Programs with Flexible Time

When?

Contributions:
Brian Williams
Patrick Conrad
Simon Fang
Paul Morris
Nicola Muscettola
Pedro Santana
Julie Shah
John Stedl
Andrew Wang

Steve Levine
Tuesday, Feb 16th

courtesy of JPL
Assignments

Problems Sets:
• Pset 1 due tomorrow (Wednesday) at 11:59pm
• Pset 2 released tomorrow

Interesting references:
Outline

• Programs with Flexible Time
  – Intro
  – Describing temporal plans
  – Exposing implicit constraints
  – Consistency checking
  – Offline scheduling
  – Online execution
  – Reformulating for faster online execution
A single “cognitive system” language and executive.

Enterprise

Uhura
Collaboratively resolves goal failures

Kirk
Sketches mission and assigns tasks

Burton
Plans actions

Pike
Coordinates and monitors tasks

Sulu
Plans paths

Bones
Diagnoses likely failures

User
Goals & models in RMPL

Control Commands

© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/.
Flexible Time

• Flexible time = more robustness

• We tell cognitive robot:
  – Timing requirements ("engage boosters 2-4 minutes after launch but before reaching orbit")
  – Cognitive robot schedules autonomously.
Execution of Timed Model-based Programs

image

```
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 01]

Decisions include what, how, who and when.

Agents adapt to temporal disturbances in a coordinated manner by scheduling the start of activities on the fly. [Muscettola, Morris, Tsamardinos, KR 98]
To Execute a Temporal Plan

Schedule Offline

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

Schedule Online

1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan
4. Dynamically Schedule Plan
To Execute a Temporal Plan

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

Schedule Offline

Schedule Online
Describe Temporal Plan

- Activities to perform
- Relationships among activities

![Diagram showing flow of activities](image-url)
Metric Temporal Relations

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

\[10\text{min}, 30\text{min}\]

Activity: Going to the store

- Bread should be eaten within one day of baking.

\[0\text{d}, 1\text{d}\]

Activity: Bake Bread ➔ Activity: Eat Bread
Simplify by reducing interval relations to relations on timepoints.
Metric Temporal Relations

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

\[ G^-(t) \rightarrow G^+(t) \quad [10m,30m] \]
Start Going to Store \hspace{1cm} End Going to Store

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

- Bread should be eaten within one day of baking.

\[ B^+(t) \rightarrow E^-(t) \quad [0d,1d] \]
End Bake Bread \hspace{1cm} Start Eat Bread

\[ 0 \leq [E^- - B^+] \leq 1 \]
Qualitative Temporal Relations

[Allen 83]

- X before Y
- X meets Y
- X overlaps Y
- X during Y
- X starts Y
- X finishes Y
- X equals Y
- X disjoint Y

Y after X
Y met-by X
Y overlapped-by X
Y contains X
Y started-by X
Y finished-by X
Y equals X
Qualitative Temporal Relations

Expressed as timepoint inequalities:

- X before Y: $X^+ < Y^-$
- X meets Y: $X^+ = Y^-$
- X overlaps Y: $Y^- < X^+$ and $X^- < Y^+$
- X during Y: $Y^- < X^- \text{ and } X^+ < Y^+$
- X starts Y: $X^- = Y^- \text{ and } X^+ < Y^+$
- X finishes Y: $X^- < Y^- \text{ and } X^+ = Y^+$
- X equals Y: $X^- = Y^- \text{ and } X^+ = Y^+$
- X disjoint Y: $X^+ < Y^-$ or $Y^+ < X^-$

[Villain & Kautz; Simmons]
Temporal Relations Described by a Simple Temporal Network (STN)

• Simple Temporal Network
  • Tuple \(<X, C>\) where:
    • variables \(X_1, \ldots X_n\), represent time points (real-valued domains)
