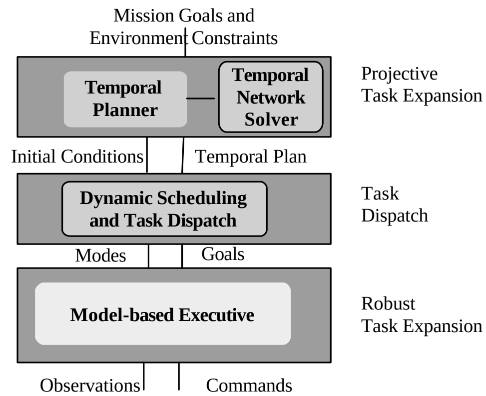
Robust Task Execution:
Procedural and Model-based

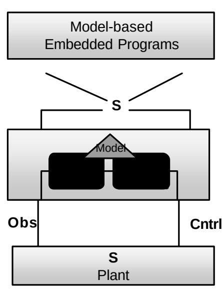
Brian C. Williams
16.412J/6.834J
March 14th, 2005

1



Desiderata: Robust Task-level Execution

Create Languages that are:
- Suspicious
  - Monitor intentions and plans
- Self-Adaptive
  - Exploits and generates contingencies
- Anticipatory
  - Predicts, plans and verifies into future
- State Aware
  - Commanded with desired state
- Fault Aware
  - Reasons about and responds to failure



Outline

- Safe Procedural Execution
- Model-Predictive Dispatch
- Model-based Reactive Planning

Robust Task Execution: RAPS [Firby PhD]

- RAPS Monitors Success Against Spec

```
(define-rap  (move-to thing place)
    (succeed (LOCATION thing place))
    (method
        (context (and (LOCATION thing loc)
                (not (= loc UNKNOWN))))
        (task-net
                (t0 (goto loc) ((TRUCK-LOCATION loc) for t1))
                (t1 (pickup thing)((TRUCK-HOLDING thing) for t2)
                ((TRUCK-HOLDING thing) for t3))
                (t2 (goto place) ((TRUCK-LOCATION place) for t3))
                (t3 (putdown thing))))
    (method
        (context (LOCATION thing UNKNOWN))
        (task-net
                (t0 (goto WAREHOUSE)))))
```

Robust Task Execution: RAPS [Firby PhD]

- RAPS Exploits contingencies by performing functionally redundant method selection

```
(define-rap (move-to thing place)
    (succeed (LOCATION thing place))
    (method
        (context(and (LOCATION thing loc)
                (not (= loc UNKNOWN))))
        (task-net
                (t0 (goto loc) ((TRUCK-LOCATION loc) for t1))
                (t1 (pickup thing)((TRUCK-HOLDING thing) for t2)
                ((TRUCK-HOLDING thing) for t3))
                (t2 (goto place) ((TRUCK-LOCATION place) for t3))
                (t3 (putdown thing))))
    (method
        (context (LOCATION thing UNKNOWN))
        (task-net
                (t0 (goto WAREHOUSE)))))
```

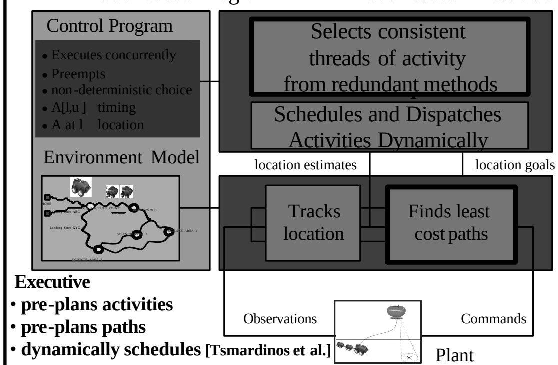
Robust Task Execution: RAPS [Firby PhD]

- RAPS Exploits contingencies by performing functionally redundant method selection
  - Methods are chosen based on the current situation.
  - If a method fails, another is tried instead.
  - Tasks do not complete until satisfied.
  - Methods can include monitoring subtasks to deal with contingencies and opportunities.

