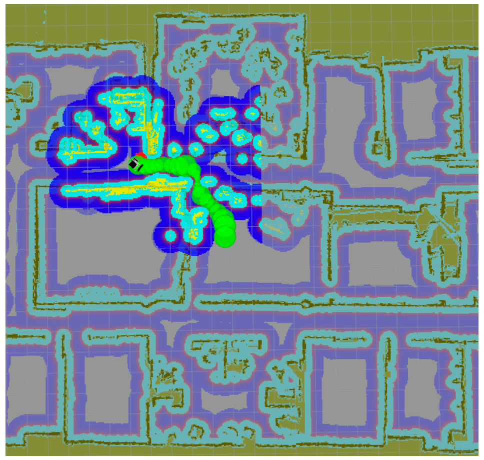
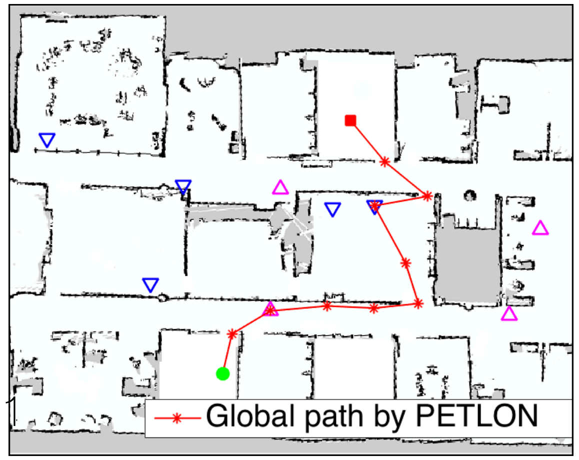
PETLON: Planning Efficiently for Task-Level-Optimal Navigation

Shih-Yun Lo[1], Shiqi Zhang[2], and Peter Stone[1]

the University of Texas at Austin
Learning Agent Research Group

SUNY Binghamton





Autonomous agents are becoming more capable of:

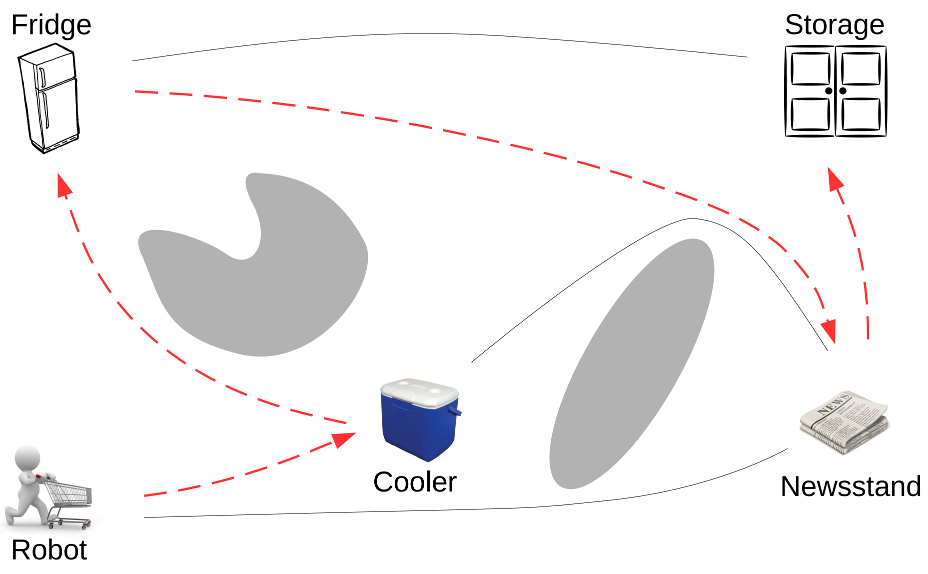
- Long-duration deployment
- Large-scale navigation
- Knowledge-intensive reasoning

Service robots will be desired to:

- Understand human requests
- Complete multiple tasks with high efficiency

Task Planning

- **Task requests?**
  Plan to satisfy certain specifications
  - Satisfiability problem
  logic programs
  Answer set programming (ASP)
  Prolog
  AI planning
  PDDL
  STRIPS

