Roboter Navigation

Temporal Task-Motion Planning

Stefan Edelkamp

Driving Research Questions

How can we improve motion planning for complex systems?

- How can we develop motion planners that are generally applicable?
- How can we achieve planning efficiency even with nonlinear dynamics?
- How far back can we push the "curse of dimensionality"?
- Is there Pareto optimality between efficiency and solution quality?
- What formal guarantees can we provide?

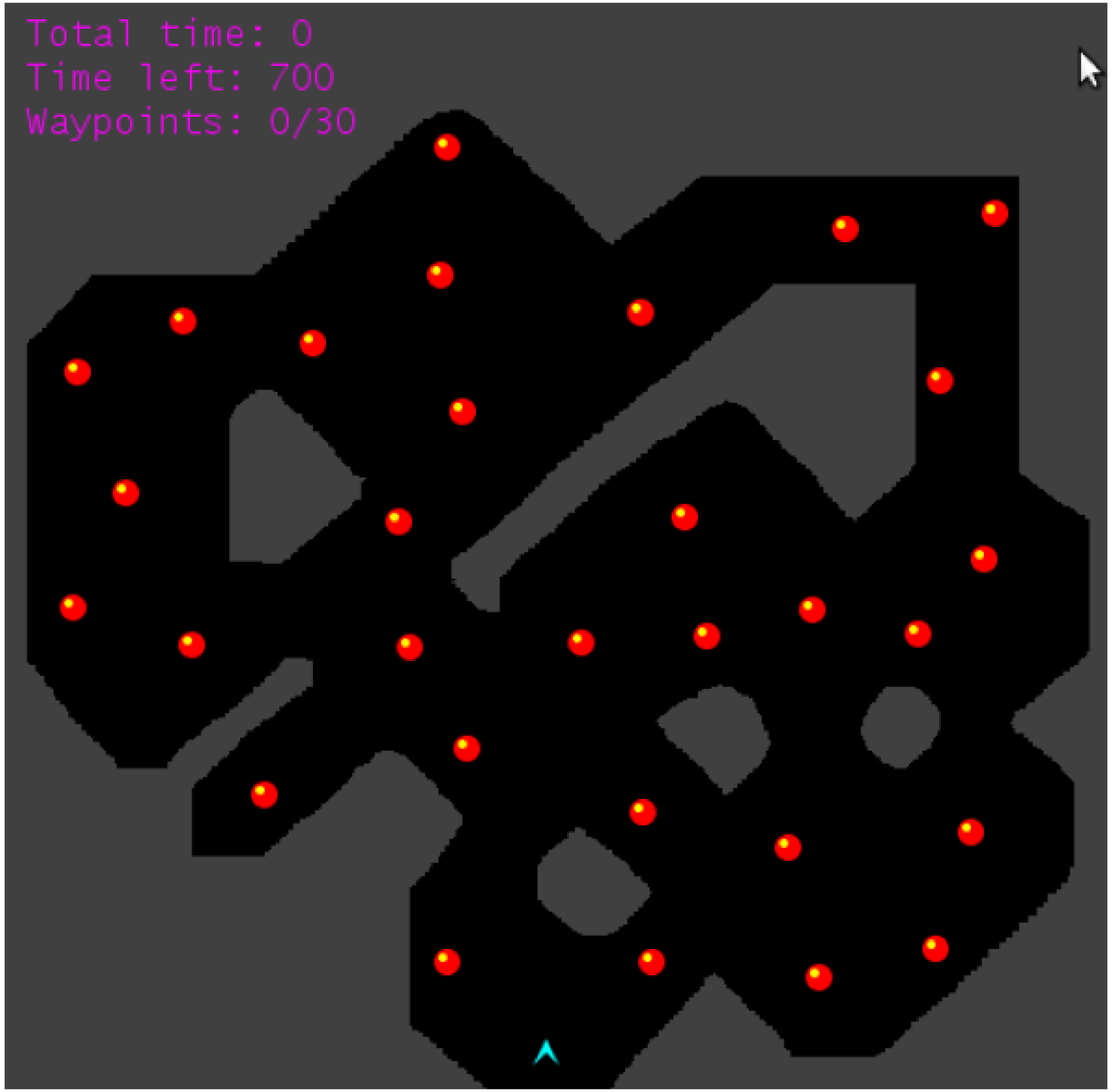
Framework

- **Sampling-based motion planning**
  - ⇒ generality: dynamics as *black-box function* $s_{\mathrm{new}} \leftarrow \text{MOTION}(s, u, dt)$
  - ⇒ continuous state/control spaces: *probabilistic sampling to make it feasible*
  - ⇒ high-dimensionality: *search to find solution*

  coupled with discrete abstractions
  - ⇒ provide simplified planning layer
  - ⇒ guide search in the continuous state/control spaces

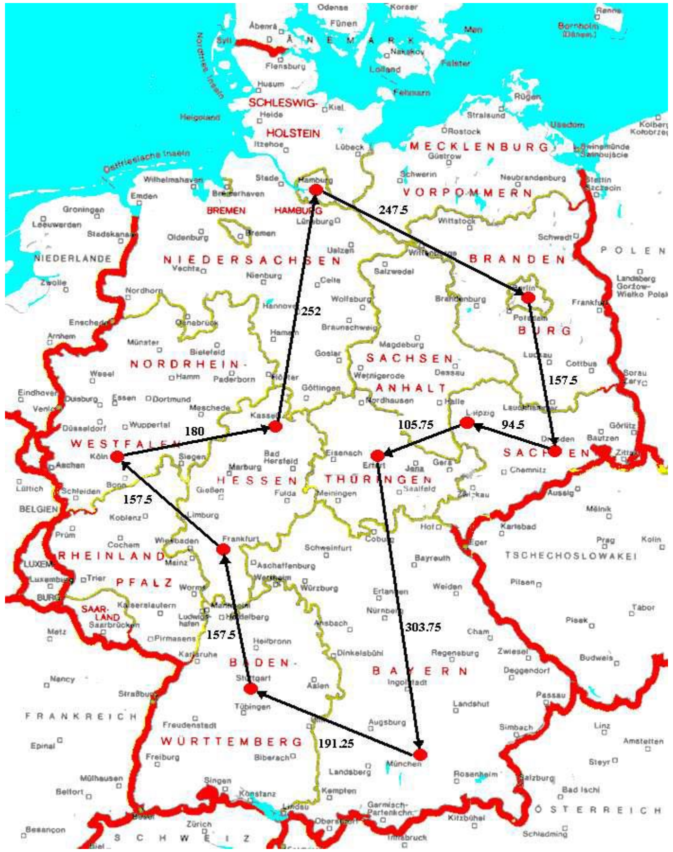
  and motion controllers
  - ⇒ open up the black-box MOTION function
  - ⇒ facilitate search expansion
- **Formal guarantees**
  - ⇒ probabilistic completeness

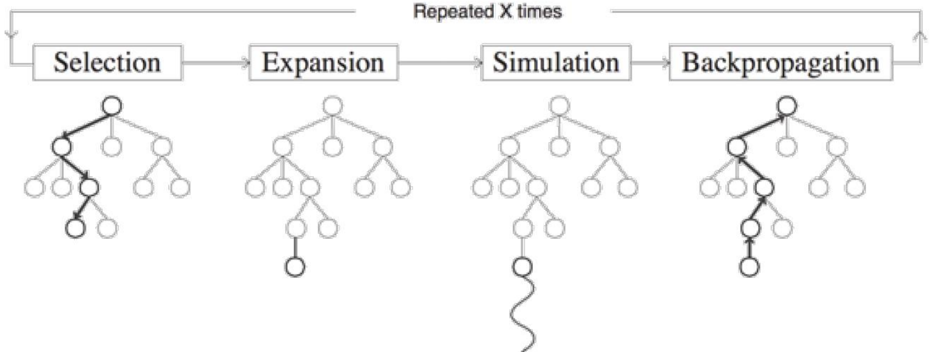
For a Start: The Physical TSP



Traveling Salesman Tours (TSP)

- Given a map, compute a minimum-cost round trip visiting certain cities
- Shortest paths graph reduction: precompute all-pairs-shortest-paths with
- Dijkstra’s algorithm (be smart: employ radix heaps)
- Traditional: Model problem as an IP and call solver (CPLEX, IPSolve,. . .)
- Neighborhood search (xOPT: SA; GA; AA; PSO; LNS,. . .)
- . . . New in the arena: Monte-Carlo Search





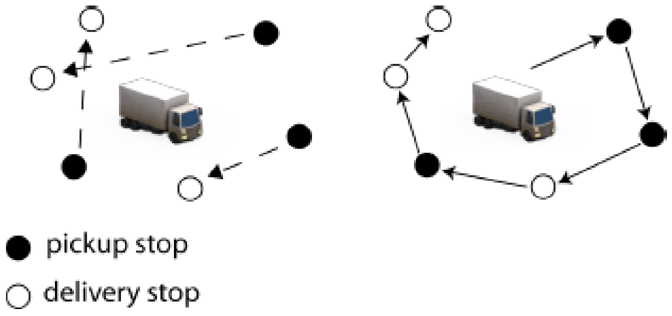
Additional Constraints

Time Windows, Capacities, Premium Services, Pickup and Deliveries

TSP+TW: Restricted time intervals / service times

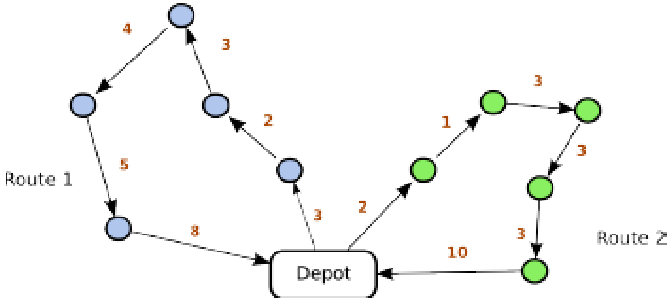
C+TSP: Limited vehicle load

TSP+PD: Pickup and deliveries (PDP)

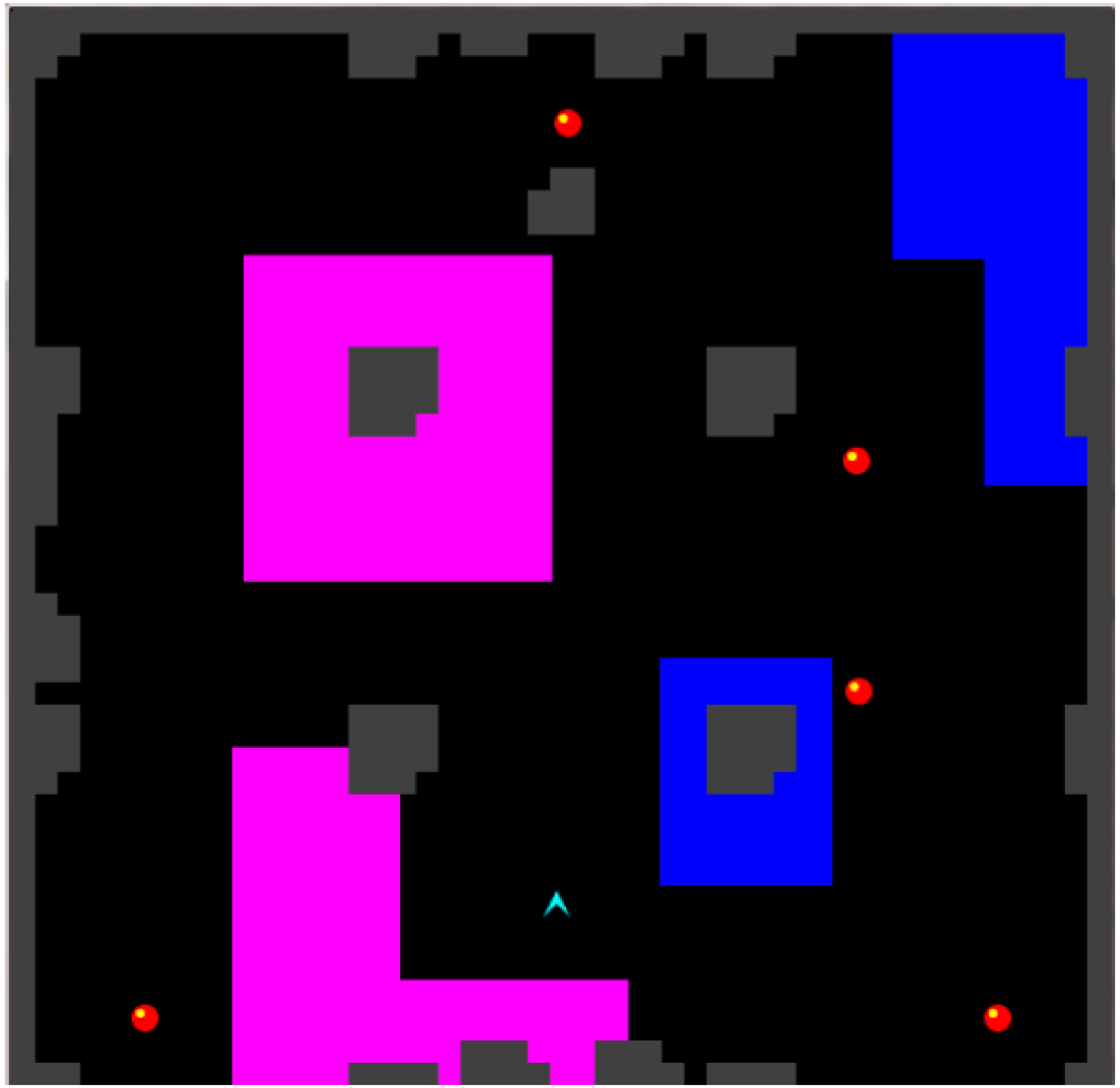


TSP*: Premium service – same-day delivery preferred

VRP: Vehicle routing – several vehicles

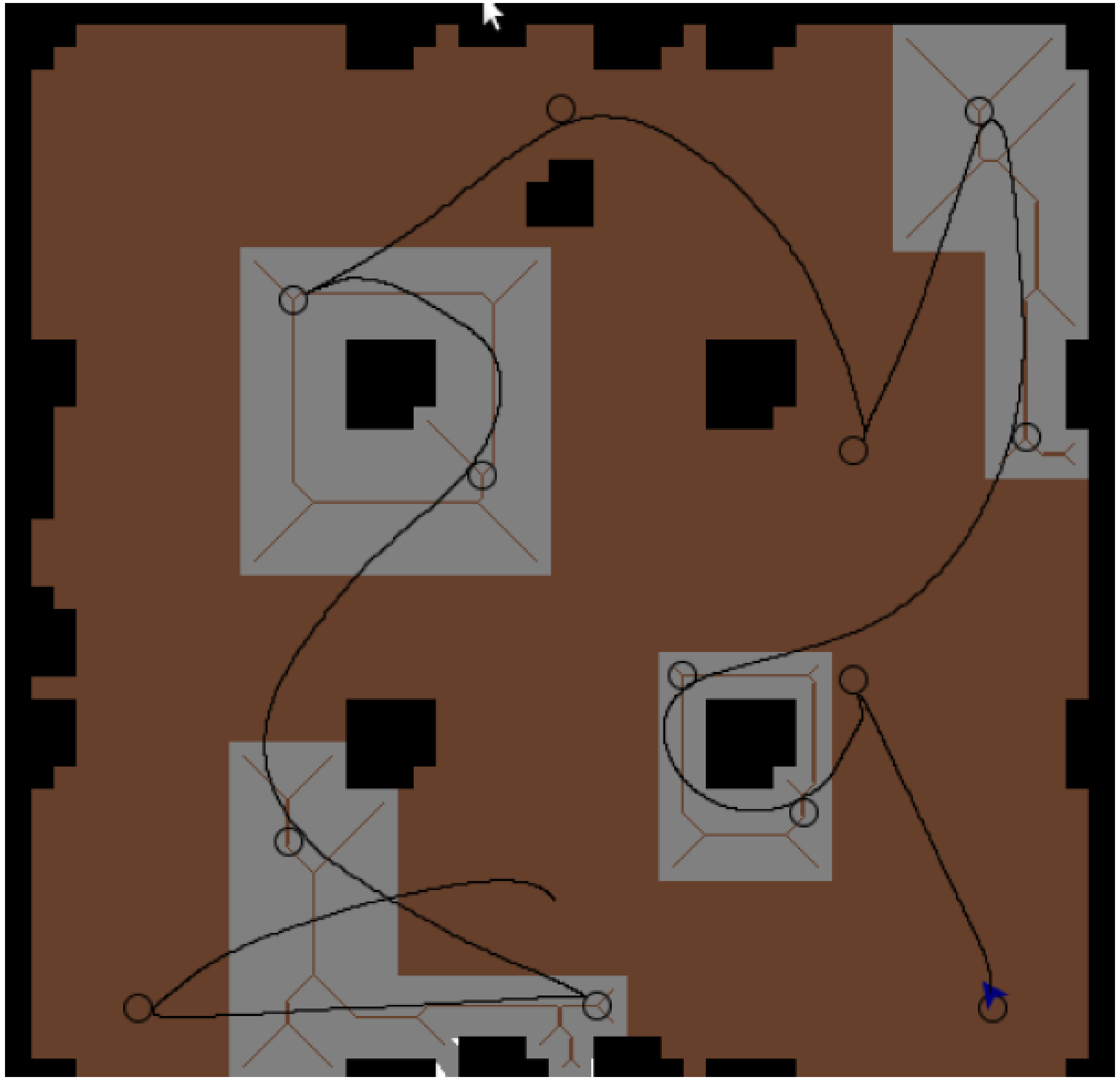


Inspection Problem

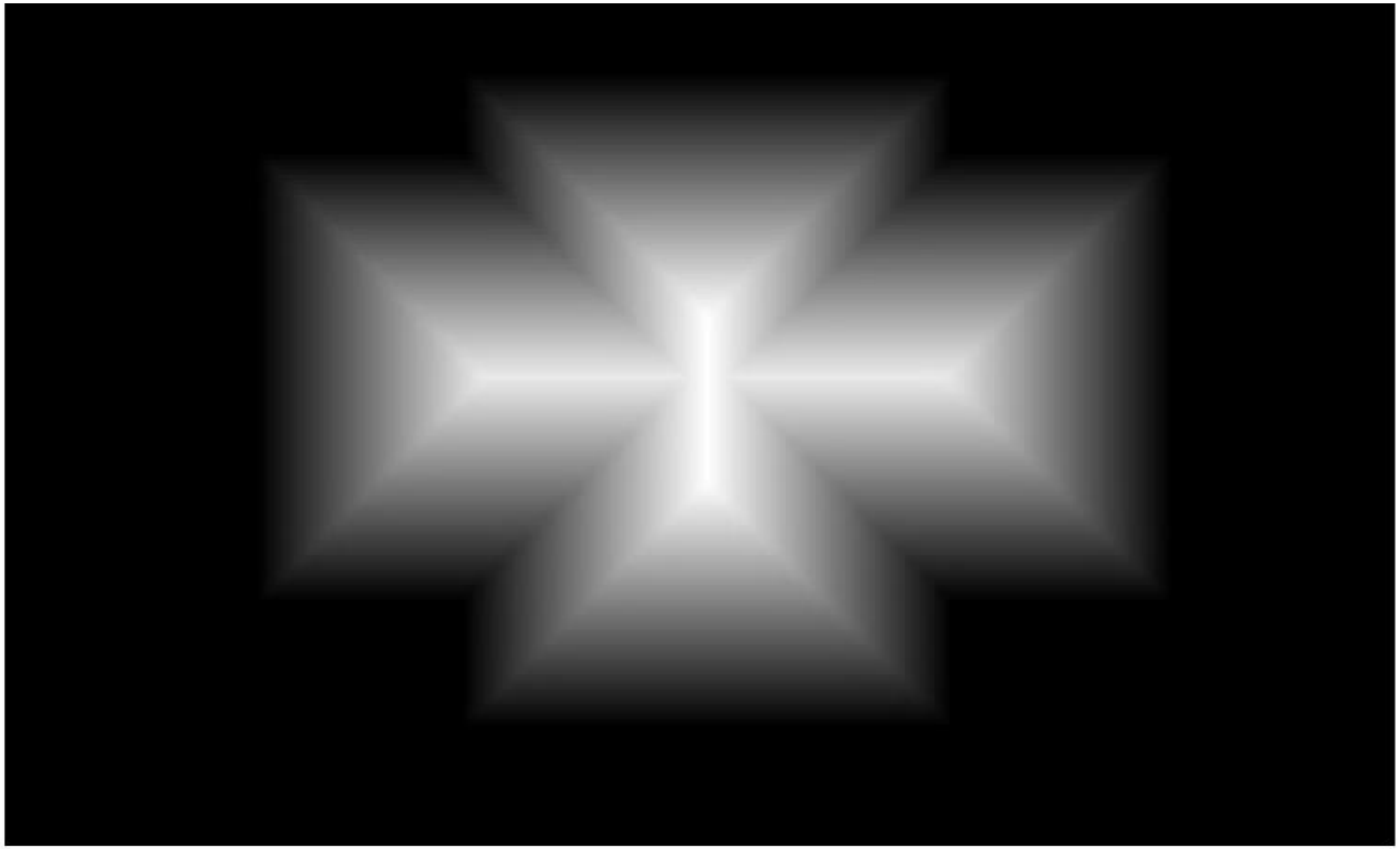


Inspection Problem

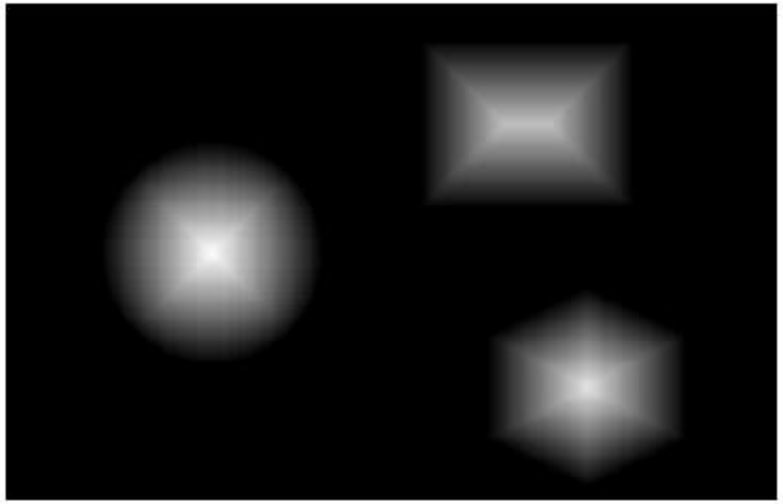
Inspection Tour



Grassfiring



Grassfiring



Generating Inspection Waypoints (1)

Input: $\mathcal{I}$: **bitmap image;** α: **desired inspection quality,**
$0 < \alpha \leq 1$
Output: a set of inspection points

$h \leftarrow \mathrm{height}(\mathcal{I})$; $w \leftarrow \mathrm{width}(\mathcal{I})$; $\mathcal{B} \leftarrow \mathrm{zeros}(h, w)$

$\diamondsuit$ ***grassfire transformation***

for $(i, j) \in \{0, \ldots, h-1\} \times \{0, \ldots, w-1\}$ **do**

  if $\mathrm{color}(\mathcal{I}(i,j)) \notin \{\mathtt{black}, \mathtt{gray}\}$ **then**

    $\mathcal{B}(i,j) \leftarrow 1 + \min\{\mathcal{B}(i-1,j), \mathcal{B}(i,j-1)\}$

for $(i, j) \in \{h-1, \ldots, 0\} \times \{w-1, \ldots, 0\}$ **do**

  if $\mathrm{color}(\mathcal{I}(i,j)) \notin \{\mathtt{black}, \mathtt{gray}\}$ **then**

    $\mathcal{B}(i,j) \leftarrow 1 + \min\{\mathcal{B}(i+1,j), \mathcal{B}(i,j+1)\}$

$\mathrm{skeleton} \leftarrow$ **extract pixels making up the most intense lines in the brightness map** $\mathcal{B}$

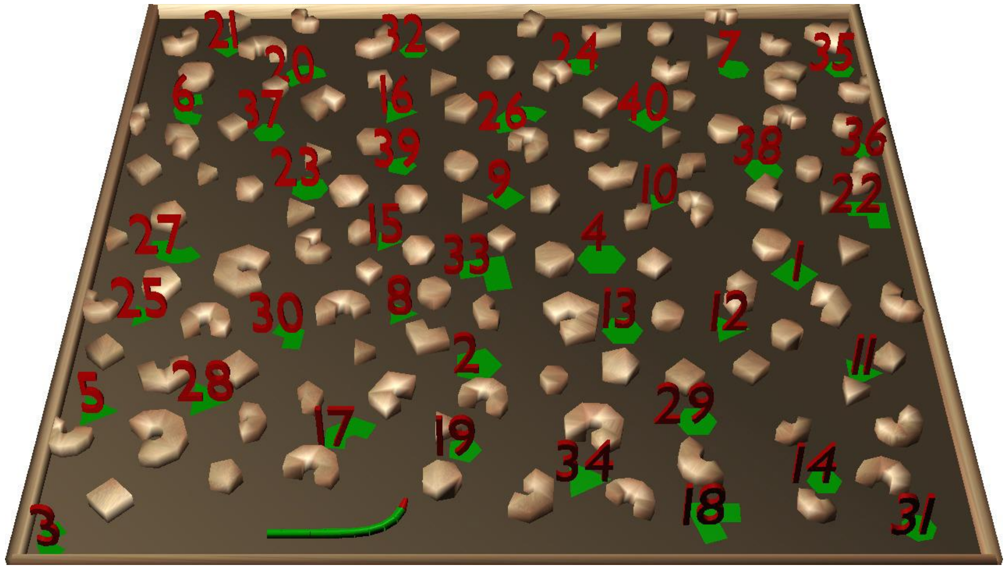
Generating Inspection Waypoints (2)

Input: $\mathcal{I}$: **bitmap image;** α: **desired inspection quality,** $0 < \alpha \leq 1$
Output: a set of inspection points

◇ ***select inspection points***

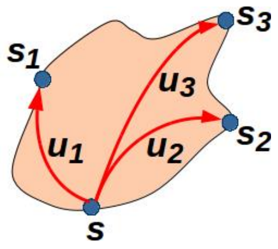
```
skeleton ← FILTER(skeleton)
inspectionPts ← skeleton
currScore ← VISSCORE(I, inspectionPts)
for p ∈ skeleton do
    newScore ← VISSCORE(I, inspectionPts \ {p})
    if newScore ≥ α ∨ currScore = newScore then
        inspectionPts ← inspectionPts \ {p}
        currScore ← newScore
return inspectionPts
```

