Quantitative Temporal Reasoning in Planning Problems

AAAI-2004 Tutorial (MP2)

Luke Hunsberger
Vassar College
hunsberg@cs.vassar.edu

Outline

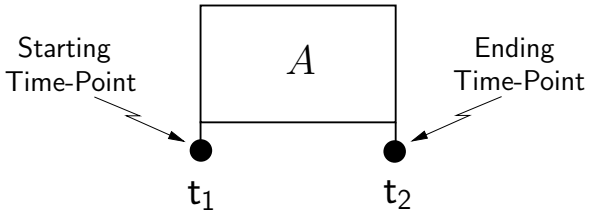
- Simple Temporal Networks
- Incremental Algs for Distance Matrix
- Collaborative Planning with STNs
- Real-time Issues
- Disjunctive Temporal Networks

Acknowledgments

- The introduction to Simple Temporal Networks draws from Dechter, Meiri and Pearl (1991).
- The section on Disjunctive Temporal Networks is based on Tsamardinos and Pollack (2003).
- Temporal Decoupling Problems and Auction Mechanisms with Temporal Constraints are discussed in my thesis (Hunsberger 2002b).

Simple Temporal Networks

Temporal Constraints on an Action



$t_1 \geq 4$ (A starts at or after 4)

$t_2 \leq 12$ (A ends at or before 12)

$3 \leq t_2 - t_1 \leq 6$ (A's dur. between 3 and 6)

Temporal Constraints on Airline Travel

Goal: Fly from Boston to Seattle:

- Leave Boston after 4 p.m. on Aug. 8;
- Return to Boston before 10 p.m., Aug. 18;
- Away from Boston no more than 7 days;
- In Seattle at least 5 days; and
- Return flight lasts no more than 7 hours.

Simple Temporal Network (STN)*

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

- $\mathcal{T}$ is a set of time-point variables: $\{t_0, t_1, \ldots, t_{n-1}\}$ and
- $\mathcal{C}$ is a set of binary constraints of the form: $t_j - t_i \leq \delta$, where δ is a real number.

* (Dechter, Meiri, & Pearl 1991)

Solutions, Consistency, Equivalence

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

  $\{t_0 = w_0,\ t_1 = w_1,\ \ldots,\ t_{n-1} = w_{n-1}\}$

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

The *Zero* Time-Point Variable

- Frequently, it is useful to fix one of the time-point variables to 0. That "variable" will often be called z.
- Binary constraints involving z are equivalent to unary constraints:

$$t_j - z \leq 5 \iff t_j \leq 5$$

$$z - t_i \leq -3 \iff t_i \geq 3$$

STN for Constrained Action

$\mathcal{T} = \{z, t_1, t_2\}$, where: $z = 0$, $t_1 =$ Start of A, $t_2 =$ End of A

$$\mathcal{C} = \left\{ \begin{array}{lll} t_2 - t_1 \leq 6 & \text{(Dur. less than 6)} \\ t_1 - t_2 \leq -3 & \text{(Dur. greater than 3)} \\ z - t_1 \leq -4 & \text{(A starts after 4)} \\ t_2 - z \leq 12 & \text{(A ends before 12)} \end{array} \right\}$$

STN for Constrained Air Travel

$\mathcal{T} = \{z, t_1, t_2, t_3, t_4\}$, where $z =$ Noon, Aug. 8.

$\mathcal{C} =$

$$\left\{ \begin{array}{lll} z - t_1 \leq -4 & \text{(Lv Bos after 4 p.m., 8/8)} \\ t_4 - z \leq 250 & \text{(Av Bos by 10 p.m., 8/18)} \\ t_4 - t_1 \leq 168 & \text{(Gone no more than 7 days)} \\ t_2 - t_3 \leq -120 & \text{(In Seattle at least 5 days)} \\ t_4 - t_3 \leq 7 & \text{(Return flight less than 7 hrs)} \end{array} \right\}$$

Graphical Representation of an STN*

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

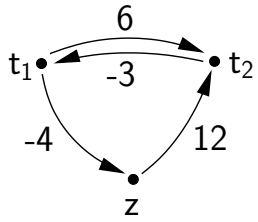
- Time-points in $\mathcal{S}$ corresp. to nodes in $\mathcal{G}$.
- Constraints in $\mathcal{C}$ correspond to edges in $\mathcal{E}$:

$t_j - t_i \leq \delta$ $\longleftrightarrow$ $t_i \bullet \xrightarrow{\delta} \bullet t_j$

* (Dechter, Meiri, & Pearl 1991)

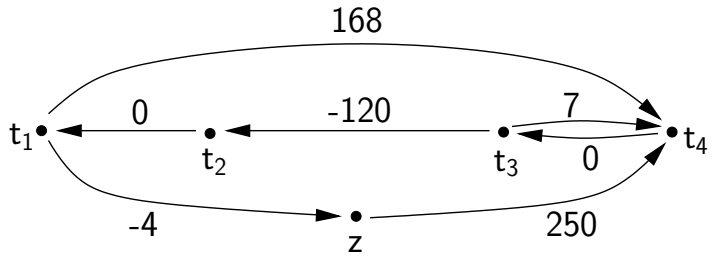
Distance Graph for Action Scenario

$$\mathcal{T} = \{z, t_1, t_2\} \qquad \mathcal{C} = \left\{ \begin{array}{rcl} t_2 - t_1 & \leq & 6 \\ t_1 - t_2 & \leq & -3 \\ z - t_1 & \leq & -4 \\ t_2 - z & \leq & 12 \end{array} \right\}$$



Distance Graph for Airline Scenario

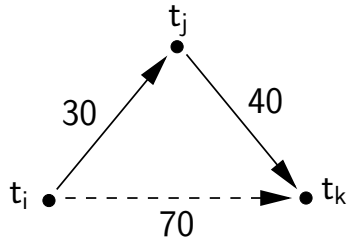
$$\left\{ \begin{array}{rcl|rcl} z - t_1 & \leq & -4, & t_4 - z & \leq & 250 \\ t_4 - t_1 & \leq & 168, & t_2 - t_3 & \leq & -120 \\ t_4 - t_3 & \leq & 7, & t_1 - t_2 & \leq & 0 \\ t_3 - t_4 & \leq & 0 & & & \end{array} \right\}$$



Implicit Constraints

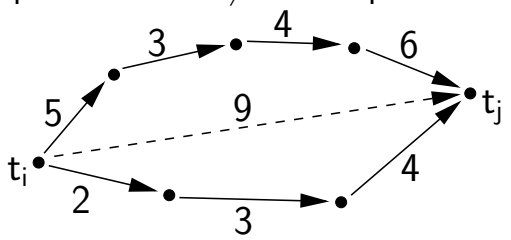
Explicit constraints in $\mathcal{C}$ can combine to form implicit constraints:

$$\begin{array}{rcl} t_j - t_i & \leq & 30 \\ t_k - t_j & \leq & 40 \\ \hline t_k - t_i & \leq & 70 \end{array}$$



Implicit Constraints as Paths

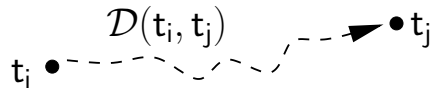
- Chains of implicit constraints in an STN correspond to *paths* in its Distance Graph.
- *Stronger/strongest* implicit constraints correspond to *shorter/shortest* paths.