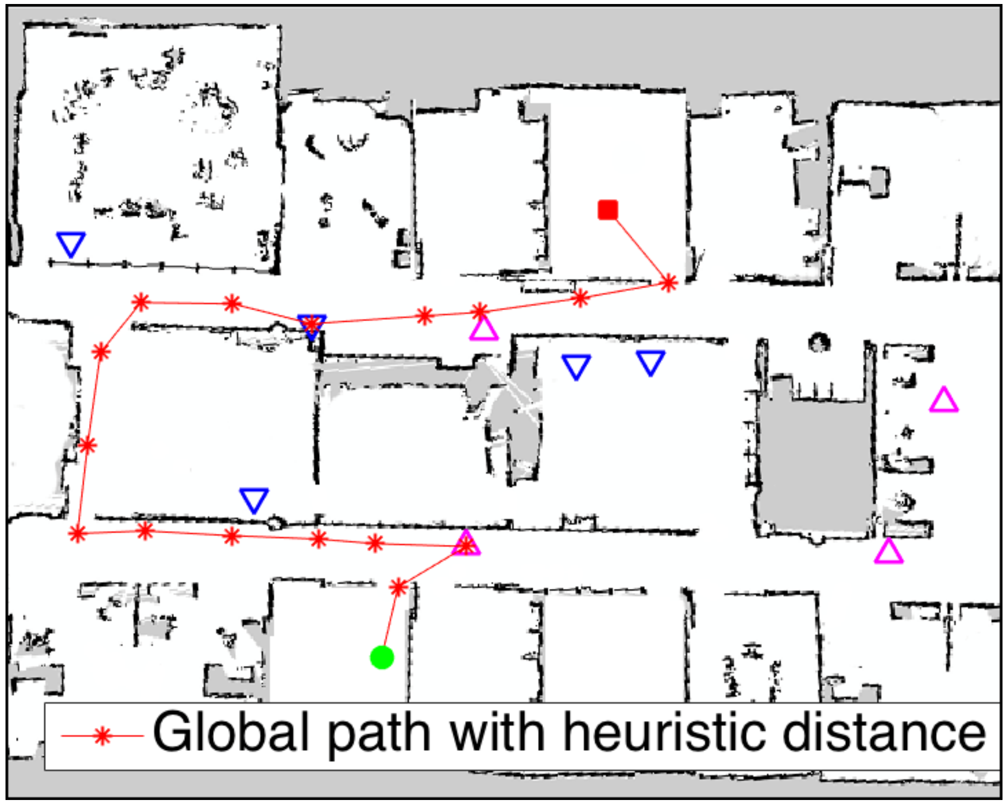


e.g. **camping preparation** (**in ASP**)**:**

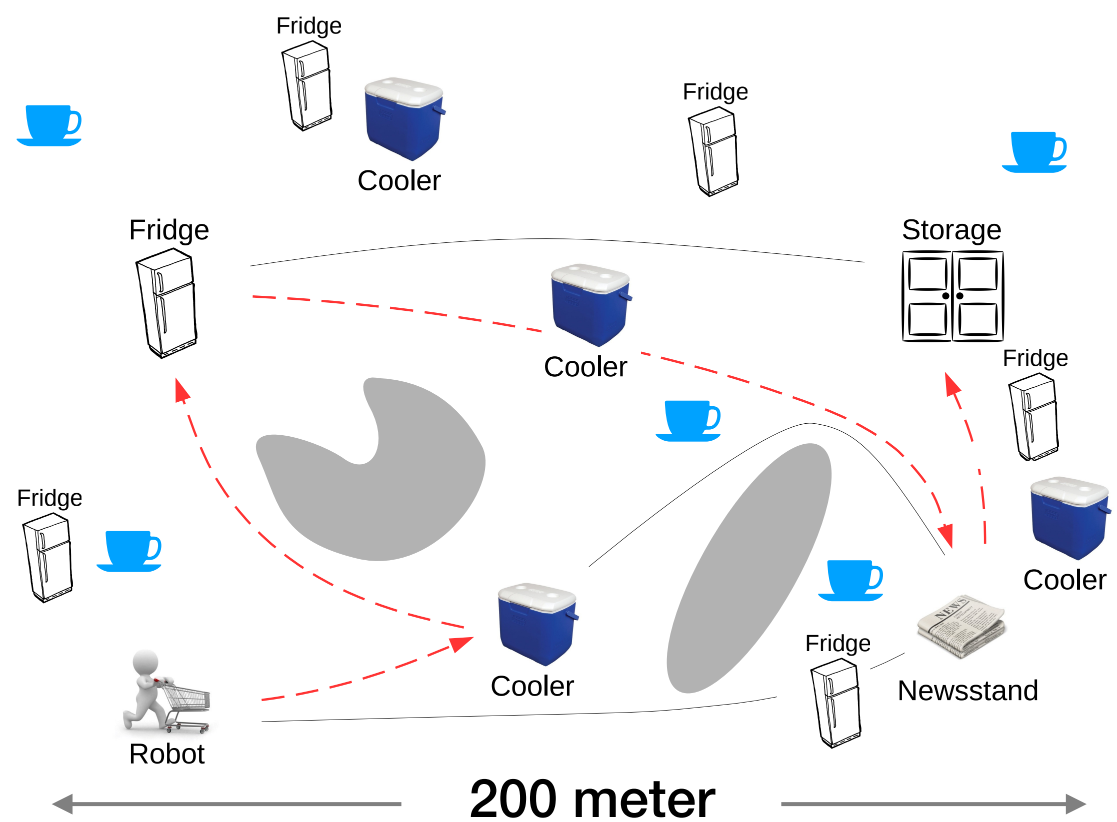
need_ice(milk)
cooling(cooler)
-spoiled(NI) :- inside(NI,C), cooling(C), need_ice(NI)
:- delivered(S,milk,maxstep), -spoiled(milk), storage(S)
#minimize{X,Y:cost(X,Y)}.

Motion Planning

- **Navigation paths for travel cost evaluation**
  Plan to satisfy kinematics/dynamics constraints
  - Collision-free, dynamically feasible
    Trajectory optimization
    CHOMP
    Sampling-based algorithms
    RRT
    PRM



The Computational Challenges in Large Domain Planning



- **Large task domains include:**
  - **- Large number of object instances N**

    cooling(cooler_east1)
    cooling(fridge_west1)
    cooling(fridge_west2)
    hascoffee(cafe_east1)
    hascoffee(cafe_west1)
    ⋮

  - **- Large search space:**

    exponential growth in task length L and N in node traversals

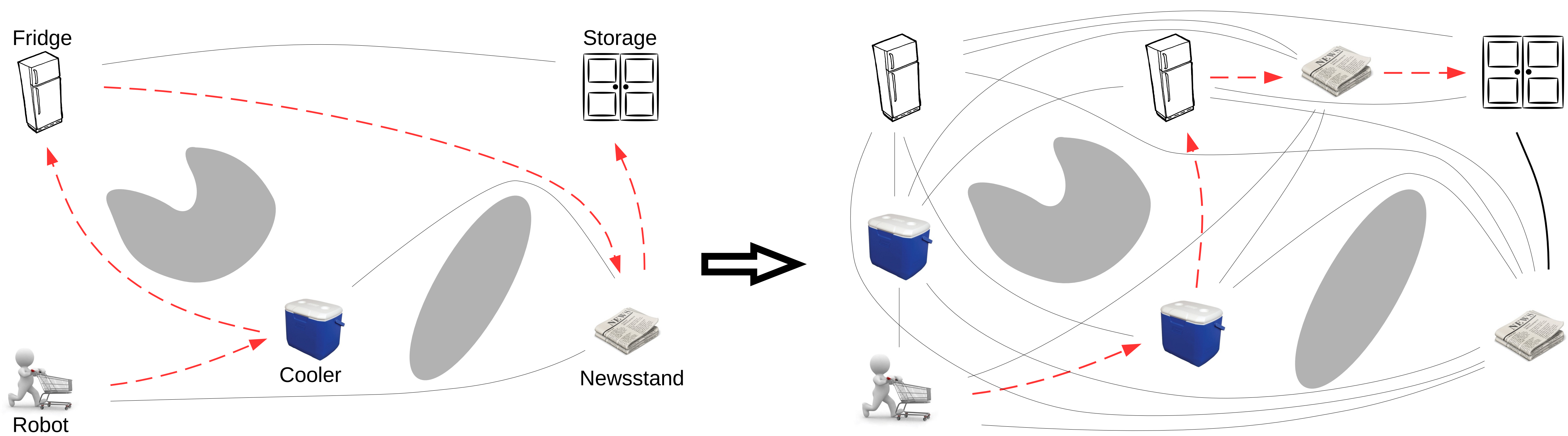
    $O(N^L)$

- **Large domain navigation cost**:
  - - evaluated by motion planner for task-level travel cost estimate

The Challenges

- **Desire for optimality guarantee**
  suboptimal plans may have much longer travel length in large maps

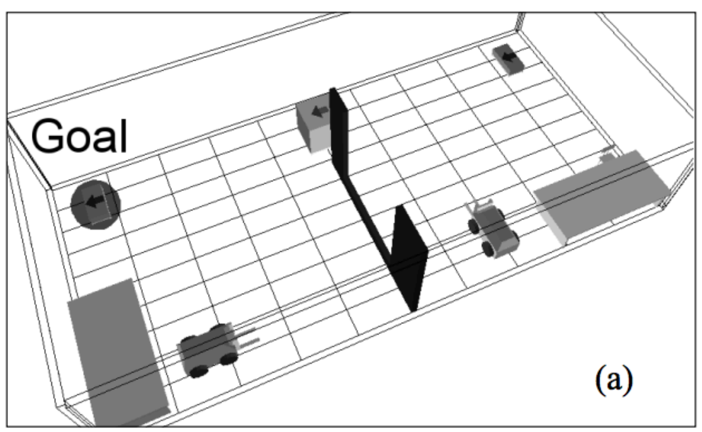
- **Minimize the calls for motion evaluation**
  the intensive search **with motion evaluation** grows exponentially in L and $\mathcal{N}$: $O(\mathcal{N}^L)$

