Executing Reactive, Model-based Programs through Graph-based Temporal Planning

Phil Kim and Brian C. Williams,
Artificial Intelligence and Space Systems Labs
Massachusetts Institute of Technology

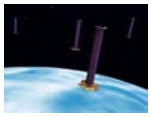
Mark Abramson
Draper Labs

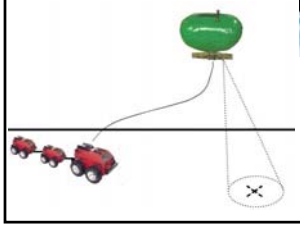
Outline

- Cooperative Vehicle Missions
- Model-based Programming
- Reactive Model-based Programming Language (RMPL)
- Temporal Plan Networks (TPN)
- Activity Planning (Kirk)
- Optional: Hybrid Activity/Path Planning

Cooperative Mars Exploration

How do we coordinate heterogeneous teams of orbiters, rovers and air vehicles to perform globally optimal science exploration?





MIT Cooperative Vehicle Testbed

- Distributed Satellites: Spheres, Spheres, TechSat21

MIT Cooperative Vehicle Testbed

- Distributed Satellites: Spheres, Spheres, TechSat21
- Aerobots: Indoor blimps

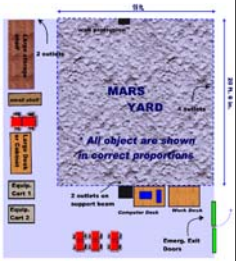
MIT Cooperative Vehicle Testbed

- Distributed Satellites: Spheres, Spheres, TechSat21
- Aerobots: Indoor blimps
- Sensing: distributed, wireless sensor net

MIT Cooperative Vehicle Testbed

- Distributed Satellites: Spheres, Spheres, TechSat21
- Aerobots: Indoor blimps
- Sensing: distributed, wireless sensor net
- Rovers: 1 ATRV Sr., 3 ATRV Jr

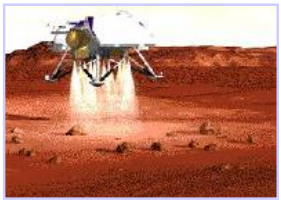





Outline

- Cooperative Vehicle Missions
- Model-based Programming
- Reactive Model-based Programming Language (RMPL)
- Temporal Plan Networks (TPN)
- Activity Planning (Kirk)
- Optional: Hybrid Activity/Path Planning

Why Model-based Programming?



Mars 98:
- Climate Orbiter
- Mars Polar Lander

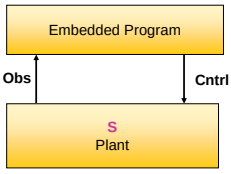
Leading Diagnosis:

- Legs deployed during descent.
- Noise spike on leg sensors latched by monitors.
- Laser altimeter registers 50ft.
- Begins polling leg monitors to determine touch down.
- Latched noise spike read as touchdown.
- Engine shutdown at ~50ft.

Create Embedded Languages That Reason from Commonsense Models

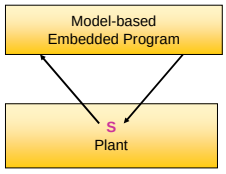
Model-based Programs Interact Directly with State

Embedded programs interact with plant sensors/actuators:
- Read sensors
- Set actuators



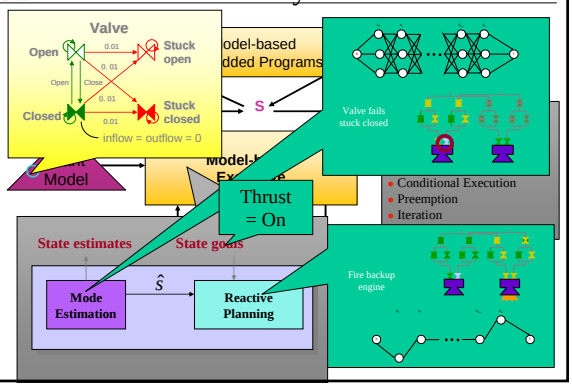
Programmer must map between state and sensors/actuators.

Model-based programs interact with plant state:
- Read state
- Write state



Model-based executive maps between sensors, actuators to states.