Distance Matrix *

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

$$\mathcal{D}(t_i, t_j) = \text{Length of Shortest Path from } t_i \text{ to } t_j \text{ in the Distance Graph for } \mathcal{S}$$

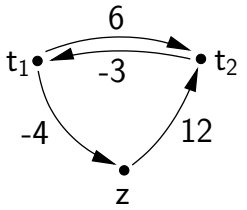


(Dechter, Meiri, & Pearl 1991)

Distance Matrix (cont'd.)

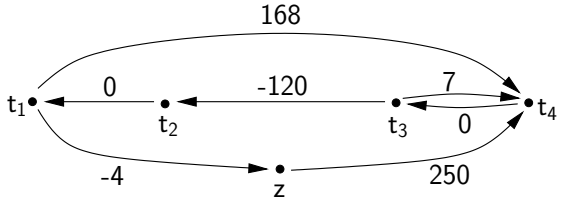
- The strongest implicit constraint on t_i and t_j in $\mathcal{S}$ is: $t_j - t_i \leq \mathcal{D}(t_i, t_j)$
- Abuse of notation: $\mathcal{D}(i, j)$ instead of $\mathcal{D}(t_i, t_j)$
- $\mathcal{D}$ is the *All-Pairs, Shortest-Path* Matrix for the Distance Graph (Cormen, Leiserson, & Rivest 1990).

Distance Matrix for Action Scenario



$\mathcal{D}$	z	t_1	t_2
z	0	9	12
t_1	-4	0	6
t_2	-7	-3	0

Distance Matrix for Airline Scenario



$\mathcal{D}$	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

Checking Consistency of an STN

Given an STN $\mathcal{S}$ with Distance Graph $\mathcal{G}$ and Distance Matrix $\mathcal{D}$, the following are equivalent (Dechter, Meiri, & Pearl 1991):

- $\mathcal{S}$ is consistent.
- Each loop in $\mathcal{G}$ has path length ≥ 0.
- The main diagonal of $\mathcal{D}$ contains only 0s.

Computing $\mathcal{D}$ from Scratch

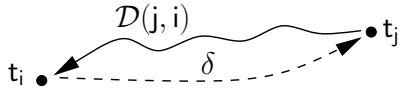
Polynomial algorithms for computing the All-Pairs, Shortest-Path Matrix (Cormen, Leiserson, & Rivest 1990):

- Floyd-Warshall Algorithm: $\mathcal{O}(n^3)$
- Johnson's Algorithm: $\mathcal{O}(n^2 \log n + nm)$

Adding Constraint to Consistent STN

- Given: $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, a consistent STN.
- Adding the new constraint, $t_j - t_i \leq \delta$, to $\mathcal{S}$ will maintain the consistency of $\mathcal{S}$ iff:

$$-\mathcal{D}(j,i) \leq \delta \quad \text{(i.e., } 0 \leq \mathcal{D}(j,i) + \delta\text{)}.$$

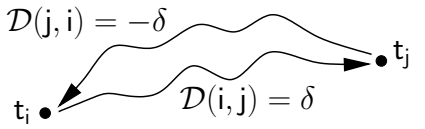


Note: This result is stated in different forms by many authors (Dechter, Meiri, & Pearl 1991; Demetrescu & Italiano 2002; Tsamardinos & Pollack 2003; Hunsberger 2003; Rohnert 1985).

Rigidly Connected Time-Points

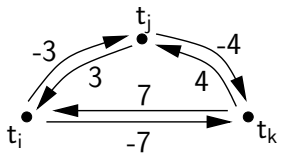
For consistent STNs, the following are equivalent:

- $(t_j - t_i) = \delta$.
- $\mathcal{D}(i,j) + \mathcal{D}(j,i) = 0$
- t_i and t_j belong to a loop of path-length 0.



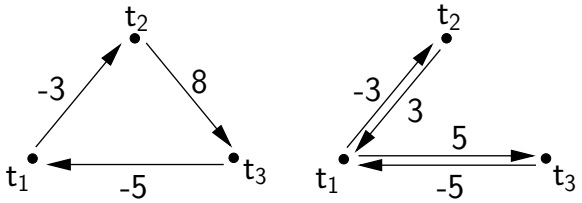
Rigidly Connected Time-Points (ctd.)

- t_i and t_j are said to be *rigidly connected* if $\mathcal{D}(i,j) = -\mathcal{D}(j,i)$.
- A set of time-points that are pairwise rigidly connected form a *rigid component*.



Note: Many authors consider rigidly connected time-points and rigid components (Tsamardinos, Muscettola, & Morris 1998; Gerevini, Perini, & Ricci 1996; Wetprasit & Sattar 1998).

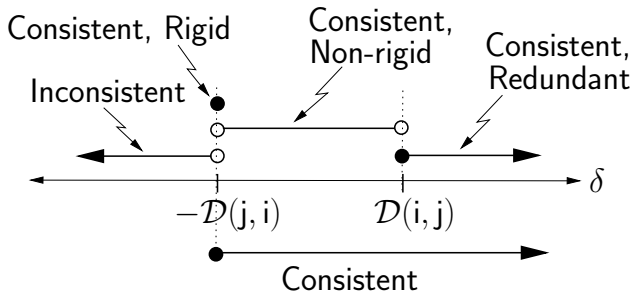
Examples of Rigid Components



Cyclical representation requires the fewest edges (Hunsberger 2002b).

Adding Constraints to Consistent STNs

Result of adding the constraint, $t_j - t_i \leq \delta$:



Rohnert (1985) distinguishes most of these cases.

Finding a Solution to an STN*

While some time-points in are *not* rigid with z,

Pick some t_i not rigidly connected to z.

Pick some $\delta \in [-\mathcal{D}(t_i, z),\ \mathcal{D}(z, t_i)]$.