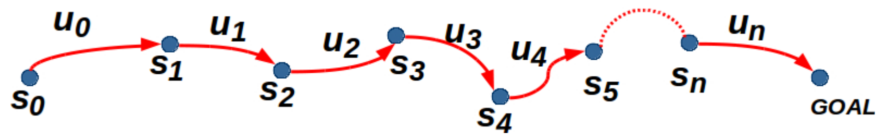
Multi-Goal Motion Planning with Dynamics



Dynamics

- **Express relation between input controls and resulting motions**
- **Necessary to plan motions that can be executed**
- **Impose significant challenges**
  - ➢ Constrain the feasible motions
  - ➢ Often are nonlinear and high-dimensional
  - ➢ Give rise to nonholonomic systems
  - ➢ State and control spaces are continuous
  - ➢ Solution trajectories are often long



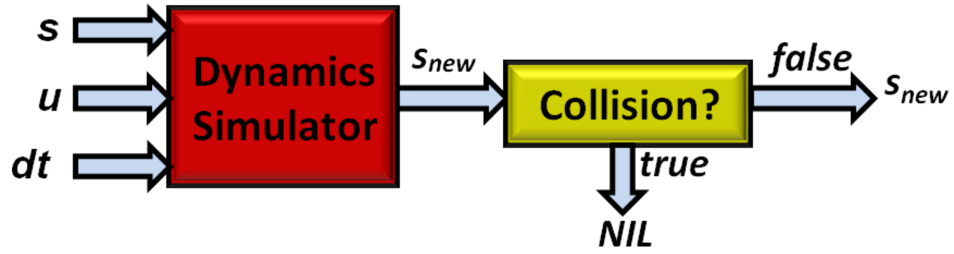


Computational complexity of motion planning with dynamics

- Point with Newtonian dynamics NP-Hard [DXCR 1993]
- Polygon Dubin’s car Decidable [CPK 2008]
- General nonlinear dynamics Undecidable [Branicky 1995]

Dynamics

- **Express relation between input controls and resulting motions**



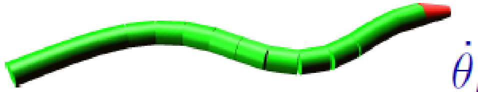
- **Modeled via physics-based engines**

$$\dot{s} = f(s, u)$$

ROS/Gazebo, ODE, Bullet, PhysX
general rigid-body dynamics
friction and gravity

$$s = (x, y, \theta_0, v, \psi, \theta_1, \ldots, \theta_n) \qquad u = (a, \omega)$$

$$\dot{x} = v\cos(\theta_0) \quad \dot{y} = v\sin(\theta_0) \quad \dot{\theta}_0 = v\tan(\psi) \quad \dot{v} = a \quad \dot{\psi} = \omega$$



$$\dot{\theta}_i = \frac{v}{d}\left(\prod_{j=1}^{i-1}\cos(\theta_{j-1} - \theta_j)\right)(\sin(\theta_{i-1}) - \sin(\theta))$$

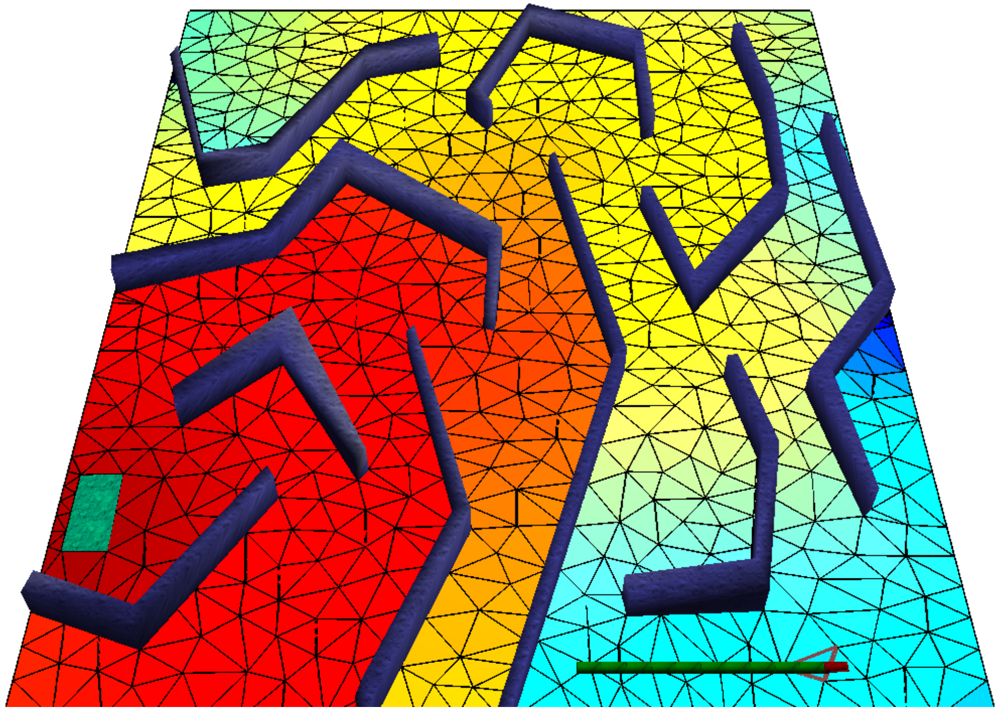
Introduce Discrete Layer to Guide the Search

Workspace decomposition provides

- discrete layer as adjacency graph G = (R,E)
- R denotes the regions of the decomposition
- E = {(ri,rj) | ri, rj in R are physically adjacent}

hcost(r) estimates the difficulty of reaching goal region from r

- defined as length of shortest path in G from r to goal [hcost(r1), hcost(r2),…, hcost(rn)]
- computed by running BFS/A* on G backwards from goal

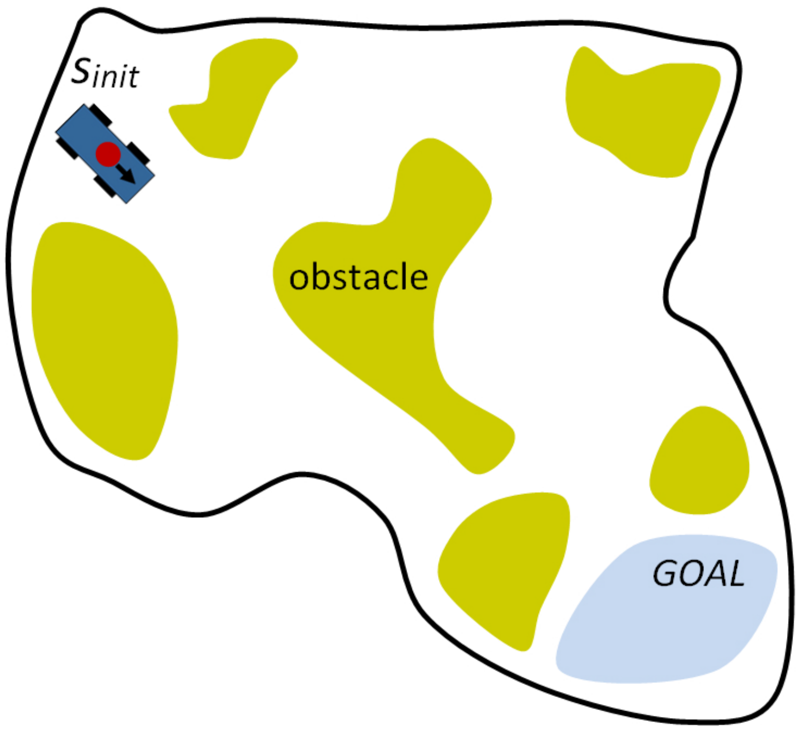


cost heuristics over discrete layer guide search in A* fashion

randomized sampling and PID controllers to expand motion tree

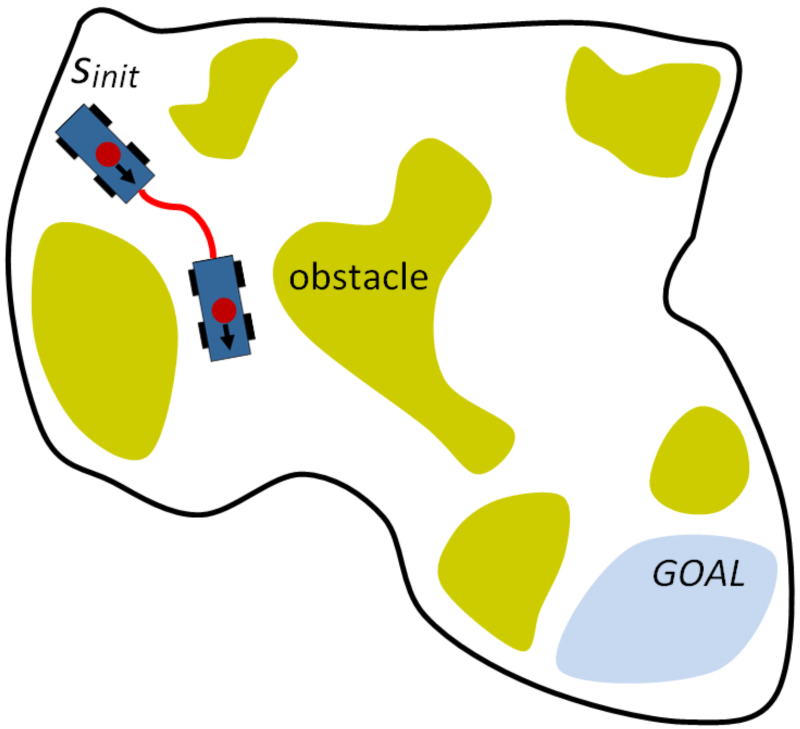
plans collision-free, dynamically-feasible, and low-cost solution trajectory

Sampling Based Motion Planning



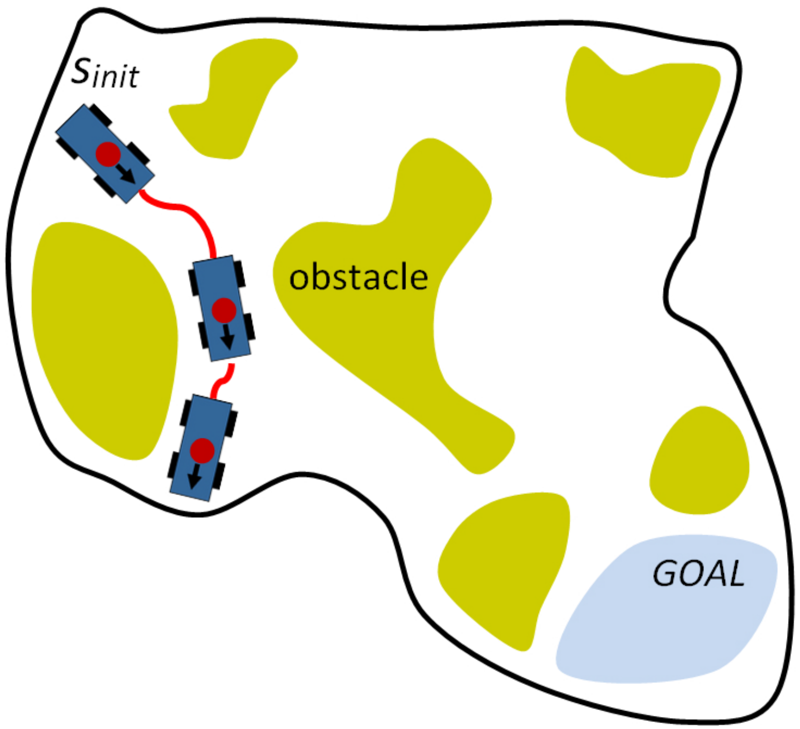
- Expand a tree T of collision-free and
- dynamically-feasible motions
  - select a state s from which to expand the tree
  - sample control input u
  - generate new trajectory by
  - applying u to s

Sampling Based Motion Planning



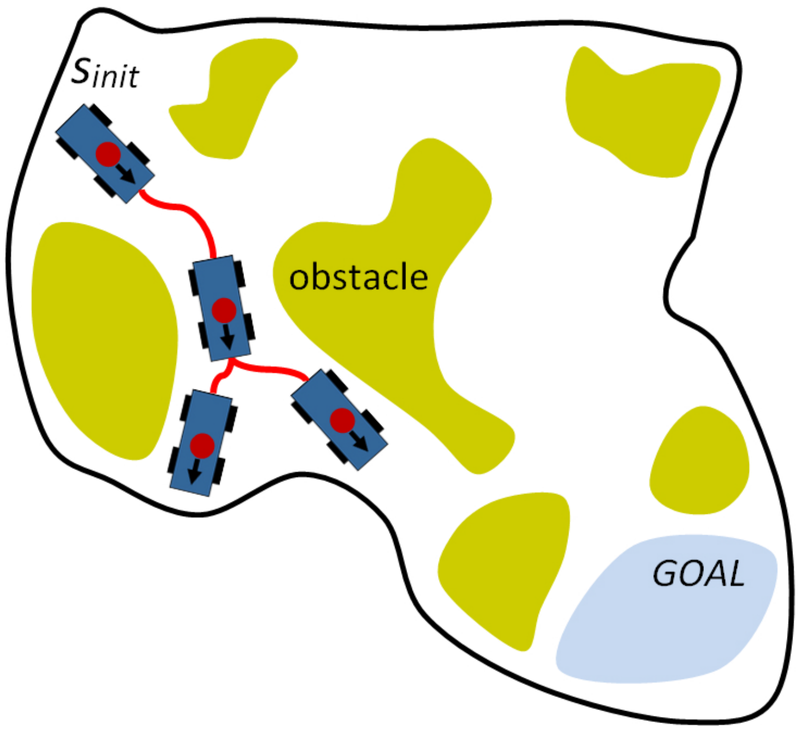
- Expand a tree T of collision-free and
- dynamically-feasible motions
  - select a state s from which to expand the tree
  - sample control input u
  - generate new trajectory by
  - applying u to s

Sampling Based Motion Planning



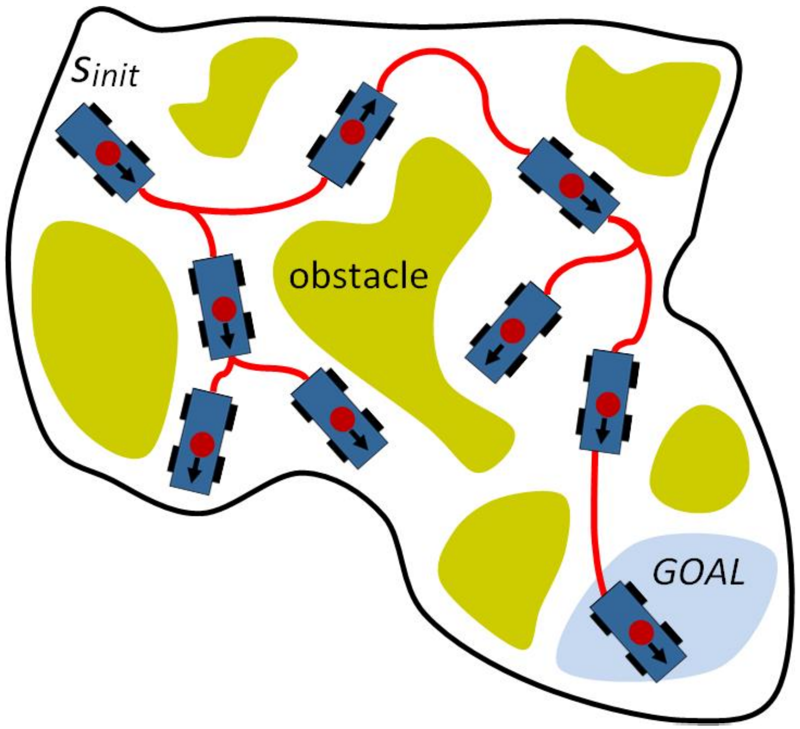
- Expand a tree T of collision-free and
- dynamically-feasible motions
  - select a state s from which to expand the tree
  - sample control input u
  - generate new trajectory by
  - applying u to s

Sampling Based Motion Planning



- Expand a tree T of collision-free and
- dynamically-feasible motions
  - select a state s from which to expand the tree
  - sample control input u
  - generate new trajectory by
  - applying u to s

Sampling Based Motion Planning

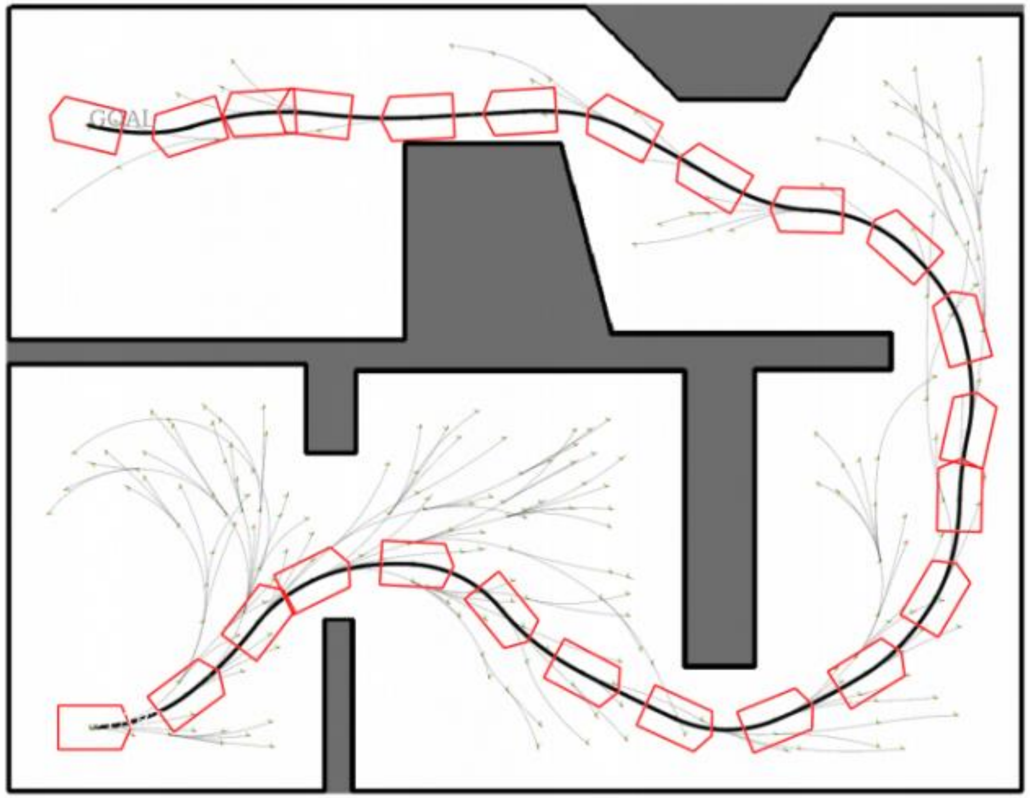


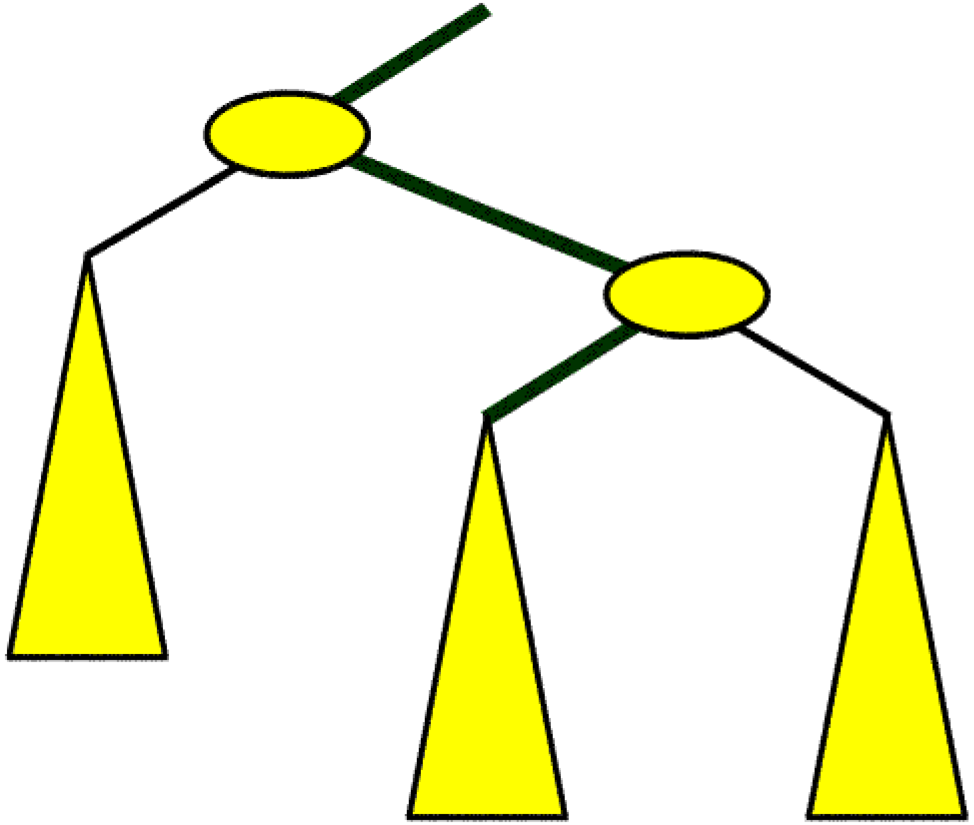
- Expand a tree T of collision-free and
- dynamically-feasible motions
  - ➢ select a state s from which to expand the tree
  - ➢ sample control input u
  - ➢ generate new trajectory by
  - ➢ applying u to s

Complex Robot Models - Often Sets of Differential Equations

Figure 2: Vehicle models of a car, snake, and blimp used in the experiments.

Sampling Based Motion Planning and Selection of Equivalence Class





$$\text{WEIGHT}(\mathcal{X}.v) = \frac{\alpha^{\text{NRSELECTIONS}(\mathcal{X}.v)}}{\text{DURATION}(\mathcal{X}.\sigma) * 2^{|\mathcal{X}.\sigma|}}.$$

Guided Expansion of Motion Tree

- **Sampling-based motion planning**

➢ generality: dynamics as black-box function s' =MOTION(s,u,dt)

➢ continuous state/control spaces: probabilistic sampling to make it feasible

➢ high-dimensionality: search to find solution

- **coupled with discrete abstractions**

➢ provide simplified planning layer

➢ guide search in the continuous state/control spaces

- **and motion controllers**

➢ open up the black-box MOTION function

➢ facilitate search expansion

- **Formal guarantees**

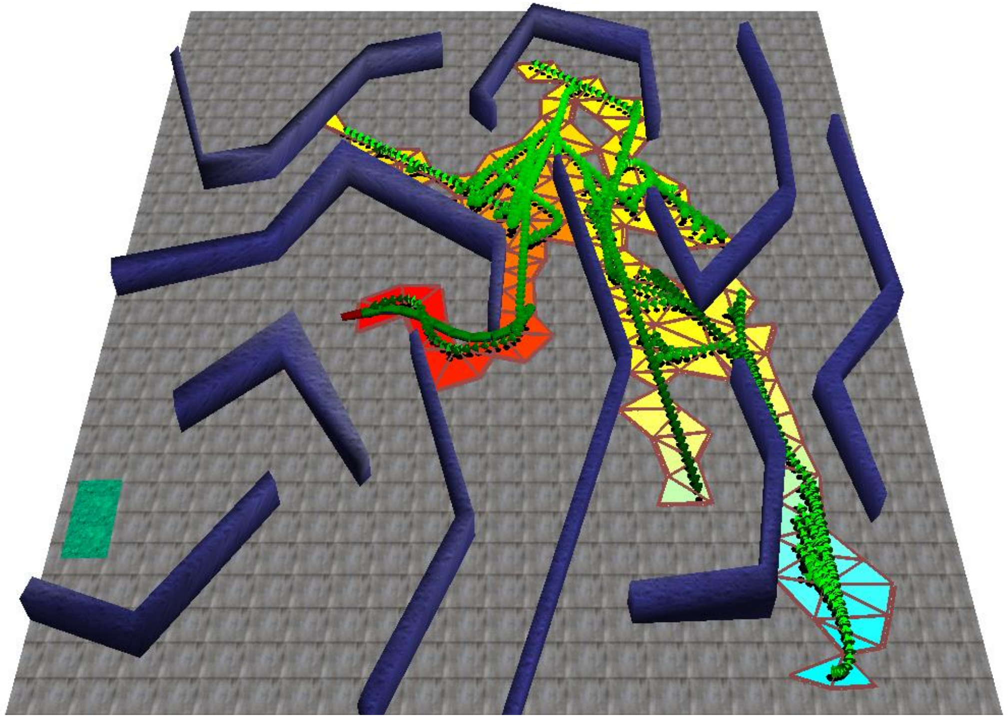
➢ probabilistic completeness

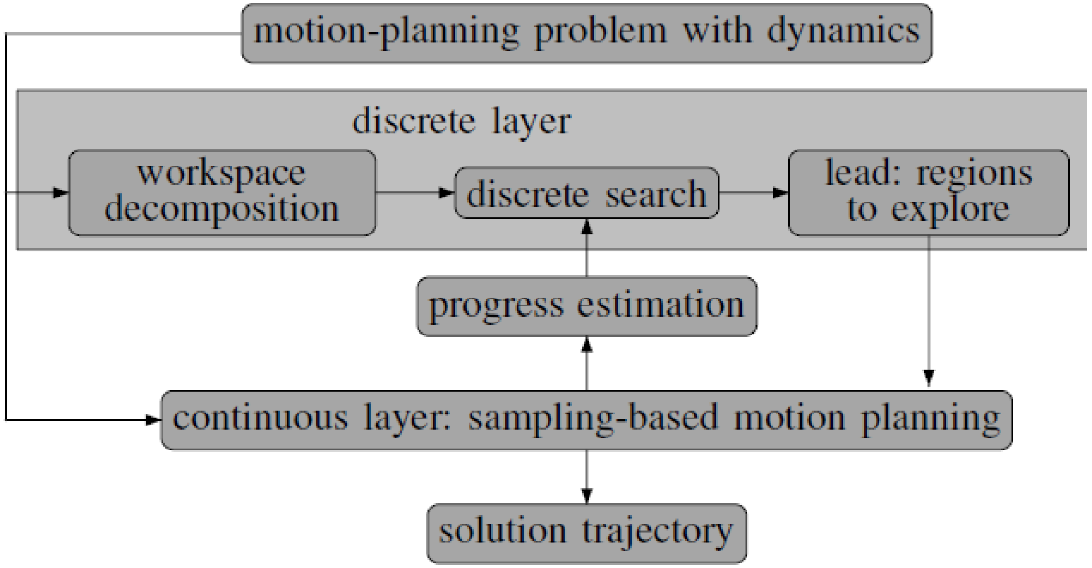
Driven by

selecting an equivalence class
from which to expand motion tree $\mathcal{T}$

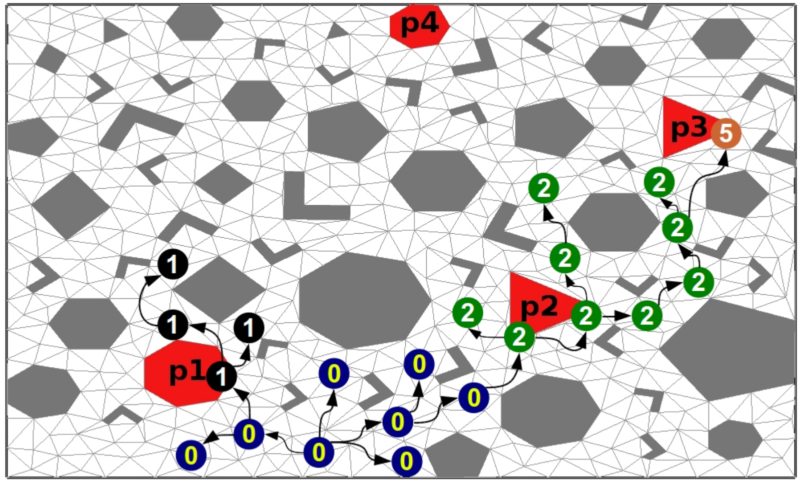
sampling-based motion planning to expand $\mathcal{T}$