    • binary constraints \(C\) of the form:
      \[
      (X_k - X_i) \in [a_{ik}, b_{ik}]
      \]
      – called \textit{links}.

  [Dechter, Meiri, Pearl 91]

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

Can’t represent:
• Allen’s disjoint relation
Modeling Visualization

- http://bicycle.csail.mit.edu/stn/
To Execute a Temporal Plan

Schedule Offline

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

Diagram:
- A (1,1)
- B (1,10)
- C (0,9)
- D (2,2)

Flow:
- 1. Describe Temporal Plan
- 2. Test Consistency
- 3. Schedule Plan
- 4. Execute Plan
Consistency of an STN

Input: STN \(<X, C>\) where \(C_j = \langle \langle X_k, X_i \rangle, \langle a_j, b_j \rangle \rangle\)

STN is \textit{consistent} iff there exists an assignment to times \(X\) satisfying \(C\).
Schedule of an STN

Input: STN <X, C> where \( C_j = \langle \langle X_k, X_i \rangle, \langle a_j, b_j \rangle \rangle \)

\[
\begin{align*}
&\text{A} & \rightarrow & \text{B} & \rightarrow & \text{D} \\
&\text{C} & \rightarrow & \text{B} & \rightarrow & \text{D} \\
&[0,9] & \rightarrow & [1,10] & \rightarrow & [1,1] \\
&[2,2] & \rightarrow & [1,1] & \rightarrow & [1,1] \\
\end{align*}
\]

**Schedule** is assignment to all timepoints X consistent with constraints.
How to find schedule?

• Idea: Transform STN
  – Transform to *distance-graph*
  – Common graph algorithms (i.e., shortest path) will apply
Transformation to distance graph

- (Board)
Map STN to Distance (D) Graph

Simple Temporal Network

Distance Graph

\[ A \rightarrow^{{[l, u]}} B \]

\[ [l, u] \leq B - A \leq u \]

\[ A - B \leq -1 \]

- Upperbound mapped to outgoing, non-negative arc.
- Lowerbound mapped to incoming, negative arc.

[Dechter, Meiri, Pearl 91]
Algorithm: offline scheduling

Initialize execution window to \([-\infty, \infty]\) for each event
while unexecuted events:

\[ x_i = \text{pick any unexecuted event} \]
\[ t_i = \text{pick any time in } x_i \text{’s execution window} \]

Propagate to all \(x_i\)’s neighbors & update their windows
Naïve (and wrong) scheduling

• (Board)
Propagating to neighbors

Tighten neighbor’s execution windows:

- outgoing edges to neighbor: $u' = \min(u, t_i + w_u)$
- incoming edges from neighbor: $l' = \max(l, t_i - w_l)$

$[u, l] \Rightarrow tightened \ [u', l']$

$x_i = t_i$
Exposing Implicit Constraints

• (Board)
**APSP Graph: Windows of Feasible Values**

### APSP d-graph

<table>
<thead>
<tr>
<th></th>
<th>$X_0$</th>
<th>Ls</th>
<th>Le</th>
<th>Ss</th>
<th>Se</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_0$</td>
<td>0</td>
<td>20</td>
<td>50</td>
<td>30</td>
<td>70</td>
</tr>
<tr>
<td>Ls</td>
<td>-10</td>
<td>0</td>
<td>40</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>Le</td>
<td>-40</td>
<td>-30</td>
<td>0</td>
<td>-10</td>
<td>30</td>
</tr>
<tr>
<td>Ss</td>
<td>-20</td>
<td>-10</td>
<td>20</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>Se</td>
<td>-60</td>
<td>-50</td>
<td>-20</td>
<td>40</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Earliest Times**
- **Latest Times**

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

• An STN or distance graph is *dispatchable* if:
  – Can be properly scheduled via local propagations to neighbors only

• Requires all implicit constraints be explicit
Checking Consistency

• (Board)
Check D-Graph Consistency

- Consistent iff D-graph has no negative cycles.
- Detect by computing shortest path from one event to all others.
  - Single Source Shortest Path (SSSP).
  - Event must reach all others.

Example of inconsistent constraint:

