Mobile Robot Planning using Action Language *BC* with an Abstraction Hierarchy

Shiqi Zhang[1], Fangkai Yang[2], Piyush Khandelwal[1], and Peter Stone[1]

[1] Department of Computer Science, UT Austin, Austin, TX
[2] Schlumberger Limited, Houston, TX

September 2015 @ LPNMR

Overview

- Objective
  - Very efficient near-optimal symbolic planning for mobile robots

- Key features
  - Different abstraction levels of domain descriptions connected by passing state constraints downward
  - Not strictly following higher-level plans: better flexibility in computing low-cost plans at low levels

Action Language *BC* (Lee et al., 2013)

- Static law

$$A_0 \ \ \mathbf{if} \ \ A_1, \cdots, A_m \ \ \mathbf{ifcons} \ \ A_{m+1}, \cdots, A_n$$

Example:

$$acc(R_1, K, R_2) \ \ \mathbf{if} \ \ hasdoor(R_1, K), \ hasdoor(R_2, K).$$

- Dynamic law

$$A_0 \ \ \mathbf{after} \ \ A_1, \cdots, A_m \ \ \mathbf{ifcons} \ \ A_{m+1}, \cdots, A_n$$

Example:

$$cross(K) \ \ \mathbf{causes} \ \ loc = R_2 \ \ \mathbf{if} \ \ loc = R_1, \ acc(R_1, K, R_2).$$

$$\mathbf{nonexecutable} \ \ cross(K) \ \ \mathbf{if} \ \ loc = R, \ \neg hasdoor(R, K).$$

Hierarchical domain representation

- Abstraction hierarchy

$$\mathcal{H} = (\mathcal{D}, \mathcal{L})$$

  - $\mathcal{D}$ is a list of action descriptions $D_1, D_2, \cdots, D_d$ such that $f(D_i) \subseteq f(D_j)$ for $1 \leq i < j \leq d$

  - $\mathcal{L}$ is the step bound estimation function

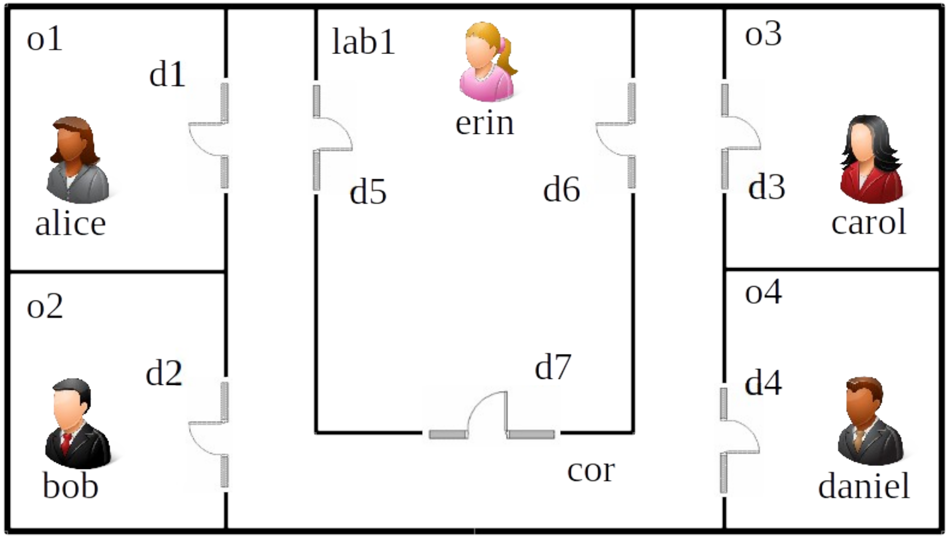
$$\mathcal{L}(a) = \max_{\langle s,a,s' \rangle \in T(D_i)} \Big(Len\big(\hat{P}(s, s')\big) \Big)$$

where $\mathcal{L}(a)$ computes the minimum number of steps needed to ensure that the effect of action a can be optimally achieved at the next level