Architecture





Abstraction



- **Used to induce partition of motion tree into equivalence classes**

$v_i \equiv v_j \quad \Longleftrightarrow \quad$ TRAJ$(\mathcal{T}, v_i)$ provides same abstract information as TRAJ$(\mathcal{T}, v_j)$

$\mathrm{region}(v_i) = \mathrm{region}(v_j)$

$\Rightarrow$ equivalence class corresponding to abstract state $\langle r \rangle$

$$\Gamma_{\langle r \rangle} = \{v : v \in \mathcal{T} \wedge \mathrm{region}(v) = r\}$$

$\Rightarrow$ partition of motion tree $\mathcal{T}$ into equivalence classes

$$\Gamma = \{\Gamma_{\langle r \rangle} : |\Gamma_{\langle r \rangle}| > 0\}$$

Graph Search for the Colored TSP

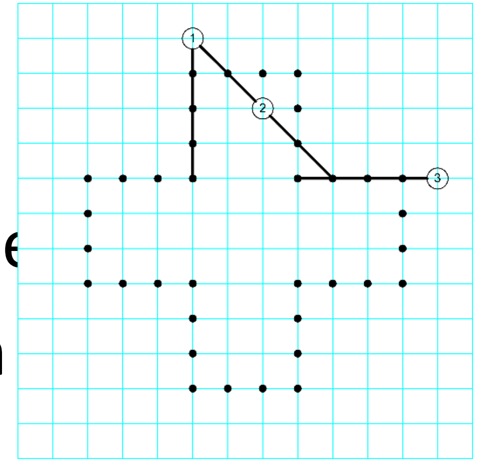
Let $G = (V, E, \mathrm{color}, \mathrm{cost})$ denote an undirected, colored, and weighted graph. Let $p_{\mathrm{start}} \in V$ denote the start vertex. A sequence of vertices $\langle p_1, \ldots, p_r \rangle$ constitutes a **valid colored tour** if

- $\{p_1, \ldots, p_r\} = V$,
- $p_1 = p_{\mathrm{start}}$,
- $\forall i \in \{1, \ldots, r-1\} : (p_i, p_{i+1}) \in E$, and
- $\forall i, j, k \in \{1, \ldots, r\} : \texttt{black} \notin \{\mathrm{color}(p_i), \mathrm{color}(p_j), \mathrm{color}(p_k)\}$ and $i < j < k \;\wedge\; \mathrm{color}(p_i) \neq \mathrm{color}(p_j) \implies \mathrm{color}(p_i) \neq \mathrm{color}(p_k)$.

An **optimal colored tour** is a colored tour with minimum cost, where the cost of the tour is defined as the sum of the weights associated with the edges of the tour.

Physical CTSP: integrate system dynamics such as angular change into cost function.

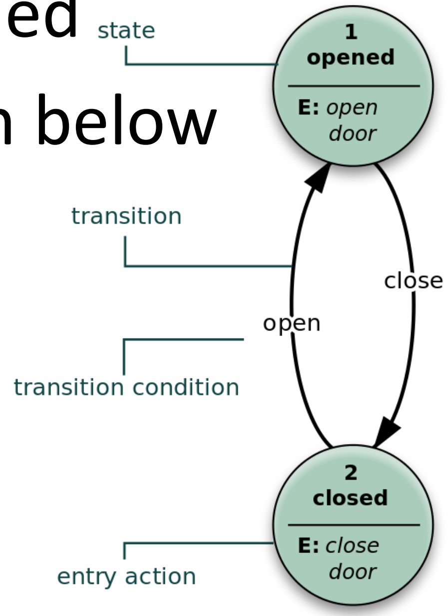
Nested Rollout Policy Adaptation



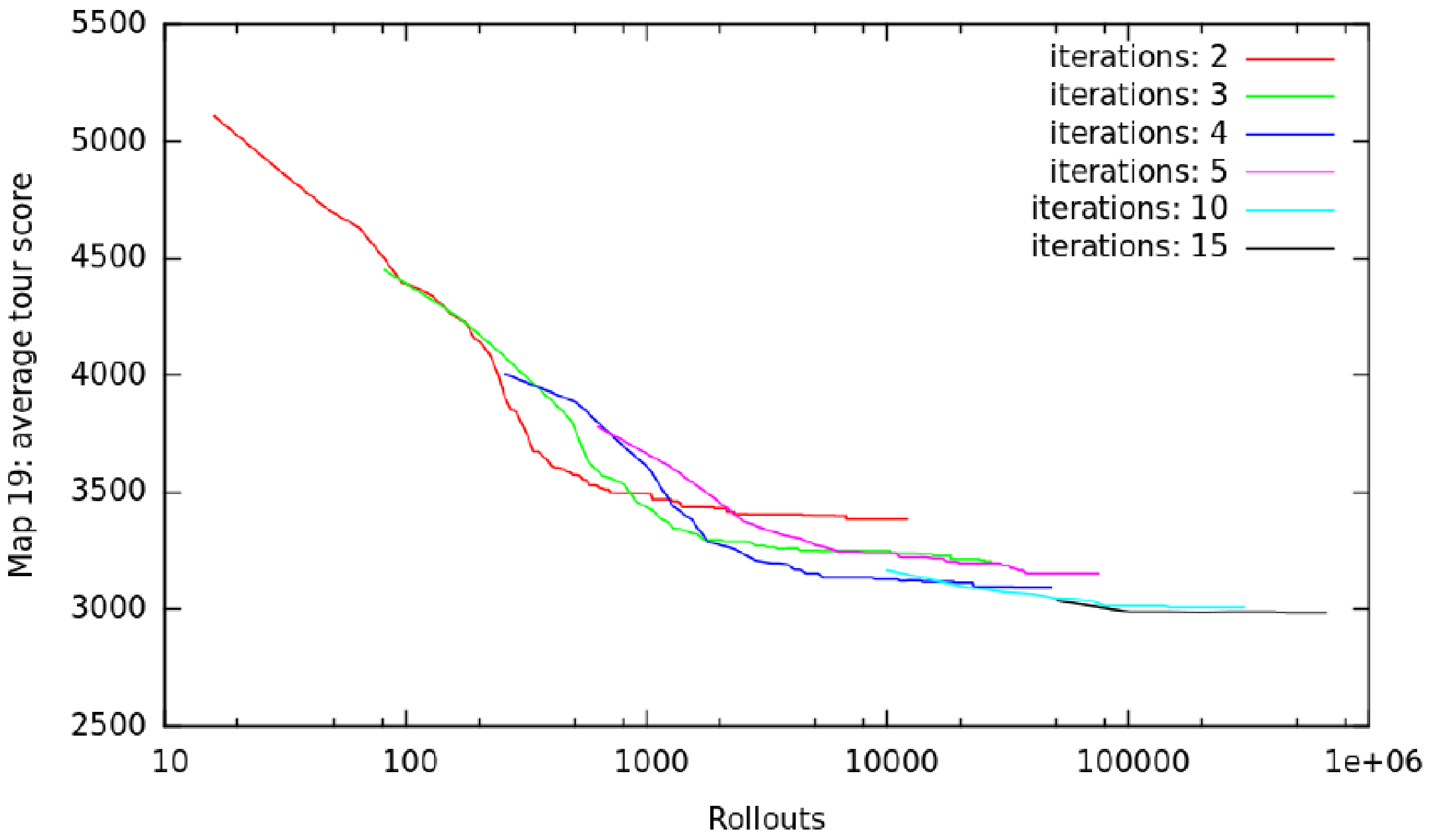
Rosin 2011 IJCAI – Best Paper, Morpion Solitaire with new Re

MCS tree based on Complete Rollout and Recursive Search

- Not really MCTS, No Search Tree.
- In Each Level a Policy is Maintained, Updated and Refreshed
- Updating Policy based on better Solutions Coming in from below
- Policy in Turn Influences the Rollouts
- Parameters: Level of Recursion, and Iteration Width
- Effective for TSPTW and many other Approaches
- Refinements: Beam / Diversity / Generalization

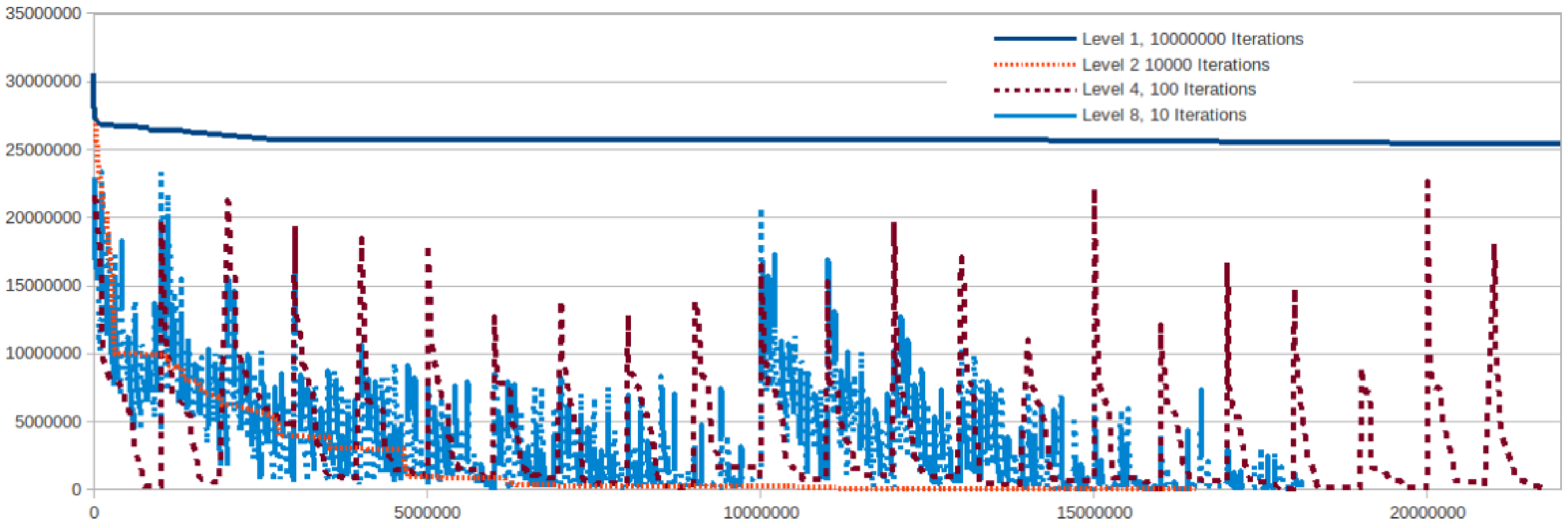


Learning Curve



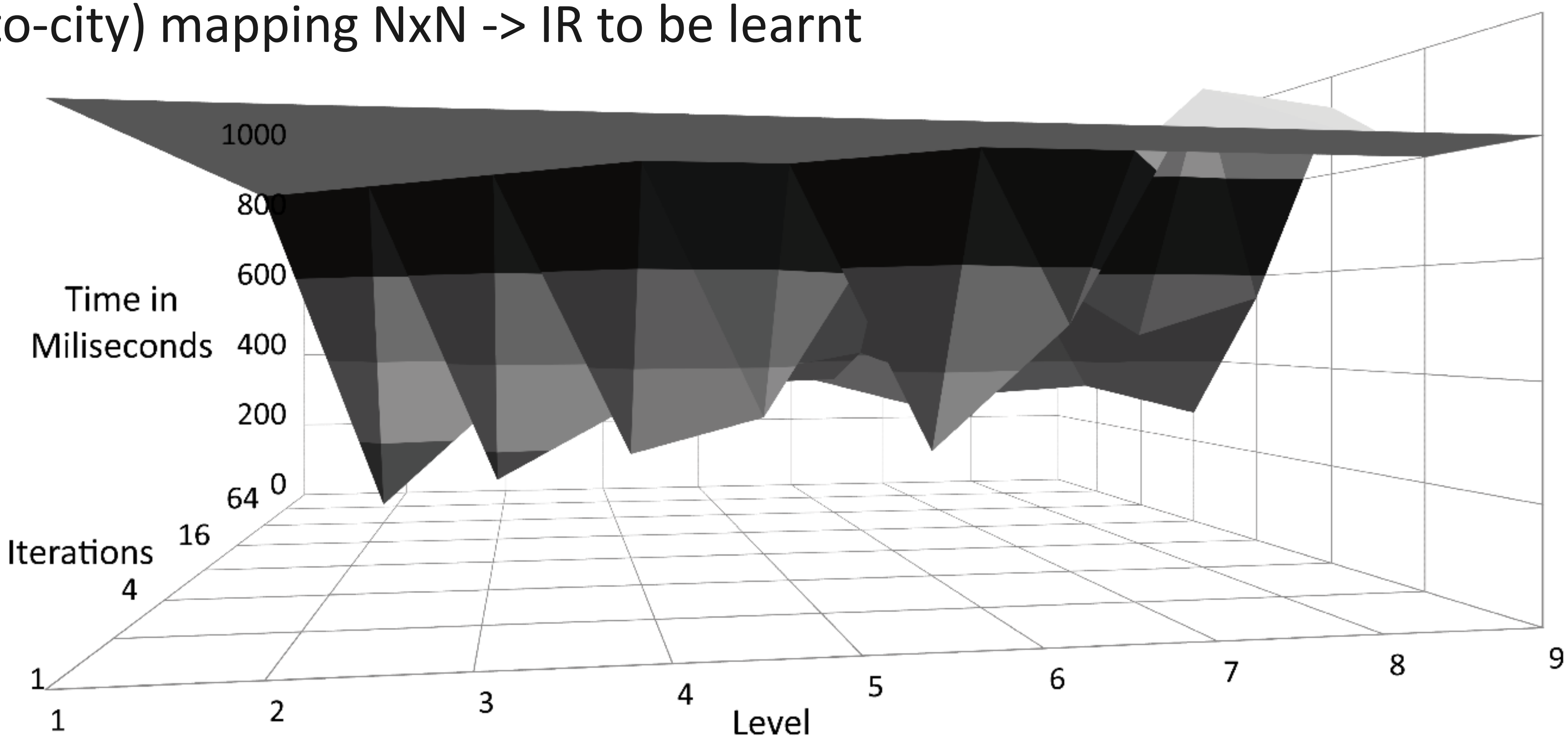
Nested Rollout Policy Adaptation

Input: Iteration width (exploitation), nestedness (exploration)
Policy: (city-to-city) mapping NxN -> IR to be learnt



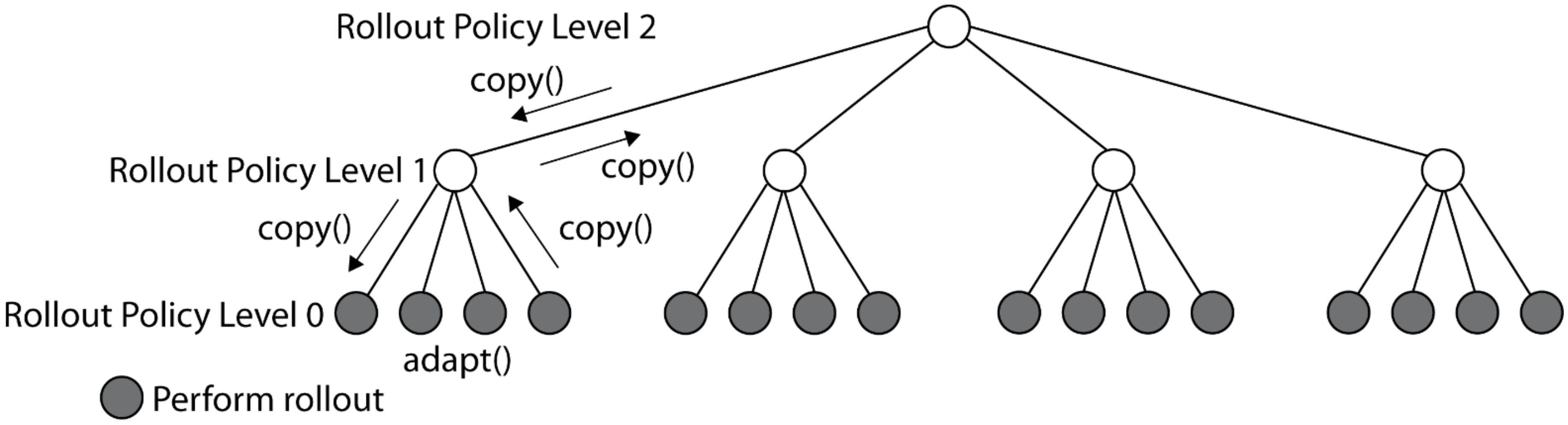
Nested Rollout Policy Adaptation

Input: Iteration width (exploitation), nestedness (exploration)
Policy: (city-to-city) mapping NxN -> IR to be learnt



Nested Rollout Policy Adaptation

Input: Iteration width (exploitation), nestedness (exploration)
Policy: (city-to-city) mapping NxN -> IR to be learnt



Playout

Algorithm 1 The playout algorithm

```
playout (state, policy)
    sequence ← []
    while true do
        if state is terminal then
            return (score (state), sequence)
        end if
        z ← 0.0
        for m in possible moves for state do
            z ← z + exp (policy [code(m)])
        end for
        choose a move with probability exp(policy[code(move)])/z
        state ← play (state, move)
        sequence ← sequence + move
    end while
```

Adapt

Algorithm 2 The Adapt algorithm

```
Adapt (policy, sequence)
    polp ← policy
    state ← root
    for move in sequence do
        polp [code(move)] ← polp [code(move)] + α
        z ← 0.0
        for m in possible moves for state do
            z ← z + exp (policy [code(m)])
        end for
        for m in possible moves for state do
            polp [code(m)] ← polp [code(m)] - α * exp(policy[code(m)]) / z
        end for
        state ← play (state, move)
    end for
    policy ← polp
```

Search

Algorithm 3 The NRPA algorithm.

```
NRPA (level, policy)
  if level == 0 then
    return playout (root, policy)
  else
    bestScore ← −∞
    for N iterations do
      (result,new) ← NRPA(level − 1, policy)
      if result ≥ bestScore then
        bestScore ← result
        seq ← new
      end if
      policy ← Adapt (policy, seq)
    end for
    return (bestScore, seq)
  end if
```

Theory…

The probability p_{ik} of choosing the move m_{ik} in a playout is the softmax function:

$$p_{ik} = \frac{e^{w_{ik}}}{\Sigma_j e^{w_{ij}}}$$

The cross-entropy loss for learning to play move m_{ib} is $C_i = -log(p_{ib})$. In order to apply the gradient we calculate the partial derivative of the loss: $\frac{\delta C_i}{\delta p_{ib}} = -\frac{1}{p_{ib}}$. We then calculate the partial derivative of the softmax with respect to the weights:

$$\frac{\delta p_{ib}}{\delta w_{ij}} = p_{ib}(\delta_{bj} - p_{ij})$$

Where $\delta_{bj} = 1$ if $b = j$ and 0 otherwise. Thus the gradient is:

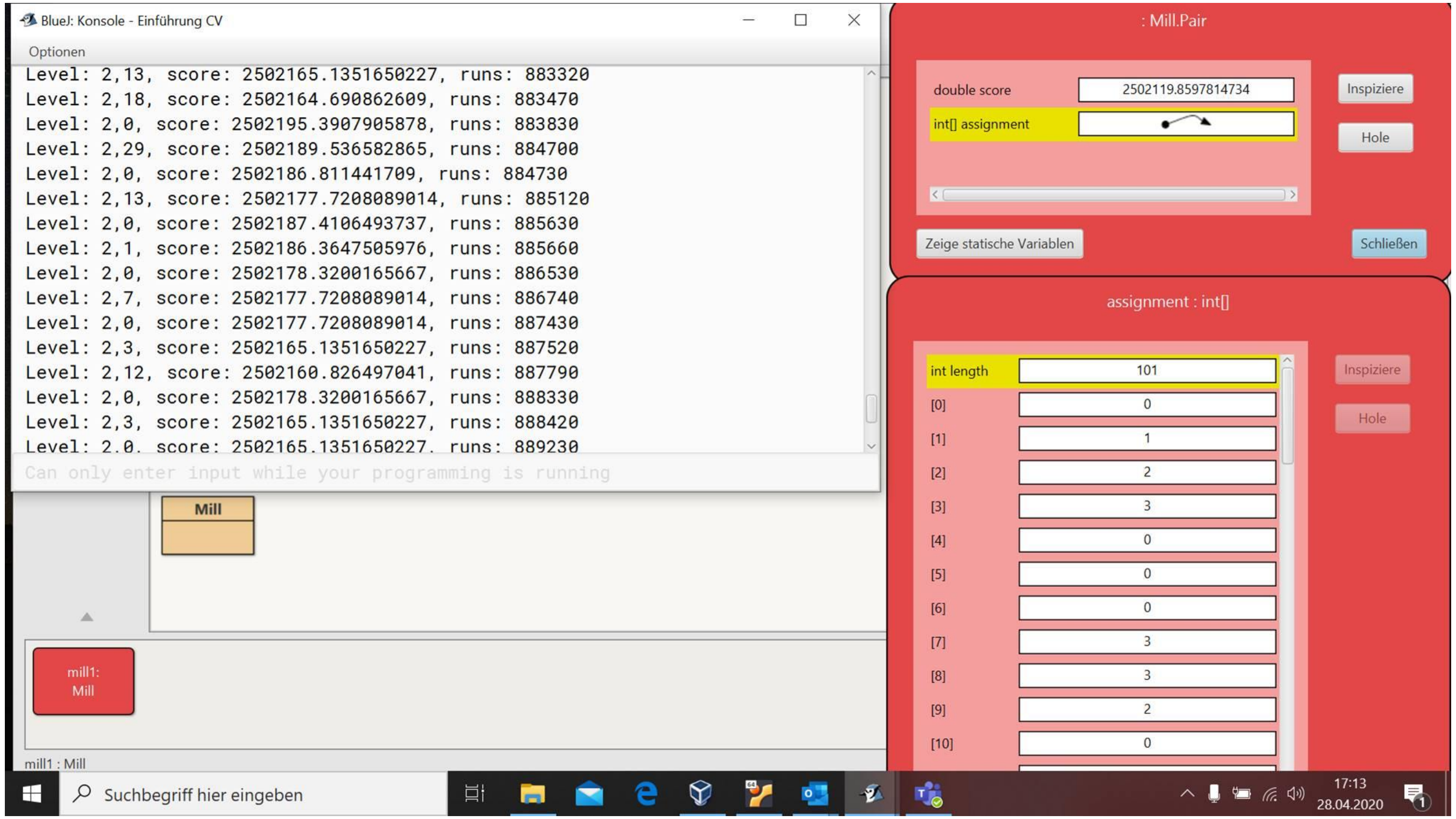
$$\nabla w_{ij} = \frac{\delta C_i}{\delta p_{ib}} \frac{\delta p_{ib}}{\delta w_{ij}} = -\frac{1}{p_{ib}} p_{ib}(\delta_{bj} - p_{ij}) = p_{ij} - \delta_{bj}$$