Simple Temporal Network

Distance Graph
Summary

• To schedule, want a simple, local-propagation algorithm
  – Requires exposing implicit constraints

• All-pairs shortest path (APSP) exposes all implicit constraints
  – Puts network in *dispatchable form*

• Negative cycle in APSP: inconsistent.
To Execute a Temporal Plan

Schedule Offline

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

Schedule Online

1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan
4. Dynamically Schedule Plan
Algorithm: offline scheduling

Compute dispatchable form (i.e., APSP)
Initialize execution window to $[-\infty, \infty]$ for each event
while unexecuted events:

\[ x_i = \text{pick any unexecuted event} \]
\[ t_i = \text{pick any time in } x_i \text{’s execution window} \]

Propagate to all $x_i$’s neighbors & update their windows
The original STN
Distance graph transformation
All pairs shortest path

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>B</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
</tr>
</tbody>
</table>
Computing a schedule

\[\begin{tikzpicture}
\node[circle, draw] (A) at (0,0) {$A$};
\node[circle, draw] (B) at (2,2) {$B$};
\node[circle, draw] (C) at (0,2) {$C$};
\node[circle, draw] (D) at (2,0) {$D$};
\draw[->] (A) to [bend left] node[above] {10} (B);
\draw[->] (B) to [bend left] node[above] {1} (D);
\draw[->] (D) to [bend left] node[above] {2} (C);
\draw[->] (C) to [bend left] node[above] {1} (B);
\draw[->] (B) to [bend left] node[above] {-1} (C);
\draw[->] (B) to [bend left] node[above] {-1} (D);
\draw[->] (C) to [bend left] node[above] {-1} (B);
\draw[->] (C) to [bend left] node[above] {1} (A);
\draw[->] (D) to [bend left] node[above] {-2} (C);
\draw[->] (D) to [bend left] node[above] {2} (B);
\draw[->] (A) to [bend left] node[above] {9} (C);
\draw[->] (A) to [bend left] node[above] {1} (B);
\draw[->] (A) to [bend left] node[above] {0} (D);
\end{tikzpicture}\]
Computing a schedule

Initialize execution windows for each event in the plan
Computing a schedule

Assign the first event
Computing a schedule

outgoing edges to neighbor: \( u' = \min(u, t_i + w_u) \)
incoming edges from neighbor: \( l' = \max(l, t_i - w_i) \)

Propagate updated time bounds to neighbors
Computing a schedule

outgoing edges to neighbor: $u' = \min(u, t_i + w_u)$

incoming edges from neighbor: $l' = \max(l, t_i - w_i)$

Propagate updated time bounds to neighbors
Computing a schedule

$t = 0$

Propagate updated time bounds to neighbors
Computing a schedule

Propagate updated time bounds to neighbors
Computing a schedule

Arbitrarily pick another time point and assign it...
Computing a schedule

Propagate updated time bounds to neighbors
Computing a schedule

$\begin{array}{c}
t = 0 \\
A \quad B \quad C \quad D
\end{array}$

$\begin{array}{c}
t = 3 \\
[2, 2] \quad [4, 4]
\end{array}$

Propagate updated time bounds to neighbors
Computing a schedule

Pick another event and assign it
Computing a schedule

Propagate to neighbors
Computing a schedule

Assign the final event
Pre-computed schedules not robust against fluctuations

• We’ve just computed a schedule:
  \( t_A = 0, \ t_B = 3, \ t_C = 2, \ t_D = 4 \)

• But what if there’s a disturbance?
  – i.e., what if \( t_B = 3.1？ \)
  – i.e., what if \( t_B = 4？ \)
  – i.e., what if \( t_B = 100？ \)

• Pre-computed schedules not robust against fluctuations!

• **Solution:** Dispatch dynamically online.
  – Schedule events “on the fly,” after observing past event times.
  – Increases robustness to many unanticipated fluctuations.