Add the constraint, $t_i = \delta$

(i.e., $t_i - z \leq \delta$ and $z - t_i \leq -\delta$).

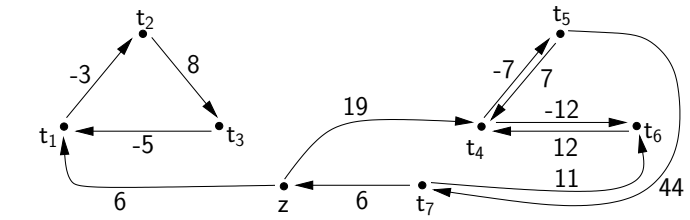
This alg derives from Dechter et al. (1991).

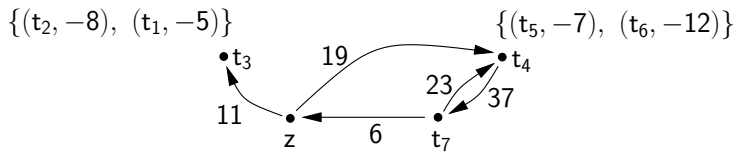
Collapsing Rigid Components

- Select one time-point from each rigid component to serve as its *representative*
- Re-orient edges involving non-representative members of rigid components
- Associate additional information with each representative sufficient to enable reconstruction of its rigid component

(Tsamardinos, Muscettola, & Morris 1998; Gerevini, Perini, & Ricci 1996; Wetprasit & Sattar 1998).

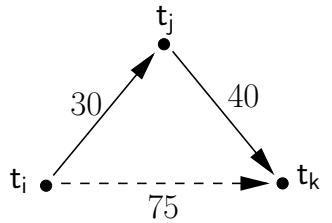
Collapsing Rigid Components: Example





Dominated Constraints

An explicit constraint, c: $t_j - t_i \leq \delta$, in an STN $\mathcal{S}$ is said to be *dominated* in $\mathcal{S}$ if *removing* c from $\mathcal{S}$ would result in no change to the distance matrix $\mathcal{D}$.

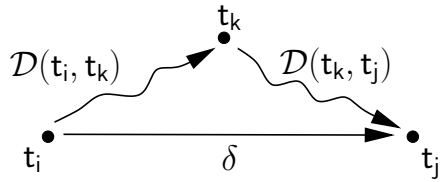


Note: Tsamardinos (2000) defines a different notion of dominance.

Dominated Constraints (cont'd.)

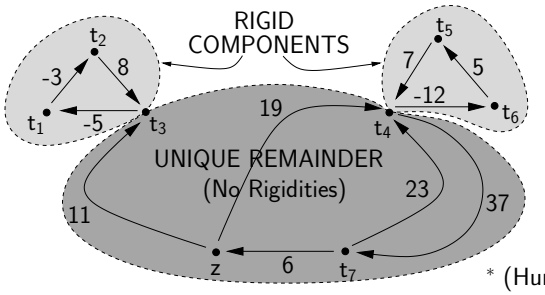
If $\mathcal{S}$ is consistent and has *no* rigid components then:

- If $\mathcal{D}(i,j) < \delta$, then c is dominated in $\mathcal{S}$.
- If $\mathcal{D}(i,j) = \delta$, then c is dominated in $\mathcal{S}$ iff there is some time-point $t_k \in \mathcal{T}$ such that: $\delta = \mathcal{D}(i,k) + \mathcal{D}(k,j)$.



Canonical Form of an STN *

- Convert rigid components to cyclical form.
- Remove all *dominated* edges from the (unique) non-rigid remainder of the STN.



* (Hunsberger 2002b)

Incremental Algs for Distance Matrix

Computing Dist. Matrix Incrementally

- Incremental algorithms compute changes resulting from adding a single constraint.
- A naïve incremental algorithm can compute such changes in $\mathcal{O}(n^2)$ time.
- Better incremental algorithms based on constraint propagation—still $\mathcal{O}(n^2)$.

Adding a Constraint to Consistent STN

Given: New constraint c: $\mathsf{t_j} - \mathsf{t_i} \leq \delta$.

- Case 1: $\delta < -\mathcal{D}(\mathsf{j},\mathsf{i})$. — Inconsistent!
- Case 2: $\delta \geq \mathcal{D}(\mathsf{i},\mathsf{j})$. — Redundant!
- Case 3: $\delta \in [-\mathcal{D}(\mathsf{j},\mathsf{i}),\ \mathcal{D}(\mathsf{i},\mathsf{j}))$.

  — Adding c would require updating $\mathcal{D}$.

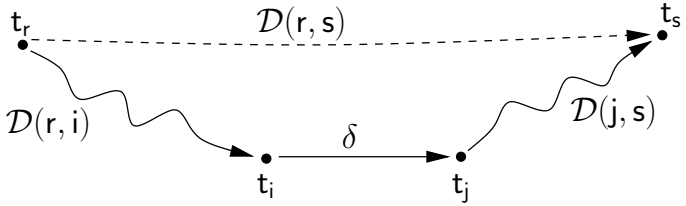
$\Rightarrow$ Incremental algorithms focus on Case 3.

Naïve Incremental Algorithm

For each entry, $\mathcal{D}(r, s)$,

If $\mathcal{D}(r, i) + \delta + \mathcal{D}(j, s) < \mathcal{D}(r, s)$, then set

$$\mathcal{D}(r, s) = \mathcal{D}(r, i) + \delta + \mathcal{D}(j, s).$$



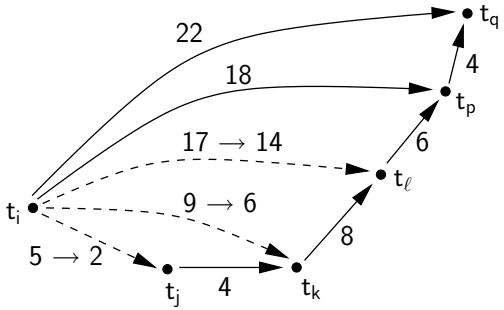
.

Constraint Propagation Algorithm*

- Propagate updates to $\mathcal{D}$ along edges in graph.
- Only propagate along *tight* edges.
  (Note: $t_s - t_r \leq \delta$ is tight iff $\mathcal{D}(r, s) = \delta$.)
- Phase I: prop. forward; Phase II: prop. bkwd.
- Checks no more than $k*\Delta$ cells of $\mathcal{D}$, where:
  Δ = number of cells needing updating; and
  k = max num edges incident on any node.

* This algorithm is based on the work of several authors (Rohnert 1985; Even & Gazit 1985; Ramalingam & Reps 1996).