Reactive Model-based Programs Interact Directly with State



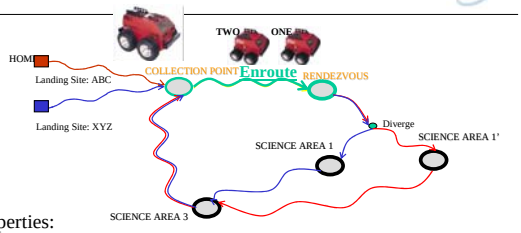
Cooperative Model-based Programming

- How do we specify the allowed behaviors of cooperative robotic networks? (RMPL)
- How do we command cooperative networks? (this talk)
- How do we monitor cooperative networks? (next talk)

Outline

- Cooperative Vehicle Missions
- Model-based Programming
- Reactive Model-based Programming Language (RMPL)
- Temporal Plan Networks (TPN)
- Activity Planning (Kirk)
- Optional: Hybrid Activity/Path Planning

Example Scenario



Properties:

- Teams are focused to a hierarchy complex strategies.
- Maneuvers are temporally coordinated.
- Novel events occur during critical phases.
- Quick response draws upon a library of contingencies.

Create Language with planner-like capabilities

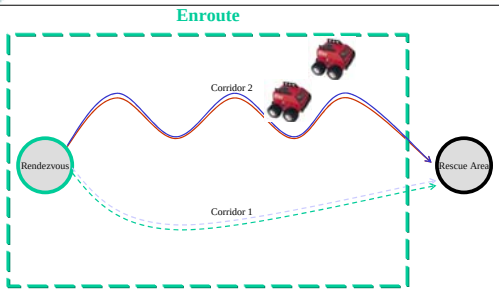
Reactive Model-based Programming

Idea: Describe team behaviors by starting with a rich concurrent, embedded programming language (RMPL,TCC, Esterel):

- c
- If c next A
- Unless c next A
- A, B
- Always A
- Sensing/actuation activities
- Conditional execution
- Preemption
- Full concurrency
- Iteration

- Add temporal constraints:
  - A [l,u]
  - Timing
- Add choice (non-deterministic or decision-theoretic):
  - Choose {A, B}
  - Contingency

Example Enroute Activity:



RMPL for Group-Enroute

```
Group-Enroute()[l,u] = {
   choose {
      do {
         Group-Traverse-
   Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH1_OK,
      do {
         Group-Traverse-
   Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH2_OK
   };
   {
      Group-Transmit(OPS,ARRIVED)[0,2],
      do {
         Group-Wait(HOLD1,HOLD2)[0,u*10%]
      } watching PROCEED
   }
}
```

RMPL for Group-Enroute

Activities:

```
Group-Enroute()[l,u] = {
   choose {
      do {
         Group-Traverse-
   Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH1_OK,
      do {
         Group-Traverse-
   Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH2_OK
   };
   {
      Group-Transmit(OPS,ARRIVED)[0,2],
      do {
         Group-Wait(HOLD1,HOLD2)[0,u*10%]
      } watching PROCEED
   }
}
```

RMPL for Group-Enroute

Conditionality and Preemption:

```
Group-Enroute()[l,u] = {
   choose {
      do {
         Group-Traverse-
   Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH1_OK,
      do {
         Group-Traverse-
   Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH2_OK
   };
   {
      Group-Transmit(OPS,ARRIVED)[0,2],
      do {
         Group-Wait(HOLD1,HOLD2)[0,u*10%]
      } watching PROCEED
   }
}
```

RMPL for Group-Enroute

Sequentiality: Concurrency:

```
Group-Enroute()[l,u] = {
   choose {
      do {
         Group-Traverse-
   Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH1_OK,
      do {
         Group-Traverse-
   Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH2_OK
   } ;
   {
      Group-Transmit(OPS,ARRIVED)[0,2],
      do {
         Group-Wait(HOLD1,HOLD2)[0,u*10%]
      } watching PROCEED
   }
}
```

RMPL for Group-Enroute

Temporal Constraints:

```
Group-Enroute()[l,u] = {
   choose {
      do {
         Group-Fly-
   Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH1_OK,
      do {
         Group-Fly-
   Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH2_OK
   };
   {
      Group-Transmit(OPS,ARRIVED)[0,2],
      do {
         Group-Wait(HOLD1,HOLD2)[0,u*10%]
      } watching PROCEED
   }
}
```