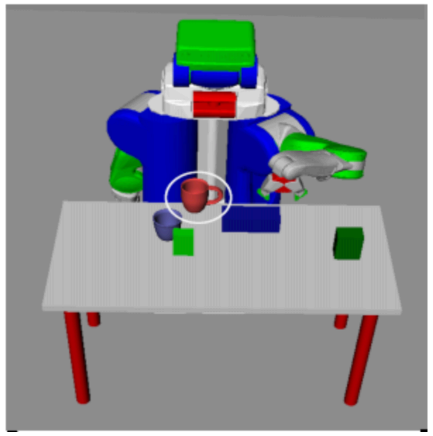


Related Work

- **Semantic attachments**
  symbolic planner with function calls
  Dornhege, et. al, 2009

- **Integrated task and motion planning in robotics**
  plan to solve for geometrically constrained problems





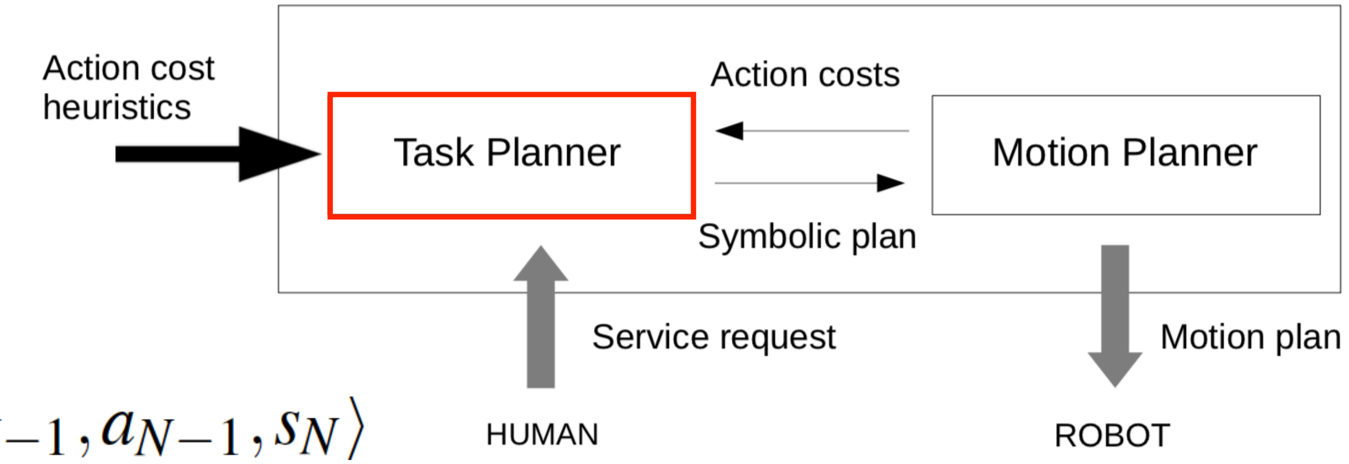
Gravot, Cambon, et. al, 2005
Wolfe, et.al, 2010
Plaku, et.al, 2010
Kaelbling, et. al, 2011
Lagriffoul, et al, 2014
Srivastava, et al, 2014
Garrett, et. al, 2017

logic-geometric programming
plan to optimize certain objective
Toussaint, 2015

- **Limitations: not addressing optimality guarantee and the corresponding motion evaluation expense problem**

PETLON:
Planning Efficiently for Task-Level-Optimal Navigation

$\mathcal{D}^t$



Symbolic state space: $s \in S$

Symbolic action space: $a \in A$

State transition function: $\mathcal{T}^t : \langle s, a \rangle \rightarrow s'$

Task plans: $p = \langle s_0, a_0, \cdots, s_{N-1}, a_{N-1}, s_N \rangle$

Task specifications: $S^G \subseteq S$

Feasible task plan space: $p \in P$ s.t. $s_0 = s^{init}$, $s_N \in S^G$

Cost function: $Cost(\langle s, a, s' \rangle) \rightarrow \mathbb{R}$

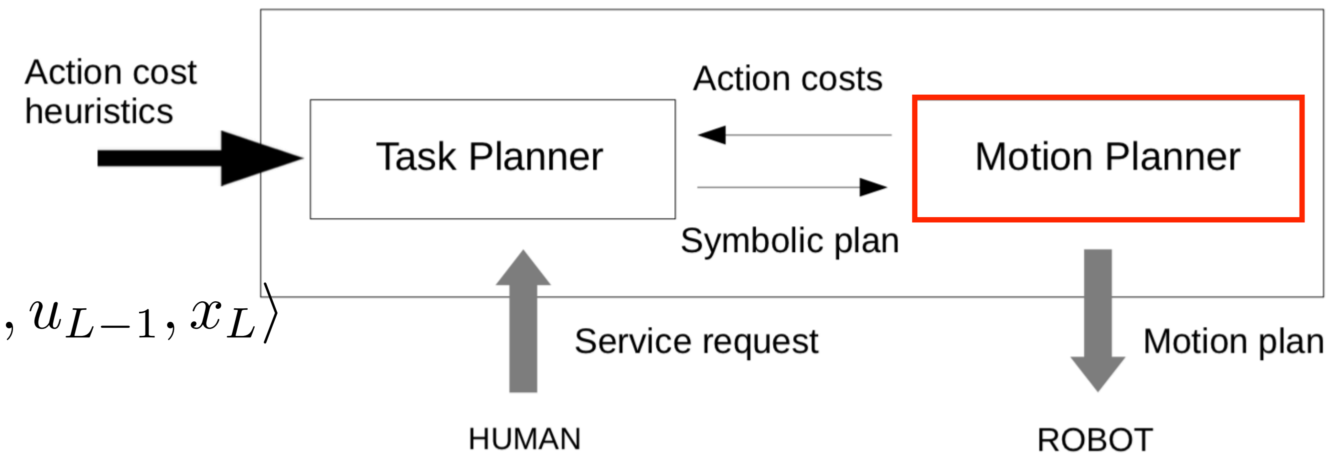
- **Optimal task-level action sequence:**

$$\mathcal{P}^t : \; p^* = \operatorname{argmin}_{p \in P} \Sigma_{\langle s,a,s' \rangle \in p} Cost(\langle s, a, s' \rangle)$$