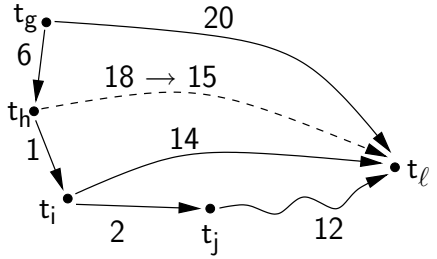
Propagating Forward



Adding the edge, $t_j - t_i \leq 2$, requires updating $\mathcal{D}(i, j)$, $\mathcal{D}(i, k)$ and $\mathcal{D}(i, \ell)$, but not $\mathcal{D}(i, p)$.

Propagating Backward

For each t_ℓ such that $\mathcal{D}(i, \ell)$ changed during Forward Propagation, propagate backward from t_i:



Here, $\mathcal{D}(h, \ell)$ needs updating, but not $\mathcal{D}(g, \ell)$.

Improvements to Incremental Alg.

- Maintain canonical form of STN.
- Only update $\mathcal{D}$ for non-rigid portion of STN.
- Propagate only along *undominated* edges.
- Case 3.1: $\delta > -\mathcal{D}(j, i)$. (No new rigidities)
- Case 3.2: $\delta = -\mathcal{D}(j, i)$. (New rigidity(ies))

The Gory Details – Case 3.1

Inputs to $\mathbf{Prop}_{3.1}$:

$\mathcal{S} = (\mathcal{T}, \mathcal{C}^u)$, an STN with only *undominated* constraints.

$\mathcal{D}$, the distance matrix for $\mathcal{S}$ (an array).

For each $t_r \in \mathcal{T}$, $\text{Succs}(t_r) = \{(t_s - t_r \leq \delta_{rs}) \in \mathcal{C}^u\}$ (a hash-table).

For each $t_r \in \mathcal{T}$, $\text{Precs}(t_r) = \{(t_r - t_q \leq \delta_{qr}) \in \mathcal{C}^u\}$ (a hash-table).

AffectedTPs, an empty hash-table.

EncounteredTPs, an empty hash-table.

$(t_j - t_i \leq \delta)$, a new constraint where: $-\mathcal{D}(j, i) < \delta < \mathcal{D}(i, j)$.

Note: This algorithm most closely resembles that of Ramalingam and Reps (1996).

The Gory Details – Case 3.1 (cont'd.)

$\mathbf{Prop}_{3.1}()$

Set: $\mathcal{D}(i, j) = \delta$.

Insert t_j into *AffectedTPs*.

PropFwd(t_j), which adds time-points to *AffectedTPs*.

For each $t_v \in$ *AffectedTPs*,

Clear *EncounteredTPs* hash-table.

PropBkwd(t_i, t_v).

The Gory Details — Case 3.1 (cont'd.)

PropFwd(t_y), where a path from t_i to t_y has already been processed and $\mathcal{D}(i, y)$ has been updated to the value $\delta + \mathcal{D}(j, y)$.

For each $t_z \in \text{Succs}(t_y)$,

If $t_z \notin$ *EncounteredTPs*,

Insert t_z into *EncounteredTPs*

If $\mathcal{D}(j, y) + \delta_{yz} = \mathcal{D}(j, z)$,

If $\delta + \mathcal{D}(j, y) + \delta_{yz} \leq \mathcal{D}(i, z)$,

Remove t_z from $\text{Succs}(t_i)$ (if in there)

Remove t_i from $\text{Precs}(t_z)$ (if in there)

If $\delta + \mathcal{D}(j, y) + \delta_{yz} < \mathcal{D}(i, z)$,

Set: $\mathcal{D}(i, z) = \delta + \mathcal{D}(j, y) + \delta_{yz}$

Insert t_z into *AffectedTPs*

PropFwd(t_z).

The Gory Details — Case 3.1 (cont'd.)

$\mathbf{PropBkwd}(t_s, t_v)$, where a path from t_s to t_v has already been processed and $\mathcal{D}(s, v)$ has been updated to the value $\mathcal{D}(s, i) + \delta + \mathcal{D}(j, v)$.

For each $t_r \in \text{Precs}(t_s)$,

If $t_r \in$ *EncounteredTPs*,

Insert t_r into *EncounteredTPs*

If $\delta_{rs} + \mathcal{D}(s, i) = \mathcal{D}(r, i)$,

If $\delta_{rs} + \mathcal{D}(s, i) + \mathcal{D}(i, v) \leq \mathcal{D}(r, v)$,

Remove t_r from $\text{Precs}(t_v)$ (if in there)

Remove t_v from $\text{Succs}(t_r)$ (if in there)

If $\delta_{rs} + \mathcal{D}(s, i) + \mathcal{D}(i, v) < \mathcal{D}(r, v)$,

Set: $\mathcal{D}(r, v) = \delta_{rs} + \mathcal{D}(s, i) + \mathcal{D}(i, v)$

$\text{PropBkwd}(t_r, t_v)$

Case 3.2: Creating New Rigidity

Adding constraint, $t_j - t_i \leq -\mathcal{D}(j, i)$.

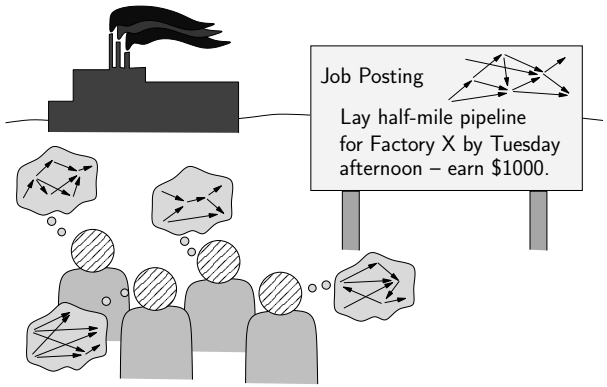
- Determine newly rigid time-points.
- Collapse new rigid component down to two points, using t_i as rep. for incoming edges and t_j as rep. for outgoing edges.
- Update set $\mathcal{C}^u$ of undominated constraints.
- Run $\text{Prop}_{3.1}$ algorithm.
- Collapse t_i and t_j into a single point.

Further Reading

- Demetrescu and Italiano (2001; 2002) consider special cases where each edge can assume a bounded number of values; or where all edge weights are non-negative.
- Ramalingham and Reps (1996) introduce *incremental complexity analysis*.
- Zaroliagis (to appear) discusses incremental and *decremental* algorithms.