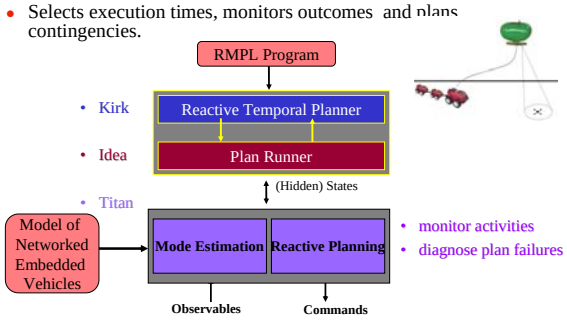
RMPL for Group-Enroute

Non-deterministic choice:

```
Group-Enroute()[l,u] = {
   choose {
      do {
         Group-Traverse-
   Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH1_OK,
      do {
         Group-Traverse-
   Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH2_OK
   };
   {
      Group-Transmit(OPS,ARRIVED)[0,2],
      do {
         Group-Wait(HOLD1,HOLD2)[0,u*10%]
      } watching PROCEED
   }
}
```

RMPL Interpreter

- Dynamically selects among alternate executions, satisfies open conditions and checks schedulability,
- Selects execution times, monitors outcomes and plans contingencies.



- How do we provide fast, temporally flexible planning?
- Graph-based planners support fast planning.
- … but plans are totally order.
- Desire flexible plans based on simple temporal networks (e.g., HSTS, Muscetola et al.).

How do we create temporally flexible plan graphs?

- Generalize simple temporal networks (temporal plan network TPN).

Kirk: Reactive Temporal Planner

Reactive Model-based Programming Language

RMPL Compiler

Temporal Plan Network (TPN) with STN

- **Represents all RMPL executions**

Reactive Temporal Planner

- **Selects schedulable execution threads of TPN**

Concurrent Plan

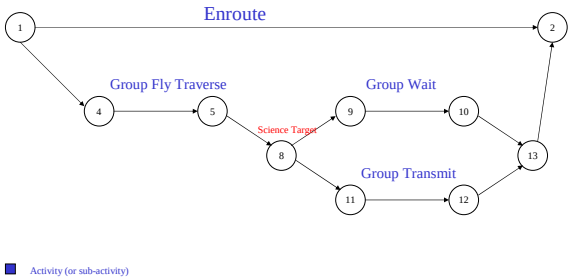
- **Plan = Execution threads related by Simple Temporal Net**

Outline

- Cooperative Vehicle Missions
- Model-based Programming
- Reactive Model-based Programming Language (RMPL)
- Temporal Plan Networks (TPN)
- Activity Planning (Kirk)
- Optional: Hybrid Activity/Path Planning

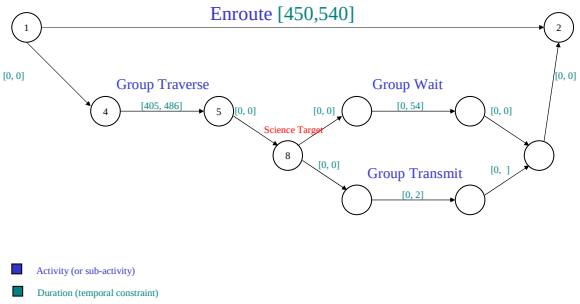
Enroute Activity:

• Start with flexible plan representation



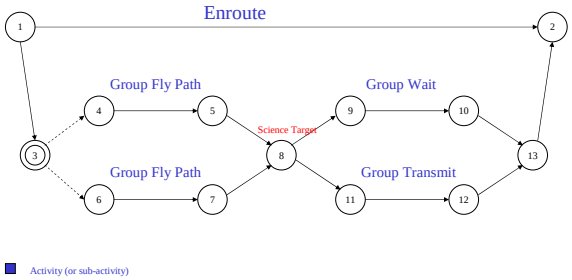
Enroute Activity:

• Start with flexible plan representation



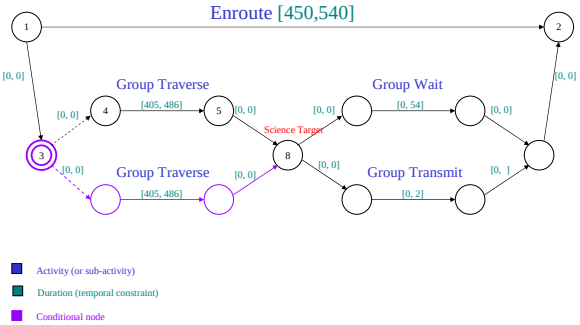
Enroute Activity:

•TPN representation of Enroute activity and sub-activities



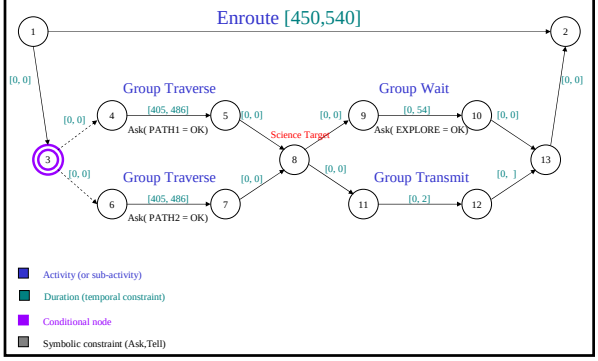
Enroute Activity:

• Add conditional nodes



Enroute Activity:

•Add temporally extended, symbolic constraints

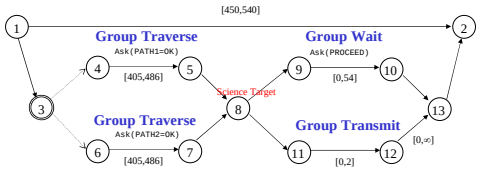


Outline

- Cooperative Vehicle Missions
- Model-based Programming
- Reactive Model-based Programming Language (RMPL)
- Temporal Plan Networks (TPN)
- Activity Planning (Kirk)
- Optional: Hybrid Activity/Path Planning

Planning Group-Enroute

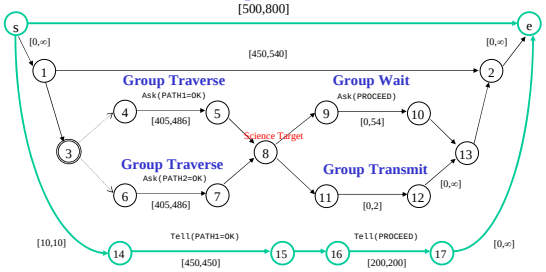
Group-Enroute



To Plan:
• Instantiate Group-Enroute

Planning Group-Enroute

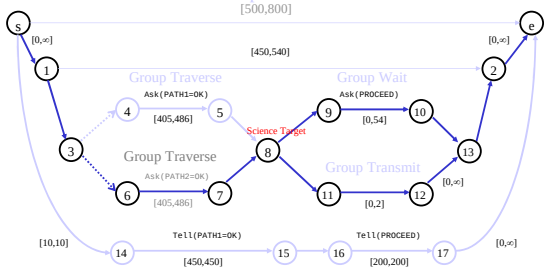
Group-Enroute
[500,800]



To Plan:
• Instantiate Group-Enroute
• Add External Constraints (Tells)

Generates Schedulable Plan

Group-Enroute
[500,800]