PETLON:
Planning Efficiently for Task-Level-Optimal Navigation

$\mathcal{D}^m$

Numeric state space: $x \in X$
Numeric action space: $u \in U$
State transition function: $\mathcal{T}^m : \langle x, u \rangle \rightarrow x'$
Motion plans: $\xi = \langle x_0, u_0, ..., x_{L-1}, u_{L-1}, x_L \rangle$
Robot model: $\mathcal{M}$
Collision-free space: $Free(\mathcal{D}^m | \mathcal{M})$
State mapping function: $f : s \rightarrow X$
Initial set: $X^{init} = f(s^{int}) \in X$
Goal set: $X^{goal} = f(s^{goal}) \in X$
Feasible motion plans: $\xi \in \Xi$ *s.t.* $x_0 \in X^{int}$, $x_L \in X^{Goal}$, $x_0, ..., x_L \in Free(\mathcal{D}^m | \mathcal{M})$
Motion cost function: $Len : \xi \rightarrow \mathbb{R}$

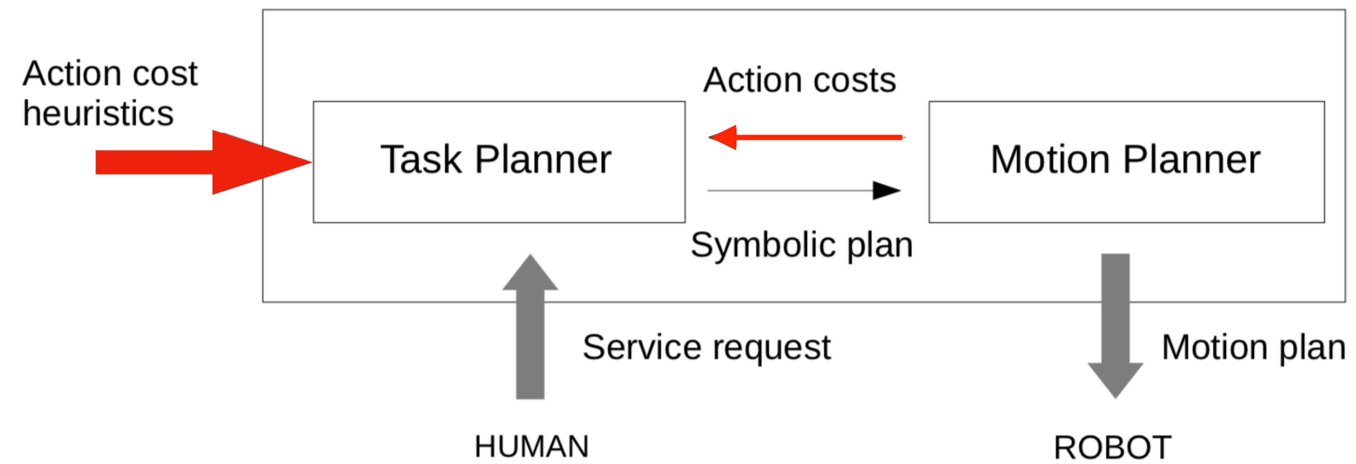


- **Motion evaluation:**

$$\mathcal{P}^m : \xi^* = \text{argmin}_{\xi \in \Xi} Len(\xi)$$

$$\hat{C}(x, x') = Len\big(\mathcal{P}^m(\langle s, a, s' \rangle, f, \mathcal{D}^m, \mathcal{M})\big)$$

PETLON:
Planning Efficiently for Task-Level-Optimal Navigation



Two Types of Cost Functions:

- **Admissible heuristic cost function** h (cheap):

$$h(x,x') = ||x - x'||_2 \qquad x \in f(s),\ \ x' \in f(s')$$

- **Evaluated cost function** $\hat{C}$ (expensive):

$$\hat{C}(x,x') = Len\big(\mathcal{P}^m(\langle s,a,s'\rangle, f, \mathcal{D}^m, \mathcal{M})\big)$$

- **Maintained cost function** $Cost$ (used in $\mathcal{D}^t$):

$$h(x,x') \leq Cost(s,a,s') \leq \hat{C}(x,x')$$

PETLON:
Planning Efficiently for Task-Level-Optimal Navigation

- The **task and motion planning** problem:

$$\Omega : \langle \mathcal{D}^t, \mathcal{D}^m, s^{init}, S^G, x^{init}, f, \mathcal{M} \rangle \rightarrow \langle p, [\xi_0, \xi_1, \cdots, \xi_{N-1}] \rangle$$

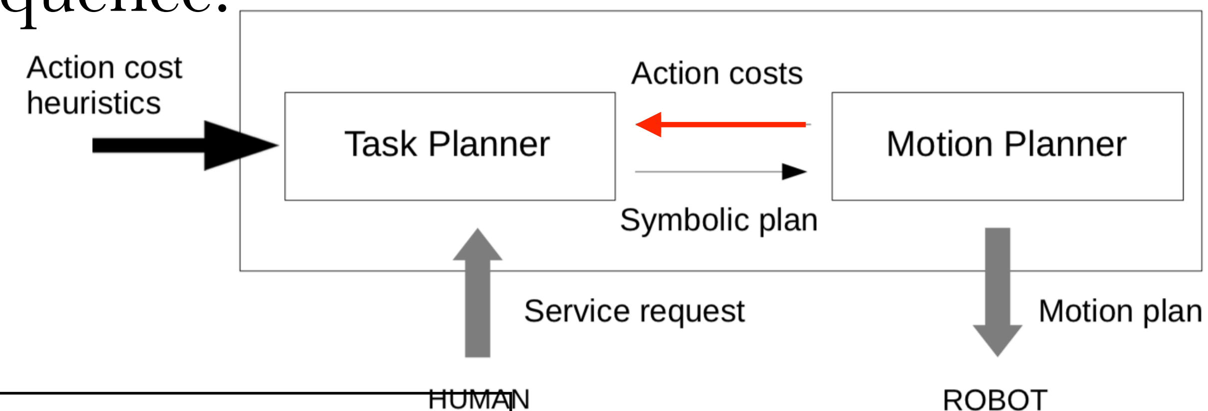
s.t.