  – Flexible temporal constraints allow this!
To Execute a Temporal Plan

Schedule Offline

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

Schedule Online

1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan
4. Dynamically Schedule Plan
How do we schedule online?

- First, consider naive (incorrect!) approach.
- Similar to offline schedule algorithm, but now online:
  - \textbf{Wait} until current time in execution window (“active”)

- (Still a problem though as we’ll see shortly)
Naïve (wrong!) online dispatcher

Compute dispatchable form (i.e., APSP)
Initialize execution window to $[-\infty, \infty]$ for each event
while unexecuted events:

\[ x_i = \text{pick any unexecuted event if current time in window} \]
\[ t_i = \text{now} \]

Propagate to all $x_i$’s neighbors & update their windows
Naïve (wrong!) online scheduling

Arbitrarily picking next event…
Naïve (wrong!) online scheduling

Arbitrarily picking next event…
Naïve (wrong!) online scheduling

...but wait! We just assigned a past time!
Enablement conditions dictate the ordering of dispatched events

- **Online**: must assign monotonically increasing times
  - whereas offline algorithms may assign in any order.

- **How can we constrain dispatcher to do this?**

- **Solution**: determine “enablement conditions” by analyzing negative edges.
  - Allows us to infer if some edges must precede other edges
Enablement conditions dictate the ordering of dispatched events

- Negative edges from APSP dictate ordering constraints
Enablement conditions dictate the ordering of dispatched events

- Negative edges from APSP dictate ordering constraints

B must occur after both A and C!
Enablement conditions dictate the ordering of dispatched events

- An event is **enabled** if all its neighbors over negative edges have already been dispatched.
  - All “predecessors” have been dispatched.

- Modify online dispatching algorithm to only dispatch events if they are enabled.

- An event is **active** if the current time is within that event’s execution window
Corrected online dispatcher

Compute dispatchable form (i.e., APSP)
Initialize execution window to \([-\infty, \infty]\) for each event

\[ E \leftarrow \{\text{events with no predecessors}\} \quad \# \text{set of enabled events} \]

\[ S \leftarrow \{\} \quad \# \text{set of executed events} \]

while unexecuted events:

- Wait until some event \( x_i \) in \( E \) is active
- \( t_i = \text{now} \quad \# \text{dispatch } x_i \text{ now at } t_i \)
- Propagate to all \( x_i \)'s neighbors & update their windows
- Add \( x_i \) to \( S \)
- Add to \( E \) any now-enabled events
Running online dispatcher

Compute dispatchable form (i.e., APSP)
Initialize execution windows to \([-\infty, \infty]\)

\[ E \leftarrow \{\text{events with no predecessors}\} \]
\[ S \leftarrow \{\} \]

while unexecuted events:
  
  Wait until some event \( x_i \) in \( E \) is active
  \( t_i = \text{now} \)
  
  Propagate to \( x_i \)’s neighbors
  
  Add \( x_i \) to \( S \)
  
  Add to \( E \) any now-enabled events

\( E = \{A, C\} \)
\( S = \{\} \)

A, C initially in \( E \) – have no negative, outgoing edges
Running online dispatcher

Compute dispatchable form (i.e., APSP)
Initialize execution windows to $[-\infty, \infty]$
$E \leftarrow \{\text{events with no predecessors}\}$
$S \leftarrow \{\}$
while unexecuted events:
  Wait until some event $x_i$ in $E$ is active
  $t_i = \text{now}$
  Propagate to $x_i$’s neighbors
  Add $x_i$ to $S$
  Add to $E$ any now-enabled events

$E = \{A, C\}$
$S = \{\}$