Collaborative Planning with STNs

Initial-Commitment Decision Prob.*

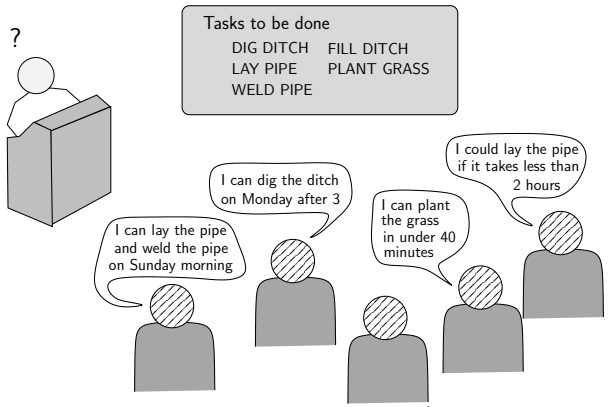


* ICDP (Hunsberger & Grosz 2000; Hunsberger 2002b)

The ICDP – in Words

- A group of agents, each with pre-existing commitments subject to temporal constraints
- A new opportunity for group action (a set of tasks also subject to temporal constraints)
- Agents must reason locally and globally about whether to commit (alone and together) to the proposed action.

ICDP Mech. using Combin'l. Auction*



* (Hunsberger & Grosz 2000; Hunsberger 2002b)

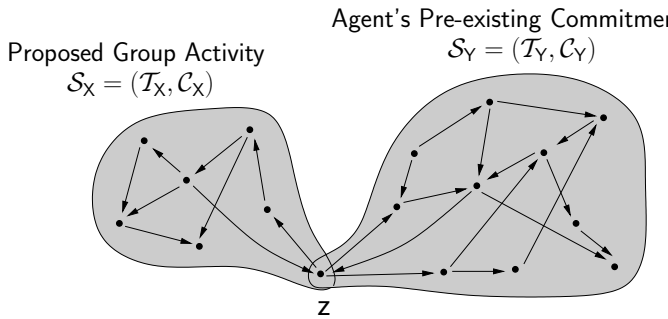
ICDP Mechanism – in Words

- Agents (reasoning locally) bid on subsets of tasks in group activity: a *combinatorial auction* (Rassenti, Smith, & Bulfin 1982).
- Agents include temporal constraints in their bids to protect their pre-existing commitments.*
- Global goal: find an *awardable* set of bids (each task covered by some bid; temporal constraints in bids jointly satisfiable).

Problems to Solve re: ICDP

- Bid Generation:
  Select tasks and generate protective temporal constraints
- Winner Determination:
  Find an awardable set of bids.
- Post-Auction Coordination:
  Deal with temporal dependencies among tasks being done by different agents without requiring excessive communication overhead.

Bid Generation using STNs

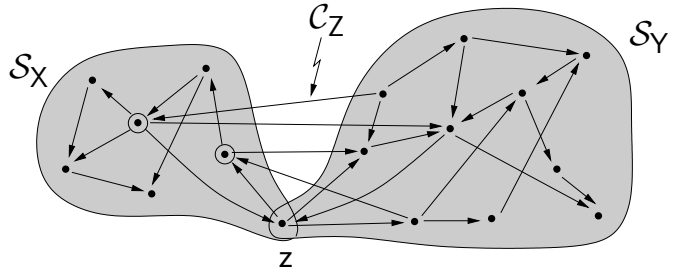


$\mathcal{S}_Z = (\mathcal{T}_Z, \mathcal{C}_Z) = (\mathcal{T}_X \cup \mathcal{T}_Y,\ \mathcal{C}_X \cup \mathcal{C}_Y)$ is consistent

if $\mathcal{S}_X$ and $\mathcal{S}_Y$ are (since they only share z).

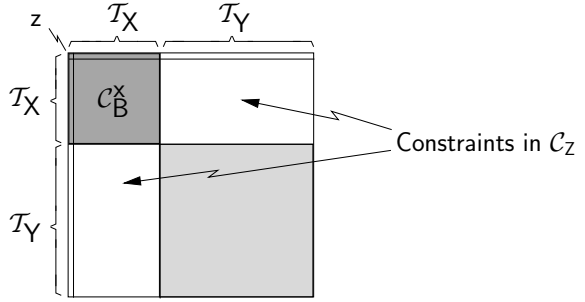
Bid Generation using STNs (cont'd.)

$\mathcal{S}_B = (\mathcal{T}_X \cup \mathcal{T}_Y,\ \mathcal{C}_X \cup \mathcal{C}_Y \cup \mathcal{C}_Z)$ includes additional constraints, $\mathcal{C}_Z$, to ensure that tasks done by the agent do not overlap.



Bid Generation using STNs (cont'd.)

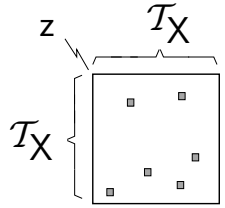
$\mathcal{D}_B$: The distance matrix for $\mathcal{S}_B$



$\mathcal{C}_B^x = \{t_j - t_i \leq \mathcal{D}_B(i,j) \mid t_i, t_j \in \mathcal{T}_X\}$ would suffice (in bid) to protect agent's pre-existing commitments.

Bid Generation using STNs (cont'd.)

. . . but necessary to include only edges in *canonical form* of $(\mathcal{T}_X, \mathcal{C}_B^x)$ that are *stronger* than the corresponding edges in $\mathcal{S}_X = (\mathcal{T}_X, \mathcal{C}_X)$ (i.e., edges for which $\mathcal{D}_B(i,j) < \mathcal{D}_X(i,j)$. (Hunsberger 2001)



Winner Determination *

- Modify existing WD algorithm (Sandholm 2002) to accommodate temporal constraints.
- Depth-first search in space of partial bid-sets
- Maintain STN, $(\mathcal{T}_X,\ \mathcal{C}_X \cup \mathcal{C}_\mathcal{B})$, containing constraints from proposed activity plus those from bids currently being considered.
- Backtrack if this STN becomes inconsistent.

* (Hunsberger & Grosz 2000)

Post-Auction Coordination

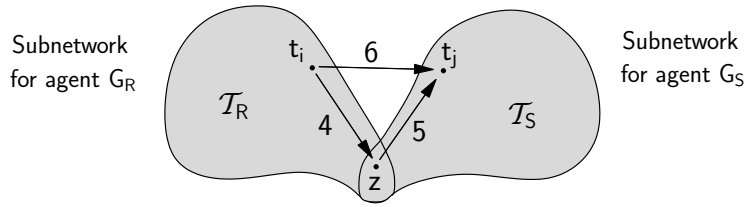
- Auction yields viable allocation of tasks, but typically results in temporal dependencies among tasks being done by different agents.
- Solution 1: *Temporally decouple* the task-sets being done by different agents (adds constraints, but no need for subsequent coord'n.).
- Solution 2: *Relative Temporal Decoupling* (weaker constraints, but requires some subsequent coordination).