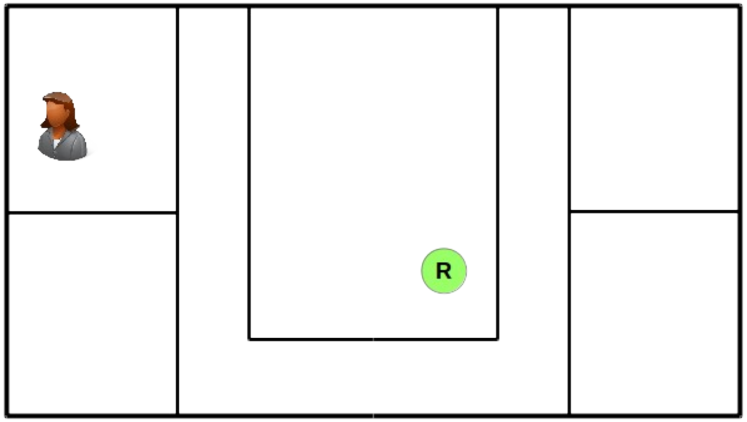
Example problem: mail collection

- D_1 formalizes if each person has been served or not
- D_2 further describes room connections through doors
- D_3 includes all domain details for primitive actions

- Planning initially at Level 3 would take too long
- The upper levels provide guidance on where to expand possible plans in Level 3



Action description: ***Level 1***



- Static laws:

$$\neg inside(P, R_2) \ \ \mathbf{if} \ \ inside(P, R_1), \ R_1 \neq R_2.$$

$$\mathbf{inertial} \ \ inside(P, R).$$

$$\boxed{mailcollected(P_1) \ \ \mathbf{if} \ \ mailcollected(P_2), \ passto(P_1, P_2).}$$

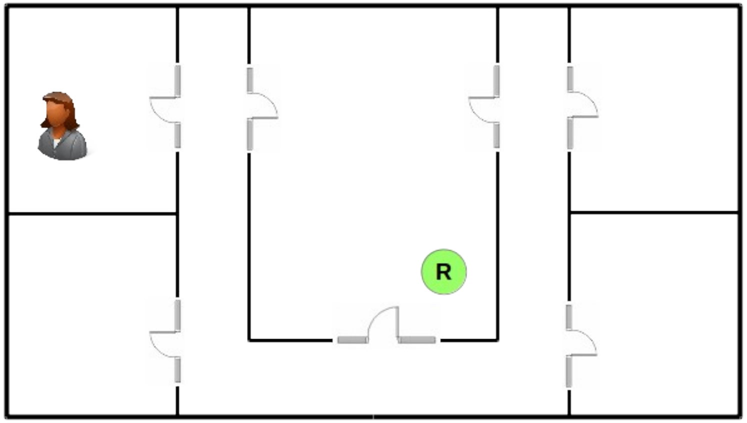
Recursively defined

- Dynamic laws:

$$serve(P) \ \ \mathbf{causes} \ \ mailcollected(P).$$

$$serve(P) \ \ \mathbf{causes} \ \ loc = R \ \ \mathbf{if} \ \ inside(P, R).$$

Action description: ***Level 2***



- Static laws:

$$acc(R_1, D, R_2) \;\; \textbf{if} \;\; hasdoor(R_1, D), \; hasdoor(R_2, D).$$

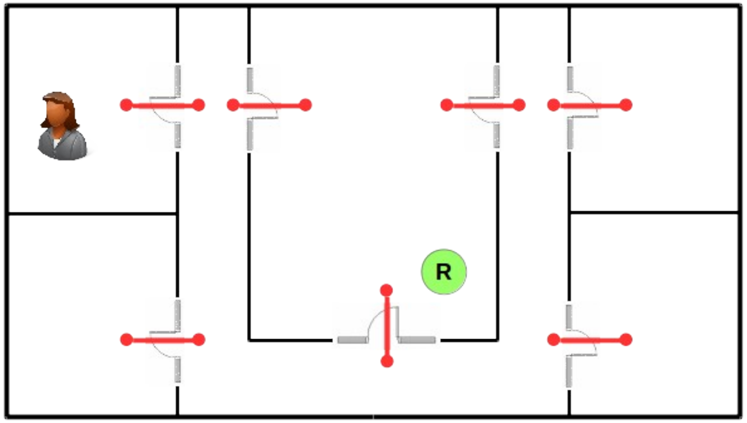
$$acc(R_1, D, R_2) \;\; \textbf{if} \;\; acc(R_2, D, R_1). \qquad \textbf{default} \; \neg acc(R_1, D, R_2).$$