A is enabled and in $E$.
(could have chosen $C$ too)
Running online dispatcher

Compute dispatchable form (i.e., APSP)
Initialize execution windows to \([-\infty, \infty]\)
\[E \leftarrow \{\text{events with no predecessors}\}\]
\[S \leftarrow \{\}\]
while unexecuted events:
  Wait until some event \(x_i\) in \(E\) is active
  \(t_i = \text{now}\)
  Propagate to \(x_i\)'s neighbors
  Add \(x_i\) to \(S\)
  Add to \(E\) any now-enabled events

\[E = \{C\}\]
\[S = \{A\}\]

Dispatch A and propagate
Running online dispatcher

Compute dispatchable form (i.e., APSP)
Initialize execution windows to \([-\infty, \infty]\)
\(E \leftarrow \{\text{events with no predecessors}\}\)
\(S \leftarrow \{\}\)

while unexecuted events:
  Wait until some event \(x_i\) in \(E\) is active
  \(t_i = \text{now}\)
  Propagate to \(x_i\)'s neighbors
  Add \(x_i\) to \(S\)
  Add to \(E\) any now-enabled events

\(E = \{C\}\)
\(S = \{A\}\)

B, D not enabled! But C still is.
Running online dispatcher

Compute dispatchable form (i.e., APSP)
Initialize execution windows to \([-\infty, \infty]\)
\(E \leftarrow \{\text{events with no predecessors}\}\)
\(S \leftarrow \{\}\)
while unexecuted events:
  
  Wait until some event \(x_i\) in \(E\) is active
  \(t_i = \text{now}\)
  Propagate to \(x_i\)'s neighbors
  Add \(x_i\) to \(S\)
  Add to \(E\) any now-enabled events

Dispatch & propagate C.
Running online dispatcher

Compute dispatchable form (i.e., APSP)
Initialize execution windows to \([-\infty, \infty]\)

\(E \leftarrow \{\text{events with no predecessors}\}\)
\(S \leftarrow \{\}\)

while unexecuted events:
  Wait until some event \(x_i\) in \(E\) is active
  \(t_i = \text{now}\)
  Propagate to \(x_i\)’s neighbors
  Add \(x_i\) to \(S\)
  Add to \(E\) any now-enabled events

\(B\) is now enabled (but still not \(D\)).

\[E = \{B\}\]
\[S = \{A, C\}\]
Running online dispatcher

Compute dispatchable form (i.e., APSP)
Initialize execution windows to \([-\infty, \infty]\)
\[ E \leftarrow \{\text{events with no predecessors}\}\]
\[ S \leftarrow \{\}\]
while unexecuted events:
  Wait until some event \(x_i\) in \(E\) is active
  \(t_i = \text{now}\)
  Propagate to \(x_i\)'s neighbors
  Add \(x_i\) to \(S\)
  Add to \(E\) any now-enabled events

Dispatch & propagate B.

\[ E = \{\}\]
\[ S = \{A, C, B\}\]
Running online dispatcher

Compute dispatchable form (i.e., APSP)
Initialize execution windows to $[-\infty, \infty]$
$E \leftarrow \{\text{events with no predecessors}\}$
$S \leftarrow \{\}$
while unexecuted events:
  Wait until some event $x_i$ in $E$ is active
  $t_i = \text{now}$
  Propagate to $x_i$’s neighbors
  Add $x_i$ to $S$
  Add to $E$ any now-enabled events

$D$ is finally enabled.
Running online dispatcher

Compute dispatchable form (i.e., APSP)
Initialize execution windows to \([-\infty, \infty]\)

\[ E \leftarrow \{\text{events with no predecessors}\} \]

\[ S \leftarrow \{\} \]

while unexecuted events:
  
  Wait until some event \( x_i \) in \( E \) is active
  
  \( t_i = \text{now} \)
  
  Propagate to \( x_i \)'s neighbors
  
  Add \( x_i \) to \( S \)
  
  Add to \( E \) any now-enabled events

Finish up by dispatching D!

\[ E = \{\} \]

\[ S = \{A, C, B, D\} \]
Online dispatching algorithm remarks

• By considering predecessors, we guarantee that events assigned monotonically increasing times online.
• Capable of responding to fluctuations that do not affect overall temporal feasibility.
• (Note: must be run on an dispatchable / APSP graph!)
Online dispatcher efficiency

• Consider an STN with $n$ edges.