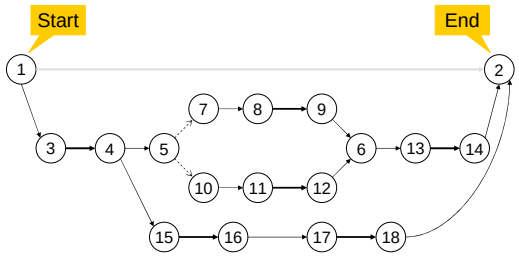


To Plan:
• Instantiate Group-Enroute
• Add External Constraints

Trace consistent trajectories
• Check Schedulability
• Satisfy and Protect Asks

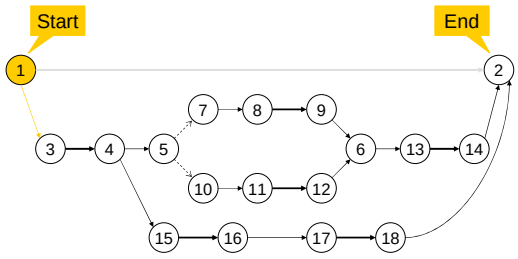
Planning Example

- Find paths from start-node to end-node



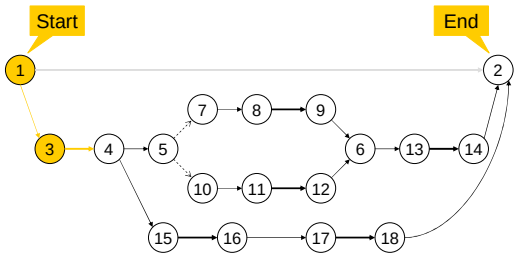
Planning Example

- Not a decision-node: Follow all outarcs



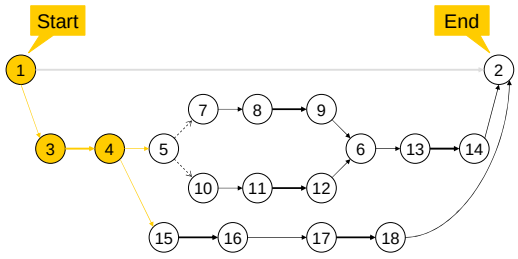
Planning Example

- Not a decision-node: Follow all outarcs



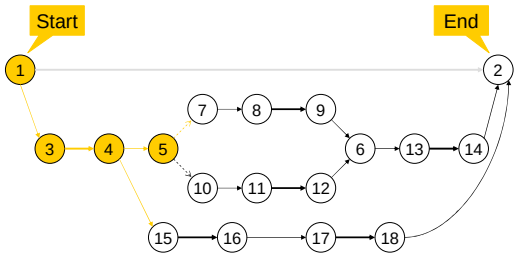
Planning Example

- Not a decision-node: Follow all outarcs



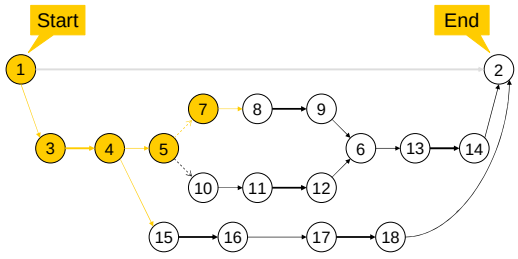
Planning Example

- Decision-node: Select a single outarc



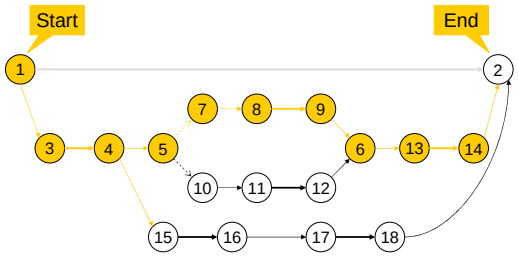
Planning Example

- Not a decision-node: Follow all outarcs



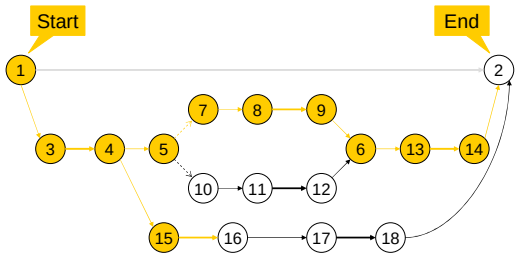
Planning Example

- Continue



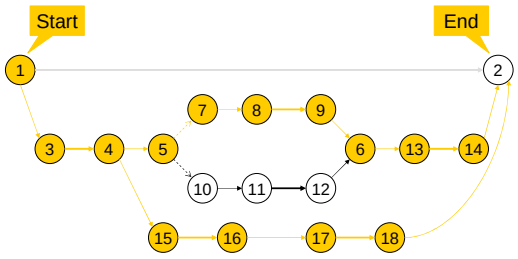
Planning Example

- Not a decision-node: Follow all outarcs

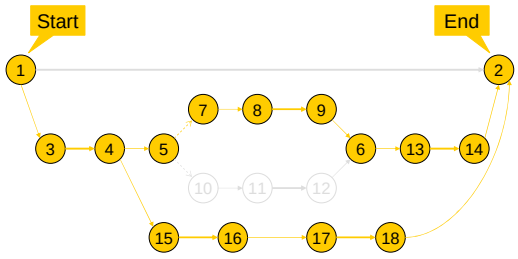


Planning Example

- Continue

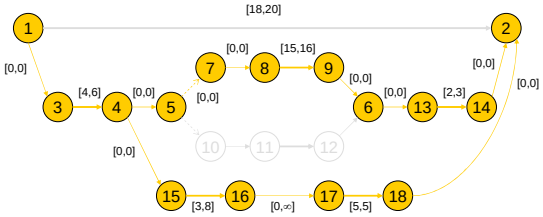


Planning Example



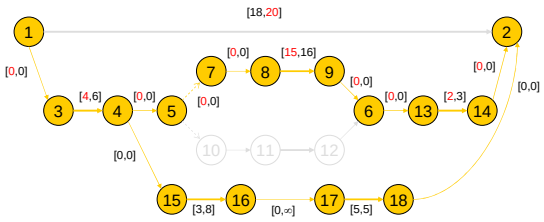
Temporal Constraint Consistency

