Robotics Reading Group
@ Instituto Superior Técnico

Session #5
13-12-2019

Rui Silva

Paper

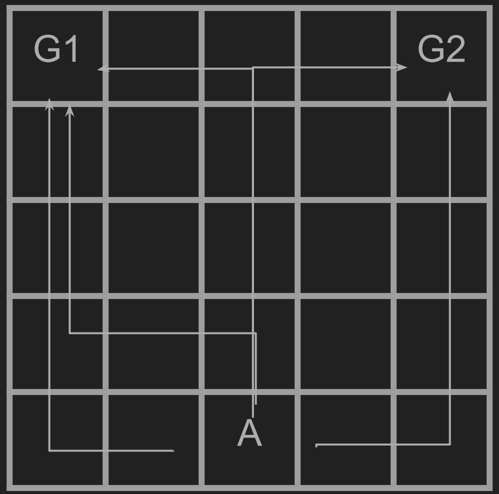
Sarah Keren, Avigdor Gal, & Erez Karpas. 2014.
Goal Recognition Design
Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling

Goal Recognition Design

Goal Recognition
+
Design

Goal Recognition

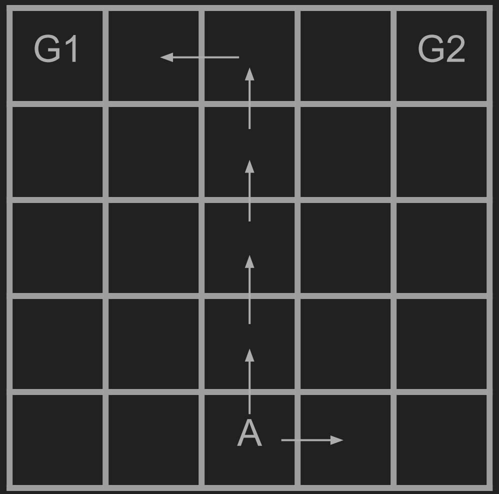
- Given a set of possible goals, and observations of an agent acting...
- Recognize the goal of the agent.



A, optimal agent

Goal Recognition

- Given a set of possible goals, and observations of an agent acting...
- Recognize the goal of the agent.

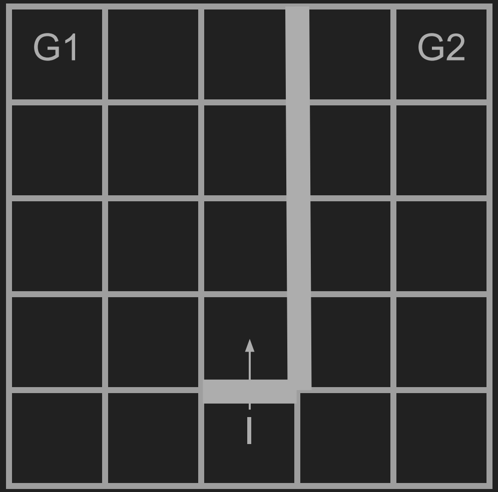


A, optimal agent

Goal Recognition **Design**

Offline design as a mechanism to facilitate **online** goal recognition

"What is the best way to modify the world so that any agent acting within it reveals its objective as early as possible"



Goal Recognition Design

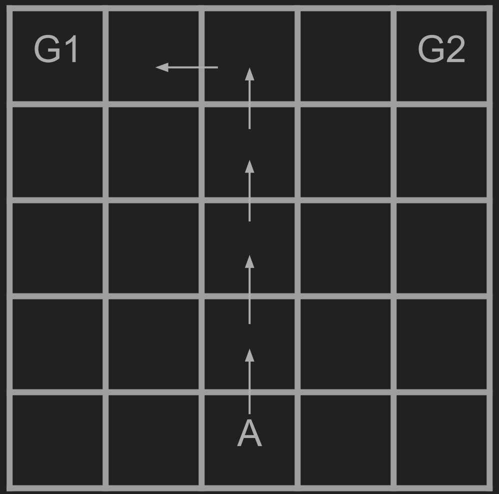
Approach:

1. **Evaluate:** Measure how long it takes to recognize the agent's goal in the worst case
2. **Optimize:** Reduce this worst case time

Evaluate: Measuring how hard to detect agent's goal

Worst case distinctiveness ***(wcd)***

- maximal length of a path until the objective of the agent becomes clear



Wcd = 4

Evaluate: Computing wcd

Naive algorithm (high level)

