d_{01}, \ldots, d_{0n})$

Scheduling with All Pairs Shortest Path Graph

	X_0	Ls	Le	Ss	Se
X_0	0	20	50	30	70
Ls	-10	0	40	20	60
Le	-40	-30	0	-10	30
Ss	-20	-10	20	0	50
Se	-60	-50	-20	40	0

APSP d-graph

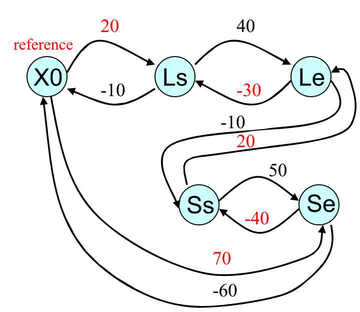


Scheduling: Latest Solution

	X_0	Ls	Le	Ss	Se
X_0	0	20	50	30	70
Ls	-10	0	40	20	60
Le	-40	-30	0	-10	30
Ss	-20	-10	20	0	50
Se	-60	-50	-20	40	0

APSP d-graph

Active constraints



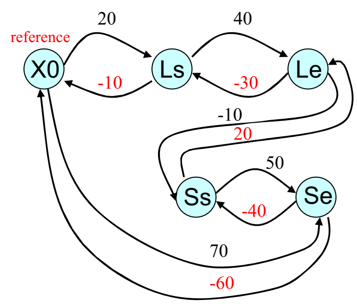
$S_{latest} = (d_{01}, \dots, d_{0n})$

Scheduling: Earliest Solution

	X_0	Ls	Le	Ss	Se
X_0	0	20	50	30	70
Ls	-10	0	40	20	60
Le	-40	-30	0	-10	30
Ss	-20	-10	20	0	50
Se	-60	-50	-20	40	0

APSP d-graph

Active constraints



$S_{earliest} = (-d_{10}, \ldots, d_{n0})$

Scheduling: Window of Feasible Values

	X_0	Ls	Le	Ss	Se
X_0	0	20	50	30	70
Ls	-10	0	40	20	60
Le	-40	-30	0	-10	30
Ss	-20	-10	20	0	50
Se	-60	-50	-20	40	0

Latest Times

Earliest Times

APSP d-graph

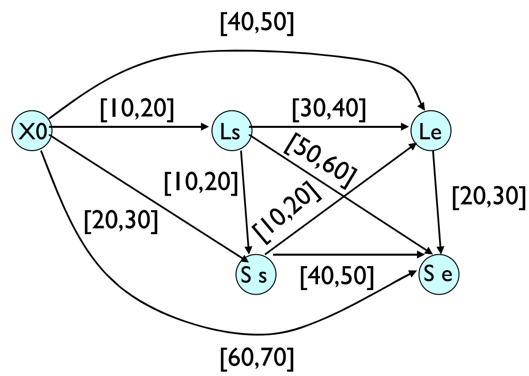
- Ls in [10, 20]
- Le in [40, 50]
- Ss in [20, 30]
- Se in [60, 70]

Scheduling without Search: Solution by Decomposition

- Can assign variables in any order, without backtracking.

Key ideas

- **Incrementally tighten feasible intervals, as commitments are made.**
- **Perform on demand.**



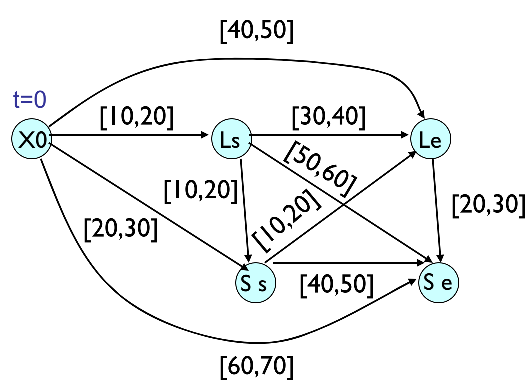
Scheduling without Search: Solution by Decomposition

- Can assign variables in any order, without backtracking.