- Dynamic laws:

$$collectmail(P) \;\; \textbf{causes} \;\; mailcollected(P).$$

$$cross(D) \;\; \textbf{causes} \;\; loc = R_2 \;\; \textbf{if} \;\; loc = R_1, \; acc(R1, D, R_2).$$

Action description: ***Level 3***



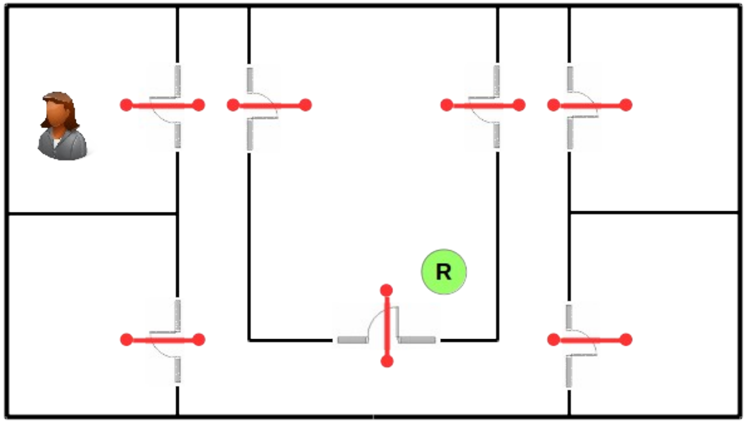
- Dynamic laws:

$$approach(D). \quad gothrough(D). \quad opendoor(D).$$

- Examples:

$$approach(D) \ \textbf{causes} \ facing(D).$$
$$\textbf{nonexecutable} \ approach(D) \ \textbf{if} \ loc = R, \ \neg hasdoor(R, D).$$
$$\textbf{nonexecutable} \ approach(D) \ \textbf{if} \ facing(D).$$

Hierarchical planning: passing state constraints downward



- Level 1:
  - Plan: $serve(alice)$.
  - State constraints for the next level ($\mathcal{L}(serve) = 5$):

    $\{0 : loc = lab1, \ldots\}, \ \{5 : mailcollected(alice)\}.$
- Level 2:
  - Plan: $0 : cross(d5), \ 1 : cross(d1), \ 2 : collectmail(alice).$

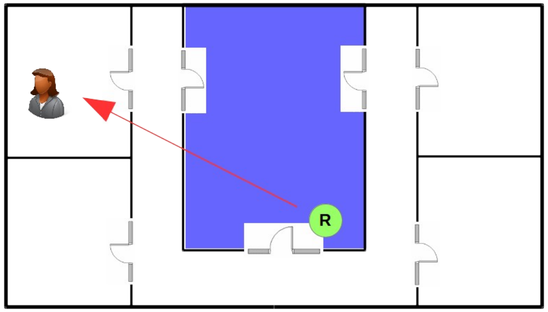
    $3 : noop, \ 4 : noop.$
  - State constraints for the next level:

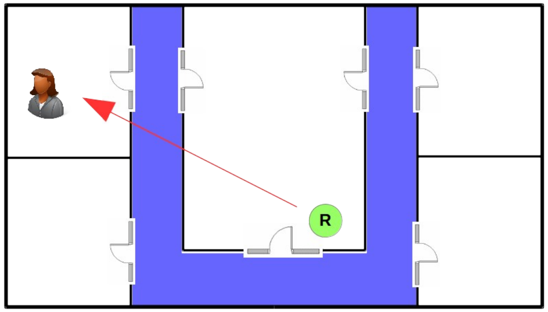
    $\{0 : loc = lab1, \ldots\}, \ \{3 : loc = cor, \ldots\}, \ \{6 : loc = o1, \ldots\},$