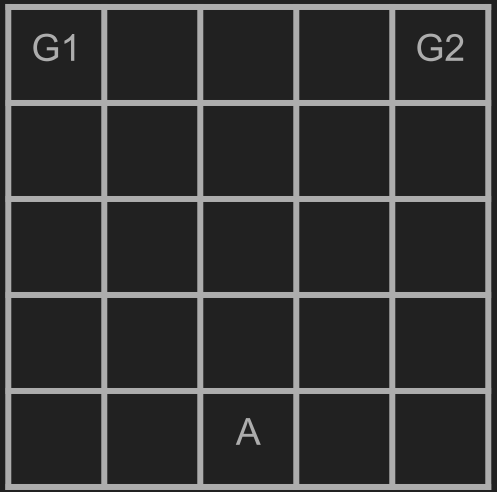
- Compute all optimal paths.
- Return the longest subpath common to >1 goals

Evaluate: Computing wcd

Naive algorithm

1. BFS to find optimal path to goals
2. Backtrack from each goal. Stop once we find a node that is on a path to both goals.

First node in path to both goals reveals the wcd.

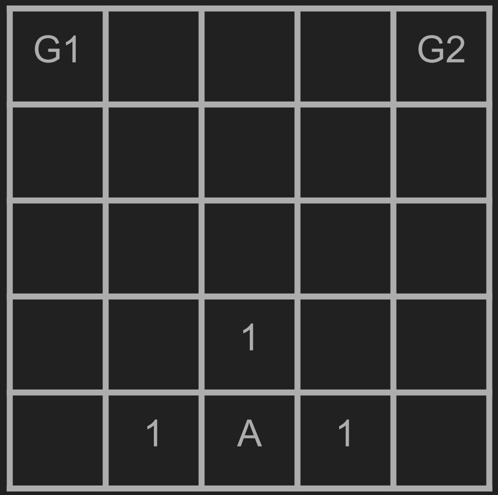


Evaluate: Computing wcd

Naive algorithm

1. **BFS to find optimal path to goals**
2. Backtrack from each goal. Stop once we find a node that is on a path to both goals.

First node in path to both goals reveals the wcd.

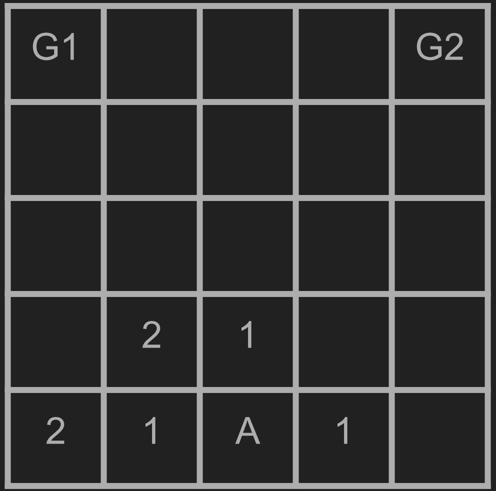


Evaluate: Computing wcd

Naive algorithm

1. **BFS to find optimal path to goals**
2. Backtrack from each goal. Stop once we find a node that is on a path to both goals.

First node in path to both goals reveals the wcd.



Evaluate: Computing wcd

Naive algorithm

1. **BFS to find optimal path to goals**
2. Backtrack from each goal. Stop once we find a node that is on a path to both goals.

First node in path to both goals reveals the wcd.

G1				G2
		2		
	2	1	2	
2	1	A	1	

Evaluate: Computing wcd

Naive algorithm

1. **BFS to find optimal path to goals**
2. Backtrack from each goal. Stop once we find a node that is on a path to both goals.

First node in path to both goals reveals the wcd.

G1				G2
		2		
	2	1	2	
2	1	A	1	2

Evaluate: Computing wcd

Naive algorithm

1. **BFS to find optimal path to goals**
2. Backtrack from each goal. Stop once we find a node that is on a path to both goals.

First node in path to both goals reveals the wcd.

G1	5	4	5	G2
5	4	3	4	5
4	3	2	3	4
3	2	1	2	3
2	1	A	1	2

Evaluate: Computing wcd

Naive algorithm

1. BFS to find optimal path to goals
2. **Backtrack from each goal.**
   Marks nodes on optimal path to goal.
   Stop once we find a node that is on a path to both goals.