➢ **Methods selected reactively**
  - ⇨ **Model-predictive dispatch**

➢ **Goals explicitly observable and controllable**
  - ⇨ **Model-based execution**

RMPL Model-based Program   Kirk Model-based Executive



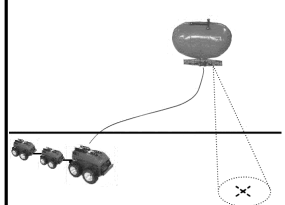
Executive
- **pre-plans activities**
- **pre-plans paths**
- **dynamically schedules [Tsmardinos et al.]**

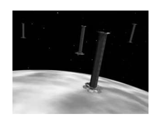
Outline

- Safe Procedural Execution
- Model-Predictive Dispatch
  - Model-based Programming
  - Temporal Plan Networks (TPN)
  - Activity Planning (Kirk)
  - Unifying Activity and Path Planning
- Model-based Reactive Planning

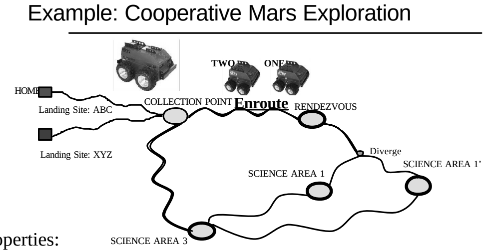
Example: Cooperative Mars Exploration

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





Example: Cooperative Mars Exploration



Properties:
- Teams exploit a hierarchy of complex strategies.
- Maneuvers are temporally coordinated.
- Novel events occur during critical phases.
- Quick responses draw upon a library of contingencies.
- Selected contingencies must respect timing constraints.

Reactive Model-based Programming

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

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

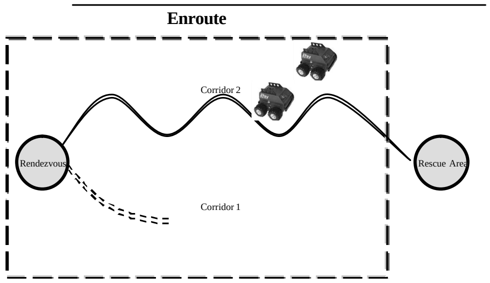
Add temporal constraints:

• 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
   }
}
```

Model-Predictive Dispatch for RMPL

How do we provide fast, temporally flexible planning for contingent method selection?

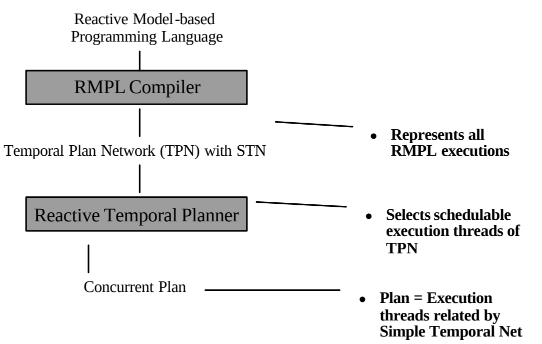
- Graph-based planners support fast planning.
- … but plans are totally order.
- Desire flexible plans based on simple temporal networks (e.g., Constrain-based Interval Planning).

How do we create temporally flexible plan graphs?

- Augment simple temporal networks with activities & choice.

⇨ temporal plan network TPN).

Model-Predictive Dispatch for RMPL



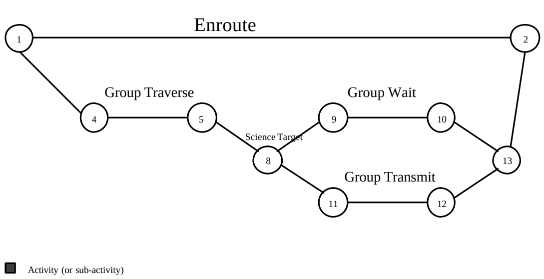
- **Represents all RMPL executions**
- **Selects schedulable execution threads of TPN**
- **Plan = Execution threads related by Simple Temporal Net**

Outline

- Safe Procedural Execution
- Model-Predictive Dispatch
  - Model-based Programming
  - Temporal Plan Networks (TPN)
  - Activity Planning (Kirk)
  - Unifying Activity and Path Planning
- Model-based Reactive Planning

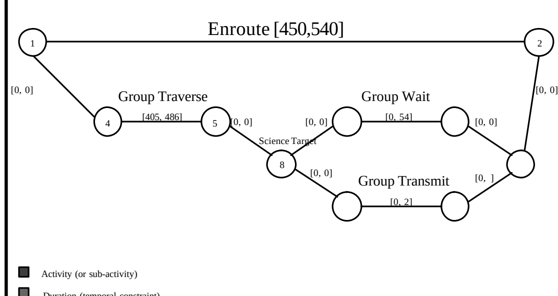
Enroute Activity:

- Start with flexible plan representation



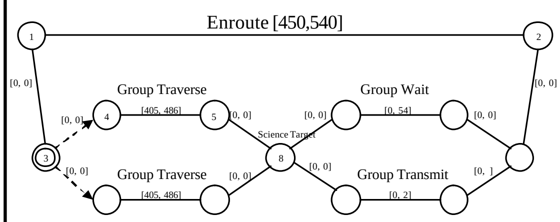
Enroute Activity:

- Start with flexible plan representation



Enroute Activity:

• Add conditional nodes



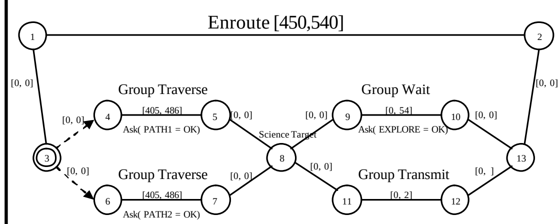
Activity (or sub-activity)

Duration (temporal constraint)

Conditional node

Enroute Activity:

•Add temporally extended, symbolic constraints



Activity (or sub-activity)

Duration (temporal constraint)

Conditional node

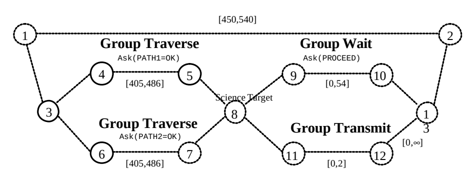
Symbolic constraint (Ask,Tell)

Outline

- Safe Procedural Execution
- Model-Predictive Dispatch
  - Model-based Programming
  - Temporal Plan Networks (TPN)
  - Activity Planning (Kirk)
  - Unifying Activity and Path Planning
- Model-based Reactive Planning

Planning Group-Enroute

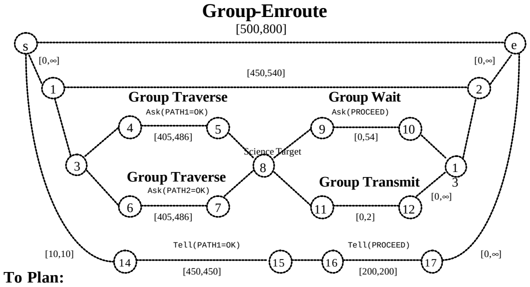
Group-Enroute



To Plan:
• Instantiate Group-Enroute

Planning Group-Enroute

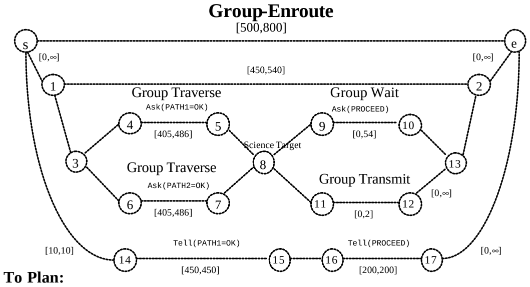
Group-Enroute
[500,800]



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

Generates Schedulable Plan

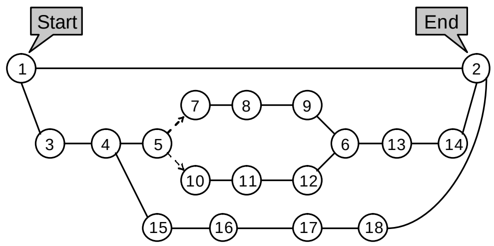
Group-Enroute
[500,800]



To Plan:
• Instantiate Group-Enroute
• Add External Constraints
• Trace Trajectories
• Check Schedulability
• Satisfy and Protect Asks

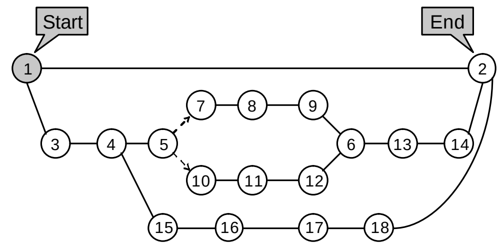
Trace Trajectories

- Find paths from start-node to end-node



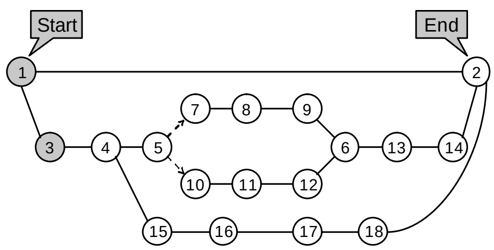