If we use α as a learning rate we update the weights with:

$$w_{ij} = w_{ij} - \alpha(p_{ij} - \delta_{bj})$$

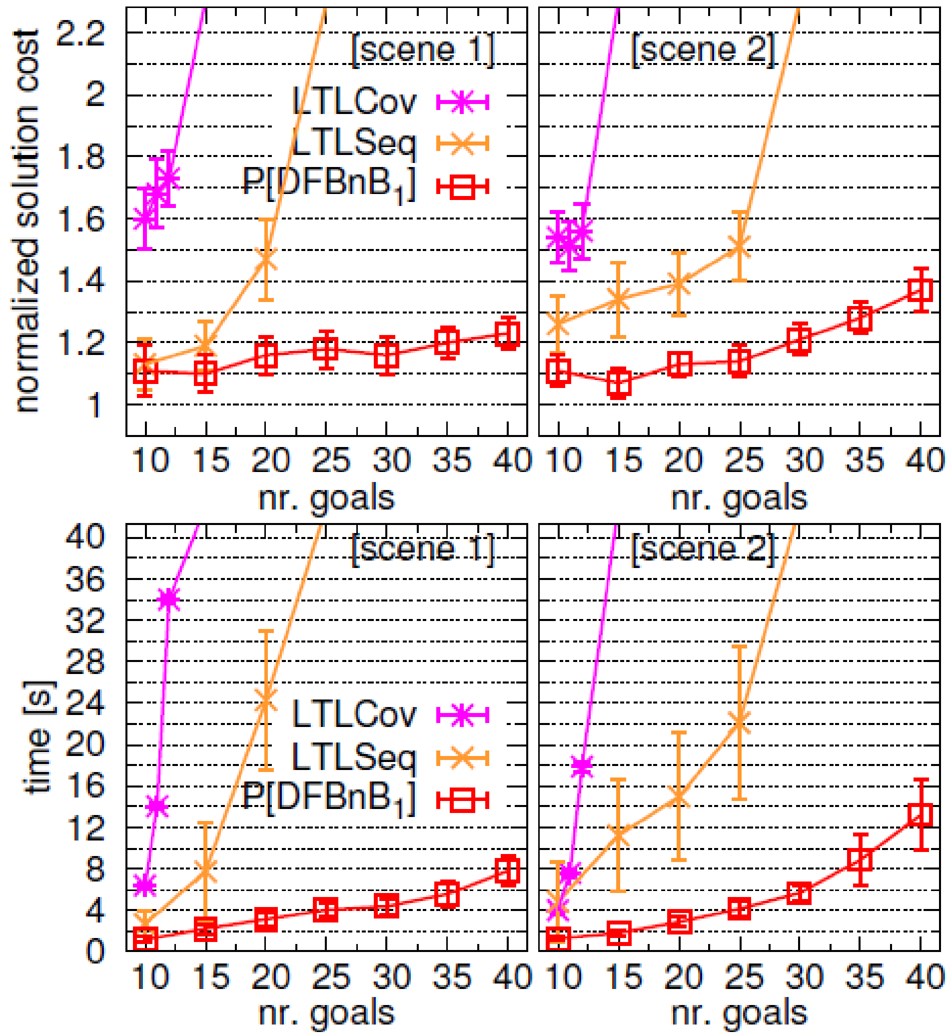
Praxis…

- https://nms.kcl.ac.uk/stefan.edelkamp/lectures/pi1/programs/VRP.java



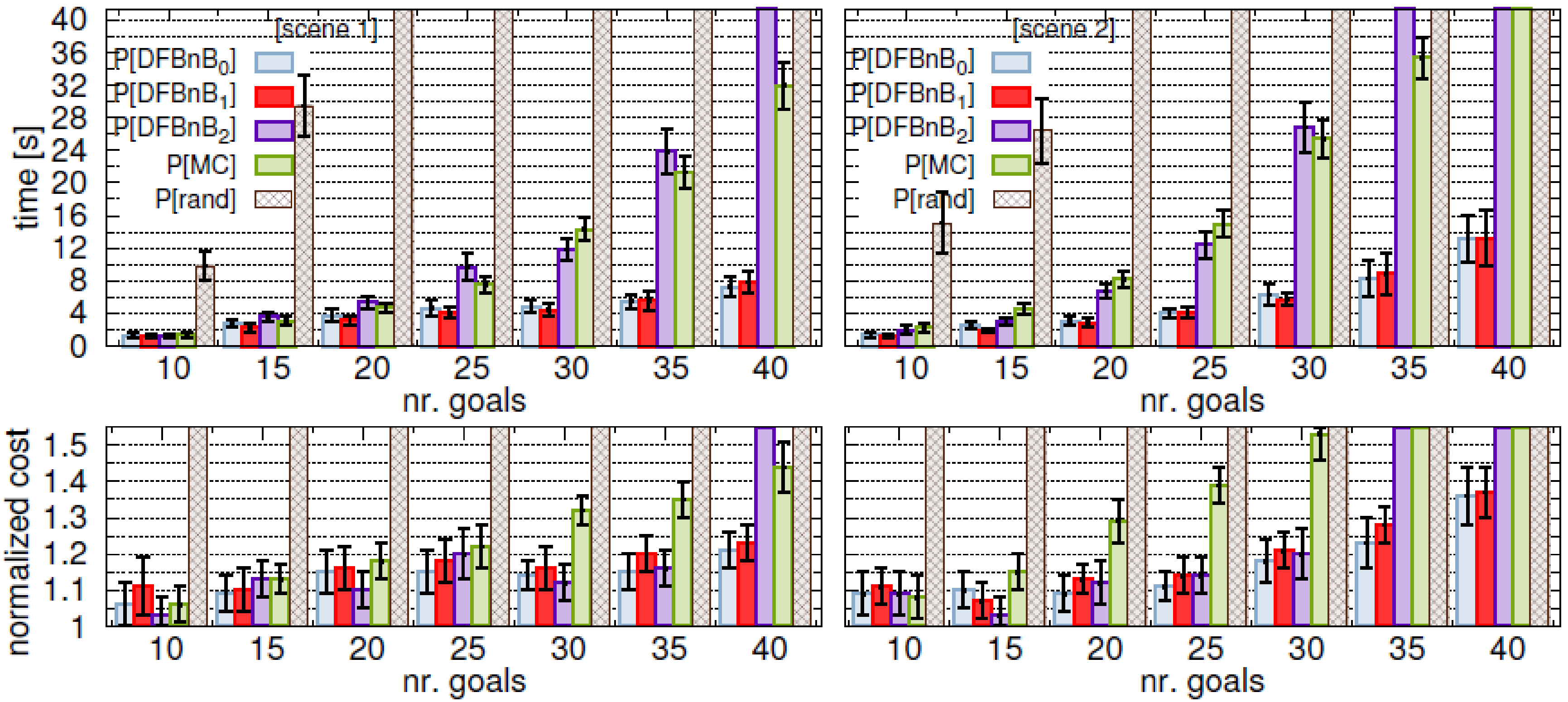
Some Results Multi-Goal

Branch and Bound Search vs. Precurser LTLSyslop



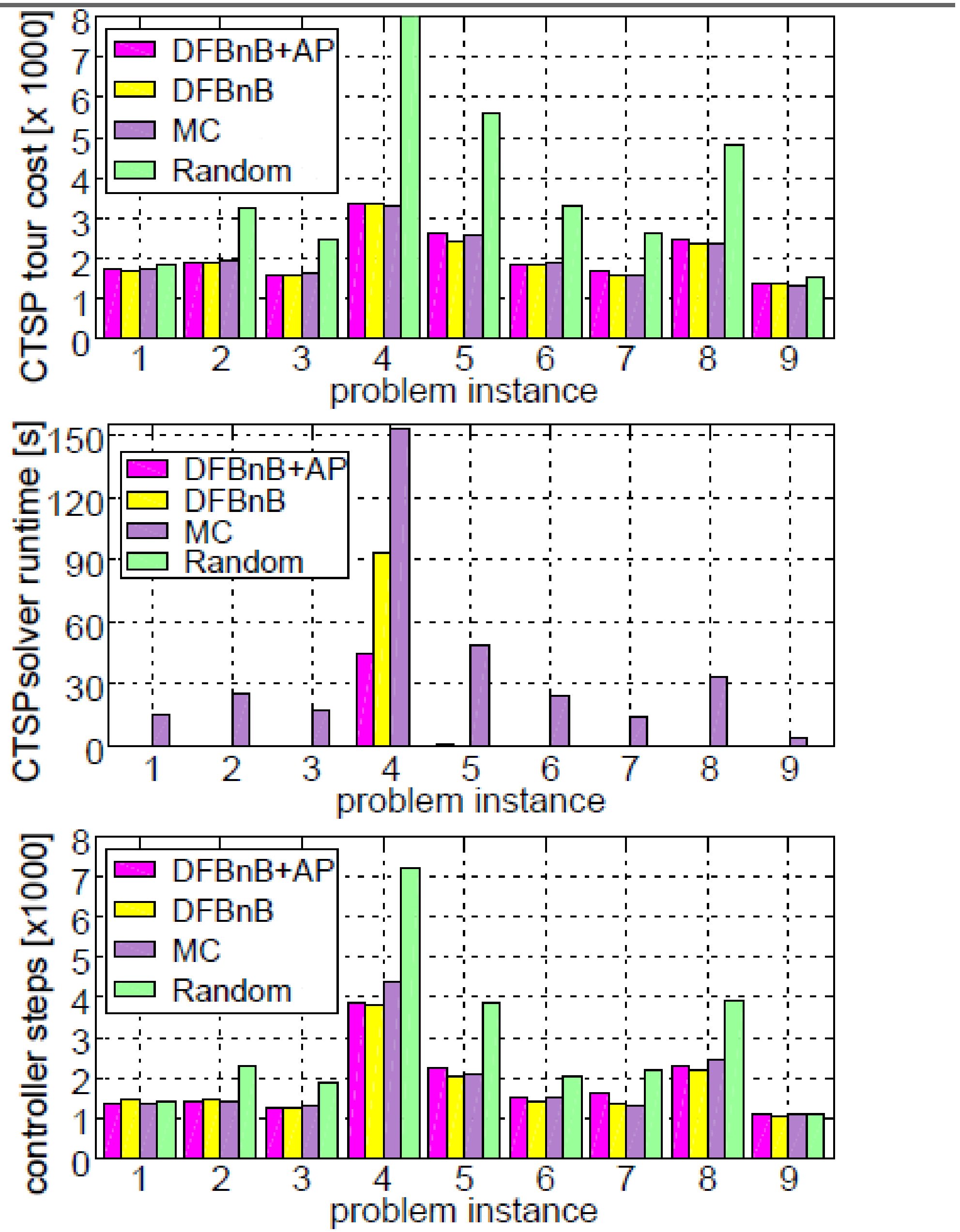
More Results Multi-Goal

Branch and Bound Search with various heuristics and Monte-Carlo Tree Search



Inspection Benchmarks

Results Inspection



Intermediate Summary Multi-Goal Task-Motion Planning

- **Approach makes it possible to consider**
  - ⇒ high-dimensional robotic systems with nonlinear dynamics and nonholonomic constraints
  - ⇒ visit all goal regions fast in suitable cost-minimizing order
  - ⇒ unstructured, complex environments

  and efficiently computes
  - ⇒ collision-free, dynamically-feasible, low-cost trajectories that enable the robot to satisfy the task specification ϕ
- **Offers probabilistic completeness**

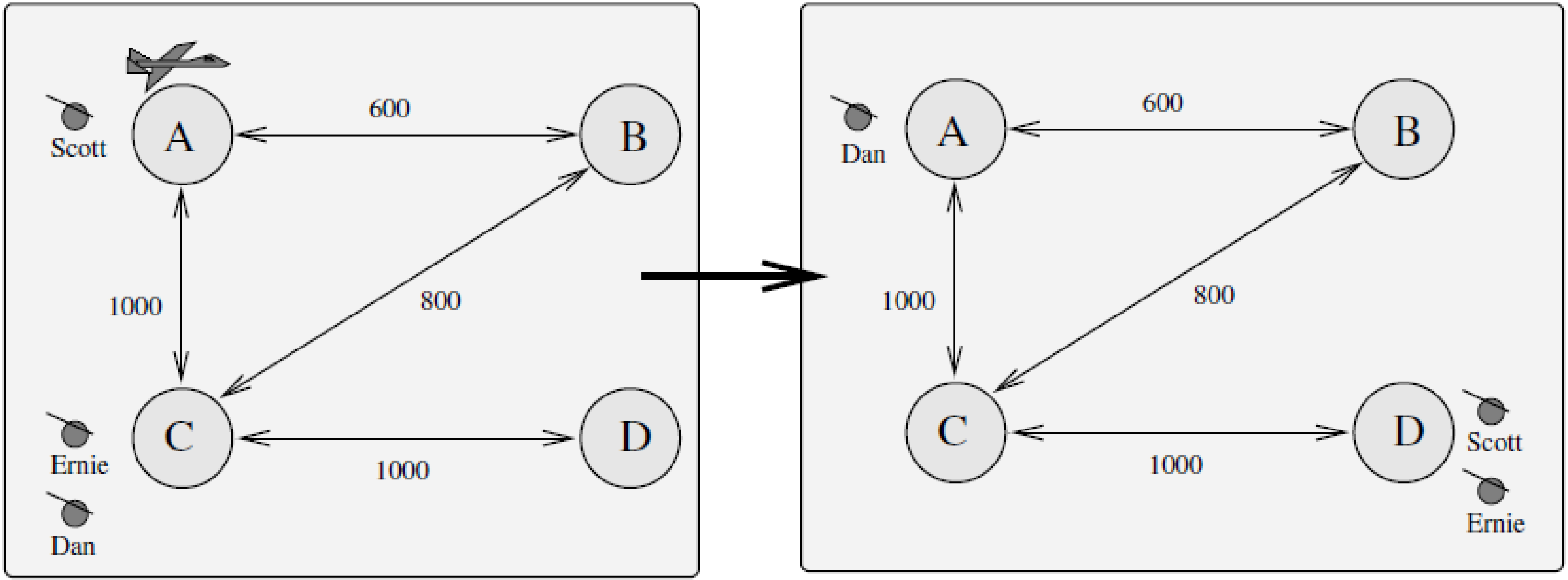
Temporal Planning: Time Does Matter

In general, activities have **varying durations**:

- Loading a package onto a truck is much quicker than driving the truck;
- Drinking a cup of tea takes longer than making it;
- Procrastinating tasks takes longer than doing them
- …

Example: Zeno Domain

Initial State Goal State



Zeno PDDL domain File

```
(define (domain zeno-travel)
(:requirements :durative-actions :typing :fluents)
(:types aircraft person city)
(:predicates (at ?x - (either person aircraft) ?c - city)
             (in ?p - person ?a - aircraft))
(:functions (fuel ?a - aircraft) (distance ?c1 - city ?c2 - city)
            (slow-speed ?a - aircraft) (fast-speed ?a - aircraft)
            (slow-burn ?a - aircraft)  (fast-burn ?a - aircraft)
            (capacity ?a - aircraft)   (refuel-rate ?a - aircraft)
            (total-fuel-used) (boarding-time) (debarking-time))
(:durative-action board
 :parameters (?p - person ?a - aircraft ?c - city)
 :duration (= ?duration boarding-time)
 :condition (and (at start (at ?p ?c))
                 (over all (at ?a ?c)))
 :effect (and (at start (not (at ?p ?c)))
              (at end (in ?p ?a))))
[...]
(:durative-action zoom
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :duration (= ?duration (/ (distance ?c1 ?c2) (fast-speed ?a)))
 :condition (and (at start (at ?a ?c1))
                 (at start (>= (fuel ?a) (* (distance ?c1 ?c2) (fast-burn ?a)))))
 :effect (and (at start (not (at ?a ?c1)))
              (at end (at ?a ?c2))
              (at end (increase total-fuel-used
                         (* (distance ?c1 ?c2) (fast-burn ?a))))
              (at end (decrease (fuel ?a)
                         (* (distance ?c1 ?c2) (fast-burn ?a))))))
```

Zeno PDDL Problem FILE

```
(define (problem zeno-travel-1)
    (:domain zeno-travel)
    (:objects plane - aircraft
              ernie scott dan - person
              city-a city-b city-c city-d - city)
    (:init (= total-fuel-used 0) (= debarking-time 20) (= boarding-time 30)
           (= (distance city-a city-b) 600)  (= (distance city-b city-a) 600)
           (= (distance city-b city-c) 800)  (= (distance city-c city-b) 800)
           (= (distance city-a city-c) 1000) (= (distance city-c city-a) 1000)
           (= (distance city-c city-d) 1000) (= (distance city-d city-c) 1000)
           (= (fast-speed plane) (/ 600 60)) (= (slow-speed plane) (/ 400 60))
           (= (fuel plane) 750)              (= (capacity plane) 750)
           (= (fast-burn plane) (/ 1 2))     (= (slow-burn plane) (/ 1 3))
           (= (refuel-rate plane) (/ 750 60))
           (at plane city-a) (at scott city-a) (at dan city-c) (at ernie city-c))
    (:goal (and (at dan city-a) (at ernie city-d) (at scott city-d)))
    (:metric minimize total-time)
)
```

Sequential and TemPORAL PLAN

```
  0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c)     [30]
130: (board ernie plane city-c)   [30]
160: (refuel plane city-c)        [40]
200: (zoom plane city-c city-a) [100]
300: (debark dan plane city-a)    [20]
320: (board scott plane city-a)   [30]
350: (refuel plane city-a)        [40]
390: (zoom plane city-a city-c) [100]
490: (refuel plane city-c)        [40]
530: (zoom plane city-c city-d) [100]
630: (debark ernie plane city-d) [20]
650: (debark scott plane city-d) [20]
```

```
  0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c)     [30]
     (board ernie plane city-c)   [30]
100: (refuel plane city-c)        [40]
140: (zoom plane city-c city-a) [100]
240: (debark dan plane city-a)    [20]
     (board scott plane city-a)   [30]
     (refuel plane city-a)        [40]
280: (zoom plane city-a city-c) [100]
380: (refuel plane city-c)        [40]
420: (zoom plane city-c city-d) [100]
520: (debark ernie plane city-d) [20]
     (debark scott plane city-d) [20]
```

SNAG: Sequential Plan TIME VS. Parallel Plan TIME

```
(zoom city-a city-c plane), (board dan plane city-c),
(refuel plane city-c), (zoom city-c city-a plane),
(board scott plane city-a), (debark dan plane city-a), (refuel plane city-a),
```

and

```
(board scott plane city-a), (zoom city-a city-c plane),
(board dan plane city-c), (refuel plane city-c),
(zoom city-c city-a plane), (debark dan plane city-a), (refuel plane city-a)
```

Different Plan OBJECTIVES

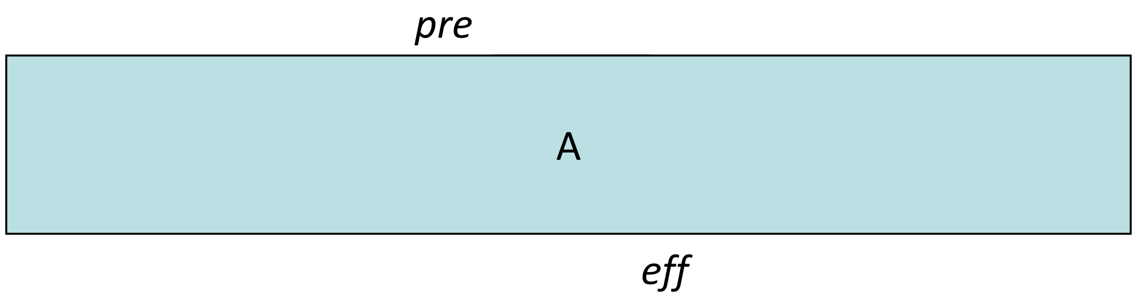
Fuel

```
     0: (board scott plane city-a) [30]
    30: (fly plane city-a city-c) [150]
   180: (board ernie plane city-c) [30]
        (board dan plane city-c)   [30]
   210: (fly plane city-c city-a) [150]
   360: (debark dan plane city-a)  [20]
        (refuel plane city-a)   [53.33]
413.33: (fly plane city-a city-c) [150]
563.33: (fly plane city-c city-d) [150]
713.33: (debark ernie plane city-d)[20]
        (debark scott plane city-d)[20]
```

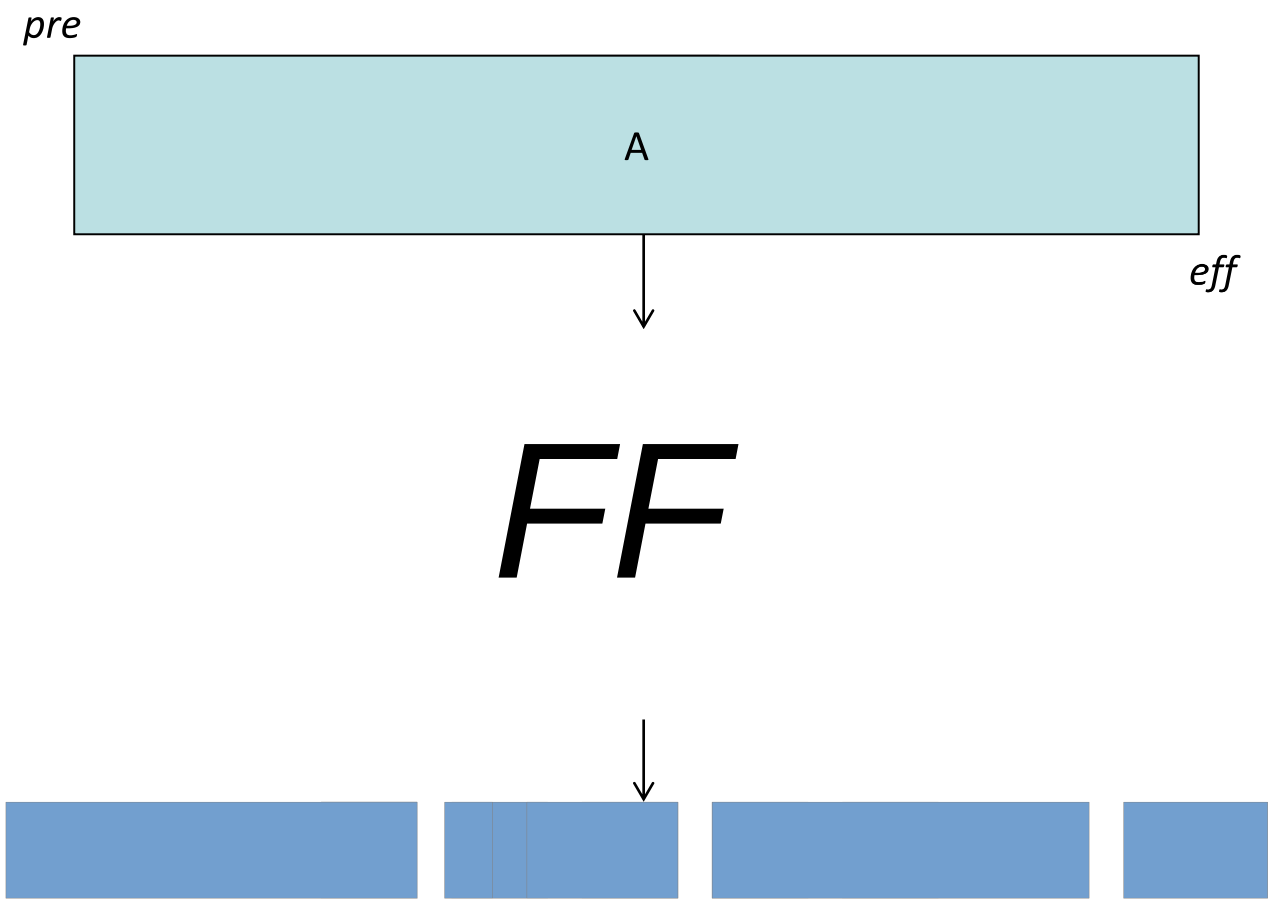
Time

```
  0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c)    [30]
     (board ernie plane city-c)  [30]
     (refuel plane city-c)       [40]
140: (zoom plane city-c city-a) [100]
240: (debark dan plane city-a)   [20]
     (board scott plane city-a)  [30]
     (refuel plane city-a)       [40]
280: (fly plane city-a city-c)  [150]
430: (fly plane city-c city-d)  [150]
580: (debark ernie plane city-d) [20]
     (debark scott plane city-d) [20]
```

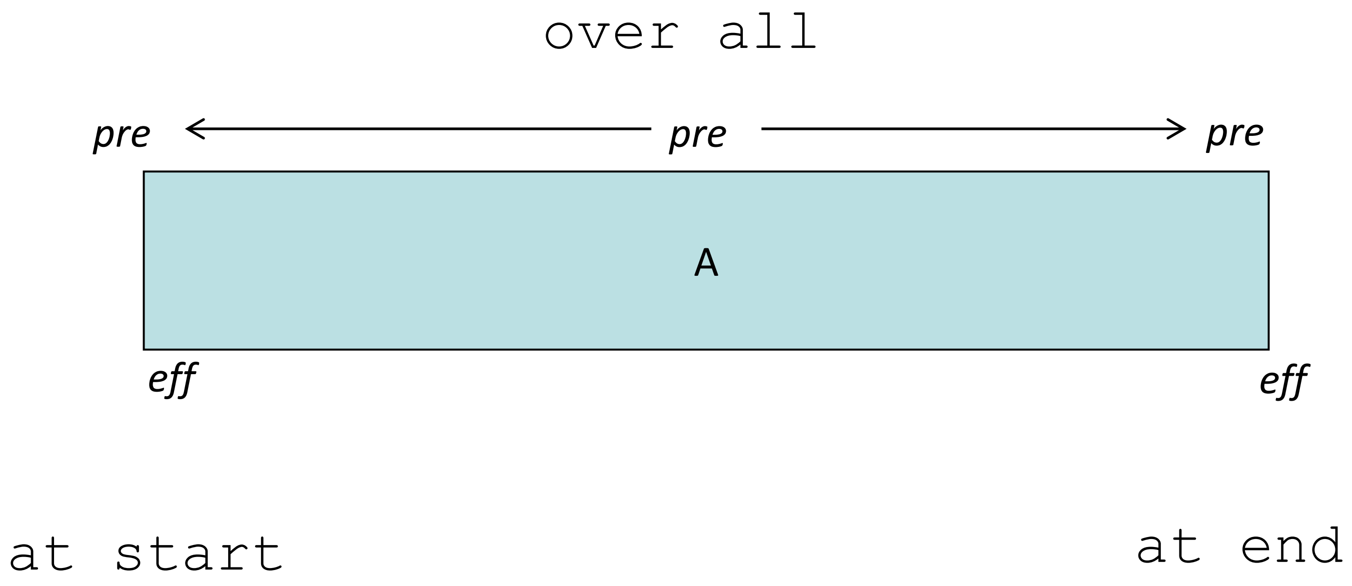
Durative Actions?



Durative Actions?



Durative Actions in PDDL 2.1



PDDL Example (i)

- ve-action LOAD-TRUCK
- :parameters
- (?obj – obj ?truck – truck ?loc - location)
- :precondition (= ?duration 2)
- :cond
- (and ll (at ?truck
- (at start (at ?obj ?loc)))
- :effect
- (ar rt (not (at ?obj
- (at end (in ?obj ?truck))))

Beware of self-overlapping actions!