- Don't test consistency at each step.
- Only when a path induces a cycle, check for negative cycle in the STN distance graph



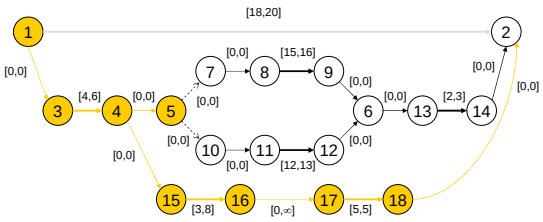
Temporal Constraint Consistency

- Example: Inconsistent



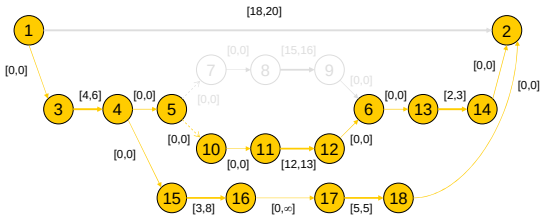
Temporal Constraint Consistency

- Backtrack to choice

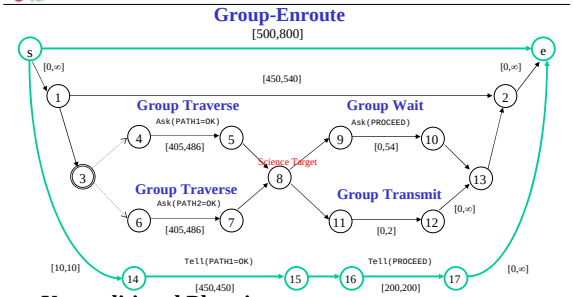


Temporal Constraint Consistency

- Complete paths



How Do Handle Asks?

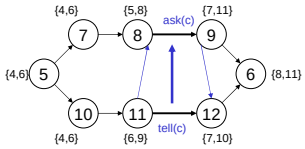


Unconditional Planning:

- **Guarantee satisfaction at compile time.**
- **Treatment similar to causal-link planning**

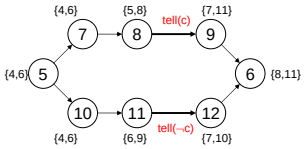
Satisfying Asks

- Compute bounds on activities.
- Link ask to equivalent, overlapping tell.
- Constrain tell to contain ask.



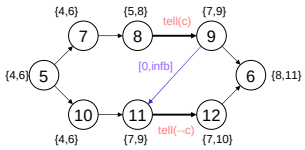
Avoiding Threats

- Identify overlapping Inconsistent activities.



Symbolic Constraint Consistency

- Promote or demote

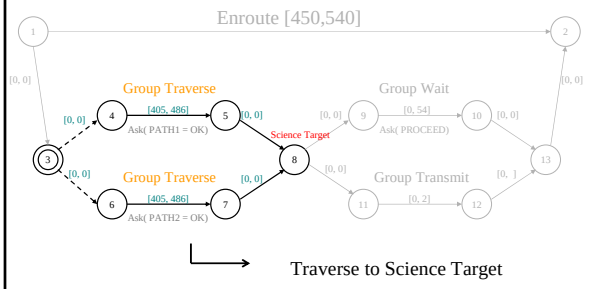


Outline

- Cooperative Vehicle Missions
- Model-based Programming
- Reactive Model-based Programming Language (RMPL)
- Temporal Plan Networks (TPN)
- Activity Planning (Kirk)
- Optional: Hybrid Activity/Path Planning