$$\mathcal{D}^t : \quad p(0) = s^{init},\ p(N) \in S^G,\ |p| = N$$

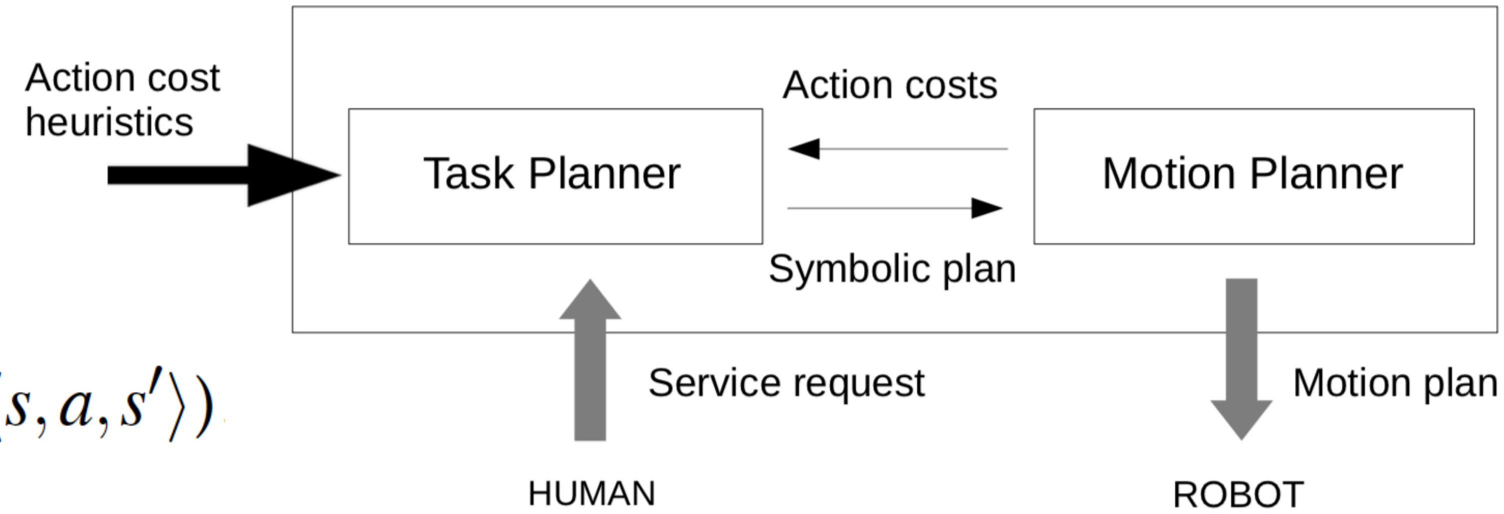
$$\mathcal{D}^m : \quad \xi_0(0) = x^{init},\ \xi_i(0) \in f(s_i),\ \xi_i(T_i) \in f(s_{i+1}) \quad \text{for } i = \{0, 1, \ldots, N-1\}$$

- **Optimal task-level navigation action** sequence:

$$p^* = \text{argmin}_{p \in P} \Big(\sum_{0 \le i < |p|} Len(\xi_i) | \mathcal{P}^m, f \Big)$$



- **PETLON:** minimizing number of calls for motion evaluation while still **guarantees task-level optimality**



Recall:

$$\mathcal{P}^t \; : \; p^* = \text{argmin}_{p \in P} \Sigma_{\langle s,a,s' \rangle \in p} Cost(\langle s,a,s' \rangle)$$

(1) (4)

Proposition 1: if all task actions in p* are evaluated, it is the optimal solution.

$$\sum_{\langle s,a,s' \rangle \in p^*} Cost(s,a,s') = \sum_{\langle s,a,s' \rangle \in p^*} \hat{C}(s,a,s') \quad (1)$$

$$\sum_{\langle s,a,s' \rangle \in p^*} Cost(s,a,s') \leq \sum_{\langle s,a,s' \rangle \in p} Cost(s,a,s') \quad (2)$$

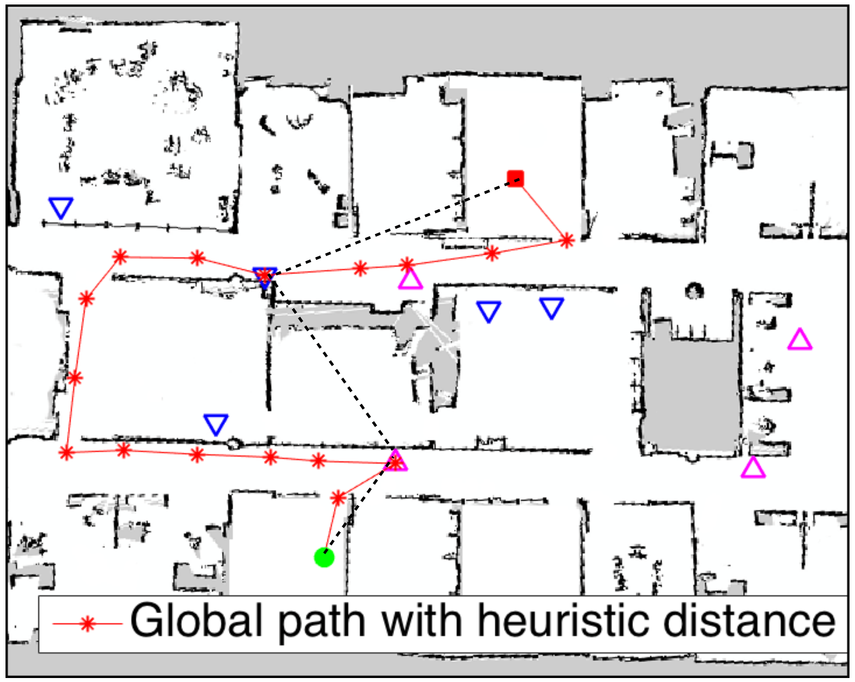
$$\sum_{\langle s,a,s' \rangle \in p} Cost(s,a,s') \leq \sum_{\langle s,a,s' \rangle \in p} \hat{C}(s,a,s') \quad (3)$$

=>

$$\sum_{\langle s,a,s' \rangle \in p^*} \hat{C}(s,a,s') \leq \sum_{\langle s,a,s' \rangle \in p} \hat{C}(s,a,s') \quad \blacksquare \quad (4)$$

PETLON:
Planning Efficiently for Task-Level-Optimal Navigation

iter 0



----- **Dash: plan value using** $Cost$**, maintained cost function**

—— **Solid: plan true value by** $\hat{C}$ **, evaluated cost function**

PETLON algorithm

---

1. Initialize $Cost(s,a,s') \leftarrow h(x,x')$
Initialize empty state-action-state array: A^{evld}

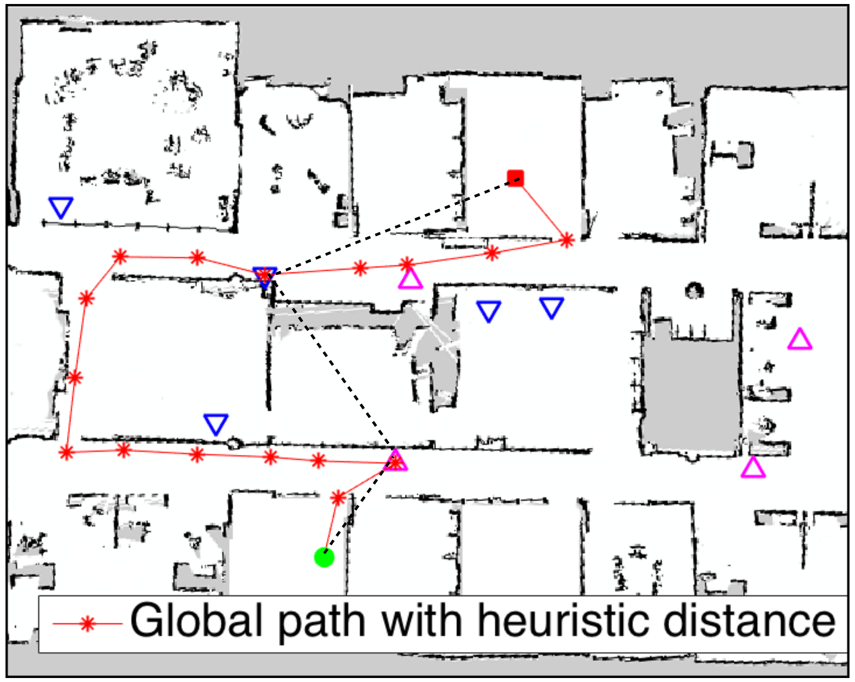
2. while true do
$[\hat{p}^*, P] \leftarrow \mathcal{P}^t(s^{init}, S^G, Cost, \mathcal{D}^t)$
if $\langle s,a,s' \rangle \in A^{evld}, \forall \langle s,a,s' \rangle \in \hat{p}^*$ **then**
return $\hat{p}^*$
end if