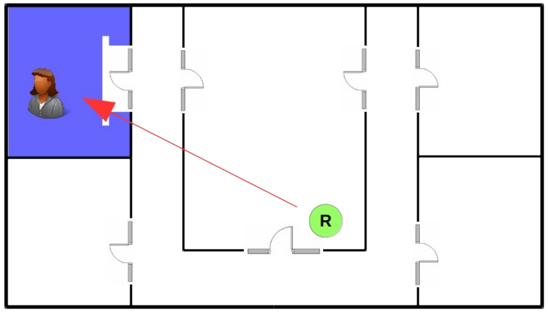
    $\{7 : mailcollected(alice), 7 : loc = o1\}.$

Planning algorithms:
PlanFG, PlanHL, and **PlanHG**

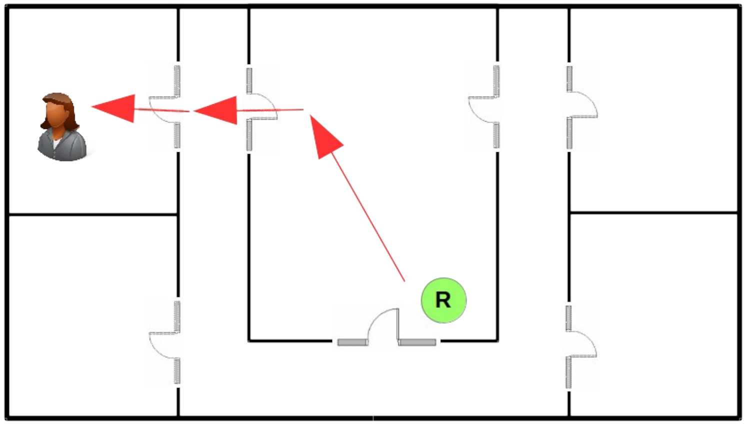
- State constraints generated at Level 2

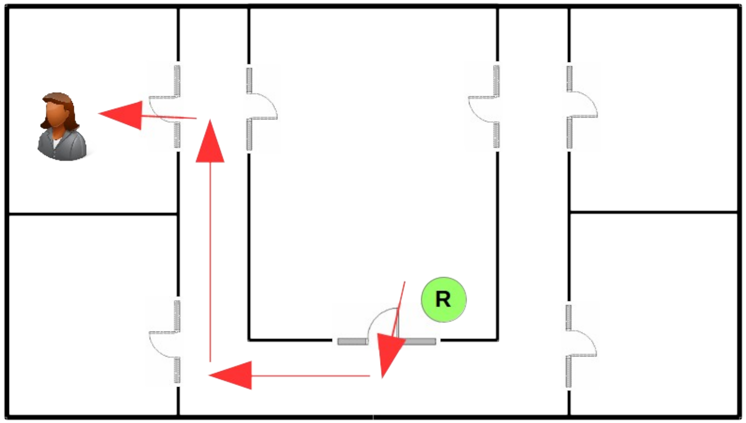






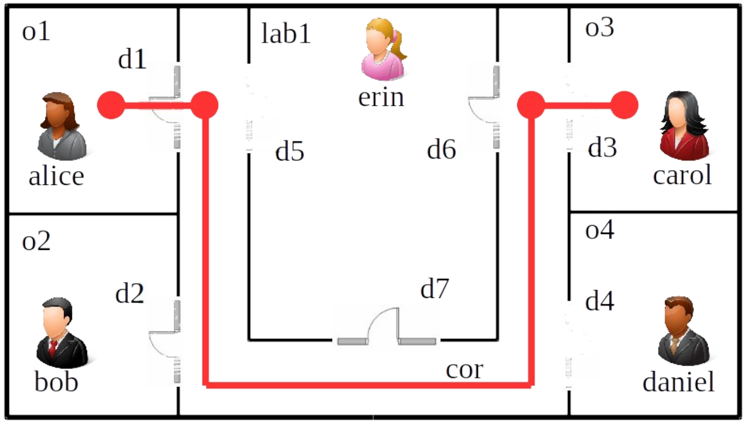
- PlanHG (global) considers all at the same time
- PlanHL (local) considers adjacent pairs



