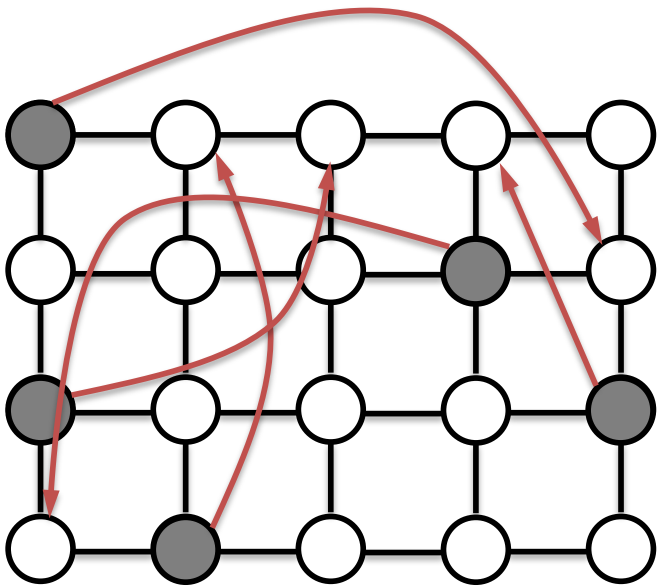
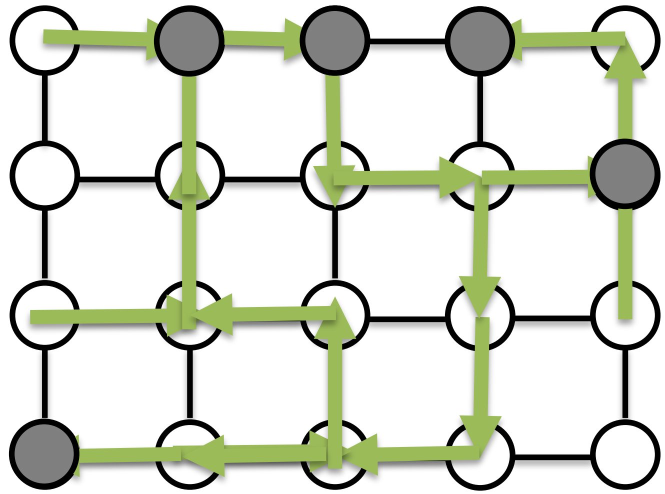
Multi-Agent Pathfinding:
Models, Solvers, and Systems

Roman Barták, Philipp Obermeier, Torsten Schaub, Tran Cao Son, Roni Stern

What is **multi-agent path finding** (MAPF)?





MAPF problem:

Find a **collision-free** plan (path) for each agent

Alternative names:

cooperative path finding (CPF), multi-robot path planning, pebble motion

Tutorial outline

Part I: Introduction to Multi-agent pathfinding (MAPF)

- *Problem formulation, variants, objectives*
- *Application areas*

Part II: Search-based solvers

- *Incomplete solvers*
- *Complete but suboptimal solvers*
- *Optimal solvers*

Part III: Reduction-based solvers

- *SAT encodings of MAPF*
- *CP encoding of MAPF*

Part IV: Demos

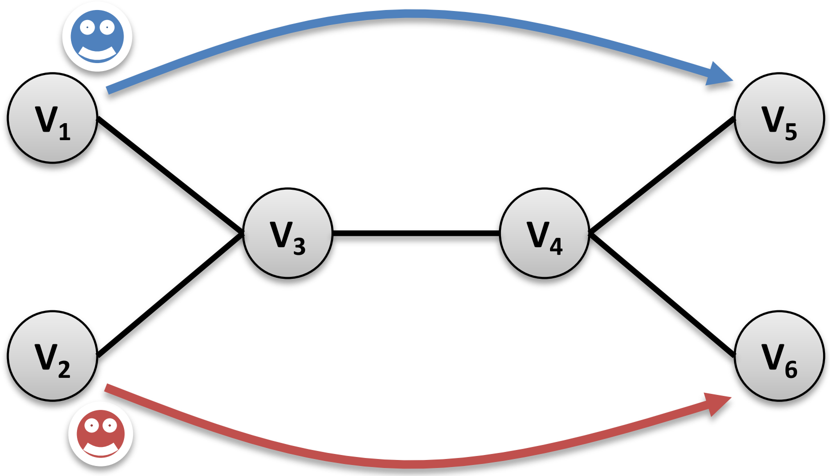
- *MAPF Scenario (MAPF for Ozobots)*
- *ASPRILO system (an abstract benchmark environment for robotic intra-logistics)*

Part V: Open challenges, conclusion, discussion

Part I:

INTRODUCTION TO MAPF

- a **graph** (directed or undirected)
- a set of **agents**, each agent is assigned to two locations (nodes) in the graph (start, destination)



Each agent can perform either **move** (to a neighboring node) or **wait** (in the same node) actions.