Trace Trajectories

- Not a decision-node: Follow all outarcs



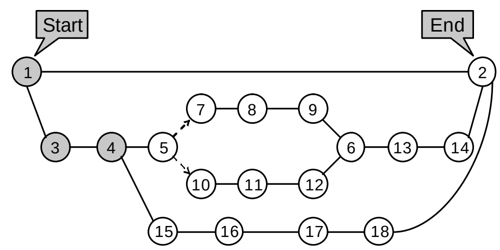
Trace Trajectories

- Not a decision-node: Follow all outarcs



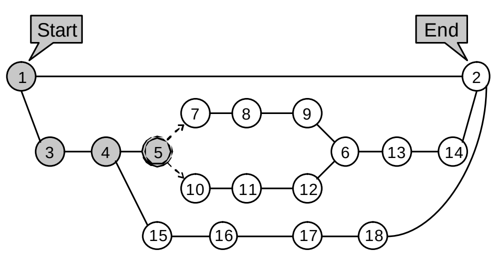
Trace Trajectories

- Not a decision-node: Follow all outarcs



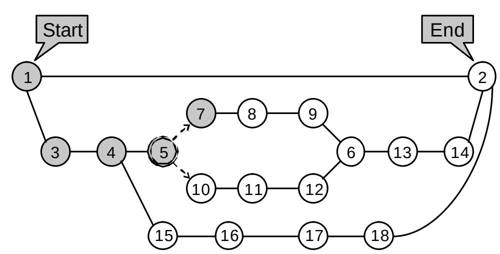
Trace Trajectories

- Decision-node: Select a single outarc



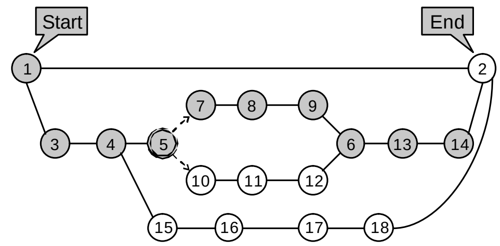
Trace Trajectories

- Not a decision-node: Follow all outarcs



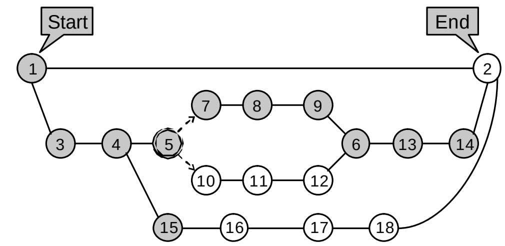
Trace Trajectories

- Continue



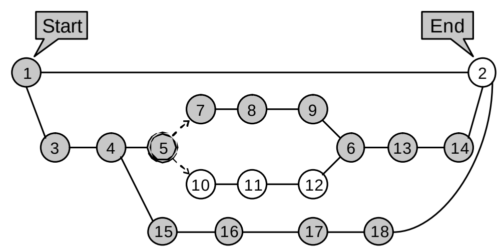
Trace Trajectories

- Not a decision-node: Follow all outarcs

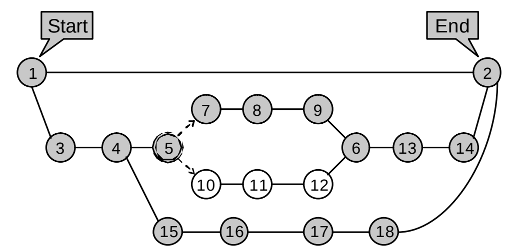


Trace Trajectories

- Continue



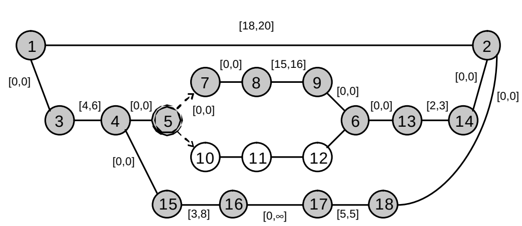
Trace Trajectories



Check Schedulability

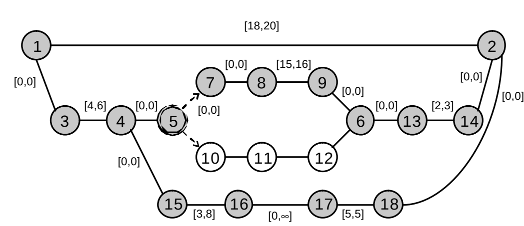
- Don't test consistency at each step.

⇨ Only when a path induces a cycle,
check for negative cycle in the STN distance graph



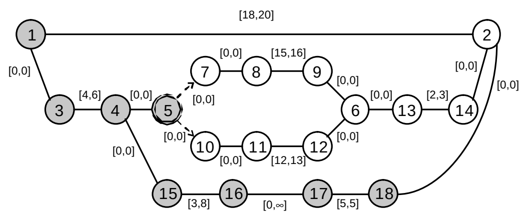
Check Schedulability

- Example: Inconsistent



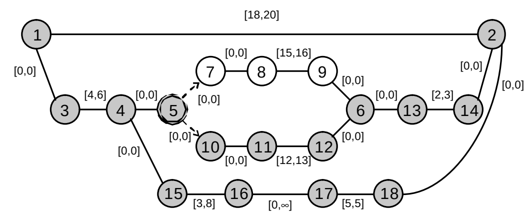
Trace Alternative Trajectories

- Backtrack to choice

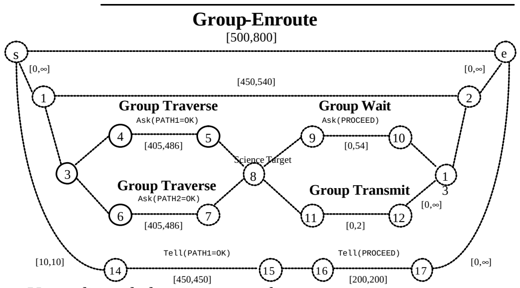