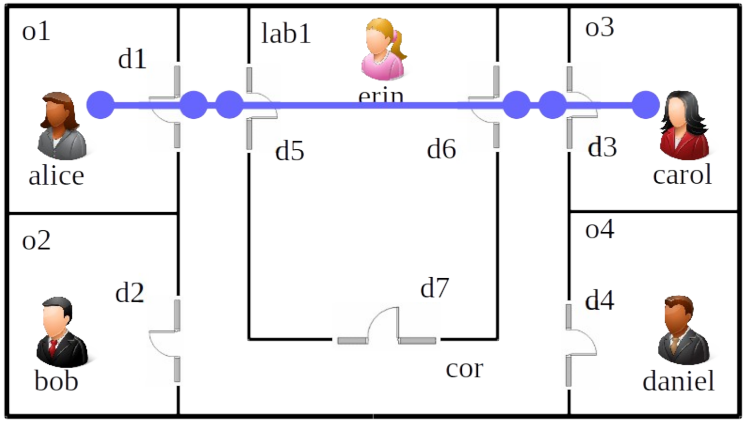


Two types of planning problems

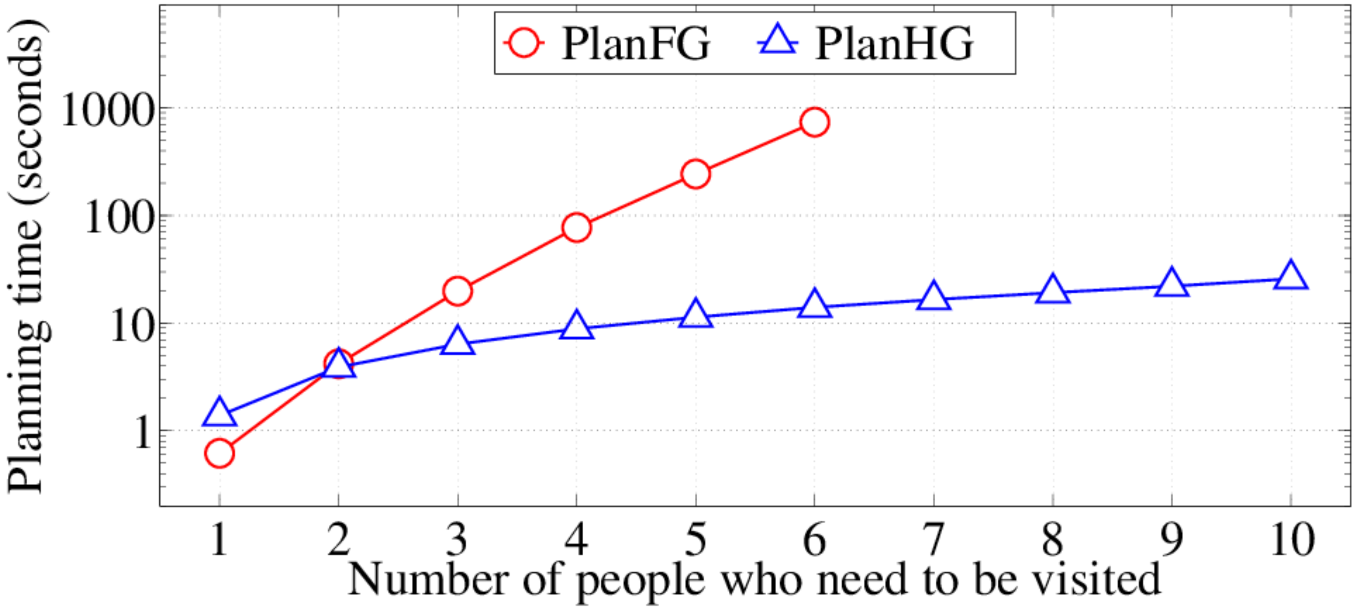
- Type-I: ***short plan*** generation