3. for each $p \in P$ **do**
Evaluate motion cost: $Cost(s,a.s') \leftarrow \hat{C}(x,x')$
Append $\langle s,a,s' \rangle$ to A^{evld}

end for
end while

PETLON:
Planning Efficiently for Task-Level-Optimal Navigation

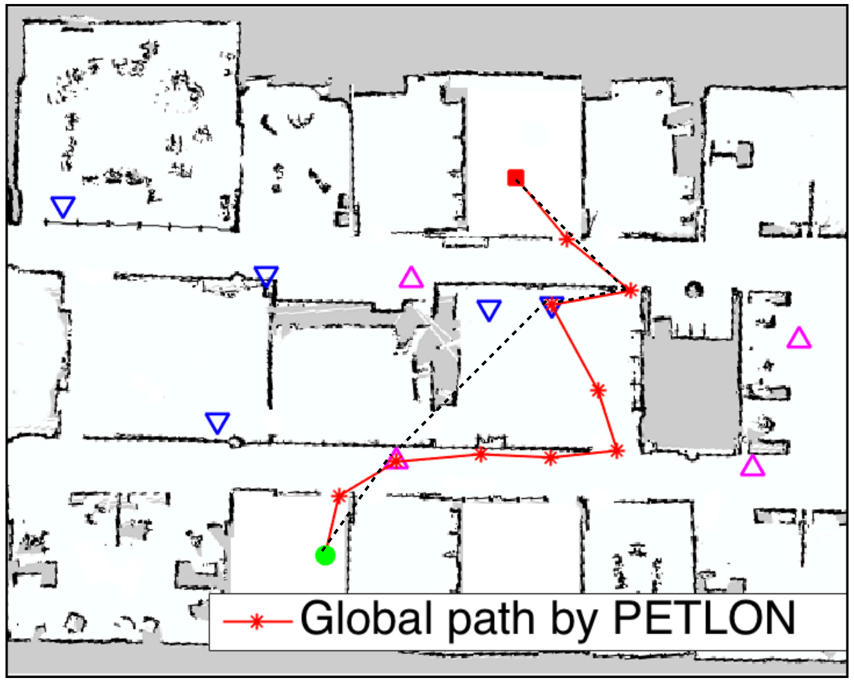
iter 0



- - - - - Dash: plan value using $Cost$, maintained cost function

——— Solid: plan true value by $\hat{C}$, evaluated cost function

iter 1



PETLON algorithm

1. Initialize $\quad Cost(s,a,s') \leftarrow h(x,x')$

Initialize empty state-action-state array: A^{evld}

2. while true do

$[\hat{p}^*, P] \leftarrow \mathcal{P}^t(s^{init}, S^G, Cost, \mathcal{D}^t)$

if $\langle s,a,s' \rangle \in A^{evld}, \forall \langle s,a,s' \rangle \in \hat{p}^*$ **then**

return $\hat{p}^*$

end if

3. for each $p \in P$ **do**

Evaluate motion cost: $Cost(s,a.s') \leftarrow \hat{C}(x,x')$

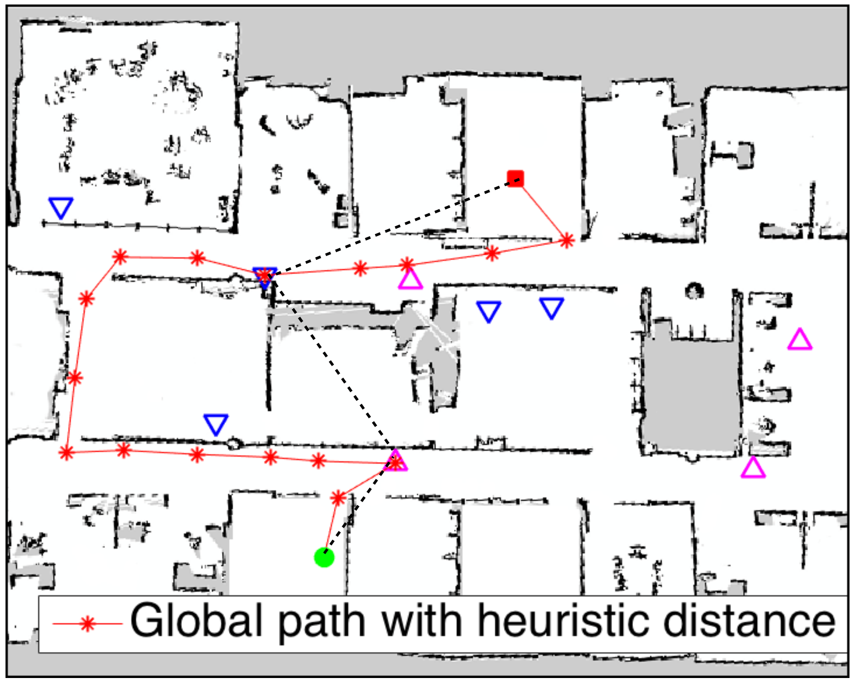
Append $\langle s,a,s' \rangle$ to A^{evld}