PDDL Example (ii)

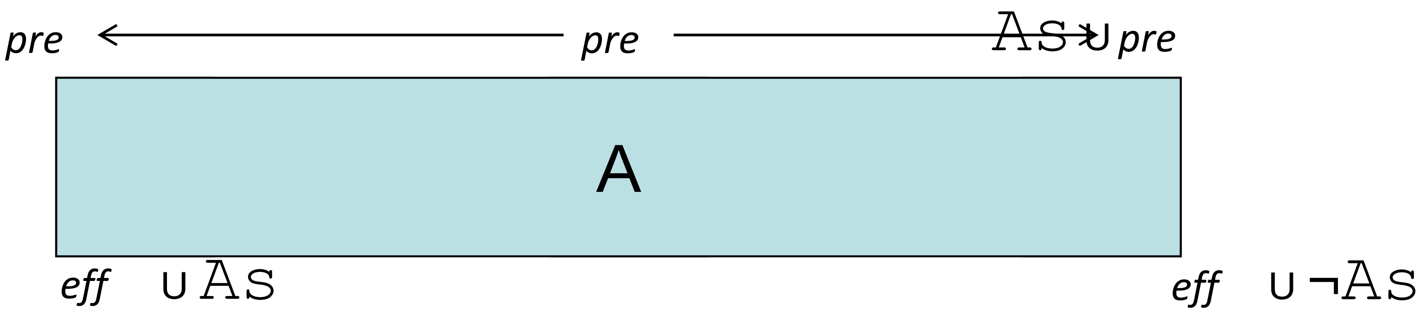
- (:durative-action open-barrier
- :parameters
- (?loc – location ?p - person)
- :duration (= ?duration 1)
- :condition
- (and (at start (at ?loc ?p)))
- :effect
- (and (at start (door-open ?loc))
- (at end (not (door-open ?loc))))

PDDL Example (ii)

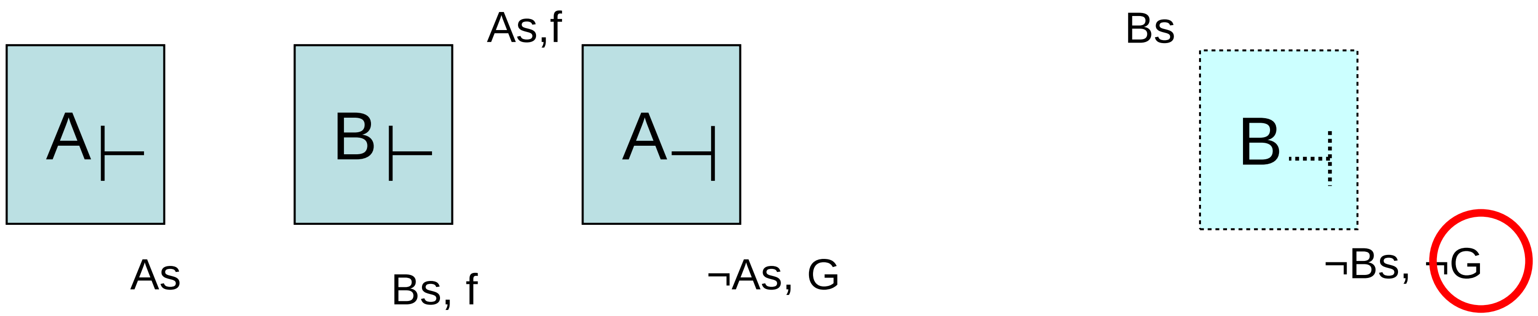
- (:durative-actio
-   :parameters
- (?loc - locatio
-   :duration (= ?
-   :condition
-    (and   (at st
-   :effect
-    (and (at start (door-open ?loc))
-          (at end (not (door-open ?loc))))

Durative Actions

(Fox and Long, ICAPS 2003)



Planning with Snap Actions (i)

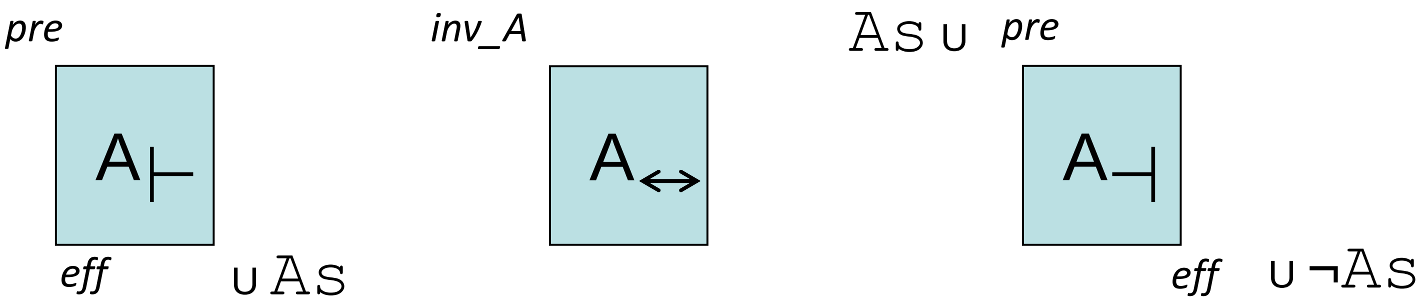


- Challenge 1: What if B⊣ interferes with the goal?

@PDDL 2.1 semantics: **no actions can be executing in a goal state.**

Solution: add ¬As, ¬Bs, ¬Cs.... to the goal (or make this implicit in a temporal planner.)

Planning with Snap Actions (ii)



Challenge 2: what about **over all** conditions?

If A is executing, inv_A must hold.

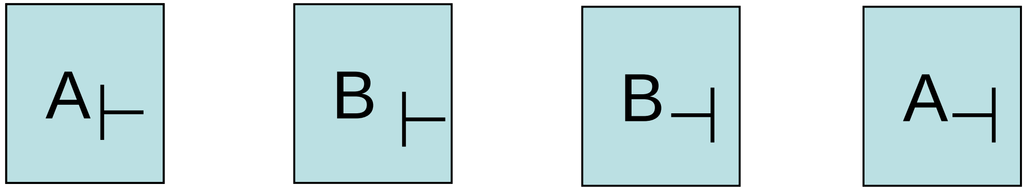
Solution:

In every state where As is true: inv_A must also be true

Or: `(imply (As) inv_A)`

Violating an invariant then leads to a **dead-end**.

Planning with Snap Actions (iii)



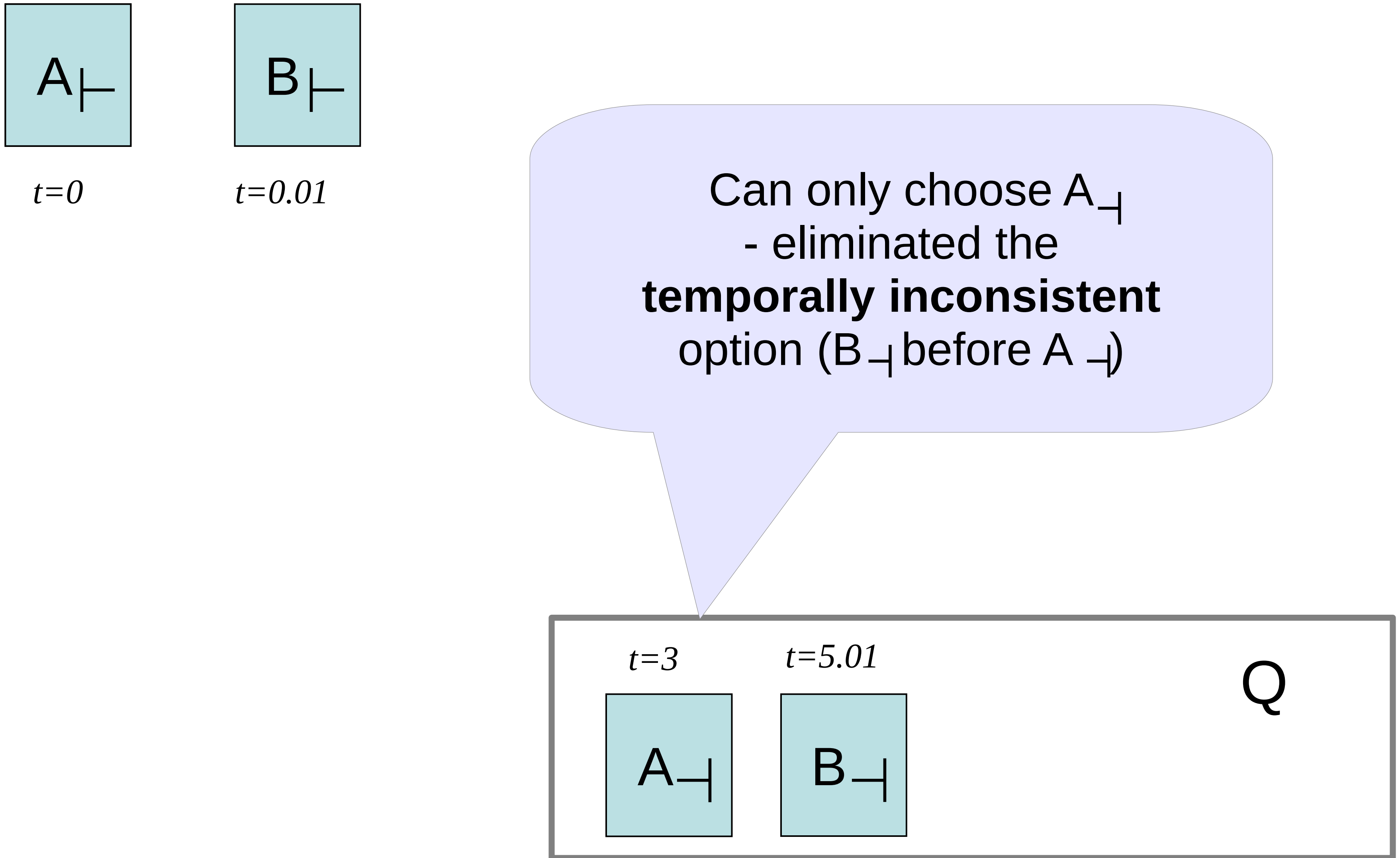
- Challenge 3: **where did the durations go?**
  - More generally, what are the temporal constraints?
  - **Logically sound ≠ temporally sound.**

Option 1: Decision Epoch Planning

Term from Cushing et al, IJCAI 2007

- Search with **time-stamped states** and a **priority queue** of pending end snap-actions.
  - See Temporal Fast Downward (Eyerich, Mattmüller and Röger, ICAPS 2009); Sapa (Do and Kambhampati, JAIR 2003), and others.
- In a state S, at time t and with queue Q, either:
  - Apply a start snap-action $A_{\vdash}$ (at time t)
    - Insert $A_{\dashv}$ into Q at time $(t + dur(A))$
    - $S'.t = S.t + \varepsilon$
  - Remove and apply the first end snap-action from Q.
    - $S'.t$ set to the scheduled time of this, plus ε

Running through our example...

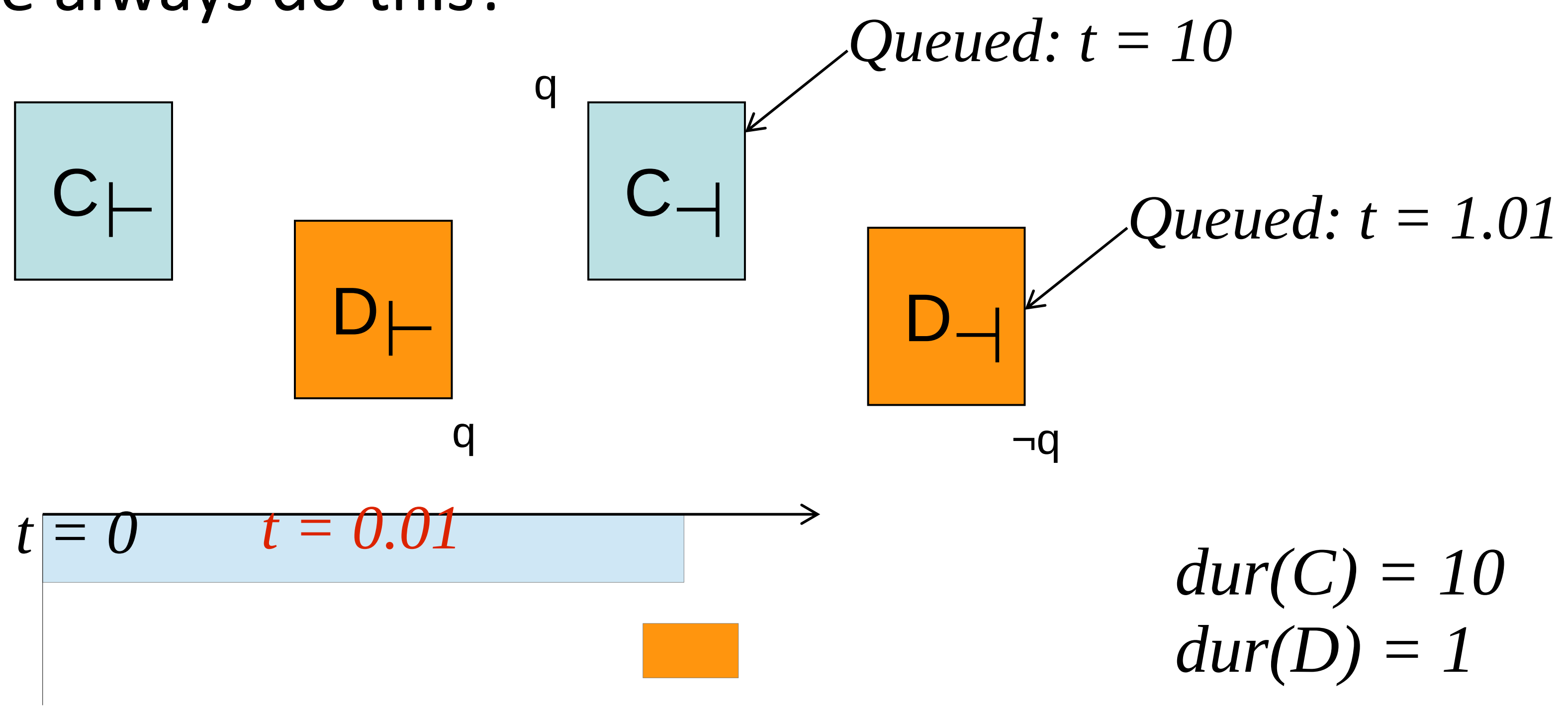


Decision Epoch Planning: The snag

Must **fix start- and end-timestamps** at the point when the action is started.

– Used for the priority queue

Can we always do this?



OPTION 2: Simple Temporal Networks

"Planning with Problems Requiring Temporal Coordination." A. I. Coles, M. Fox, D. Long, and A. J. Smith. AAAI 08.
https://local.cis.strath.ac.uk/research/publications/papers/strath_cis_publication_2248.pdf
"Managing concurrency in temporal planning using planner-scheduler interaction."
A. I. Coles, M. Fox, K. Halsey, D. Long, and A. J. Smith. Artificial Intelligence. 173 (1). 2009.

a Simple Temporal Problem?

- All our constraints are of the form:
- $\varepsilon \le t(i+1) - t(i)$ *(c.f. sequence constraints)*
- $dur_{min}(A) \le t(A_{\dashv}) - t(A_{\vdash}) \le dur_{max}(A)$
- Or, more generally, $lb \le t(j) - t(i) \le ub$

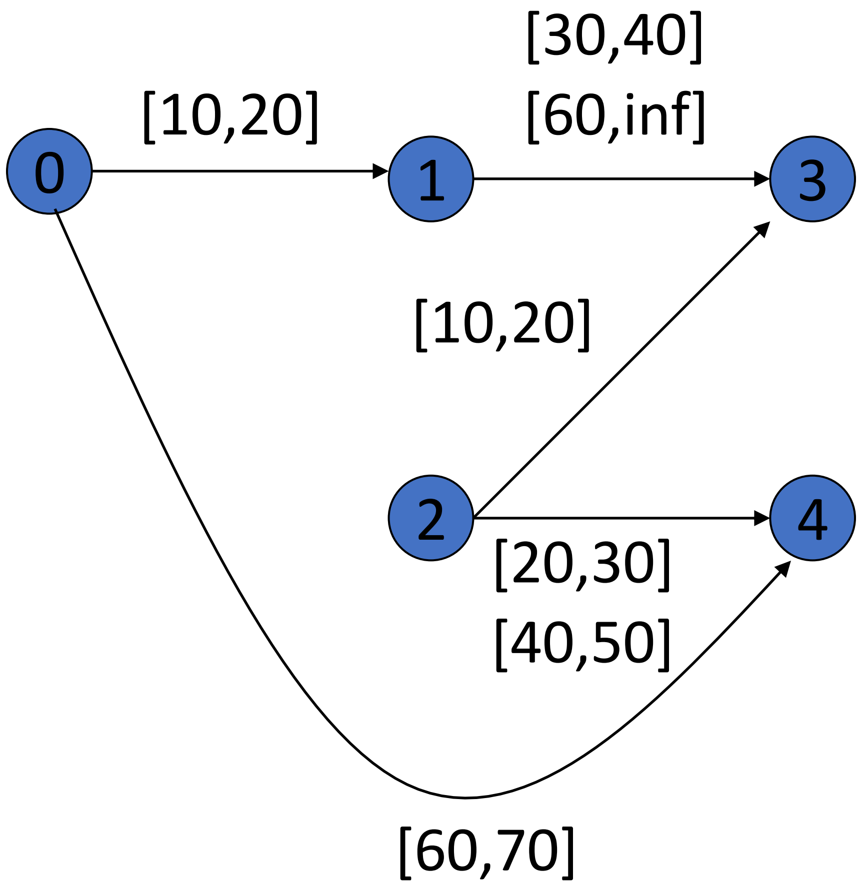
- Is a **Simple Temporal Problem**
- “Temporal Constraint Networks”, Dechter, Meiri and Pearl, AIJ, 1991

- Good news – is **polynomial**
- Bad news – in planning, we need to solve it a lot....

Example

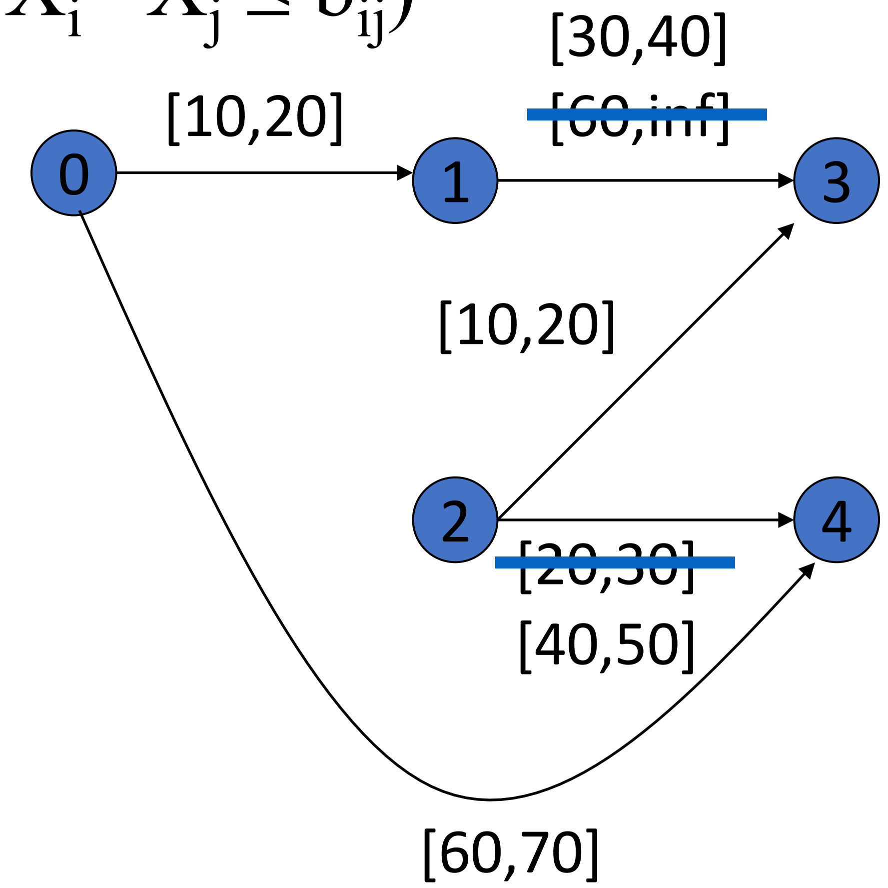
- John travels to work either by car (30-40 min) or by bus (>= 60 min)
- Fred travels to work either by car (20-30 min) or in a carpool (40-50 min)
- Today John left between 7:10 and 7:30am.
- Fred arrived at work between 8:00 and 8:10am.
- John arrived at work 10-20min after Fred left home.

Visualize TCSP as Directed Constraint Graph



Simple Temporal Network

- $T_{ij} = (a_{ij} \leq X_i - X_j \leq b_{ij})$



Simple Temporal Network:

A set of time points X_i at which events occur.

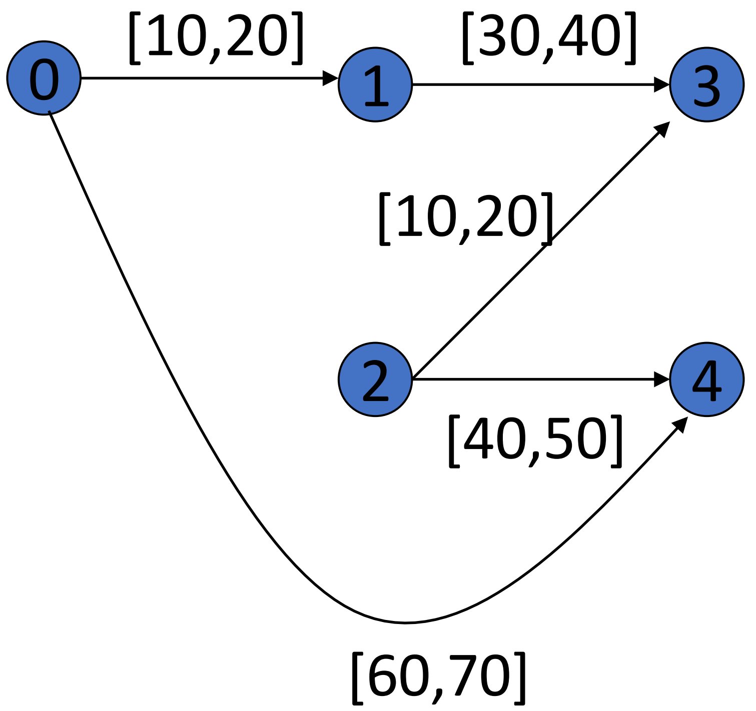
Unary constraints

$(a_0 \le X_i \le b_0)$ or ~~$(a_1 \le X_i \le b_1)$ or . . .~~

Binary constraints

$(a_0 \le X_j - X_i \le b_0)$ or ~~$(a_1 \le X_j - X_i \le b_1)$ or . . .~~

STN



Shostak (1981) A simple temporal problem is consistent if and only if the distance graph has no cycles.

→ The consistency and the minimal network of an STP can be determined in cubic time using all-pairs shortest path search.

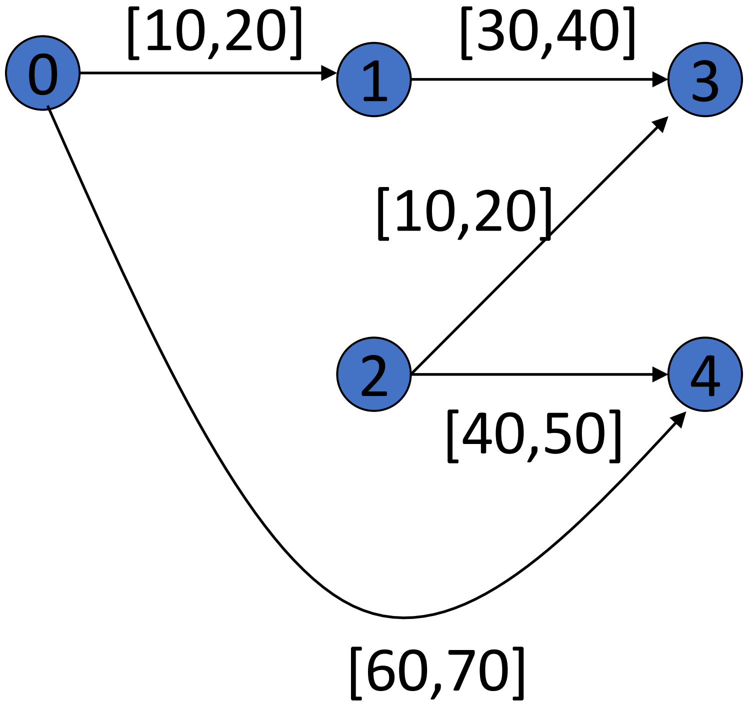
To Query STN Map to Distance Graph G_d

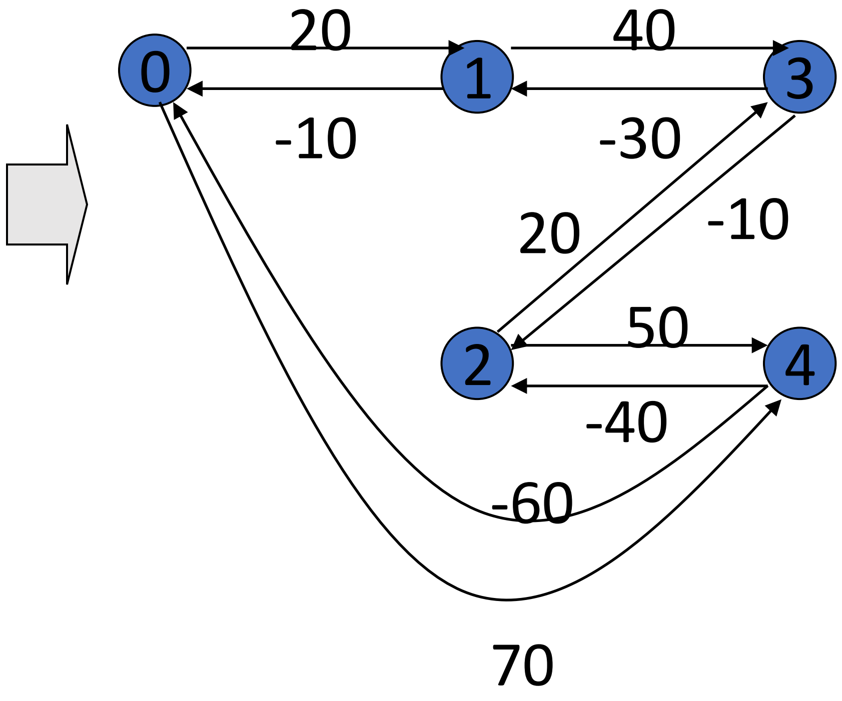
Edge encodes an upper bound on distance to target from source.