Temporal Decoupling (TD)*

- Goal: Enable agents to operate independently —and hence without communication.
- Method: Add new constraints to ensure *mergeable solutions property*.
- Will focus on two-agent case, but works for arbitrarily many agents.

(Hunsberger 2002a; 2002b)

Typical Case for TD Problem



- Edge from t_i to t_j not dominated by a path through z.
- Can fix by strengthening edge from t_i to z, or edge from z to t_j, or both.

TD Algorithm*

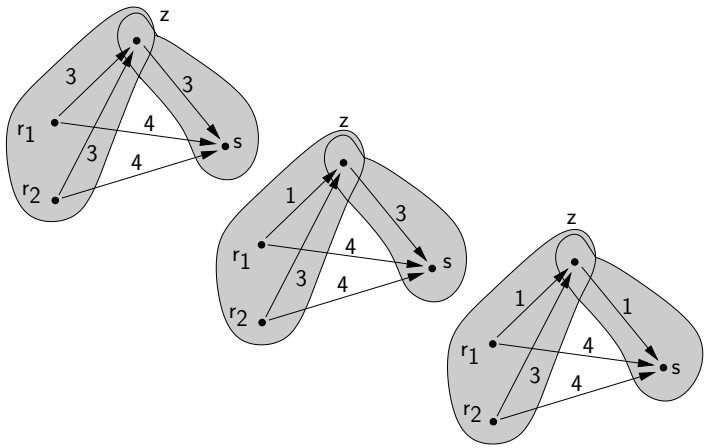
- Add *intra*-subnetwork constraints to ensure that each tight, proper, *inter*-subnetwork constraint is dominated by a path through z.
- Requires processing each such edge only once.
- Afterward, no matter how each agent tightens constraints in its own subnetwork, all inter-subnetwork constraints will be satisfied.

(Hunsberger 2002b)

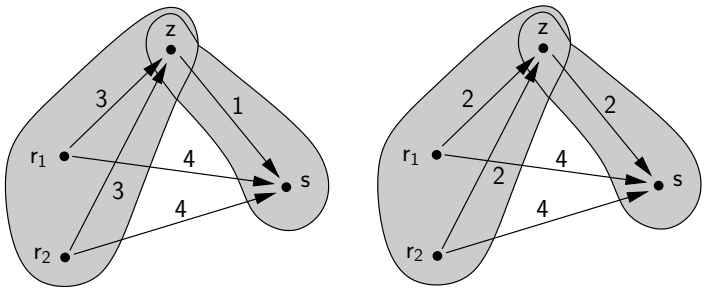
Improvements to TD Algorithm*

- When selecting inter-subnetwork edges to work on, and when deciding how much to tighten each intra-subnetwork edge, use heuristics to increase flexibility in resultant decoupled subnetworks.
- Use *Iterative Weakening* algorithm to ensure *minimal* temporal decoupling (i.e., one in which any further weakening would foil the decoupling).

Generating a Non-Minimal Decoupling

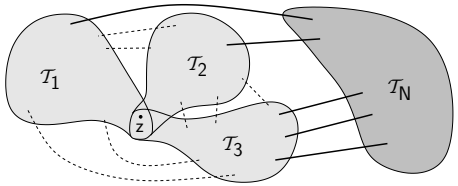


Alternative Minimal Decouplings



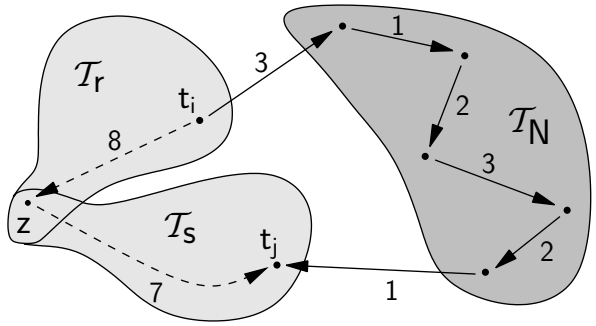
Relative Temporal Decoupling (RTD)*

- Goal: Use weaker constraints, but allow some inter-subnetwork dependence to remain.
- Method: Given N subnetworks, (N-1) are fully decoupled; but N^{th} dependent on the rest.



(Hunsberger 2003; 2002b)

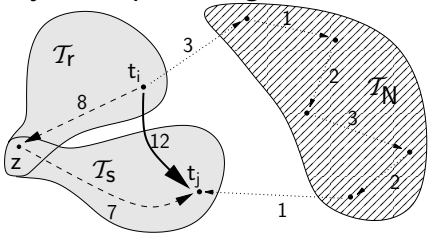
Typical Case for RTD Problem



Inter-subnetwork path from t_i to t_j is not dominated by path through z.

RTD Algorithm*

(1) Replace each *tight*, *proper*, inter-subnetwork path by an explicit edge.



(2) Run TD algorithm ignoring N^{th} subnetwork.

(Hunsberger 2003; 2002b)

Lambda Bounds for RTD*

- After RTD, agent controlling Nth subnetwork is dependent on the rest.
- Must not re-introduce any inter-subnetwork paths that would threaten the RTD. (Requirements captured in *Lambda Bounds*.)
- Unlike other agents, Nth agent may add edges linking Nth subnetwork with other subnetworks.

(Hunsberger 2003; 2002b)

Other Applications, Etc.

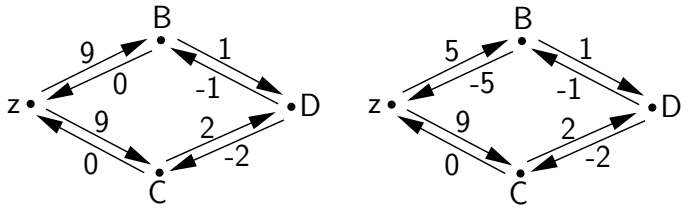
- Submitting a bid imposes restrictions on the bidder that are precisely captured by the Lambda Bounds (where N = 2).
- The RTD algorithm may be recursively applied yielding an arbitrarily complex hierarchy of dependence and independence.
- Hadad et al. (2001) present an alternative approach to temporal reasoning in the context of collaboration.

Real-time Issues

Executing a Temporal Network

- To *execute* a time-point means to assign that time-point to the current moment.
- Goal: Maintain consistency of network while executing its time-points.
- Challenges:
  Decisions must be made in real-time.
  Updating $\mathcal{D}$ takes time.