Key ideas

- **Incrementally tighten feasible intervals, as commitments are made.**
- **Perform on demand.**

- Select value for X0



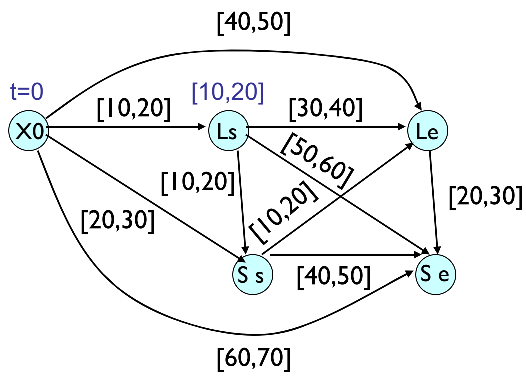
Scheduling without Search: Solution by Decomposition

- Can assign variables in any order, without backtracking.

Key ideas

- **Incrementally tighten feasible intervals, as commitments are made.**
- **Perform on demand.**

- Select value for X0
- Select value for Ls, consistent with X0



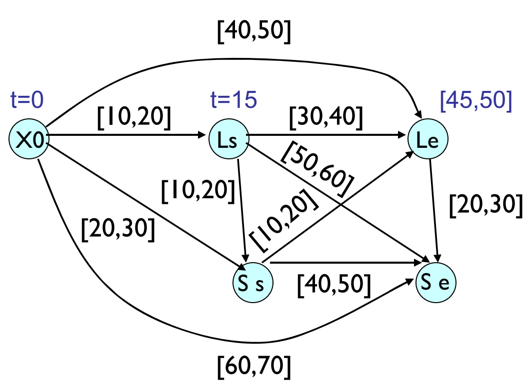
Scheduling without Search: Solution by Decomposition

- Can assign variables in any order, without backtracking.

Key ideas

- **Incrementally tighten feasible intervals, as commitments are made.**
- **Perform on demand.**

- Select value for X0
- Select value for Ls, consistent with X0
- Select value for Le, consistent with X0, Ls



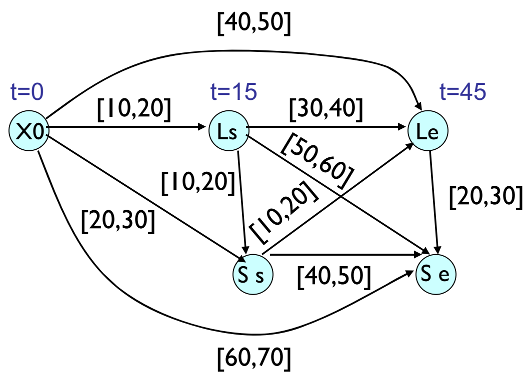
Scheduling without Search: Solution by Decomposition

- Can assign variables in any order, without backtracking.

Key ideas

- **Incrementally tighten feasible intervals, as commitments are made.**
- **Perform on demand.**

- Select value for X0
- Select value for Ls, consistent with X0
- Select value for Le, consistent with X0, Ls



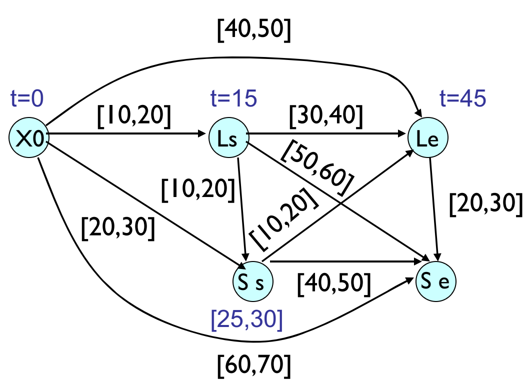
Scheduling without Search: Solution by Decomposition

- Can assign variables in any order, without backtracking.

Key ideas

- **Incrementally tighten feasible intervals, as commitments are made.**
- **Perform on demand.**

- Select value for X0
- Select value for Ls, consistent with X0
- Select value for Le, consistent with X0, Ls
- Select value for Ss, consistent with X0, Ls, Le

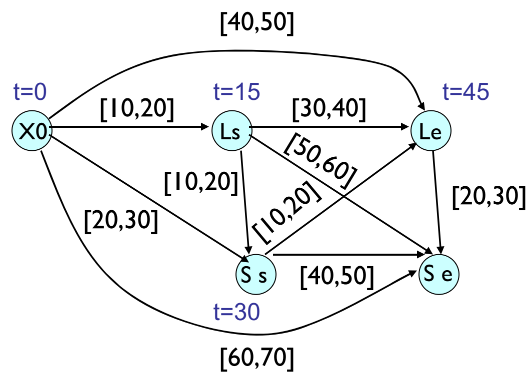


Scheduling without Search: Solution by Decomposition

- Can assign variables in any order, without backtracking.