$$T_{ij} = (a_{ij} \leq X_j - X_i \leq b_{ij})$$

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

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

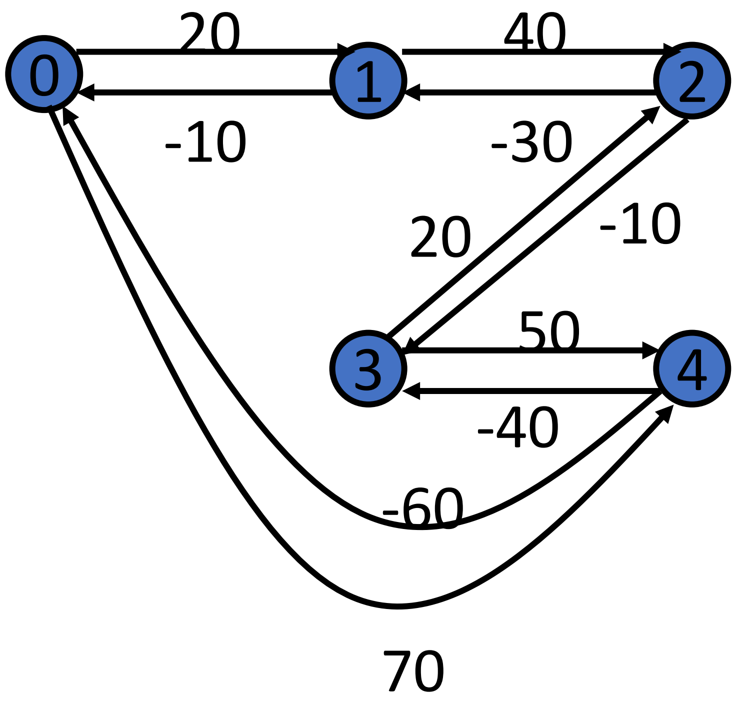




Shortest Paths of G_d

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

d-graph



STN Minimum Network

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

d-graph

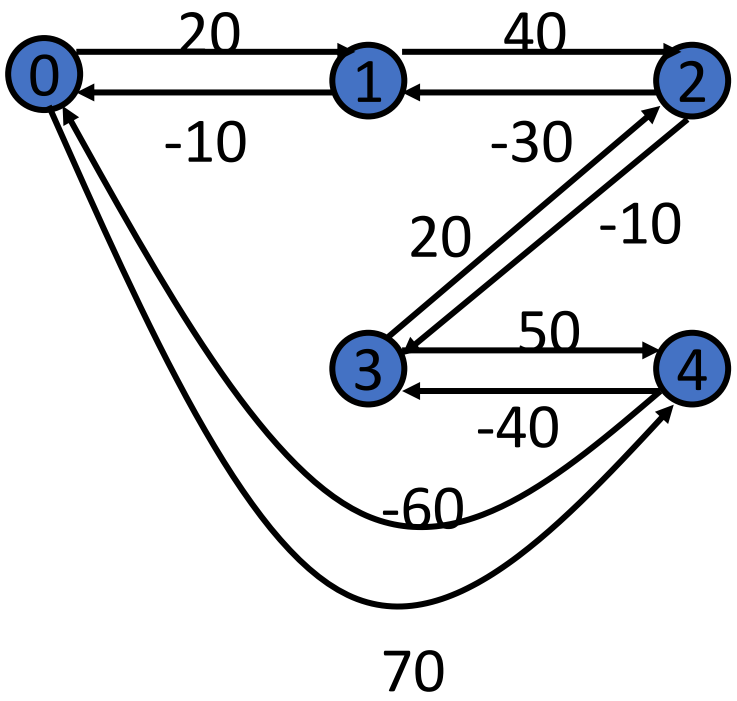
	0	1	2	3	4
0	[0]	[10,20]	[40,50]	[20,30]	[60,70]
1	[-20,-10]	[0]	[30,40]	[10,20]	[50,60]
2	[-50,-40]	[-40,-30]	[0]	[-20,-10]	[20,30]
3	[-30,-20]	[-20,-10]	[10,20]	[0]	[40,50]
4	[-70,-60]	[-60,-50]	[-30,-20]	[-50,-40]	[0]

STN minimum network

Test Consistency:
No Negative Cycles

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

d-graph

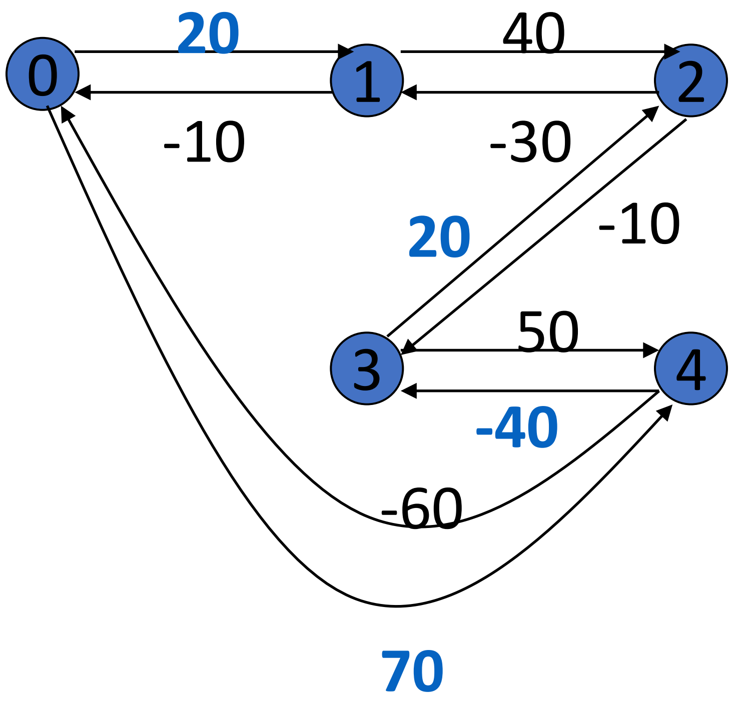


Latest Solution

Node 0 is the reference.

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

d-graph

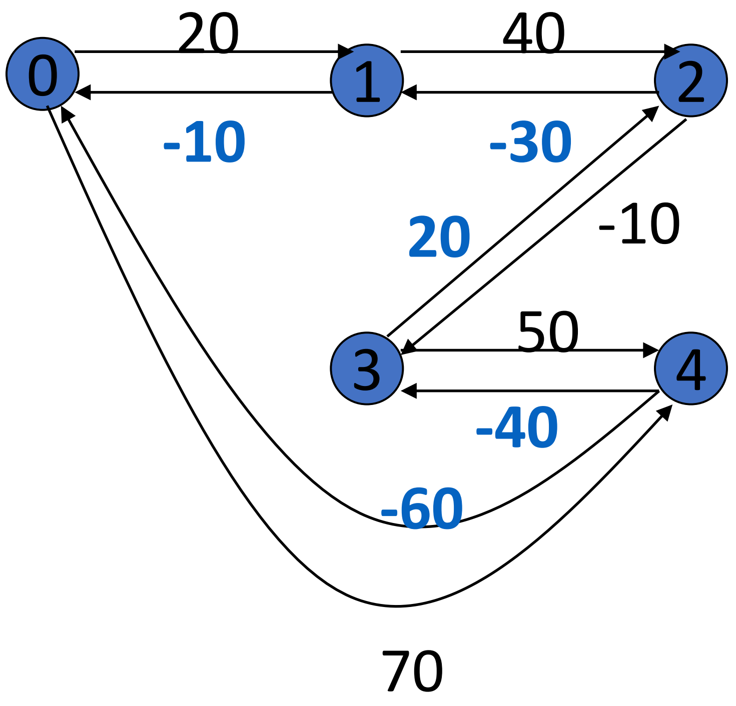


Earliest Solution

Node 0 is the reference.

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

d-graph



Feasible Values

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

d-graph

- X_1 in [10, 20]
- X_2 in [40, 50]
- X_3 in [20, 30]
- X_4 in [60, 70]

Back to Planning: Latest possible times? (Maximum Separation)

$t(A) - t(Z) <= 4$
$t(B) - t(Z) <= 8$
$t(C) - t(Z) <= 10$

('A comes no more than 4 time units after Z')

Latest possible times?

$t(A) - t(Z) \leq 4$
$t(B) - t(Z) \leq 8$
$t(C) - t(Z) \leq 10$

$t(B) - t(A) \leq 2$
$t(C) - t(B) \leq 1$

('B comes no more than 2 time units after A')

Earliest possible times? (Minimum Separation)

- For latest possible time: find the **shortest path**
- For earliest possible times...?

Earliest possible times?

$$2 \le t(A) - t(Z)$$
$$4 \le t(B) - t(Z)$$

$$3 \le t(B) - t(A)$$
$$1 \le t(C) - t(B)$$

Hacking algorithms

- Longest path from Z to C?
- = Shortest **negative** path from C to Z

$2 <= t(A) - t(Z)$

Multiply both sides by -1:

$-2 > - t(A) + t(Z)$

p >= q is the same as q <= p:

$- t(A) + t(Z) < -2$

Rearrange LHS:

$t(Z) - t(A) < -2$

Earliest possible times?

$$2 \le t(A) - t(Z) \qquad -2 \ge -t(A) + t(Z) \qquad t(Z) - t(A) \le -2$$

$$4 \le t(B) - t(Z) \qquad -4 \ge -t(B) + t(Z) \qquad t(Z) - t(B) \le -4$$

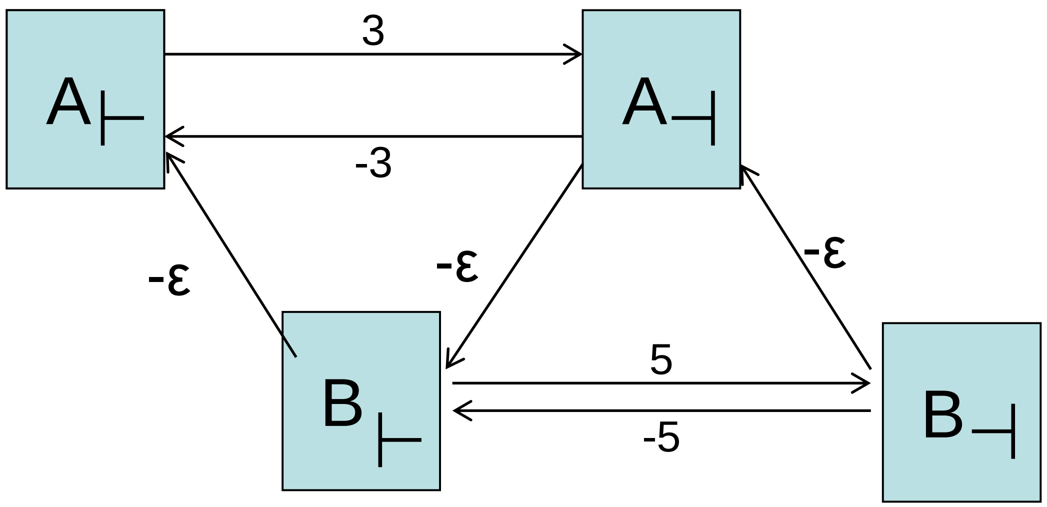
$$3 \le t(B) - t(A) \qquad -3 \ge -t(B) + t(A) \qquad t(A) - t(B) \le -3$$

$$1 \le t(C) - t(B) \qquad -1 \ge -t(C) + t(B) \qquad t(B) - t(C) \le -1$$

Simple Temporal Networks (i)

- Can map STPs to an equivalent digraph:
- One vertex per time-point (and one for 'time zero');
- For $lb \le t(j) - t(i) \le ub$:
- An edge (i $\rightarrow$ j) with weight *ub*.
- An edge (j $\rightarrow$ i), with weight *-lb*
- (c.f. $lb \le t(j) - t(i) \quad \rightarrow \quad t(j) - t(i) \le -lb$)

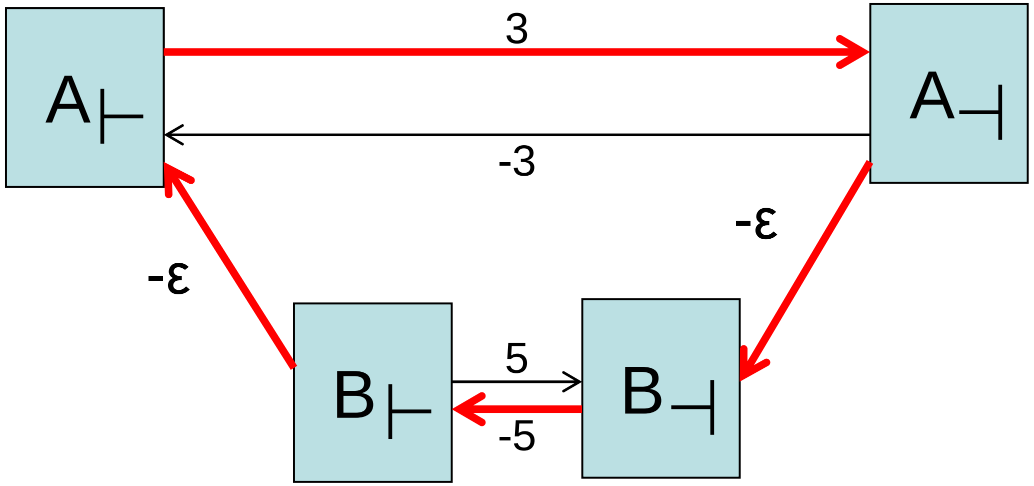
Example STN



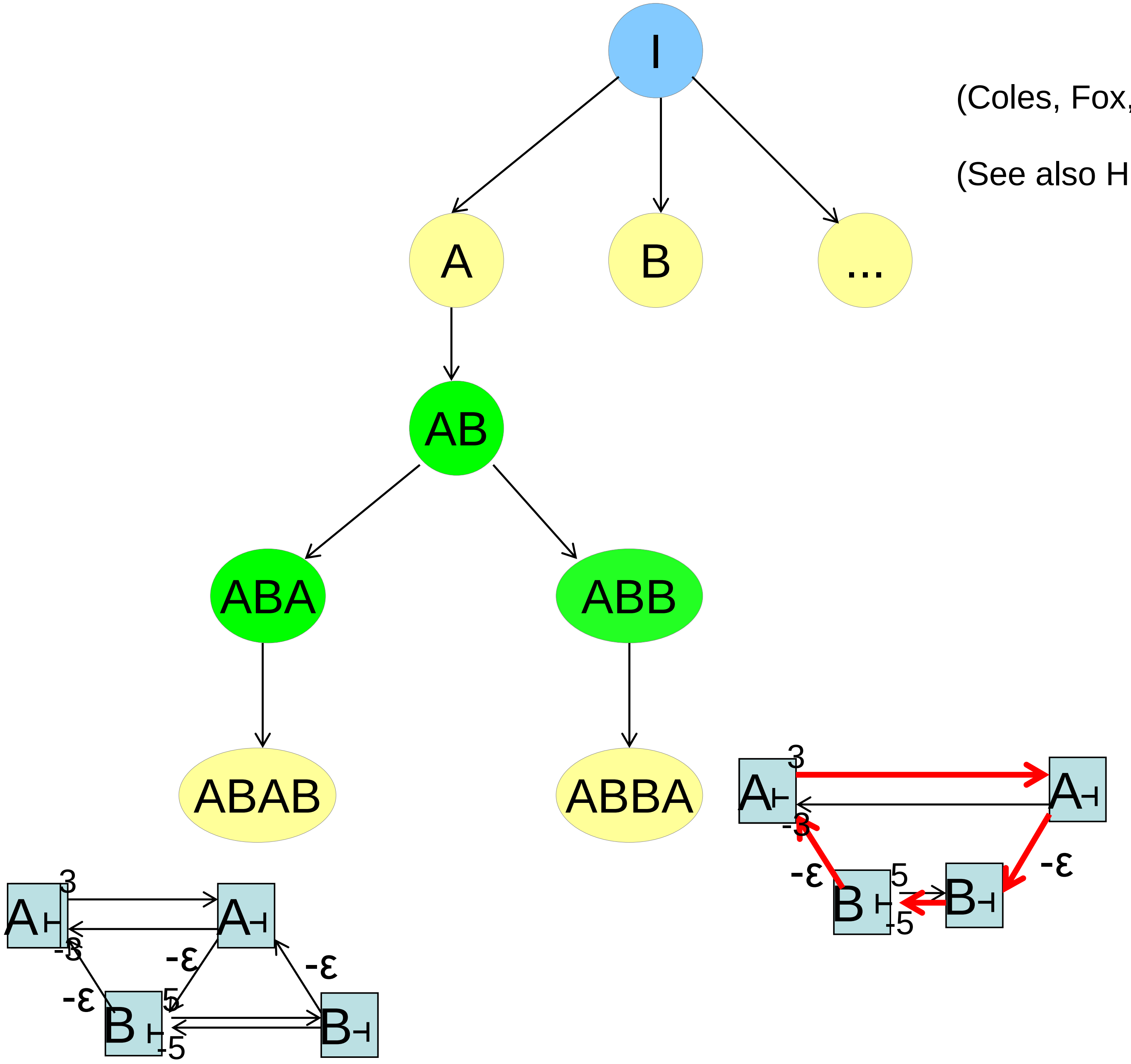

```
0.00: (A) [3]
0.01: (B) [5]
```

Simple Temporal Networks (ii)

- Solve the shortest path problem (e.g. using Bellman-Ford) from/to zero
  - dist(0,j)=x → maximum timestamp of j = x
  - dist(j,0)=y → minimum timestamp of j = -y
- If we find a **negative cycle** then the temporal constraints are inconsistent:



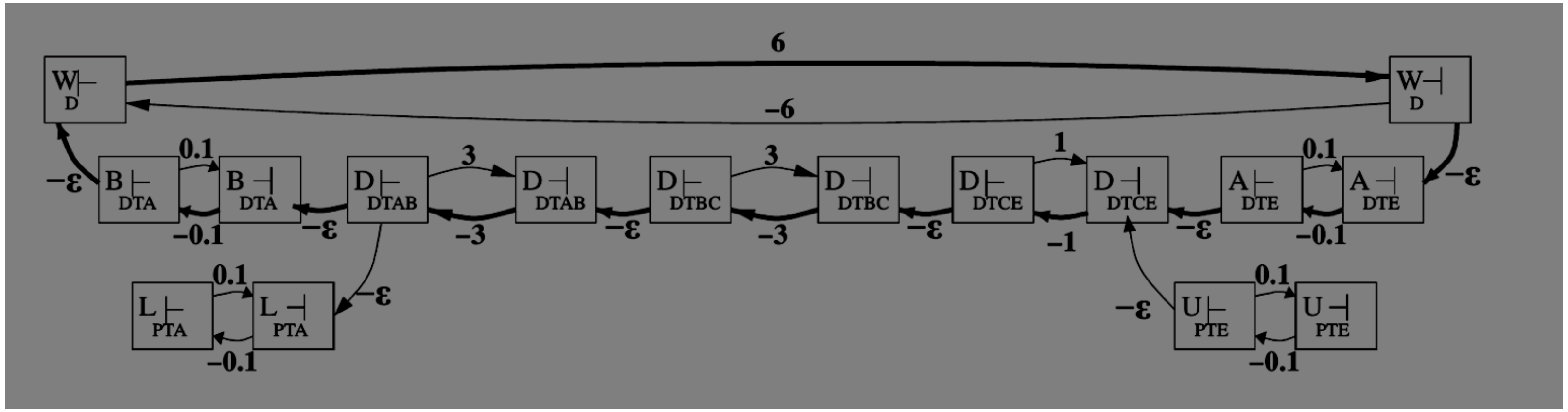
"Incremental Constraint-Posting Algorithms in Interleaved Planning and Scheduling." A. J. Coles, A. I. Coles, M. Fox, and D. Long. Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling, ICAPS09 https://local.cis.strath.ac.uk/research/publications/papers/strath_cis_publication_2409.pdf



(Coles, Fox, Long and Smith, AAAI 2008);

(See also Halsey, Fox and Long, ECAI 2004)

STN Simplifies For Partially Ordered Plans

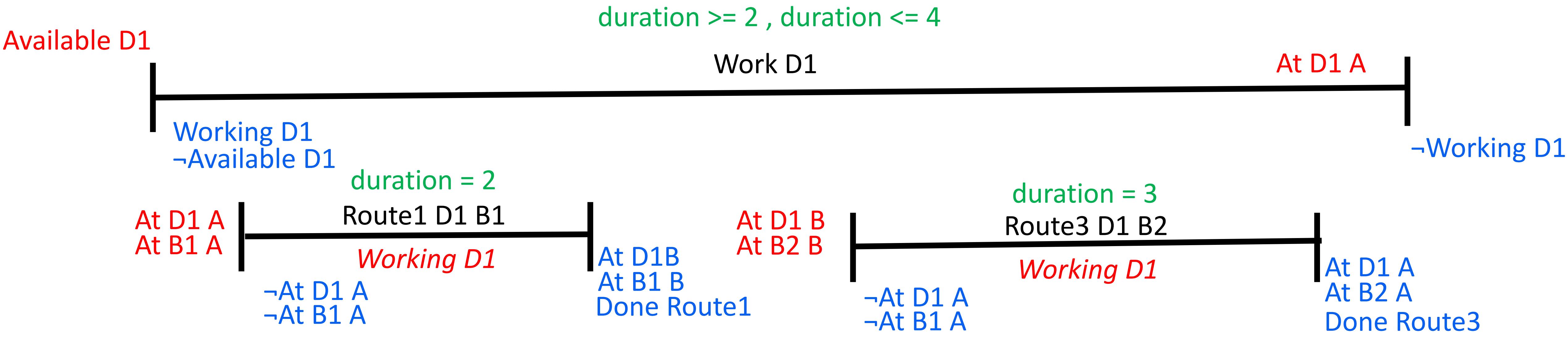


- Transitively implied edges omitted for clarity:
  - e.g. all the drive/board ends before work end;
  - All the drive/board starts after work start.

Public Transport Example

- Drivers have working hours;
- Bus routes have fixed durations and start and end locations.
- Goals are that each bus route is done.
- The routes have timetables that they must follow.

Temporal Planning: Public Transport



Actions have:

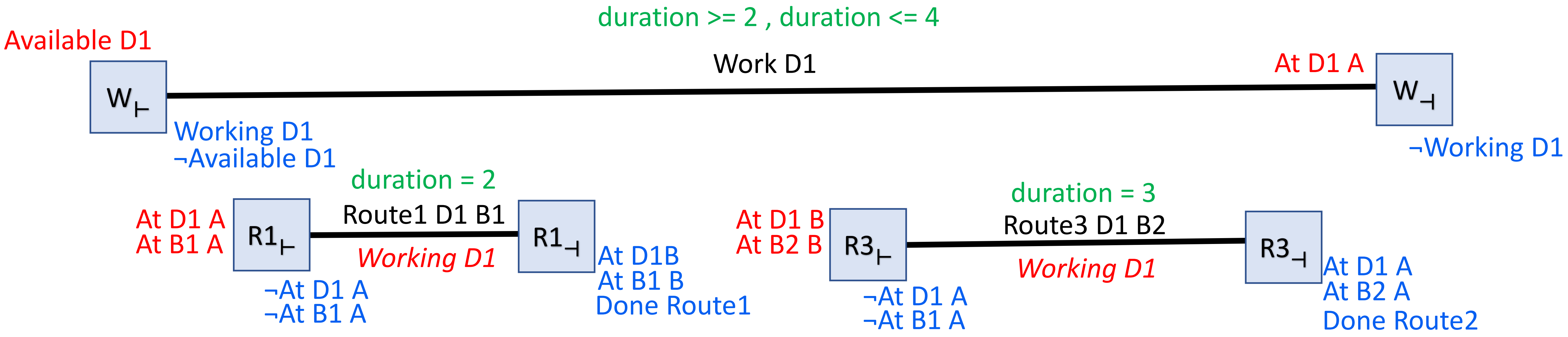
- Conditions and Effects at the start and at the end;
- Invariant/overall conditions;
- Durations constraints:

  (= ?duration 4)

  (and (>= ?duration 2) (<= ?duration 4))

"Planning with Problems Requiring Temporal Coordination." A. I. Coles, M. Fox, D. Long, and A. J. Smith. AAAI 2008.
"Managing concurrency in temporal planning using planner-scheduler interaction." A. I. Coles, M. Fox, K. Halsey, D. Long, and A. J. Smith. Artificial Intelligence. 173 (1) (2009).

Planning with Snap Actions

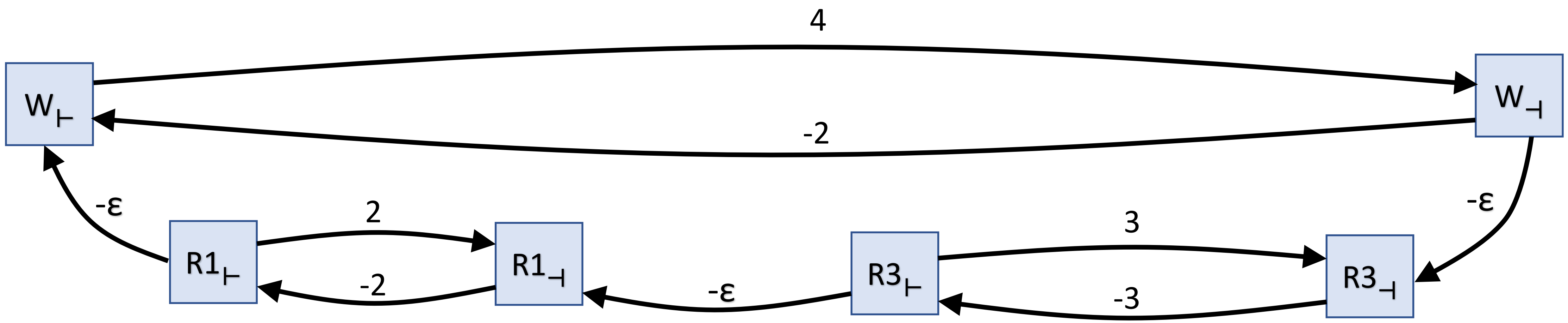


Three Challenges:

- Make sure ends can't be applied unless starts have.
- Overall Conditions.
- Duration constraints.

"Planning with Problems Requiring Temporal Coordination." A. I. Coles, M. Fox, D. Long, and A. J. Smith. AAAI 2008.
"Managing concurrency in temporal planning using planner-scheduler interaction." A. I. Coles, M. Fox, K. Halsey, D. Long, and A. J. Smith. Artificial Intelligence. 173 (1) (2009).