First node in path to both goals reveals the wcd.

G1	G1	4	5	G2
G1	4	3	4	5
4	3	2	3	5
3	2	1	2	3
2	1	A	1	2

Evaluate: Computing wcd

Naive algorithm

1. BFS to find optimal path to goals
2. **Backtrack from each goal.**
   Marks nodes on optimal path to goal.
   Stop once we find a node that is on a path to both goals.

First node in path to both goals reveals the wcd.

G1	G1	4	G2	G2
G1	4	3	4	G2
4	3	2	3	5
3	2	1	2	3
2	1	A	1	2

Evaluate: Computing wcd

Naive algorithm

1. BFS to find optimal path to goals
2. **Backtrack from each goal.**
   Marks nodes on optimal path to goal.
   Stop once we find a node that is on a path to both goals.

First node in path to both goals reveals the wcd.

G1	G1	4	G2	G2
G1	G1	3	4	G2
G1	3	2	3	5
3	2	1	2	3
2	1	A	1	2

Evaluate: Computing wcd

Naive algorithm

1. BFS to find optimal path to goals
2. **Backtrack from each goal.**
   Marks nodes on optimal path to goal.
   Stop once we find a node that is on a path to both goals.

First node in path to both goals reveals the wcd.

G1	G1	G1	G2	G2
G1	G1	3	4	G2
G1	3	2	3	5
3	2	1	2	3
2	1	A	1	2

Evaluate: Computing wcd

Naive algorithm

1. BFS to find optimal path to goals
2. **Backtrack from each goal.**
   Marks nodes on optimal path to goal.
   Stop once we find a node that is on a path to both goals.

First node in path to both goals reveals the wcd.

G1	G1	G1	G2	G2
G1	G1	3	G2	G2
G1	3	2	3	G2
3	2	1	2	3
2	1	A	1	2

Evaluate: Computing wcd

Naive algorithm

1. BFS to find optimal path to goals
2. **Backtrack from each goal.**
   Marks nodes on optimal path to goal.
   Stop once we find a node that is on a path to both goals.

First node in path to both goals reveals the wcd.

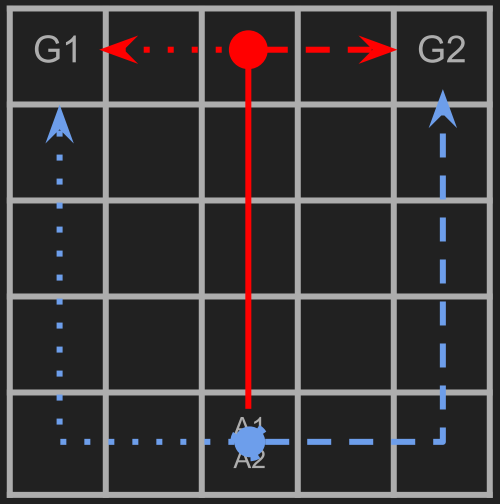
Inefficient when there are many optimal paths to each goal!

G1	G1	G1 G2	G2	G2
G1	G1	3	G2	G2
G1	3	2	3	G2
3	2	1	2	3
2	1	A	1	2

Evaluate: Computing wcd

Latest split algorithm (high level)

- Create a new 2 agent planning problem, where agents:
    - Have a different goal
    - Can act separately or together
        - Can only act together in the beginning. Once they split, they must act separately
        - Encouraged to act together with a smaller cost
- Optimal solution for new problem yields the wcd path
    - Wcd = time when agents decided to split



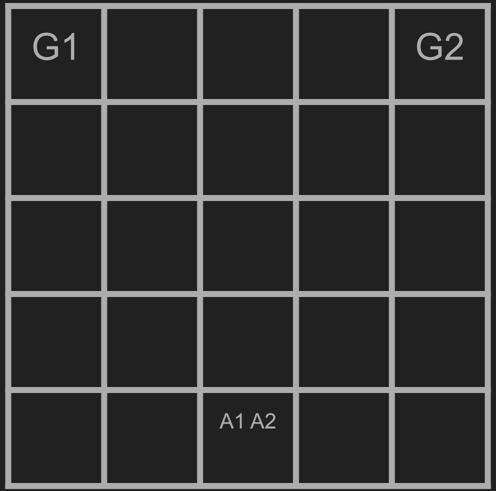
C > C

Evaluate: Computing wcd

Latest split algorithm

- Extend planning problem as:
  - Extend state space with the position of each agent
  - Extend action space with:
    - Independent actions for each agent - a1 and a2
    - Joint actions - a12
    - Split
  - Cost function C’ such as:
    - C’(a1) = C’(a2)
    - C’(a12) = 2C(a) - eps