end for

end while

PETLON:
Planning Efficiently for Task-Level-Optimal Navigation

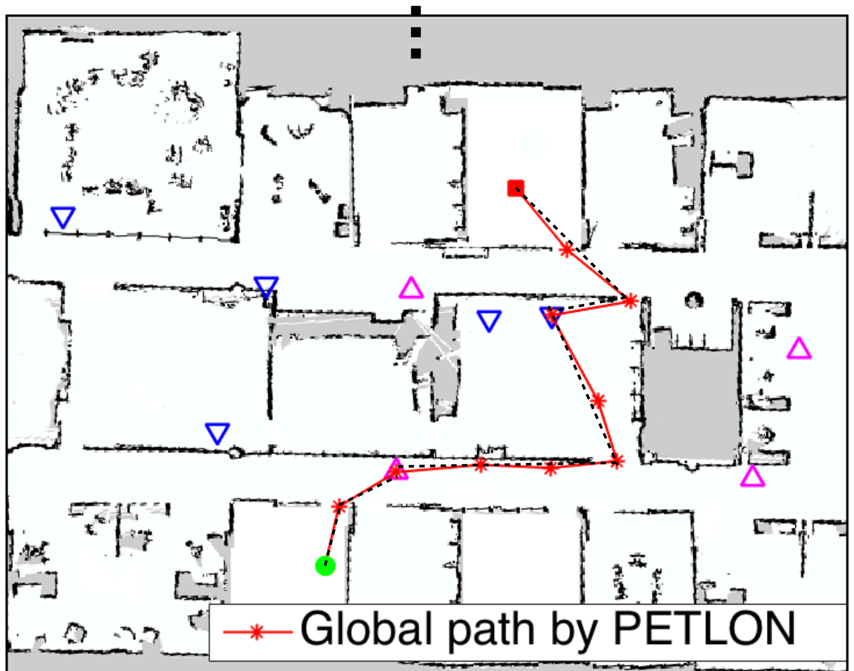
iter 0



- - - - - **Dash: plan value using** $Cost$**, maintained cost function**

—— **Solid: plan true value by** $\hat{C}$ **, evaluated cost function**

iter N



PETLON algorithm

1. Initialize $Cost(s,a,s') \leftarrow h(x,x')$
Initialize empty state-action-state array: A^{evld}

2. while true do
$[\hat{p}^*, P] \leftarrow \mathcal{P}^t(s^{init}, S^G, Cost, \mathcal{D}^t)$
if $\langle s,a,s' \rangle \in A^{evld}, \forall \langle s,a,s' \rangle \in \hat{p}^*$ **then**
return $\hat{p}^*$
end if

3. for each $p \in P$ **do**
Evaluate motion cost: $Cost(s,a.s') \leftarrow \hat{C}(x,x')$
Append $\langle s,a,s' \rangle$ to A^{evld}

end for
end while

- **guarantees task-level optimality**
- **minimal calls for motion evaluation**

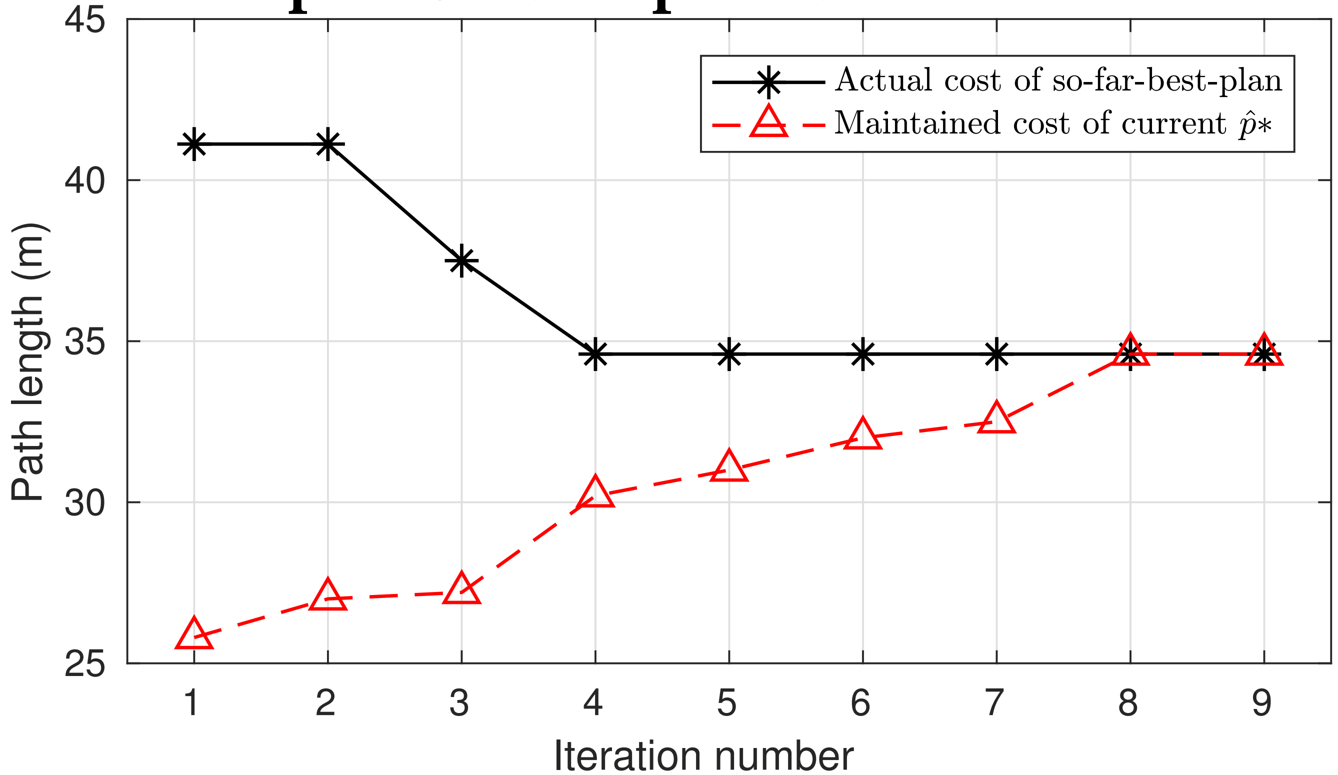
- **OptOne**: evaluate the best plan p*

PETLON:
Planning Efficiently for Task-Level-Optimal Navigation

- **So-far-best cost C^{sfb}:**
  the cost value of the so-far best plan

- **Anytime** property:
  C^{sfb} monotonically decreases

example run of OptOne



PETLON algorithm

Initialize $Cost(s,a,s') \leftarrow h(x,x')$
Initialize empty state-action-state array: A^{evld}

while true do
  $[\hat{p}^*, P] \leftarrow \mathcal{P}^t(s^{init}, S^G, Cost, \mathcal{D}^t)$
  if $\langle s,a,s'\rangle \in A^{evld}, \forall \langle s,a,s'\rangle \in \hat{p}^*$ **then**
    return $\hat{p}^*$
  end if
  for each $p \in P$ **do**
    Evaluate motion cost: $Cost(s,a.s') \leftarrow \hat{C}(x,x')$
    Append $\langle s,a,s'\rangle$ to A^{evld}
  end for
end while