Trace Alternative Trajectories

- Complete paths



How Do We Handle Asks?

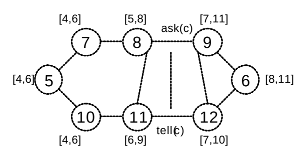


Unconditional planning approach:

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

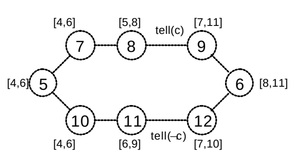
Satisfying Asks

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



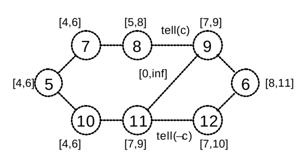
Avoiding Threats

- Identify overlapping Inconsistent activities.



Symbolic Constraint Consistency

- Promote or demote



How do we optimally select activities and paths?

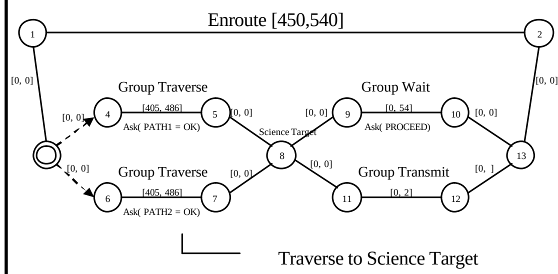
Background: Can perform global path planning using Rapidly-exploring Random Trees (RRTs) (la Valle).

Approach:

1. Search for globally optimal activity and path plan by
   - unifying TPN & RRT graphs, and
   - by searching hybrid graph best first.
2. Refine plan using receding horizon control.

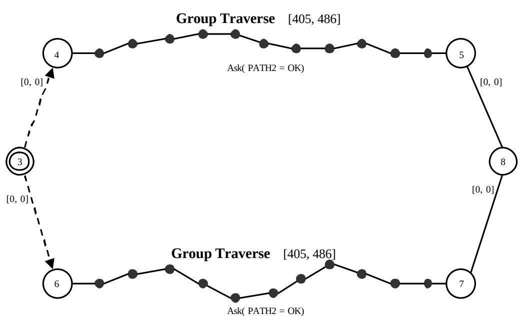
Enroute Activity:

•Closer look at Group Traverse sub-activity



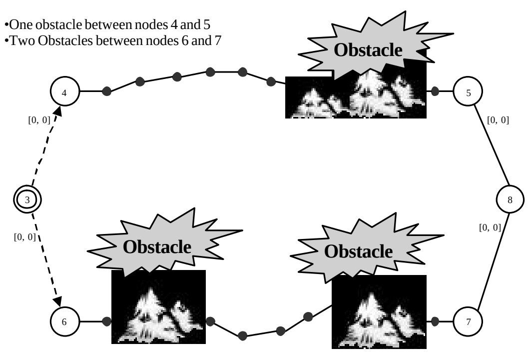
Group Traverse sub-activity:

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



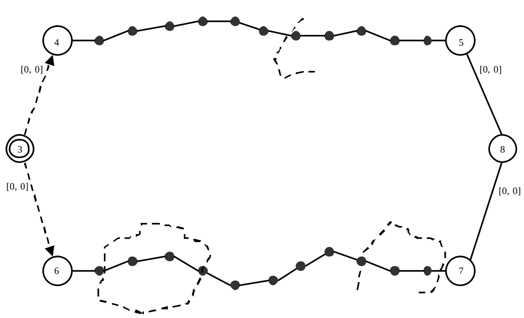
Group Traverse sub-activity:

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