A Sample Execution*



After executing B at time 5, it turns out that C must be executed at time 4 (which is already past).

* (Muscettola, Morris, & Tsamardinos 1998)

Greedy Dispatcher*

While some time-points not yet executed:

Wait until some time-point is executable.

If more than one, pick one to execute.

Propagate updates only to *neighboring* time-points (i.e., do not fully update $\mathcal{D}$).

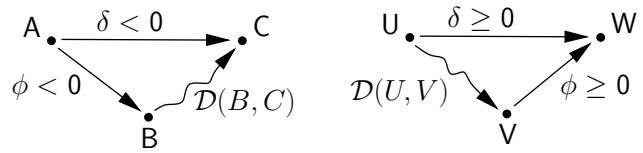
* (Muscettola, Morris, & Tsamardinos 1998)

Dispatchability*

- An STN that is guaranteed to be satisfied by the Greedy Dispatcher is called *dispatchable*.
- Any *consistent* STN can be transformed into an equivalent *dispatchable* STN.
- Step I: The corresponding All-Pairs graph is equivalent and dispatchable.
- Step II: Remove *lower- and upper-dominated* edges (does not affect dispatchability).

* (Muscettola, Morris, & Tsamardinos 1998).

Lower and Upper Dominance*



- The *negative edge* AC is *lower-dominated* if: $\delta = \phi + \mathcal{D}(B,C)$.
- The *non-negative edge* UW is *upper-domin'd* if: $\delta = \mathcal{D}(U,V) + \phi$.

* (Muscettola, Morris, & Tsamardinos 1998)

Controllability Issues*

- In real-world applications, an agent may only control some time-points directly; others may be controlled by other agents or Nature.
- Such a network is called *controllable* if there exists a strategy for the agent to execute the time-points under its direct control that will ensure the consistency of the network—no matter how the other agents or Nature execute their time-points.

* (Vidal & Ghallab 1995; Vidal & Fargier)

Varieties of Controllability*

- Weak Controllability: Presumes agent gets notified in advance of all choices to be made by others.
- Strong Controllability: Presumes agent may not get *any* information about choices made by others.
- Dynamic Controllability: Agent makes its choices and learns about the choices of others in real-time.

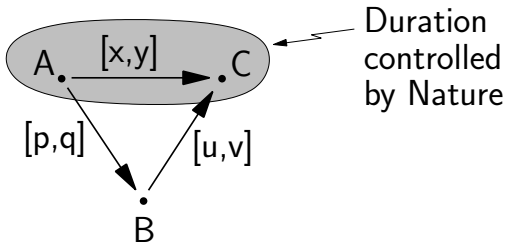
* (Vidal & Ghallab 1995; Vidal & Fargier)

Checking Dynamic Controllability*

Morris et al. (2001) present a sound and complete algorithm for checking dynamic controllability using:

- *Triangular Reductions*
- *Wait Propagation*

Triangular Reductions



- If v < 0, then B follows C – no reduction.
- If u ≥ 0, then B precedes C. Must tighten bounds on interval AB to: [y-v, x-u].

Triangular Reductions (ctd.)

- If u < 0 and v ≥ 0, then the order of B and C is not yet determined. Derive a *WAIT*: If C has not yet been executed, B must *wait* to be executed until (y-v) after A.

Waits can be propagated much like binary constraints.

Distributing Control Among a Group

- Temporal Decoupling Algorithms enable control over an STN to be distributed among a group.
- Prior research on dynamic controllability assumes only one agent (plus Nature).
- The Nth agent in an RTD is in a role similar to that of the single agent in dynamic controllability scenarios.
- Future research aimed at merging multi-agent and real-time issues (Hunsberger 2003).

Disjunctive Temporal Networks

Disjunctive Temporal Network (DTN)*

A DTN is a pair, $(\mathcal{T}, \mathcal{C})$, where:

- $\mathcal{T}$ is a set of time-point variables; and
- $\mathcal{C}$ is a set of constraints among those variables, each constraint $C_i \in \mathcal{C}$ having the form:

$$c_{i1} \vee c_{i2} \vee \ldots \vee c_{in}$$

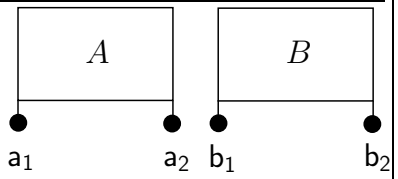
where, in turn, each c_{ij} has the form:

$$y - x \leq \delta \quad \text{where : } x, y \in \mathcal{T} \text{ and } \delta \in \mathbf{R}$$

* This section is based on Tsamardinos & Pollack (2003)

Example of a DTN

Actions A and B must be completed between times 5 and 25 and must not overlap.



- $\mathcal{T} = \{z, a_1, a_2, b_1, b_2\}$
- $\mathcal{C} = \left\{ \begin{array}{ll} z - a_1 \leq -5, & z - b_1 \leq -5 \\ a_2 - z \leq 25, & b_2 - z \leq 25 \\ a_2 - z \leq b_1 \ \vee & b_2 - z \leq a_1 \end{array} \right\}$

Component STNs for a DTN

- Given a DTN, $(\mathcal{T}, \mathcal{C})$, a *component STN* is found by selecting a *single* disjunct c_{ij} from each constraint in $\mathcal{C}$.
- A DTN is consistent if and only if it has at least one consistent component STN.
- Standard Approach: Solve DTN by finding one of its consistent component STNs.

Solving DTN using Meta-CSP

Given a DTN, $(\mathcal{T}, \mathcal{C})$, construct a (meta) Constraint Satisfaction Problem (meta-CSP):

Variables	One variable v_i for each $C_i \in \mathcal{C}$
Values	Each v_i may be assigned one of the disjuncts c_{ij} from corresponding $C_i \in \mathcal{C}$
Constraint	Current assignments are satisfactory iff they correspond to consistent STN
Solution	Satisfactory & complete set of assts.

Meta-CSP for Sample DTN

$\mathcal{T} = \{z, a_1, a_2, b_1, b_2\}$

Variables	Domains
v_1	$\{(z - a_1 \leq -5)\}$
v_2	$\{(z - b_1 \leq -5)\}$
v_3	$\{(a_2 - z \leq b_1), (b_2 - z \leq a_1)\}$
v_4	$\{(a_2 - z \leq 25)\}$
v_5	$\{(b_2 - z \leq 25)\}$

Forward Checking

- Use incremental algorithm to maintain distance matrix for STN corresponding to current assignment.
- Use *negative-transpose criterion* for the checks:

  $y - x \leq \delta$ will introduce loop with negative path-length iff $\delta < -\mathcal{D}(y, x)$

Tsamardinos & Pollack (2003) refer to the Negative-Transpose Criterion as the Forward-Check (FC) Condition.