Key ideas

- **Incrementally tighten feasible intervals, as commitments are made.**
- **Perform on demand.**



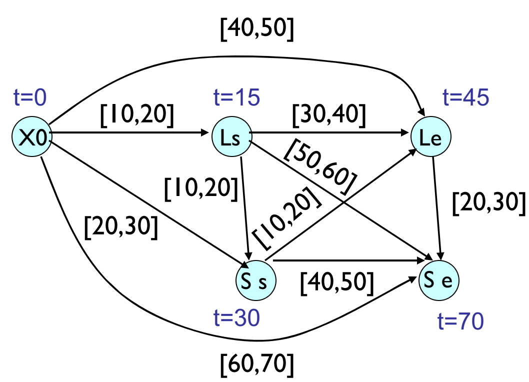
- Select value for X0
- Select value for Ls, consistent with X0
- Select value for Le, consistent with X0, Ls
- Select value for Ss, consistent with X0, Ls, Le

Scheduling without Search: Solution by Decomposition

- Can assign variables in any order, without backtracking.

Key ideas

- **Incrementally tighten feasible intervals, as commitments are made.**
- **Perform on demand.**



- Select value for X0
- Select value for Ls, consistent with X0
- Select value for Le, consistent with X0, Ls
- Select value for Ss, consistent with X0, Ls, Le
- Select value for Se…

To Execute a Temporal Plan

Part I : Schedule Off-line

1. Describe Temporal Plan
2. Test Consistency
3. Schedule Plan

offline

online

4. Execute Plan

Problem: delays and fluctuations in task duration can cause plan failure.

Observation: Least commitment temporal plans leave room to adapt.

Flexible Execution adapts through dynamic scheduling [Muscettola et al]

- Assigns time to event when executed

To Execute a Temporal Plan

Part I : Schedule Off-line

1. Describe Temporal Plan
2. Test Consistency
3. Schedule Plan

offline

online

4. Execute Plan

Part II: Schedule Online

1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan

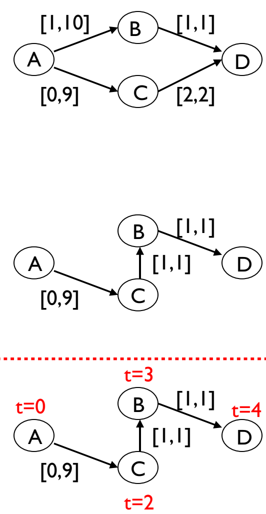
offline

online

4. Dynamically Execute Plan

To Execute a Temporal Plan

How do we schedule on line?



Part II: Schedule Online

1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan

offline

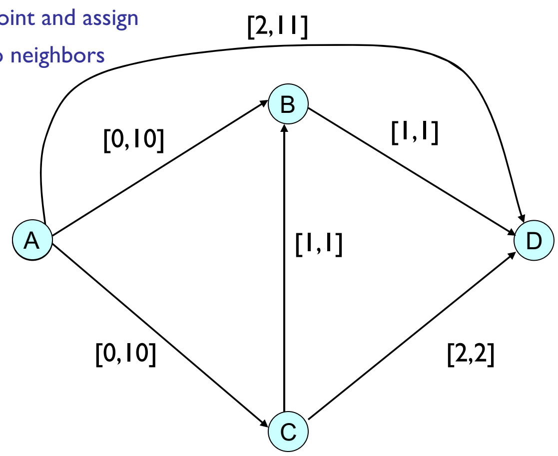
online

4. Dynamically Execute Plan

Dynamic Scheduling by Decomposition?