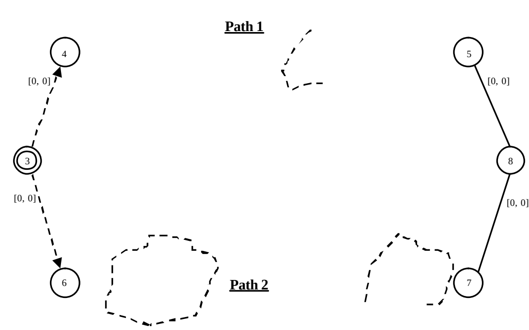


Group Traverse sub-activity:

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

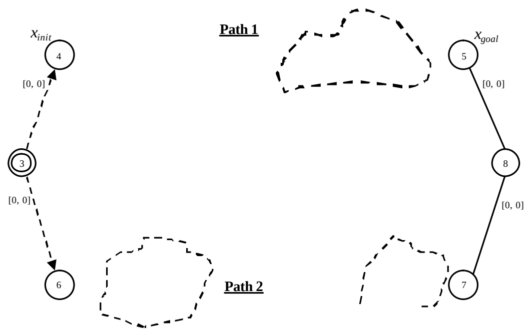


RRT: **Example**



RRT: Example

Planner considers rovers taking Path 1:

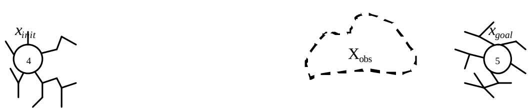


RRT: Example

Path 1

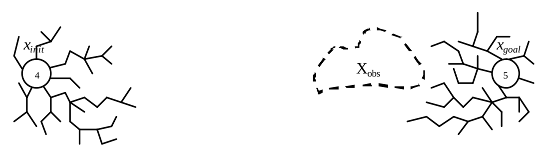
RRT: Example

Path 1



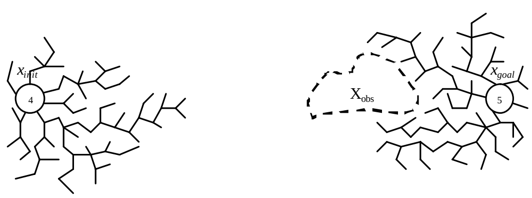
RRT: Example

Path 1



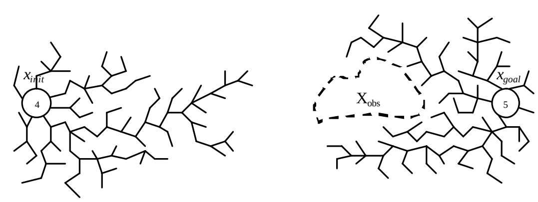
RRT: Example

Path 1



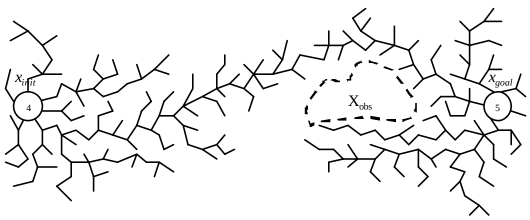
RRT: Example

Path 1



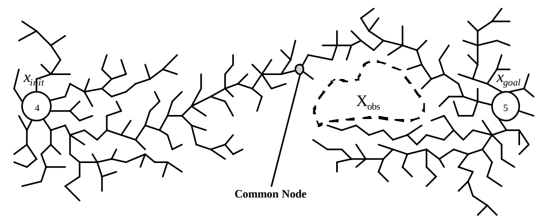
RRT: **Example**

Path 1



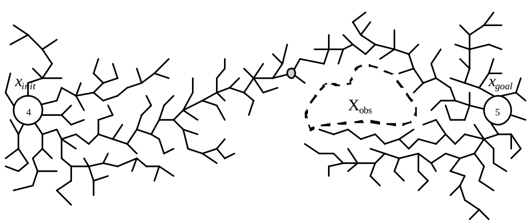
RRT: **Example**

Path 1



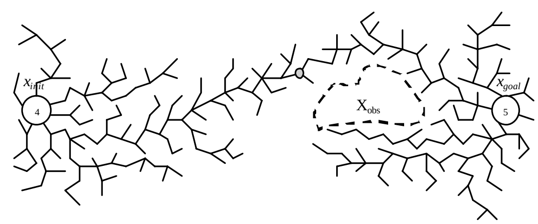
RRT: **Example**

Path 1



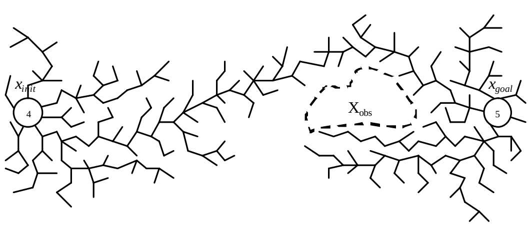
RRT: **Example**

Path 1



RRT: **Example**

Path 1



Model-Predictive Dispatch

Goal: Fast, robust, temporal execution with contingencies, in an uncertain environments.

Solution: Model-predictive Dispatch, a middle ground between non-deterministic programming 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.
- Global optimality achieved by unifying activity planning and global kino-dynamic path planning.