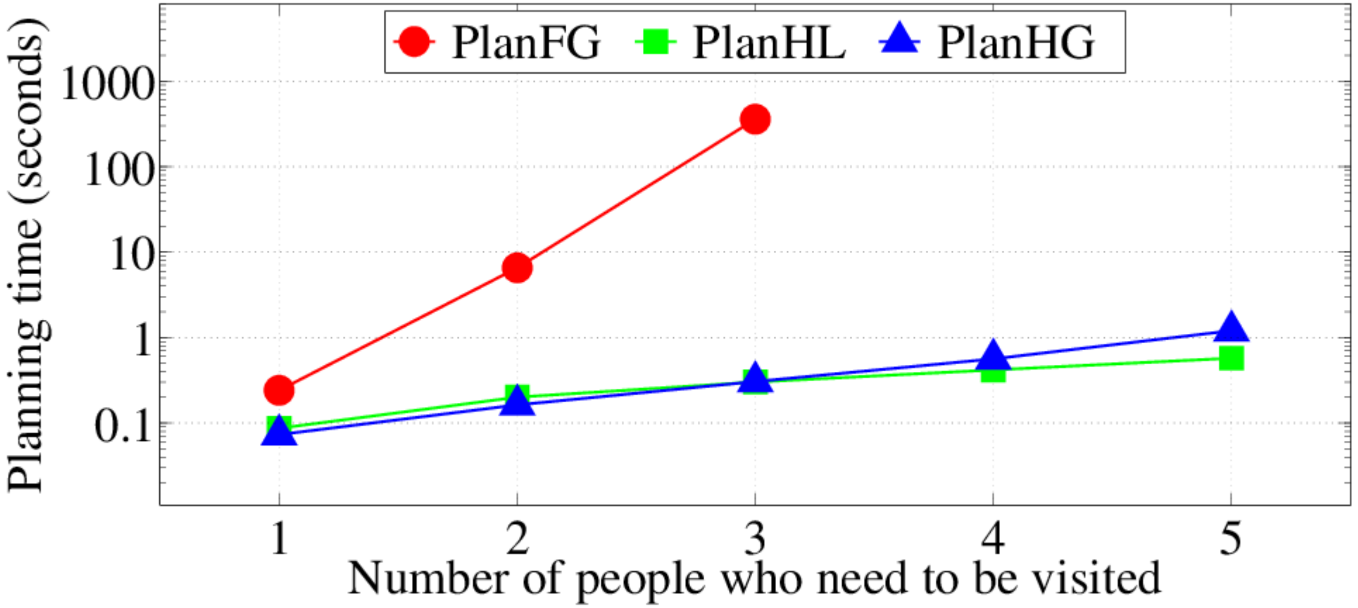
- Type-II: ***low-cost plan*** generation



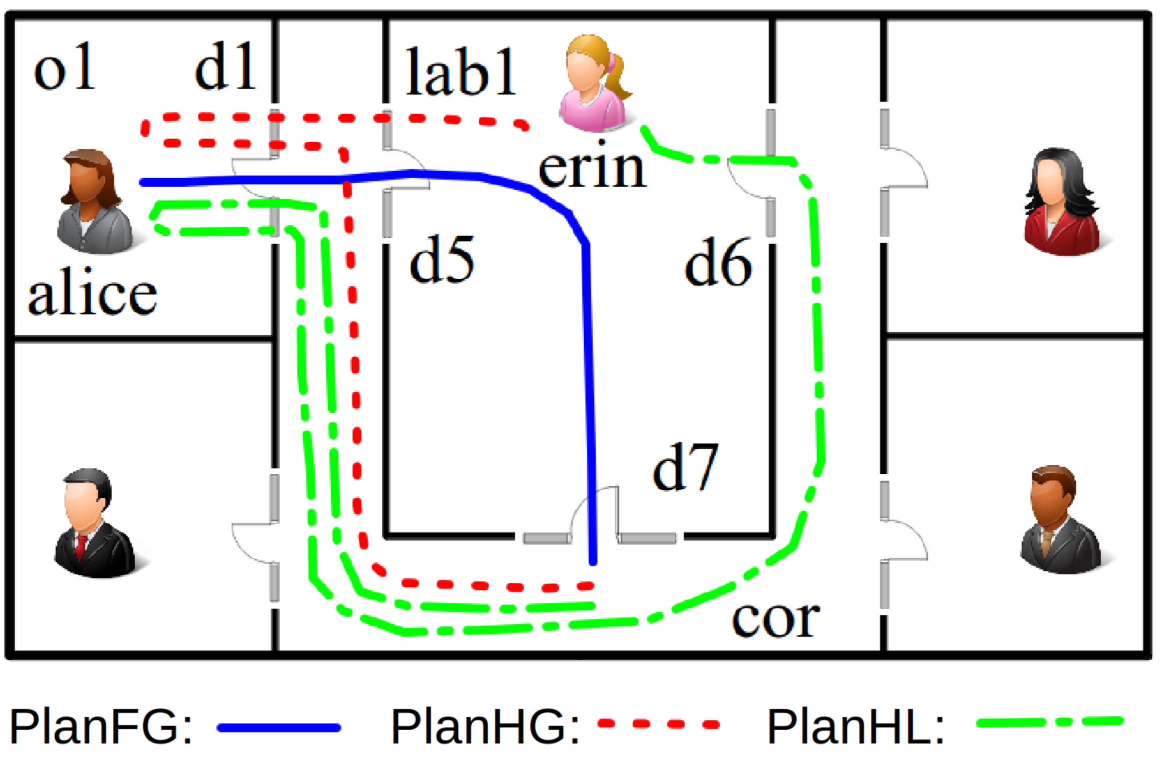
Experiments: ***short plan*** generation