PETLON: Planning Efficiently for Task-Level-Optimal Navigation

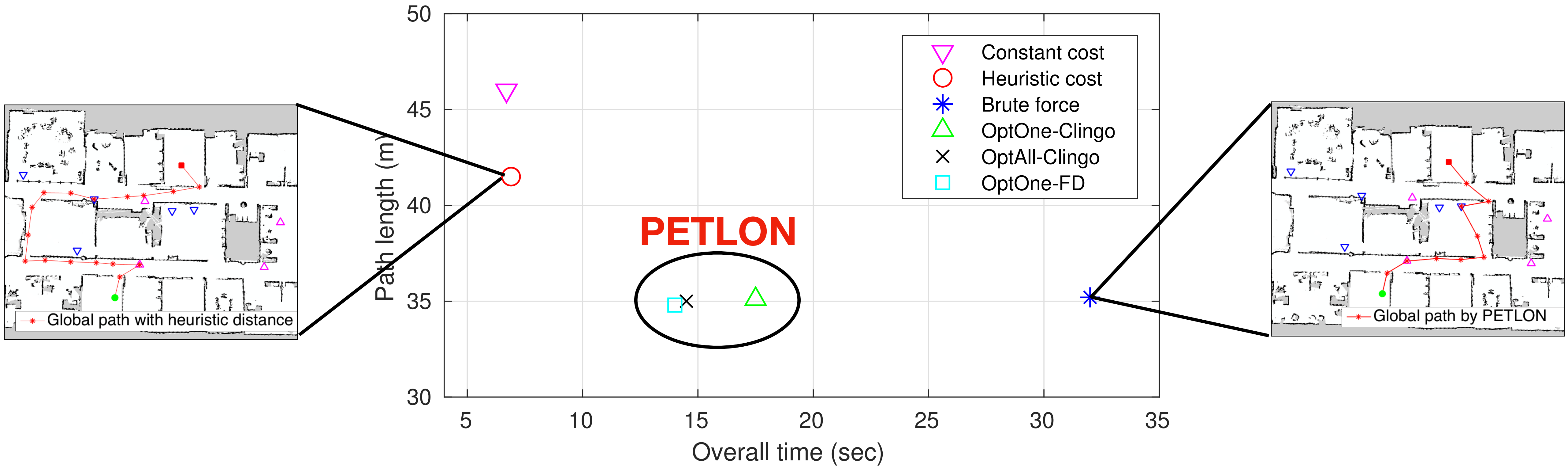
- **OptAll**: evaluate all feasible plans with costs $< C^{sfb}$ (also guarantees optimality)

PETLON algorithm

Initialize $Cost(s,a,s') \leftarrow h(x,x')$
Initialize empty state-action-state array: A^{evld}
\+ Initialize the *so-far-best* cost: $C^{sfb} \leftarrow Inf$
while true do
  $[\hat{p}^*, P] \leftarrow \mathcal{P}^t(s^{init}, S^G, Cost, \mathcal{D}^t)$
  if $\langle s,a,s' \rangle \in A^{evld}, \forall \langle s,a,s' \rangle \in \hat{p}^*$ **then**
    return $\hat{p}^*$
  end if
\+ **for each** $p \in P$ **and** $Cost(p) < C^{sfb}$ **do**
    Evaluate motion cost: $Cost(s,a.s') \leftarrow \hat{C}(x,x')$
    Append $\langle s,a,s' \rangle$ to A^{evld}
\+ $C_{plan} = \Sigma_{\langle s,a,s' \rangle \in p} Cost(s,a,s')$
\+ $C^{sfb} = min(C_{plan},\ C^{sfb})$
  end for
end while

- **guarantees task-level optimality**
- **more calls for motion evaluation than OptOne, fewer calls for task planner**

Plan Quality v.s. Efficiency

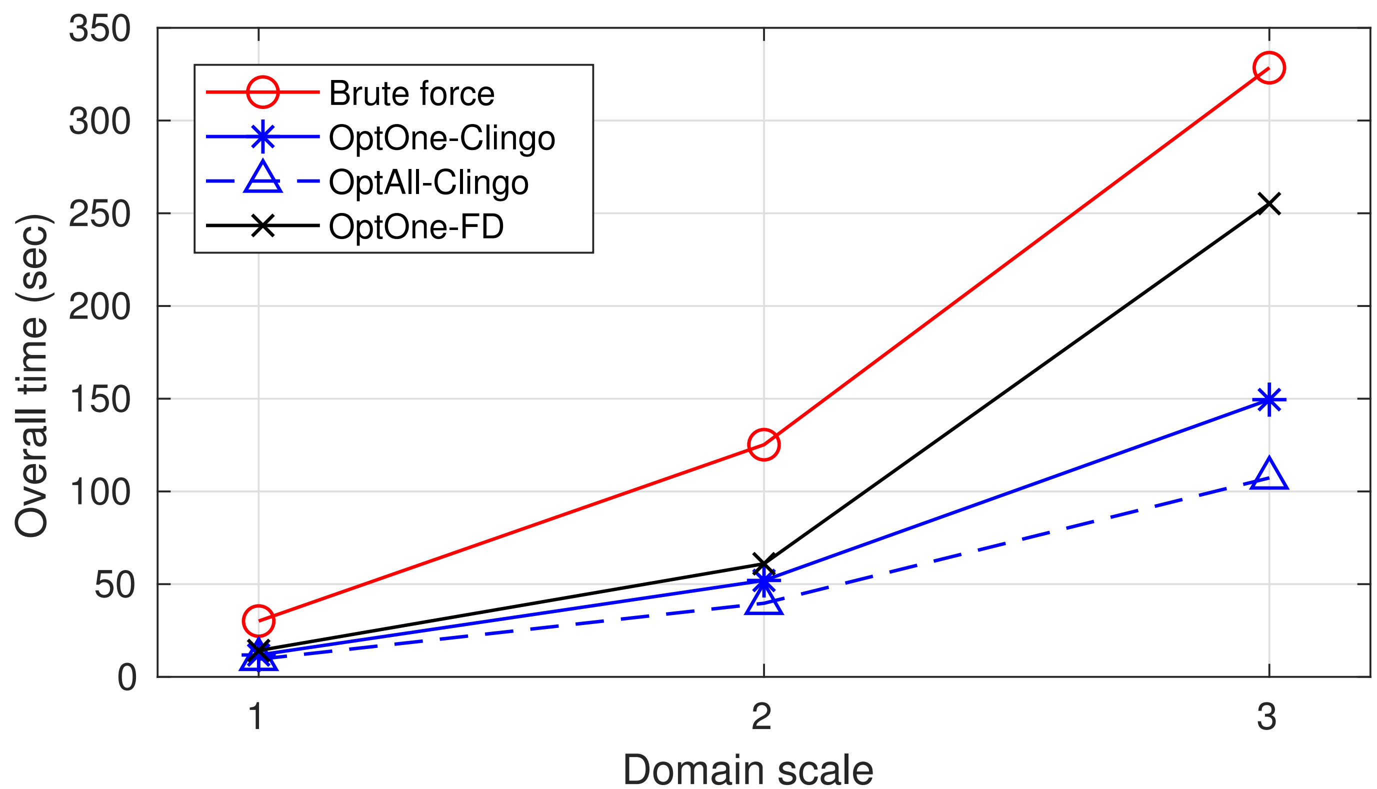


	Domain scale		
	1	2	3
OptOne-Clingo	10.75		
OptOne-FD	13.60		
OptAll-Clingo	16.00		
brute-force baseline	325.00		

PETLON

Total number of motion evaluations

Experiments with Domain Scale-ups



PETLON

	Domain scale		
	1	2	3
OptOne-Clingo	10.75	9.00	11.00
OptOne-FD	13.60	11.00	12.00
OptAll-Clingo	16.00	17.50	25.75
brute-force baseline	325.00	1295.00	2850.00

PETLON

Total number of motion evaluations

Conclusion

- **PETLON:**
  An algorithm that *saves motion evaluations computation* yet *guarantees optimality* in large-domain navigation problem

- **Future work:**
  To more efficiently re-use the task-level search
  To include cost estimate in *dynamic environments*, and
  to impose an *exploration mechanism* in uncertain environments

Thank you!