Consider a Simple Example

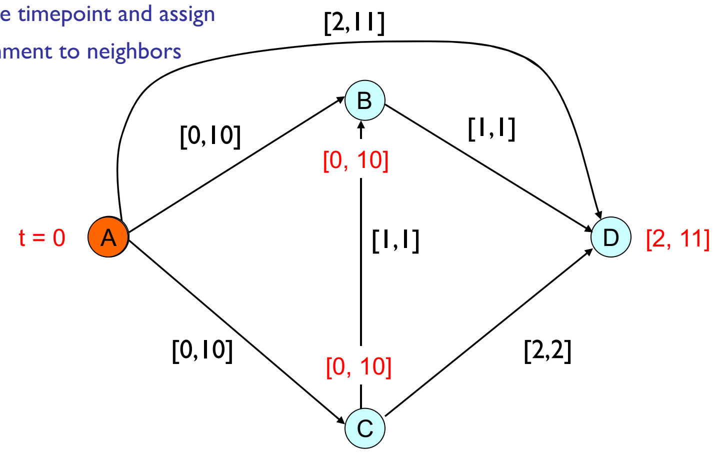
- Select executable timepoint and assign
- Propagate assignment to neighbors



Dynamic Scheduling by Decomposition?

Consider a Simple Example

- Select executable timepoint and assign
- Propagate assignment to neighbors



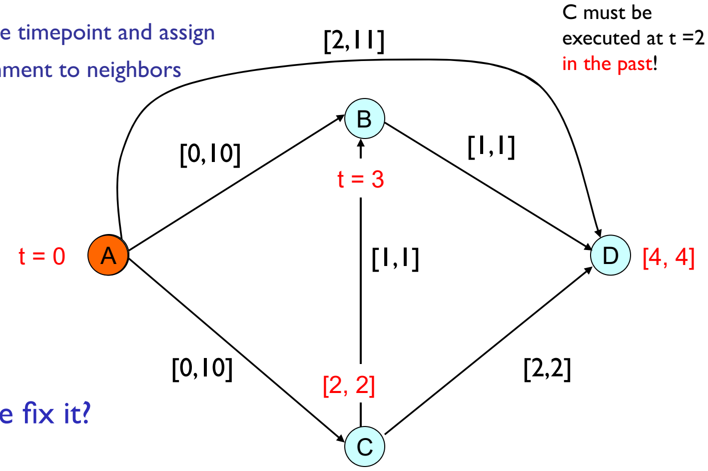
~~Dynamic Scheduling by Decomposition?~~

Consider a Simple Example

- Select executable timepoint and assign
- Propagate assignment to neighbors

Uh oh!

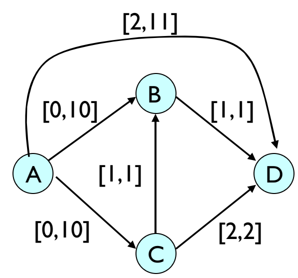
C must be executed at t =2 in the past!



How can we fix it?

Dispatching Execution Controller

- How can we fix it?
  - Assignments must monotonically increase in value.
  - Respect induced orderings.
- Execute an event when enabled and alive
  - Enabled – Predecessors are completed
  - Alive – Current time within bound of task



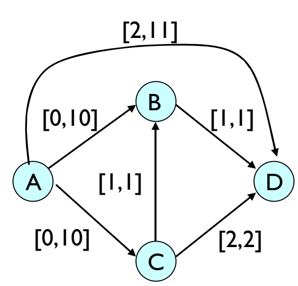
Dispatching Execution Controller

Initially:

- E = Time points w/o predecessors
- S = { }

Repeat:

1. Wait until current time has advanced such that some TP in E is active
2. Set TP's execution time to current time.
3. Add TP to S.
4. Propagate time of execution to TP's immediate neighbors
5. Add to E, all immediate neighbors that become enabled
   - TP enabled if all +lb edges starting at TP have their destination in S.



Dynamic Scheduling through Dispatchable Execution

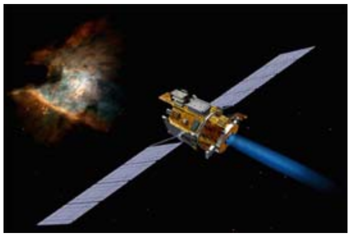


Image credit: NASA.