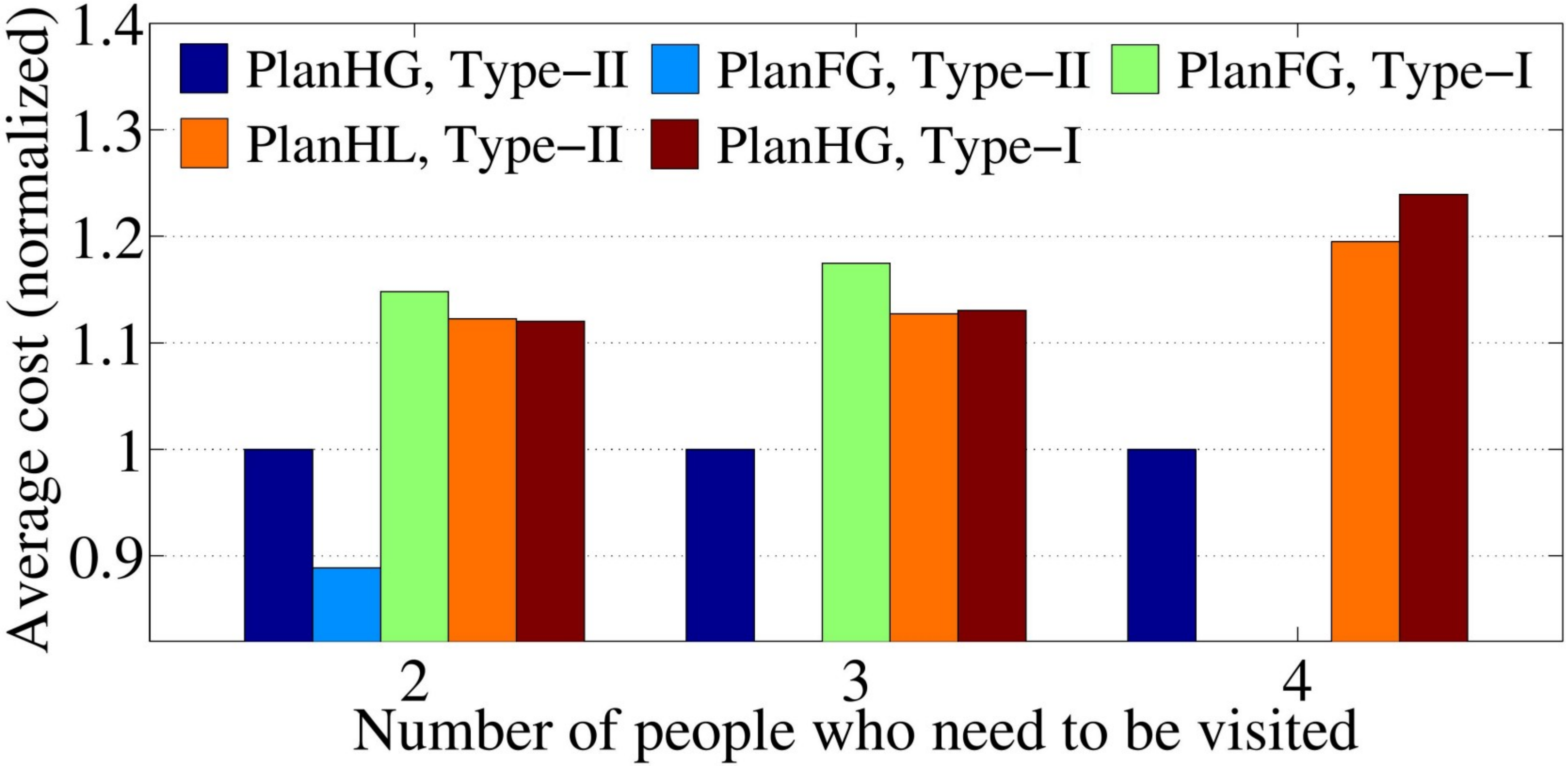
Experiments: ***low-cost plan*** generation