Planning with Snap Actions and STNs



Constraints:

$W_{\dashv} - W_{\vdash} >= 2$

$W_{\dashv} - W_{\vdash} <= 4$

$R1_{\vdash} >= W_{\vdash} + \varepsilon$

$R1_{\dashv} - R1_{\vdash} = 2$

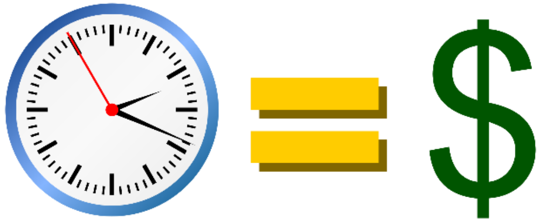
$R3_{\vdash} >= R1_{\vdash} + \varepsilon$

$R3_{\dashv} - R3_{\vdash} = 3$

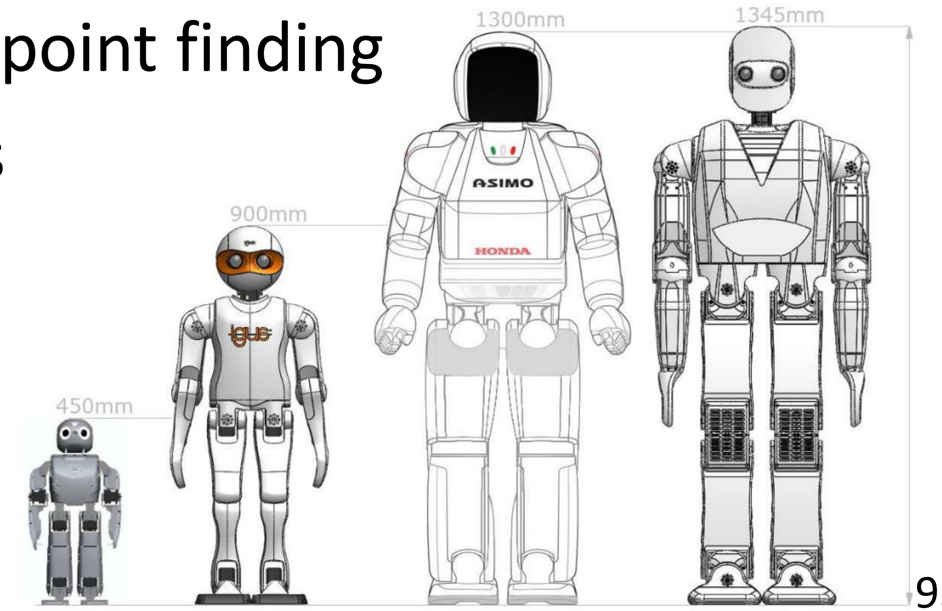
$W_{\dashv} >= R3_{\dashv} + \varepsilon$

"Planning with Problems Requiring Temporal Coordination." A. I. Coles, M. Fox, D. Long, and A. J. Smith. AAAI 2008.
"Managing concurrency in temporal planning using planner-scheduler interaction." A. I. Coles, M. Fox, K. Halsey, D. Long, and A. J. Smith. Artificial Intelligence. 173 (1) 2009.

Temporal Task-Motion Planning



- Time is money
- Real-world has and needs time constraints
- Combining task with motion planning “holy grail” in robotics
- Multiple goals is planning for longer-term plans in form of tours
- Inspection problems can be solved via waypoint finding
- Robots have complex, non-linear dynamics

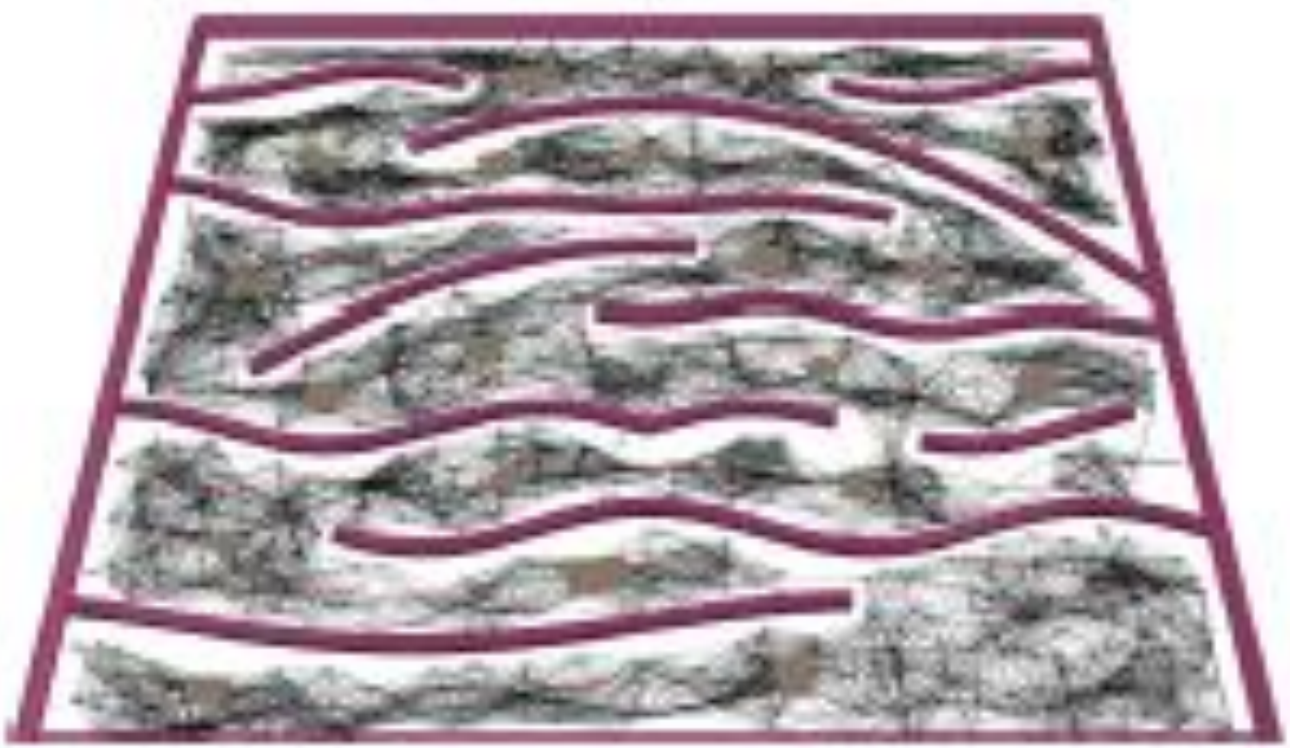


Temporal Task-Motion Planning

- Crucial for Robotics, Logistics, Surgery, VR, …
- No (convincing) solution so far
- High-Level Task, Low-Level Motion Planning
- Solutions in Discrete World only Approximate

  ➔ Replanning needed.

Randomized Roadmaps



PDDL Task Planning and TILS

coffee_errors.pddl

```
(define COFFEE

  (requirements
    :typing)

  (:types room - location
          robot human _ agent
          furniture door - (at ?l - location)
          kettle ?coffee cup water - movable
          location agent movable - object)

  (:predicates (at ?l - location ??o - object)
               (have ?m - movable ?a - agent)
               (hot ?m - movable) = true
               (on ?f - furniture ?m - movable))

  (:action boil
    :parameters (?m - movable $k - kettle ?a - agent)
    :preconditions (have ?m ?a)
    :effect (hot ?m))
```

Line 20, Column 22 Spaces: 2 PDDL

TIL = Timed Initial Literal

(at timepoint (fact))
(at timepoint (not fact))

Specified in initial state

Leads to time windows for actions

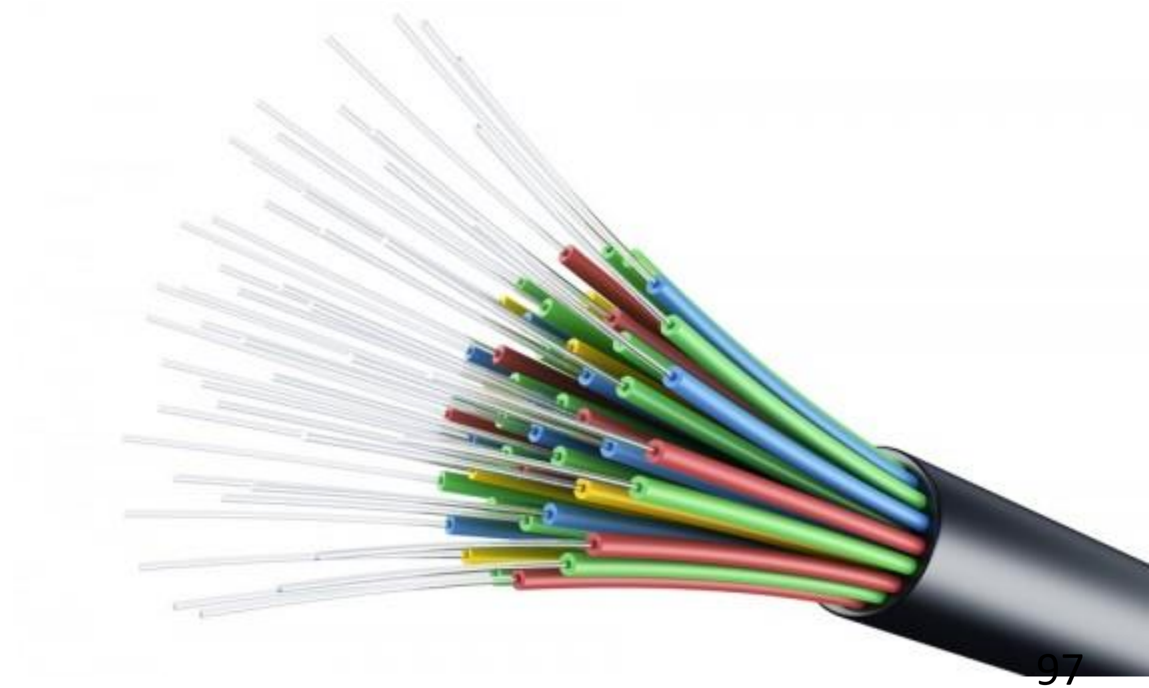
Interface with PDDL Temporal Planner (OPTIC)

Input

```
(at auv v0)
(connected v0 v1)
(connected v0 v2)
(connected v1 v2)
(= (traveltime v0 v1) 0.8)
(= (traveltime v0 v2) 1.5)
(= (traveltime v1 v2) 0.7)
(located task1 v1)
(located task2 v2)
(at 1.1 (tw_open task1))
(at 2.1 (not (tw_open task1)))
(at 2.3 (tw_open task2))
(at 3.3 (not (tw_open task2)))
```

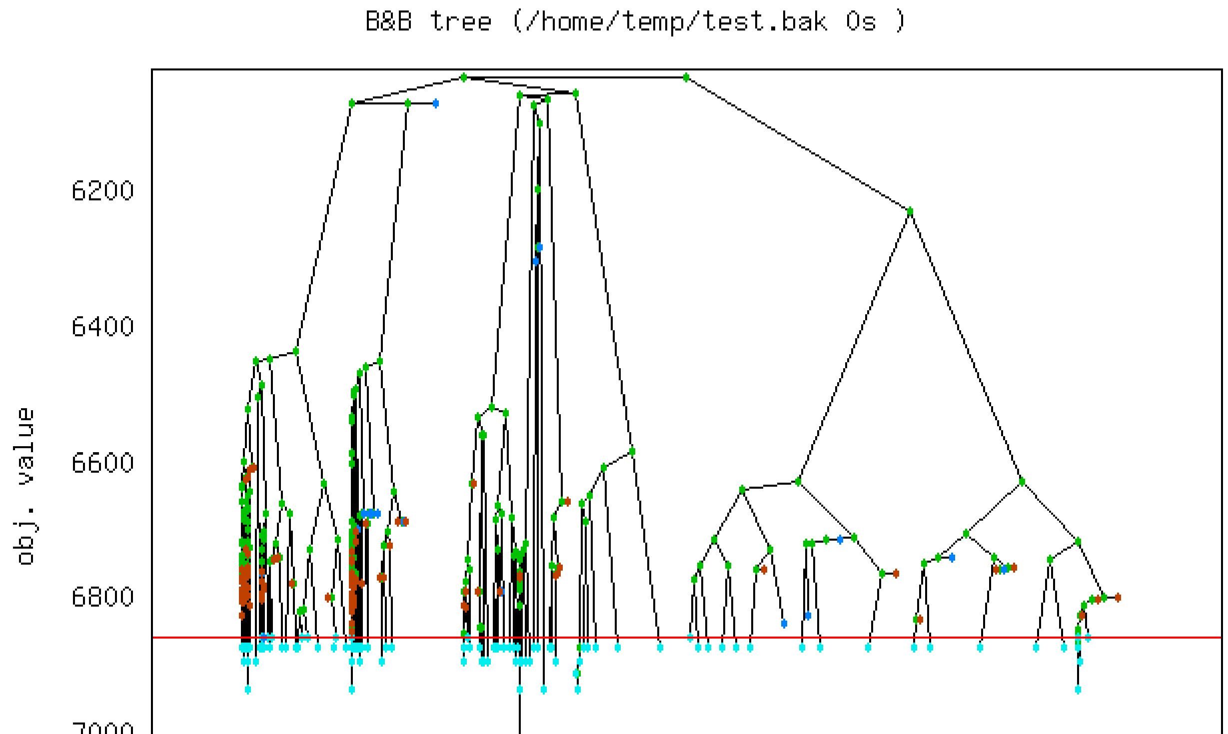
Output

```
v1 v3 v5 v4 v2
0.0  1.26  3.22  12.55  21.11
```

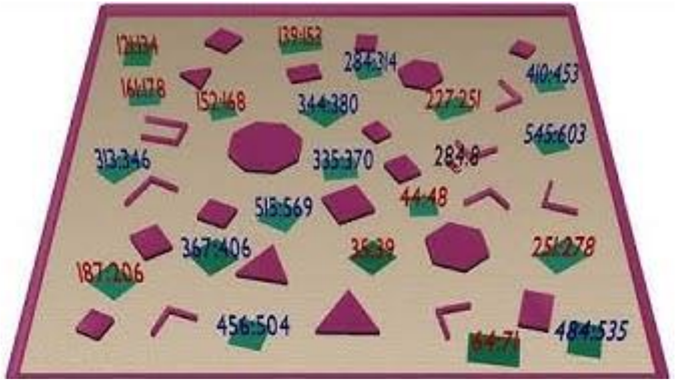


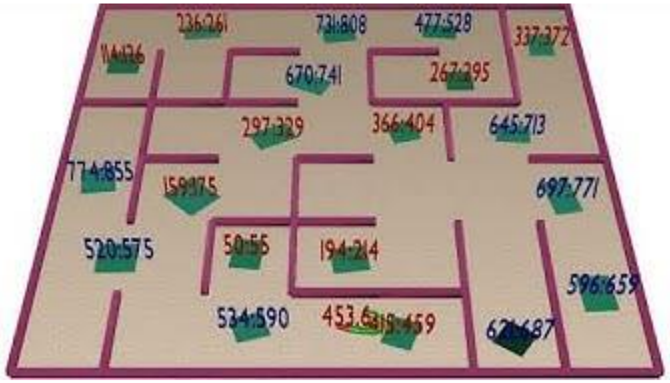
Interface with Specialised Solvers (BnB, MCS, Random)

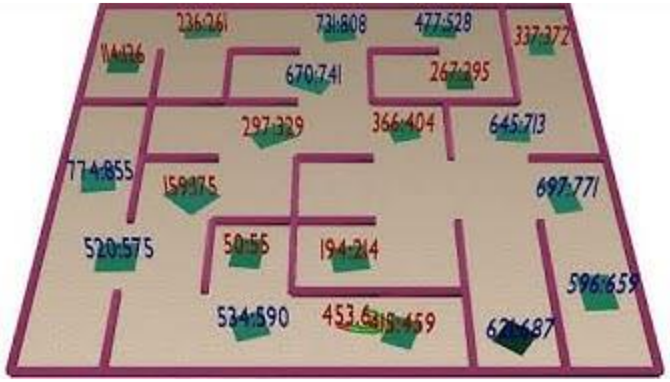
- SSSP-Reduced Roadmap Graph via Dijkstra Calls (Computed Prior to the Search)
- Cities = Waypoints, Current Position of Robot = Depot
- Open Tour
- Pairwise SSSP Distance Table
- Time Windows

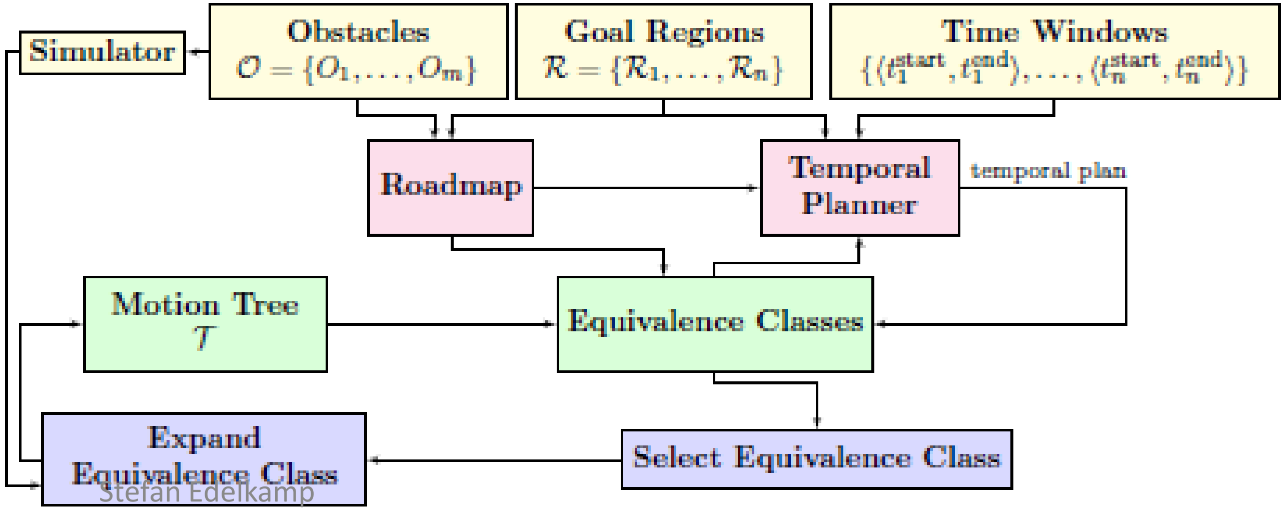


Example



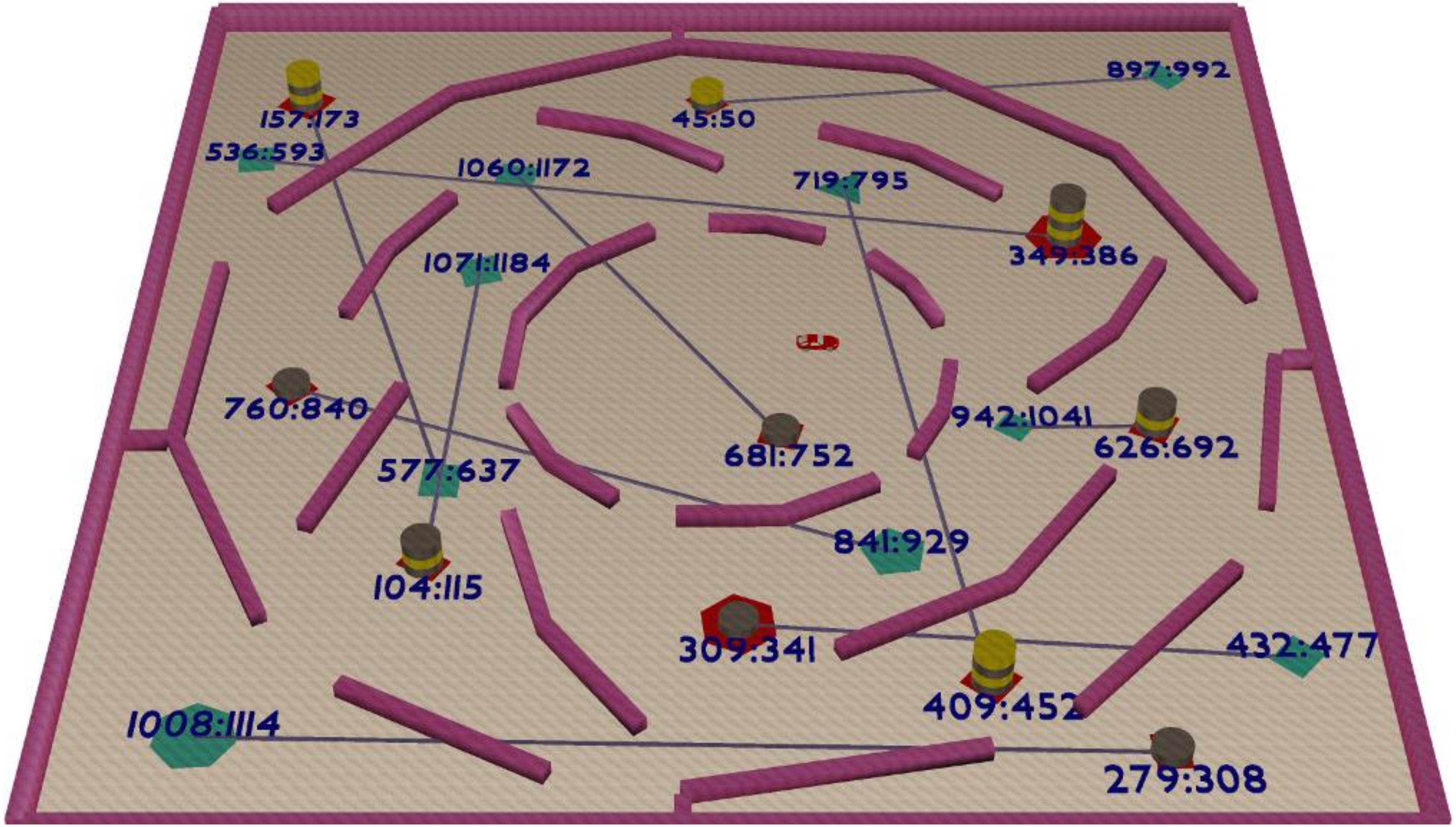






Simulator
Obstacles
$\mathcal{O} = \{O_1, \dots, O_m\}$
Goal Regions
$\mathcal{R} = \{\mathcal{R}_1, \dots, \mathcal{R}_n\}$
Time Windows
$\{\langle t_1^{start}, t_1^{end} \rangle, \dots, \langle t_n^{start}, t_n^{end} \rangle\}$
Roadmap
Temporal Planner
temporal plan
Motion Tree
$\mathcal{T}$
Equivalence Classes
Expand Equivalence Class
Select Equivalence Class

Pickup and Deliveries



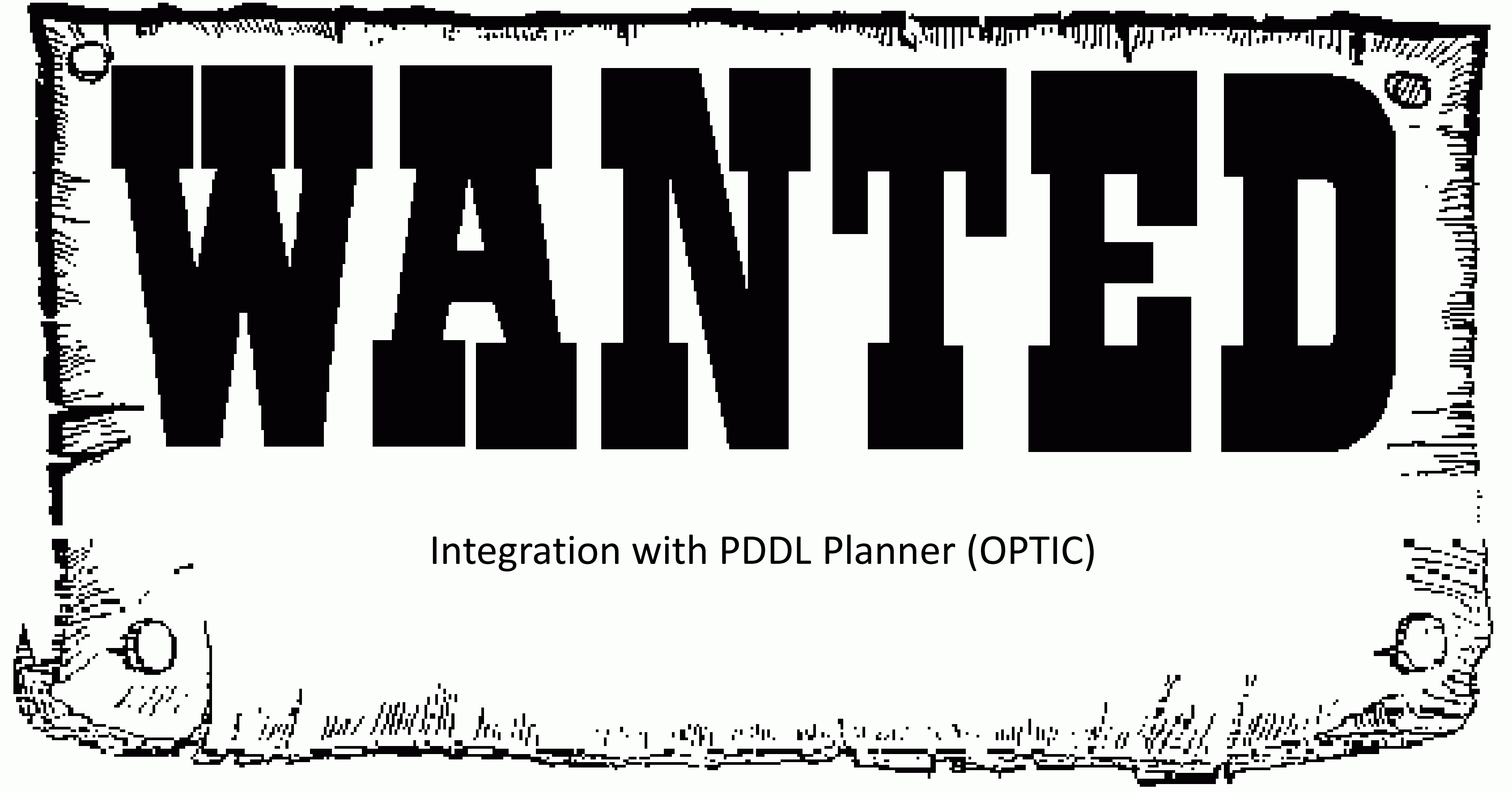
Framework

Input: multi-goal motion planning problem with time windows, pickups, deliveries, capacities

Output: collision-free and dynamically-feasible trajectory ζ that seeks to maximize $\textsc{goals}(\zeta)$