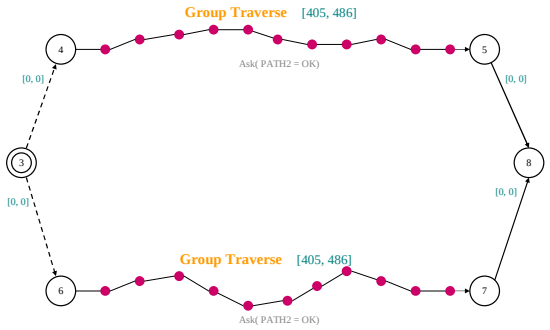
Enroute Activity:

•Closer look at Group Traverse sub-activity



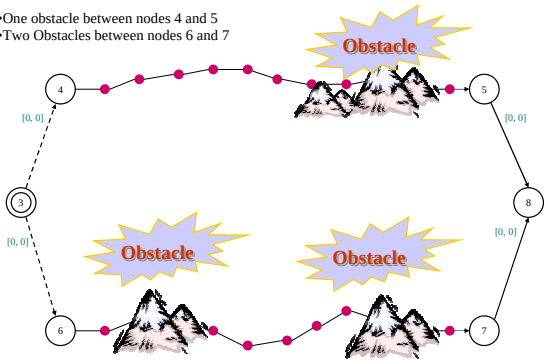
Group Traverse sub-activity:

•Traverse through **way points** to science target



Group Traverse sub-activity:

•One obstacle between nodes 4 and 5
•Two Obstacles between nodes 6 and 7



How do we optimally select activities and paths?

Current Research:

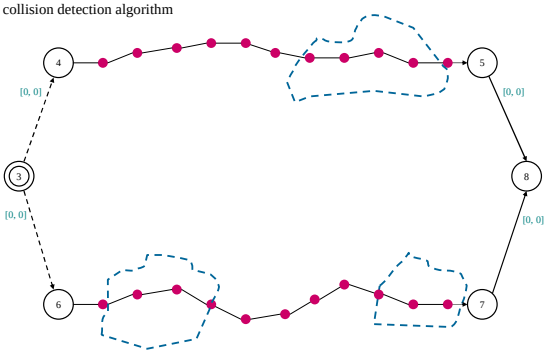
• Perform global path planning using Rapidly-exploring Random Trees (RRTs) (la Valle).

• Search for globally optimal plan by unifying TPN & RRT graphs, and by searching hybrid graph best first.

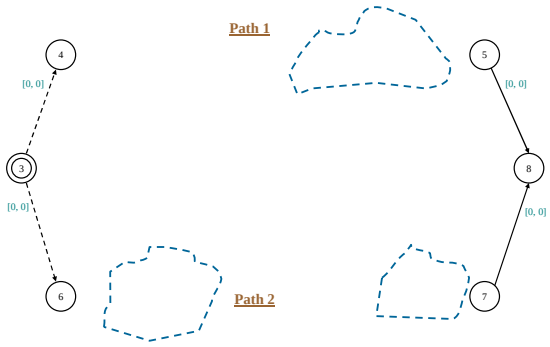
•Perform local kino-dynamic path planning along path segments using hybrid maneuver automata (Frazzoli, Dahleh, Feron).

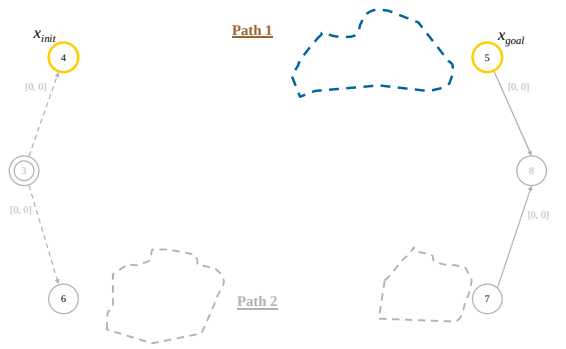
Group Traverse sub-activity:

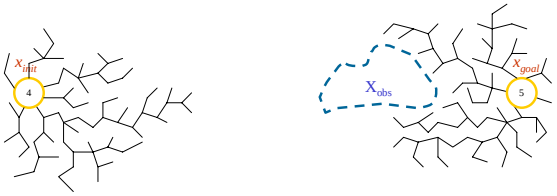
•Non-explicit representations of obstacles obtained from an incremental collision detection algorithm