Conflict-Directed Backtracking*

- Forward checking removes disjuncts that would create loops having negative path-length.
- Keep track of variables that contributed edges (i.e., disjuncts) to such loops.
- Use *Predecessor Matrix*: $\mathcal{P}(x, y) =$ immediate predecessor of y in a shortest path from x to y.
- Subsequent backtracking can skip over variables that did not contribute to such loops.

* Tsamardinos & Pollack (2003) cite Stergiou & Koubarakis (2000).

Removal of Subsumed Variables*

Let $\mathcal{S}$ be the STN corresponding to the current assignment of values (i.e., disjuncts) to variables.

- Subsumed Values: A disjunct, $y - x \leq \delta$, is *subsumed* by an STN iff $\mathcal{D}(x, y) \leq \delta$.
- During search, currently-unassigned variables that have one or more values subsumed by $\mathcal{S}$ may be ignored.

* Tsamardinos & Pollack (2003) cite Oddi & Cesta (2000).

Semantic Branching*

If all attempts to expand a partial assignment $\mathcal{A}$ using the value $(y - x \leq \delta)$ fail, then subsequent attempts to expand $\mathcal{A}$ may assume that $x - y \leq -\delta$ holds.

* Tsamardinos & Pollack (2003) cite Armando et al. (1999).

No-Good Recording

For an arbitrary CSP, $\langle V, C \rangle$, a *no-good* is a pair, $\langle A, J \rangle$, where:

- A is a set of forbidden value assignments to a subset of V; and
- J (the justification) is a subset of V such that no solution of the CSP restricted to constraints involving only variables in J contains the assignments in A.

(Tsamardinos & Pollack 2003)

Conclusions

- STNs expressive and computationally appealing
- Many algorithms ready for implementation in planning systems
- Much work to do on extensions to include real-time and controllability issues
- DTNs increase expressiveness, but require more computational resources.

Putting it all Together

Tsamardinos & Pollack (2003) implemented all of the above features in a general-purpose DTN solver called *Epilitis*, demonstrating "a nearly two order-of-magnitude speed-up over the previously published algorithms on benchmark problems in the DTP literature."

Note: DTP = Disjunctive Temporal Problem.

References

- Armando, A.; Castellini, C.; and Giunchiglia, E. 1999. Sat-based procedures for temporal reasoning. In *Fifth European Conference on Planning (ECP-99)*, volume 1809 of *Lecture Notes in Computer Science*. Springer. 97–108.
- Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms*. Cambridge, MA: The MIT Press.
- Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.
- Demetrescu, C., and Italiano, G. F. 2001. Fully dynamic all pairs shortest paths with real edge weights. In *42nd Annual Symposium on Foundations of Computer Science (FOCS 2001)*. IEEE Computer Society. 260–267.
- Demetrescu, C., and Italiano, G. 2002. A new approach to dynamic all pairs shortest paths. Technical Report ALCOMFT-TR-02-92, ALCOM-FT.

To appear in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC'03), San Diego, California, June 2003.

- Even, S., and Gazit, H. 1985. Updating distances in dynamic graphs. *Methods of Operations Research* 49:371–387.
- Gerevini, A.; Perini, A.; and Ricci, F. 1996. Incremental algorithms for managing temporal constraints. Technical Report IRST-9605-07, IRST.
- Hadad, M.; Kraus, S.; Gal, Y.; and Lin, R. 2001. Time reasoning for a collaborative planning agent in a dynamic environtment. *?* [LH-0305].
- Hunsberger, L., and Grosz, B. J. 2000. A combinatorial auction for collaborative planning. In *Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, 151–158. IEEE Computer Society.
- Hunsberger, L. 2001. Generating bids for group-related actions in the context of prior commitments. In Meyer, J.-J. C., and Tambe, M., eds., *Intelligent Agents VIII (ATAL-01)*, volume 2333 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.
- Hunsberger, L. 2002a. Algorithms for a temporal decoupling problem in multi-agent planning. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*.
- Hunsberger, L. 2002b. *Group Decision Making and Temporal Reasoning*. Ph.D. Dissertation, Harvard University. Available as Harvard Technical Report TR-05-02.
- Hunsberger, L. 2003. Distributing the control of a temporal network among multiple agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS-03)*.
- Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. 494–499.
- Muscettola, N.; Morris, P.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*.
- Oddi, A., and Cesta, A. 2000. Incremental forward checking for the disjunctive temporal problem. In *European Conference on Artificial Intelligence*.
- Ramalingam, G., and Reps, T. 1996. On the computational complexity of dynamic graph problems. *Theoretical Computer Science* 158:233–277.
- Rassenti, S.; Smith, V.; and Bulfin, R. 1982. A combinatorial auction mechanism for airport time slot allocation. *Bell Journal of Economics* 13:402–417.
- Rohnert, H. 1985. A dynamization of the all pairs least cost path problem. In Mehlhorn, K., ed., *2nd Symposium of Theoretical Aspects of Computer Science (STACS 85)*, volume 182 of *Lecture Notes in Computer Science*. Springer. 279–286.
- Sandholm, T. 2002. An algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence* 135:1–54.
- Stergiou, K., and Koubarakis, M. 2000. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence* 120(1):81–117.
- Tsamardinos, I., and Pollack, M. E. 2003. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence* 151:43–89.
- Tsamardinos, I.; Muscettola, N.; and Morris, P. 1998. Fast transformation of temporal plans for efficient execution. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*. Cambridge, MA: The MIT Press. 254–261.
- Tsamardinos, I. 2000. Reformulating temporal plans for efficient execution. Master's thesis, University of Pittsburgh.
- Vidal, T., and Fargier, H. Contingent durations in temporal csps: from consistency to controllabilities.
- Vidal, T., and Ghallab, M. 1995. Temporal constraints in planning: Free or not free? In *CONSTRAINT-95 Workshop*.
- Wetprasit, R., and Sattar, A. 1998. Qualitative and quantitative temporal reasoning with points and durations (an extended abstract). In *Fifth International Workshop on Temporal Representation and Reasoning (TIME-98)*, 69–73.
- Zaroliagis, C. D. to appear. Implementations and experimental studies of dynamic graph algorithms. In Fleischer, R.; Moret, B.; and Meineche-Schmidt, E., eds., *Experimental Algorithmics—The State of the Art*. Springer-Verlag. 229–278.