$RM \leftarrow \textsc{ConstructRoadmap}(\mathcal{O}, \mathcal{G})$
$\Xi \leftarrow \textsc{ShortestPaths}(RM, \mathcal{G})$
$\mathcal{T} \leftarrow \textsc{InitializeMotionTree}(s_{\text{init}})$
$\mathcal{X} \leftarrow \textsc{InitializeEquivalenceClasses}(s_{\text{init}})$
while $\textsc{time}() < t_{\max}$ **and** not solved **and** not converged **do**
    $\mathcal{X}_{\text{key}} \leftarrow \textsc{SelectEquivalenceClass}(\mathcal{X})$
    $\mathcal{X}_{\text{key}}.\sigma \leftarrow \textsc{DiscreteSolver}(RM, \Xi, \text{key})$
    $\textsc{ExpandMotionTree}(\mathcal{T}, \mathcal{X}_{\text{key}}.\sigma)$
    $\textsc{UpdateEquivalenceClasses}(\mathcal{X})$
return trajectory ζ in $\mathcal{T}$ that maximizes $\textsc{goals}(\zeta)$

Integration with Specialized Solvers



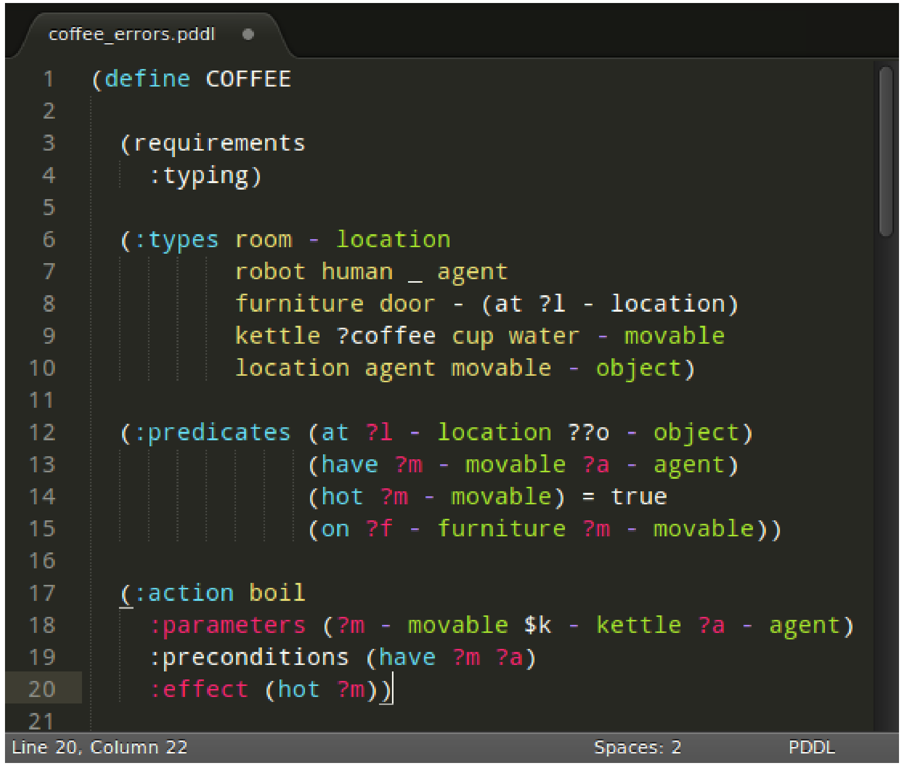
Interface with PDDL

Input

```
(:durative-action execute_task_pickup
 :parameters (?v - vehicle ?wp - waypoint ?t - task)
 :duration ( = ?duration (taskduration ?t))
 :condition (and
   (at start (at ?v ?wp)) (at start (located ?t ?wp)
   (at start (todo ?t))
   (at start (<= (+ (customer ?wp) (cap ?v)) (max_cap ?v)))
   (at start (is-pickup ?wp)) (at start(tw_open ?t)))
 :effect (and
   (at start (not (todo ?t))) (at end (visited ?wp))
   (at end (increase (cap ?v) (customer ?wp)))
   (at end (decrease (profit ?v) (customer ?wp)))
   (at end (completed ?t)))
(:durative-action execute_task_delivery
 :parameters (?v - vehicle ?wp1 ?wp2 - waypoint ?t - task)
 :duration ( = ?duration (taskduration ?t))ov
 :condition (and
   (at start (at ?v ?wp1)) (at start (located ?t ?wp1))
   (at start (todo ?t)) (at start (is-delivery ?wp1))
   (at start (and (visited ?wp2) (link ?wp2 ?wp1)))
   (at start (tw_open ?t)))
 :effect (and
   (at start (not (todo ?t))) (at end (visited ?wp1))
   (at end (increase (cap ?v) (customer ?wp1)))
   (at end (decrease (profit ?v) (customer ?wp1)))
   (at end (completed ?t))))
```

Output

```
v1  v3  v5  v4  v2
0.0   1.26   3.22   12.55   21.11
```

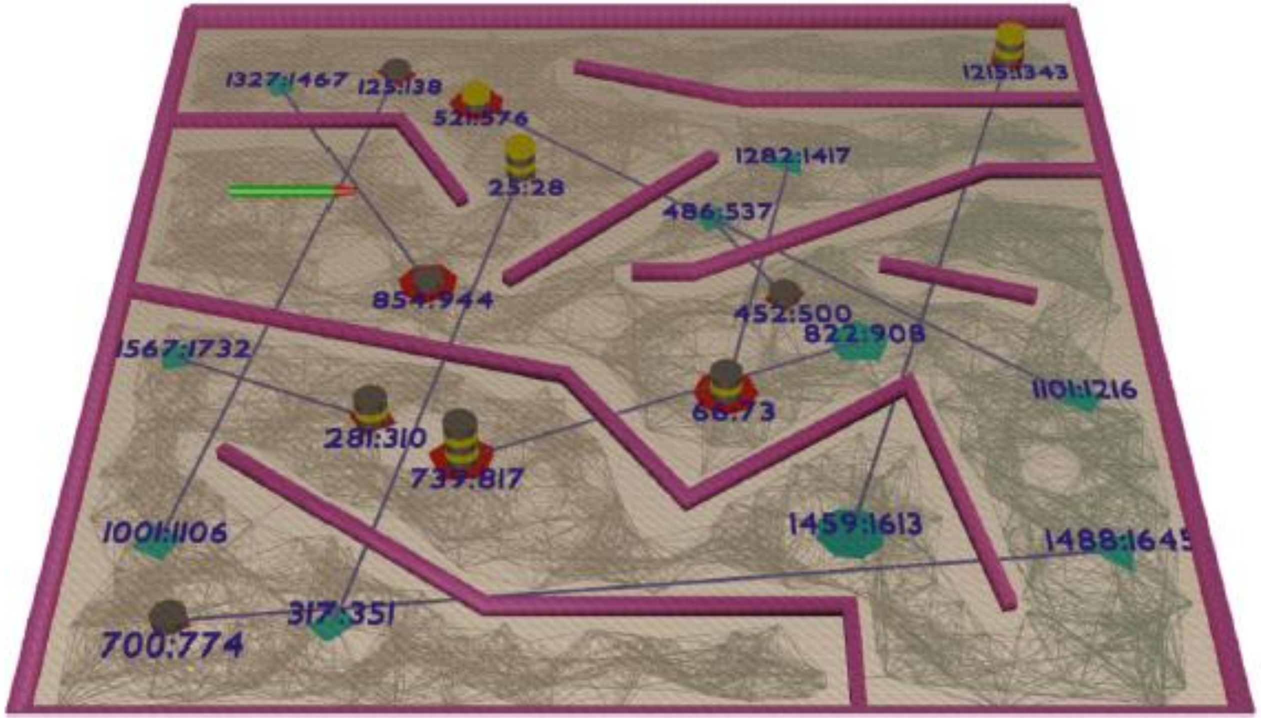

coffee_errors.pddl

```
(define COFFEE

  (requirements
    :typing)

  (:types room - location
          robot human _ agent
          furniture door - (at ?l - location)
          kettle ?coffee cup water - movable
          location agent movable - object)

  (:predicates (at ?l - location ??o - object)
               (have ?m - movable ?a - agent)
               (hot ?m - movable) = true
               (on ?f - furniture ?m - movable))

  (:action boil
    :parameters (?m - movable $k - kettle ?a - agent)
    :preconditions (have ?m ?a)
    :effect (hot ?m))
```

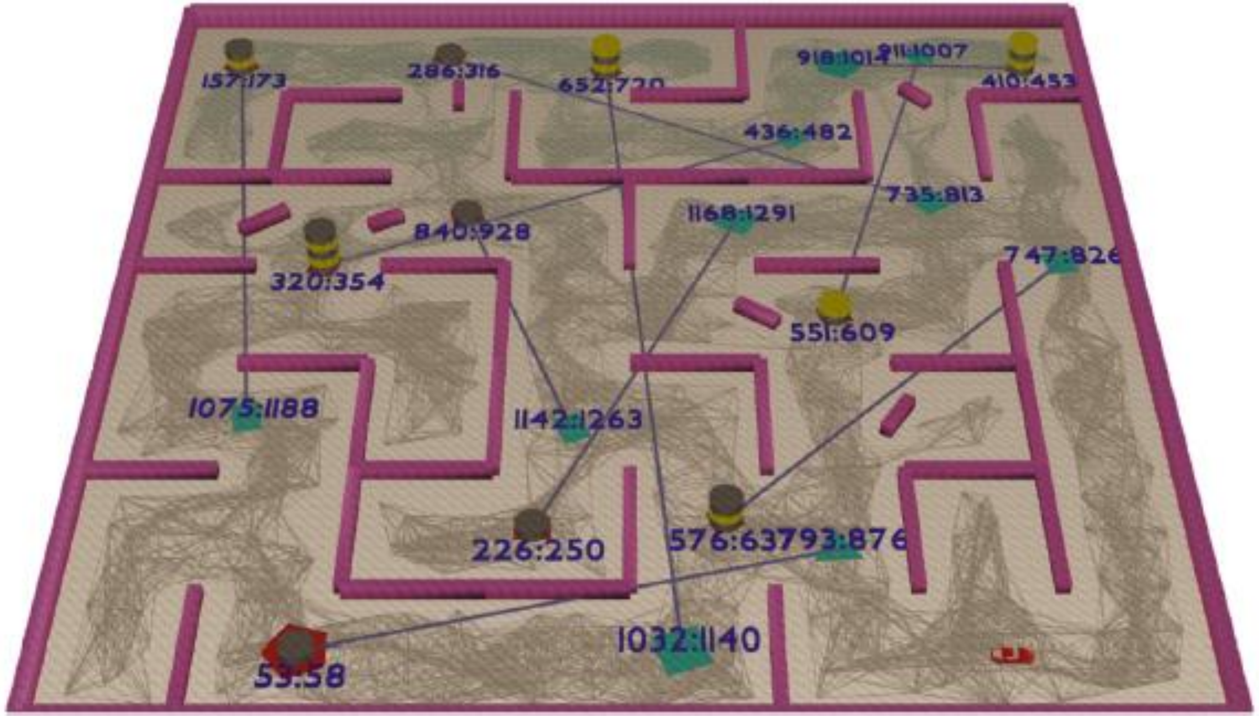
Line 20, Column 22 Spaces: 2 PDDL

Scenes (With Randomized Road Maps)

Snake Model

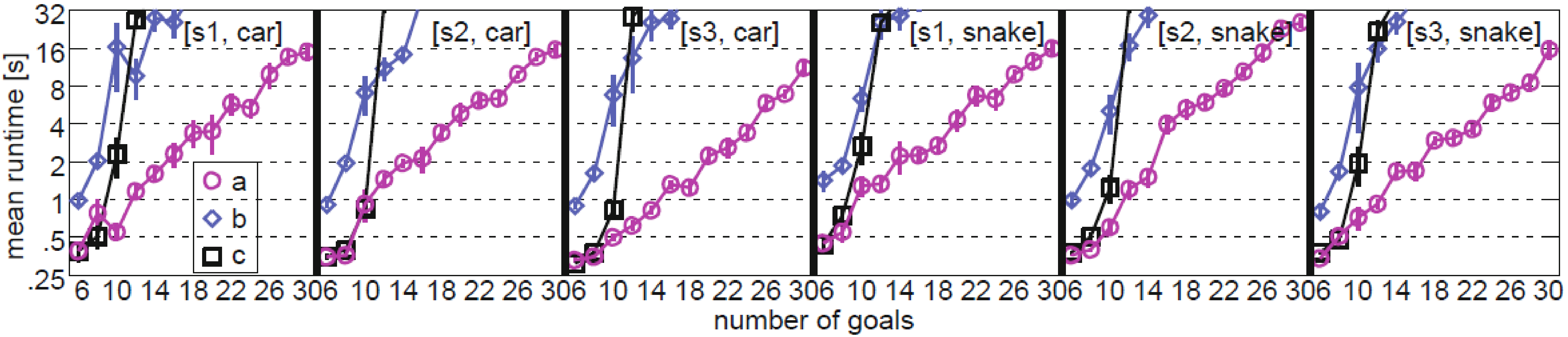


Car Model

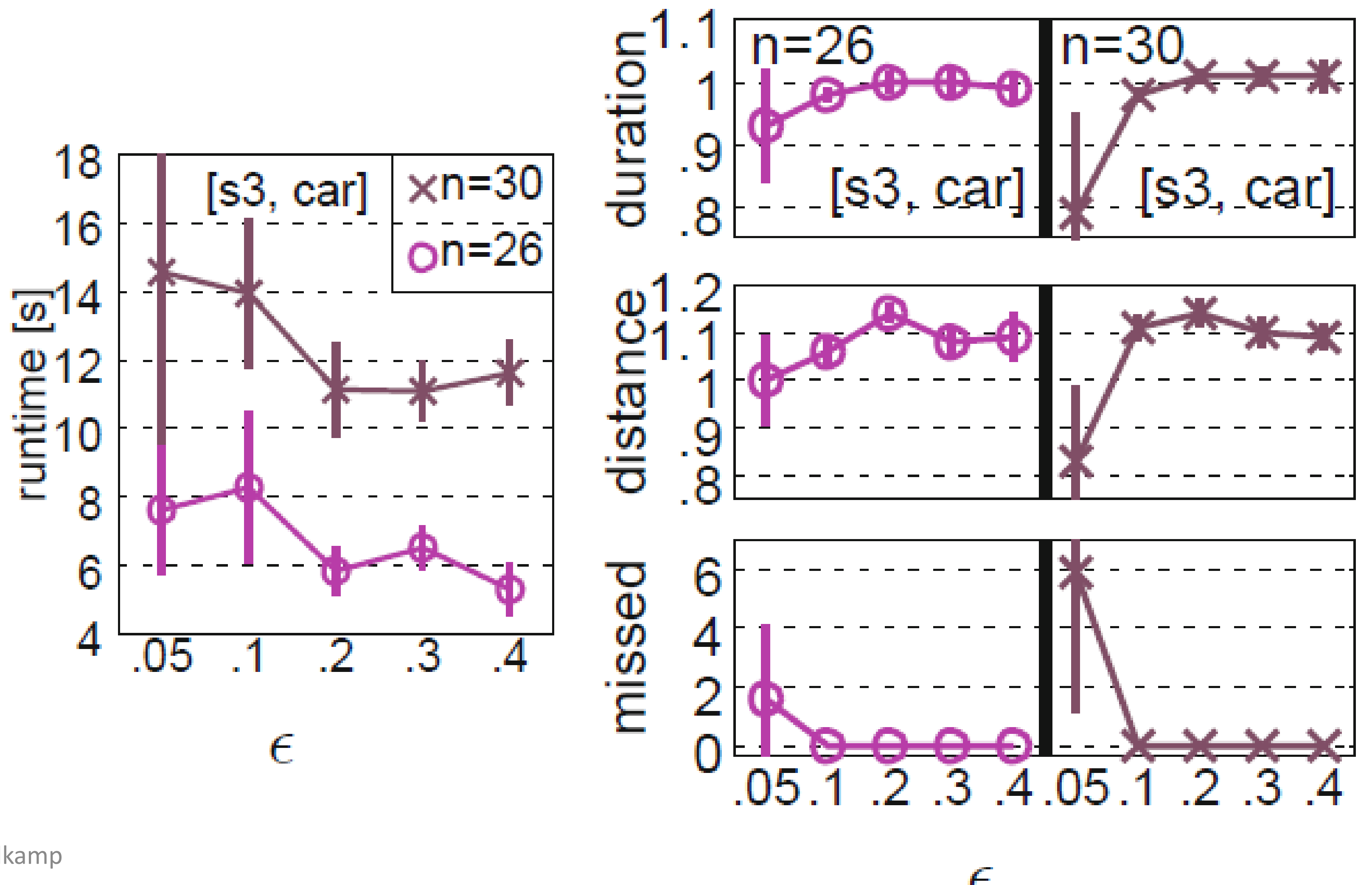


Results

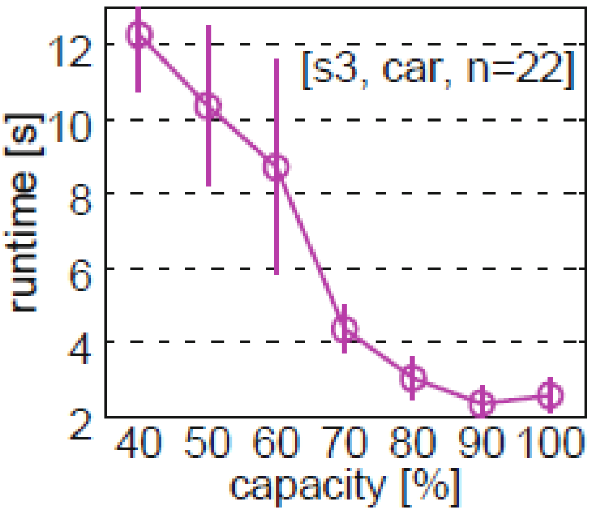
(a) MC (B) OPtic (C) Random

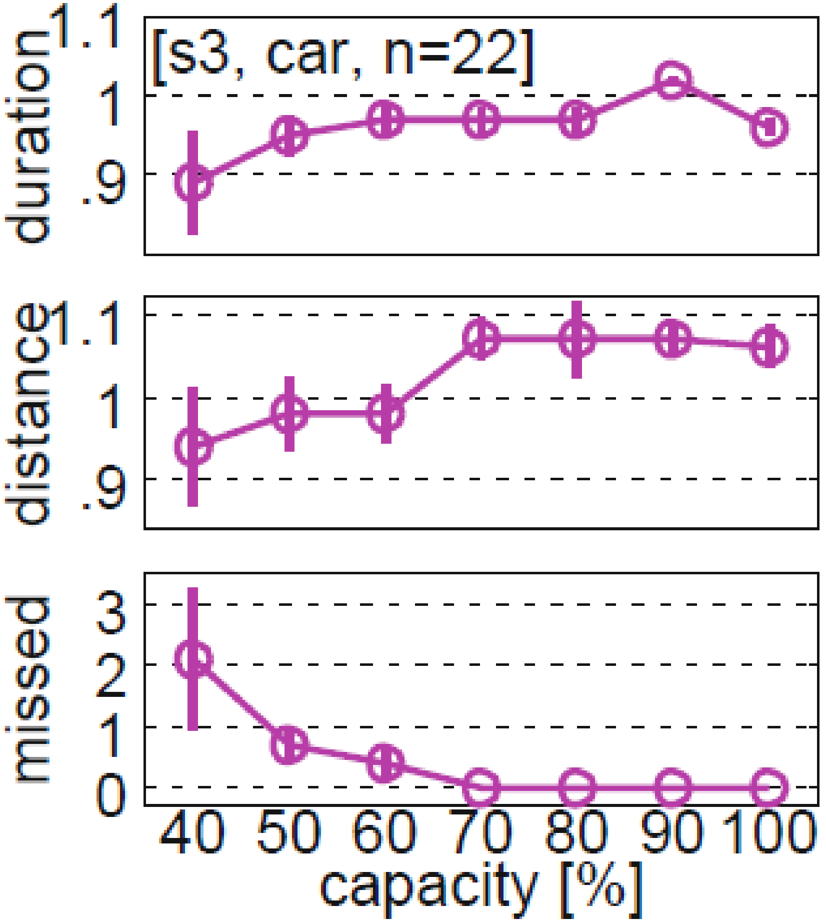


Adjusting Time Windows $[(1-\epsilon)t_i, (1+\epsilon)t_i]$



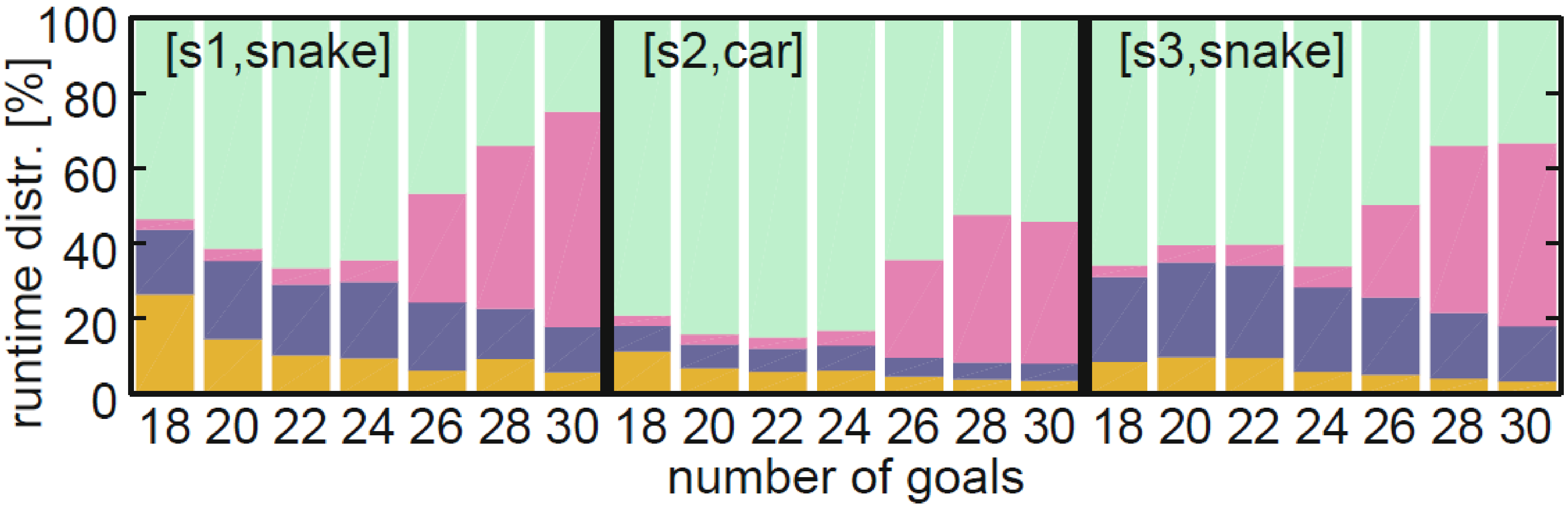
Adjusting Vehicle Capacity





RunTime Distribution

Bottom to Top:
RoadMap Constr., APSP, Collision & Simulate, Discr. Solver, Other



Conclusion

Full-Fledged Solution:

- ➢ high-dimensional robotic systems with nonlinear dynamics and
- ➢ nonholonomic constraints
- ➢ visit all goal regions fast in suitable cost-minimizing order
- ➢ unstructured, complex environments

- **and efficiently computes**
  - ➢ collision-free, dynamically-feasible, low-cost trajectories that
  - ➢ enable the robot to satisfy the CPSPTW+PD task specification

References

- [1] **Stefan Edelkamp** , Baris Secim, and **Erion Plaku** . *Surface Inspection via Hitting Sets and Multi-Goal Motion Planning.* Towards Autonomous Robotic Systems (TAROS), 134-149, Guildford, UK, 2017.
- [2] **Stefan Edelkamp** , Mihai Pomarlan, **Erion Plaku** . *Multi-Region Inspection by Combining Clustered Traveling Salesman Tours with Sampling-Based Motion Planning* . IEEE RAL. 2(2): 428-435, 2017.
- [3] **Stefan Edelkamp** , Max Gath, Tristan Cazenave, and Fabien Teytaud: Algorithm and knowledge engineering for the TSPTW problem. CISched 2013: 44-51.
- [4] **Erion Plaku** , Sarah Rashidian, and **Stefan Edelkamp** . *Multi-Group Motion Planning in Virtual Environments.* Computer Animation and Virtual Worlds, 2016.
- [5] Sara Rashidian, **Erion Plaku** , and **Stefan Edelkamp** . *Motion Planning with Rigid-Body Dynamics for Generalized Traveling Salesman Tours* . Proc. of ACM Conference on Motion in Games, 2014.
- [6] **Stefan Edelkamp** and Christoph Greulich. *Solving Physical Traveling Salesman Problems with policy adaptation* . Proc. of IEEE Conference on Computer in Games (CIG) 2014.
- [7] **Stefan Edelkamp** and **Erion Plaku** . *Multi-goal motion planning with physics-based game engines* . Proc. of IEEE Conference on Computer in Game (CIG), 2014.
- [8] **Stefan. Edelkamp** , Stefan Schroedl. *Heuristic Search, Theory and Practice. Morgan Kaufmann* , 2011.
- [9] **Stefan Edelkamp** , Christoph Greulich, and Denis Golubev . *Solving the Physical Vehicle Routing Problem for Improved Multi-Robot Freespace Navigation.* Proc. of KI, 2016.
- [10] Enrico Scala, Patrik Haslum, **Daniele Magazzeni** , Sylvie Thiébaux. *Landmarks for Numeric Planning Problems.* Proc. of IJCAI 2017: 4384-4390.
- [11] Michael Cashmore, Maria Fox, Derek Long, **Daniele Magazzeni** , et al . *ROSPlan: Planning in the Robot Operating System.* Proc. of ICAPS 2015.
- [12] Malik Ghallab, Nick Hawes, **Daniele Magazzeni** , Brian C. Williams, Andrea Orlandini. *Planning and Robotics* (Dagstuhl Seminar 17031). Dagstuhl Reports 7(1): 32-73, 2017.
- [13] Senka Krivic, Michael Cashmore, **Daniele Magazzeni,** Bram Ridder, Sándor Szedmák, Justus H. Piater: *Decreasing Uncertainty in Planning with State Prediction.* IJCAI 2017: 2032-2038.
- [14] Marcello Balduccini, **Daniele Magazzeni,** Marco Maratea, Emily Leblanc:*CASP solutions for planning in hybrid domains* . TPLP 17(4): 591-633 (2017).