- Empirical evaluation shows *latest split* is significantly more efficient than naive algorithm

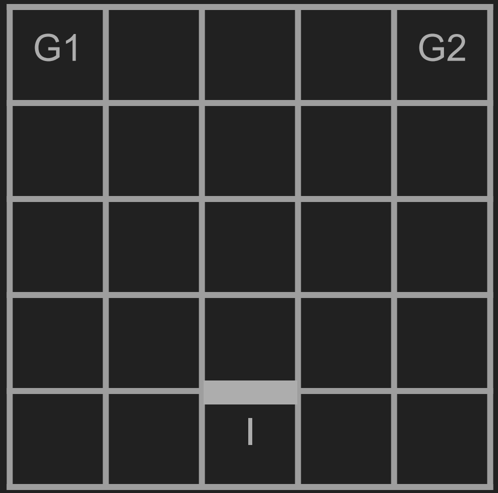


Optimize: Reduce wcd

Given a planning problem and the possible goals, how to reduce wcd?

Approach:

- Disallow the execution of **some actions** in **some states**



Removes (I, UP)

Optimize: Reduce wcd

Formally:

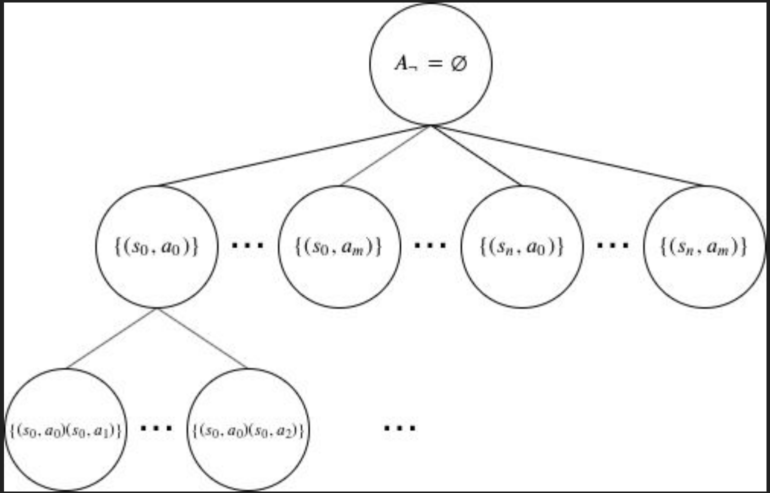
- Planning problem D
- $A_{\neg}$, set of pairs (s, a) --- action a disallowed in state s
- $D_{A \setminus A_{\neg}}$ new problem with disallowed state-action pairs
- **Goal:**

$$\begin{aligned} \underset{A_{\neg}}{\text{minimize}} \quad & (wcd(D_{A \setminus A_{\neg}}), |A_{\neg}|) \\ \text{subject to} \quad & \forall G,\ C^*_D(G) = C^*_{D_{A \setminus A_{\neg}}}(G) \end{aligned}$$

Optimize: Reduce wcd

Exhaustive search

- Each node represents a set $A_\neg$
- Start with empy set
- For each node,
  - Compute wcd and optimal costs of achieving each goal
- Successors formed by concatenating each state-action pair to set $A_\neg$ of previous node
- Search continues, increasing $A_\neg$, until reaching:
  - Model with wcd = 0
  - No more nodes to explore



Result:

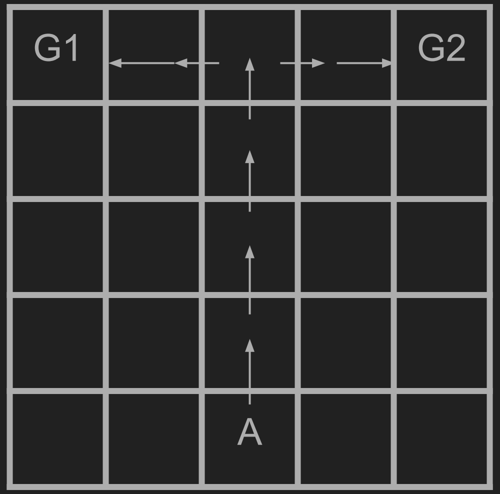
- $A_\neg^*$ for which wcd is minimized, and also smallest size.