• How many edges in APSP distance graph?

Compute dispatchable form (i.e., APSP)
Initialize execution windows to $[-\infty, \infty]$
$E \leftarrow \{\text{events with no predecessors}\}$
$S \leftarrow \{}$
while unexecuted events:
  Wait until some event $x_i$ in $E$ is active
  $t_i = \text{now}$
  Propagate to $x_i$’s neighbors
  Add $x_i$ to $S$
  Add to $E$ any now-enabled events
Online dispatcher efficiency

• Consider an STN with $n$ edges.
• How many edges in APSP distance graph? $n^2$.

Compute dispatchable form (i.e., APSP)
Initialize execution windows to $[-\infty, \infty]$
$E \leftarrow \{\text{events with no predecessors}\}$
$S \leftarrow \{\}$
while unexecuted events:
    Wait until some event $x_i$ in $E$ is active
    $t_i = \text{now}$
    Propagate to $x_i$’s neighbors
    Add $x_i$ to $S$
    Add to $E$ any now-enabled events
Online dispatcher efficiency

• Consider an STN with $n$ edges.
• How many edges in APSP distance graph? $n^2$.
• How many neighbors to propagate to each step?

Compute dispatchable form (i.e., APSP)
Initialize execution windows to $[-\infty, \infty]$
$E \leftarrow \{\text{events with no predecessors}\}$
$S \leftarrow \{\}$
while unexecuted events:
    Wait until some event $x_i$ in $E$ is active
    $t_i = \text{now}$
    Propagate to $x_i$’s neighbors
    Add $x_i$ to $S$
    Add to $E$ any now-enabled events
Online dispatcher efficiency

• Consider an STN with $n$ edges.
• How many edges in APSP distance graph? $n^2$.
• How many neighbors to propagate to each step? $n$.

Compute dispatchable form (i.e., APSP)
Initialize execution windows to $[-\infty, \infty]$
$E \leftarrow \{\text{events with no predecessors}\}$
$S \leftarrow \{\}$
while unexecuted events:
  Wait until some event $x_i$ in $E$ is active
  $t_i = \text{now}$
  Propagate to $x_i$’s neighbors
  Add $x_i$ to $S$
  Add to $E$ any now-enabled events
Online dispatcher efficiency

• Consider an STN with $n$ edges.
• How many edges in APSP distance graph? $n^2$.
• How many neighbors to propagate to each step? $n$.
• Large STNs: propagation slow. Want to reduce this.

Compute dispatchable form (i.e., APSP)
Initialize execution windows to $[-\infty, \infty]$
$E \leftarrow \{\text{events with no predecessors}\}$
$S \leftarrow \{\}$
while unexecuted events:
  Wait until some event $x_i$ in $E$ is active
  $t_i = \text{now}$
  Propagate to $x_i$’s neighbors
  Add $x_i$ to $S$
  Add to $E$ any now-enabled events
To Execute a Temporal Plan

**Schedule Offline**

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

**Schedule Online**

1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan
4. Dynamically Schedule Plan
You don’t need all those edges!
You don’t need all those edges!

Equivalent *minimal dispatchable network*
You don’t need all those edges!

Let’s consider a specific triangle of edges.
You don’t need all those edges!

Let’s consider a specific triangle of edges.

Do we really need the bottom edge?
You don’t need all those edges!

Let’s consider a different triangle of edges.

Do we really need the bottom edge?
Imagine a “unidirectional rope” of length 10 constraining sliders on a track.
Imagine a “unidirectional rope” of length 10 constraining sliders on a track.
Rope analogy

Now add in ropes for other constraints
Rope analogy

Imagine pulling A and D as tightly as possible.
Rope analogy

Can we remove rope AD without changing behavior?

Yes! Same possible positions for A, B, D.
Can we remove ropes AB, BD without changing behavior?