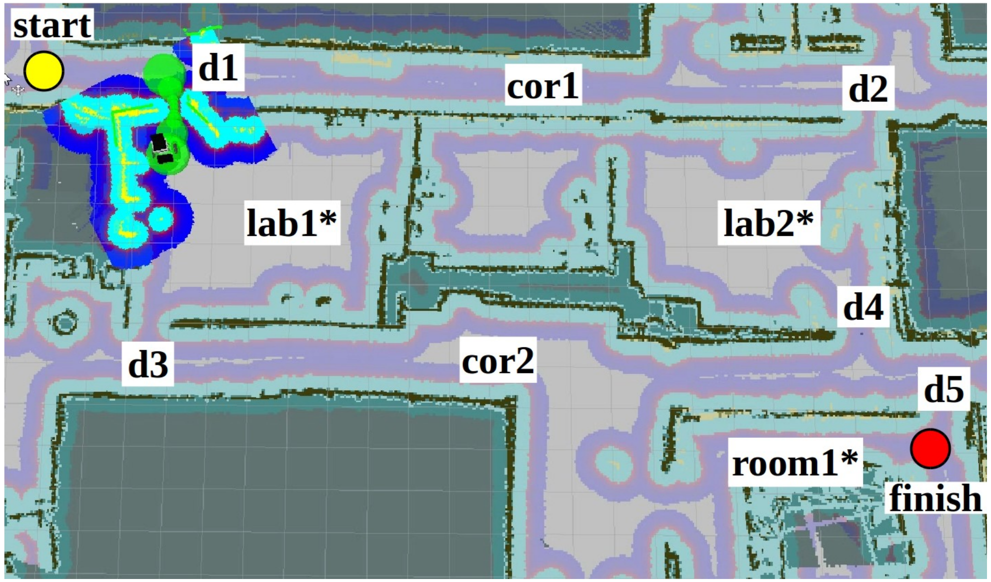
Experiments: evaluating ***plan quality***



Experiments: evaluating ***plan quality***



An illustrative trial on a real robot



(a) (b)

(a) A Segway-based robot preparing to go through a door

(b) Occupancy-grid map with a path planned for going through a door

https://youtu.be/-QpFj7BbiRU

Related work

- Xiaoping Chen, Jianmin Ji, Jiehui Jiang, Guoqiang Jin, Feng Wang, and Jiongkun Xie.
  Developing High-Level Cognitive Functions For Service Robots, AAMAS, 2010

- Jurgen Dix, Ugur Kuter, and Dana Nau.
  Planning in answer set programming using ordered task decomposition, Springer, 2003.

- Kutluhan Erol, James A. Hendler, and Dana S. Nau.
  HTN Planning: Complexity and Expressivity, AAAI, 1994.

- Piyush Khandelwal, Fangkai Yang, Matteo Leonetti, Vladimir Lifschitz, and Peter Stone.
  Planning in Action Language BC while Learning Action Costs for Mobile Robots, ICAPS, 2014

- Craig A Knoblock.
  Automatically Generating Abstractions For Planning, AIJ 1994

- Joohyung Lee, Vladimir Lifschitz, and Fangkai Yang.
  Action Language BC: A Preliminary Report, IJCAI 2013

- Tran Cao Son and Jorge Lobo.
  Reasoning about policies using logic programs.
  In AAAI Spring Symposium on Answer Set Programming, 2001

Thank you