Optimize: Reduce wcd

Pruned reduce

- Key insight: no point in removing actions not belonging to a *wcd* path

- We can prune a lot of search branches
    - Only create successors for state-action pairs that appear in the wcd path of parent node.



Empirical evaluation

Scenarios considered:

- Grid-navigation - simple navigation task
- Logistics - moving packages with trucks and airplanes
- Ipc-grid+ - complex navigation task
- Blockwords - block stacking

Empirical evaluation

Table 1: Calculating *wcd*

	time		expanded		% solved	
	bfs	split	bfs	split	bfs	split
grid navigation	20.3	0.4	928.4	48.1	100	100
ipc-grid+	7.4	1.4	1263.3	2328.7	100	100
logistics	59.6	27.5	51947.8	86596.1	89	92
block-words	45.6	1.9	3816.2	3473.1	99	100

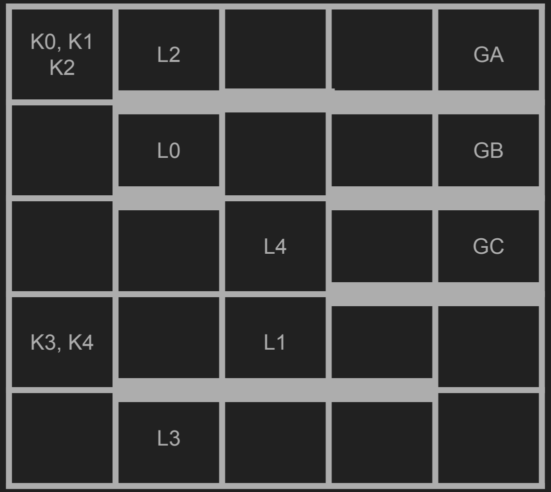
Empirical evaluation

Table 2: Reducing *wcd*

	% completed		% reduced		average reduction	
	exhaustive	reduce	exhaustive	reduce	exhaustive	reduce
grid navigation	9	95	18	21	1.64	3.45
ipc-grid+	44	85	28	47	2.07	3.36
logistics	22	86	5	14	2.1	3.46
block-words	11	63	2	9	1	1

Empirical evaluation

- Original domain:
  - Grab K2; Right; Unlock L2; Right; Right; Right
  - Grab K2; Right; Unlock L2; Right; Bottom; Right; Right
  - Wcd = 4
- Modified domain
  - wcd = 0



I wanna know more about GRD!

	Agent		Environment		Metrics		Designs		
	Suboptimal Plans	Partial Obs.	Partial Obs.	Stochastic Actions	*wcd*	*ecd*	Action Removal	Sensor Refinement	Action Conditioning
Keren *et al.* (ICAPS 2014)					✓		✓		
Son *et al.* (AAAI 2016)					✓		✓		
Keren *et al.* (AAAI 2015)	✓				✓		✓		
Keren *et al.* (AAAI 2016)	✓		✓		✓		✓		
Keren *et al.* (IJCAI 2016)	✓		✓		✓		✓	✓	
Wayllace *et al.* (IJCAI 2016)				✓	✓		✓		
Wayllace *et al.* (IJCAI 2017)				✓	✓	✓	✓		
Wayllace *et al.* (HSDIP 2018)			✓	✓	✓		✓	✓	
Keren *et al.* (ICAPS 2018)	✓		✓		✓		✓	✓	✓
Keren *et al.* (JAIR 2019)	✓		✓		✓		✓	✓	✓
Keren *et al.* (HSDIP 2019)		✓			✓			✓	

Connection to GAIPS?

- João Ribeiro - Ad Hoc Teamwork
  - GRD as mechanism to speed up adhoc teamwork?
- Miguel Faria - Trajectory legibility
  - Traj. legibility and grd are dual problems? Sweet spot is hybrid approach?
- Robotics in general
  - World is typically built for humans. Could GRD help robots more easily understanding human intentions?

More resources

https://www.cse.wustl.edu/~wyeoh/GRD-Tutorial.pdf

Merry Christmas 

We will see you in 2020!