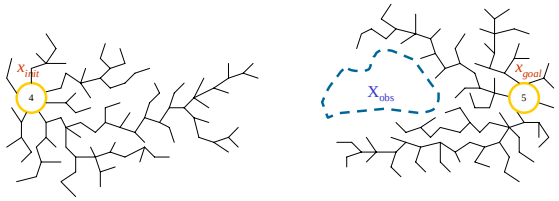


RRT: Example









RRT: **Example**

Assume rovers take Path 1:

RRT: **Example**

RRT: **Example**

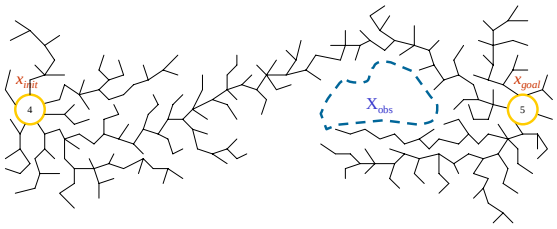
RRT: **Example**

RRT: **Example**

RRT: **Example**

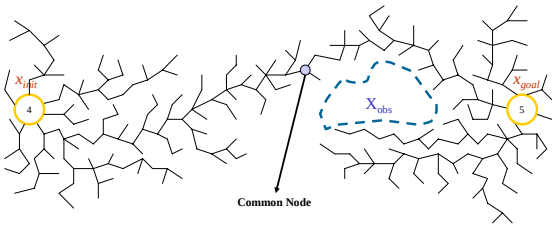
RRT: **Example**

Path 1



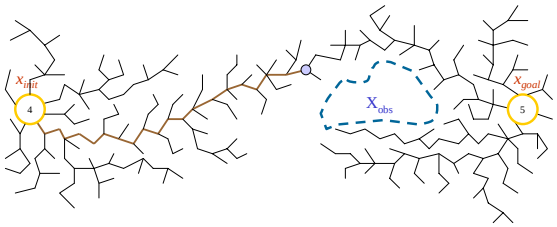
RRT: **Example**

Path 1



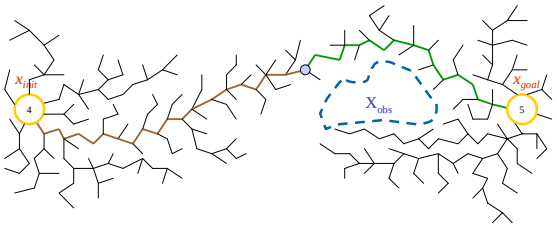
RRT: **Example**

Path 1



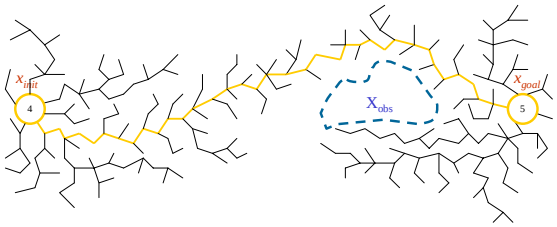
RRT: **Example**

Path 1

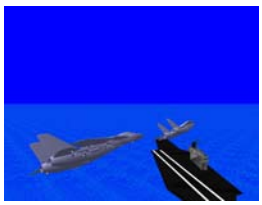


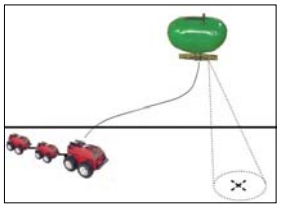
RRT: **Example**

Path 1



Current Status





- Kirk demonstrated on cooperative scenario using UAV simulation.
- Development on Multi-Rover testbed currently in progress.
- Distributed hybrid activity/path planning in progress.
- Selected for NASA Space Technology 7, Phase A
- together with IDEA (Muscettola)

Model-based Cooperative Programming

Goal: Fast, mission-directed coordination of teams of vehicles acting in an uncertain environments.

Solution: New middle ground between embedded programming, task decomposition execution, and temporal planning.

- Rich embedded language,RMPL, for describing complex concurrent team strategies extended to time and contingency.
- Kirk Interpreter “looks” for schedulable threads of execution before “leaping” to execution.
- Temporal Plan Network provides a flexible, temporal, graph-based planning paradigm built upon Simple Temporal Nets.
- Interpreter “leaps” through flexible execution (Nicola talk).
- Current work towards unifying activity planning, global path planning and kino-dynamics.