No. AD still constrained, but B could slide freely! Not the same behavior. Collectively, AB and BD entail AD (but AD does not entail both AB and AD).
Upper dominating edges - detection from APSP

If $d_{AC}, d_{BC} \geq 0$ and $d_{AB} + d_{BC} = d_{AC}$ then $BC$ dominates $AC$

(Proof omitted - based on triangle rule property of APSP. Please see notes / reading for more info)
Lower dominating edges - detection from APSP

If $d_{AB}, d_{AC} < 0$ and $d_{AB} + d_{BC} = d_{AC}$ then AB dominates AC

(Proof omitted - based on triangle rule property of APSP. Please see notes / reading for more info)
Dominance example
Dominance example

Upper dominated!
Dominance example

Upper dominated!
Dominance example

Upper dominated!

Diagram of a network showing nodes A, B, C, and D with directed edges and numerical values indicating dominance.
Dominance example

Upper dominated!
Dominance example

Upper dominated!
Dominance example

Upper dominated!
Dominance example

Lower dominated!
Dominance example

Lower dominated!
Dominance example

Lower dominated!
Dominance example

Lower dominated!
Dominance example

Lower dominated!
Dominance example

Lower dominated!
Dominance example
Dominance example

Original APSP distance graph

... now in \textit{minimal dispatchable form}!
Dominance example

Original STN

... now in \textit{minimal dispatchable form}!
FilteringAlgorithm($G$)

Input: A dispatchable APSP-graph $G$
Output: A minimal dispatchable graph

1. for each pair of intersecting edges in $G$
2. if both dominate each other
3.   if neither is marked
4.     arbitrarily mark one for elimination
5.   end if
6. else if one dominates the other
7.     mark dominated edge for elimination
8. end if
9. end for
10. remove all marked edges from graph
11. return $G$
Avoiding Intermediate Graph Explosion

• **Problem:**
  - All pairs shortest path table computation consumed $O(n^2)$ space
  - Only used as an intermediate - not needed after minimal dispatchable graph obtained.

• **Solution:**
  - Interleave process of APSP construction with edge elimination.
    • Never have to build whole APSP graph.

[Tsarmardinos 1998]
Recap

- To schedule online, times must monotonically increase - use enablement conditions
- Running online allows greater flexibility to fluctuations
- However, propagation costs can be large for large graphs
- Can reduce edges by using domination to make graph smaller
To Execute a Temporal Plan

Schedule Offline

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

Detect negative loops (SSSP).

APSP + Decomposition.

STN

D Graph

[Dechter, Meiri, Pearl 91]
To Execute a Temporal Plan

Schedule Offline

1. Describe Temporal Plan

2. Test Consistency

3. Schedule Plan

4. Execute Plan

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

Observation: temporal constraints leave room to adapt.

Flexible Execution adapts through dynamic scheduling:

- Assign time to event when executed.
  - Guarantee that all constraints will be satisfied.
  - Schedule with low latency through pre-compilation.

[Muscettola, Morris, Tsmardinou KR98]
To Execute a Temporal Plan

Schedule Offline

1. Describe Temporal Plan

2. Test Consistency

3. Schedule Plan

4. Execute Plan

Schedule Online

1. Describe Temporal Plan

2. Test Consistency

3. Reformulate Plan

4. Dynamically Execute Plan

How do we schedule on line?

~ Decomposable STN

offline

online
Outline: To Execute a Temporal Plan

Schedule Online

1. Describe Temporal Plan

2. Test Consistency

3. Reformulate Plan

4. Dynamically Execute Plan

[Muscettola, Morris, Tsamardinos KR98]