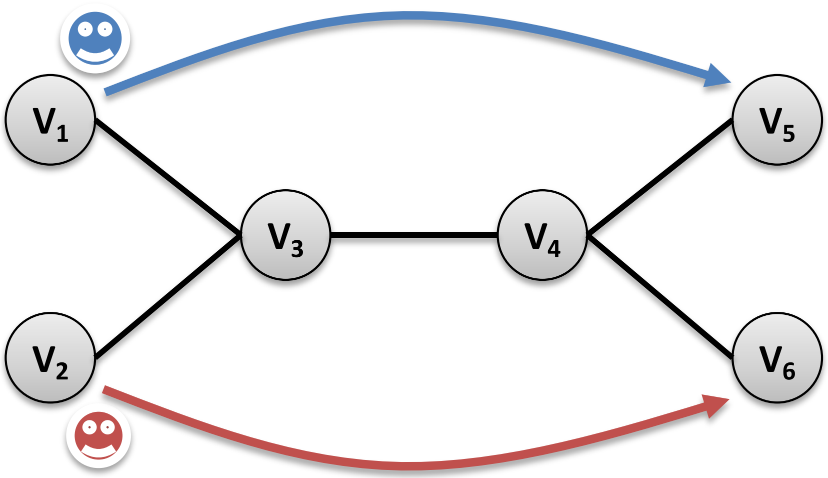
Typical assumption:

all move and wait actions have identical durations (plans for agents are synchronized)

Plan is a sequence of actions for the agent leading from its start location to its destination.

The **length of a plan** (for an agent) is defined by the time when the agent reaches its destination and does not leave it anymore.

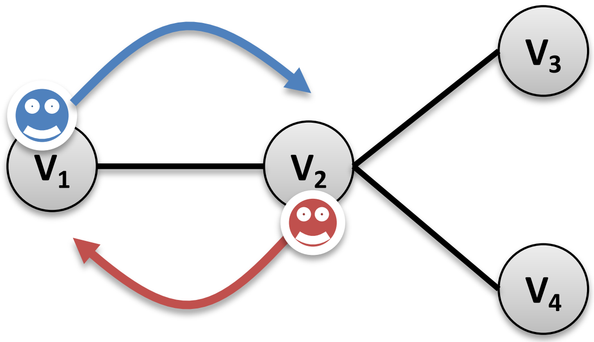
Find **plans** for all agents such that the plans **do not collide in time and space** (no two agents are at the same location at the same time).



time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	wait $\mathbf{v_1}$	move $\mathbf{v_3}$
2	move $\mathbf{v_3}$	move $\mathbf{v_4}$
3	move $\mathbf{v_4}$	move $\mathbf{v_6}$
4	move $\mathbf{v_5}$	wait $\mathbf{v_6}$

Some necessary **conditions for plan existence**:

- no two agents are at the same start node
- no two agents share the same destination node (unless an agent disappears when reaching its destination)
- the number of agents is strictly smaller than the number of nodes



> Agent at $\mathbf{v_i}$ cannot perform **move $\mathbf{v_j}$** at the same time when agent at $\mathbf{v_j}$ performs **move $\mathbf{v_i}$**

Agents may swap position

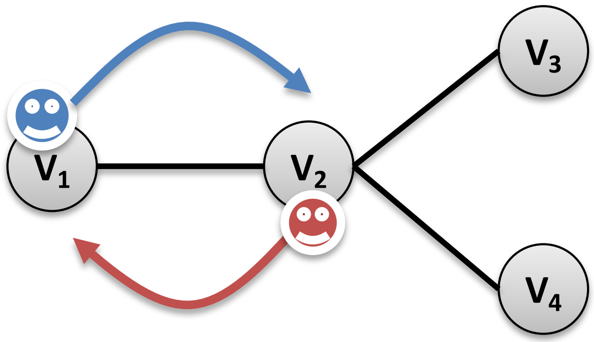
time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	move $\mathbf{v_2}$	move $\mathbf{v_1}$

Agents use the same edge at the same time!

Swap is not allowed.

time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	move $\mathbf{v_2}$	move $\mathbf{v_3}$
2	move $\mathbf{v_4}$	move $\mathbf{v_2}$
3	move $\mathbf{v_2}$	move $\mathbf{v_1}$

No-train constraint



Agent at $\mathbf{v_i}$ cannot perform **move** $\mathbf{v_j}$ if there is another agent at $\mathbf{v_j}$

Agent can approach a node that is currently occupied but will be free before arrival.

time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	move $\mathbf{v_2}$	move $\mathbf{v_3}$
2	move $\mathbf{v_4}$	move $\mathbf{v_2}$
3	move $\mathbf{v_2}$	move $\mathbf{v_1}$

Agents form a **train**.

Trains may be forbidden.

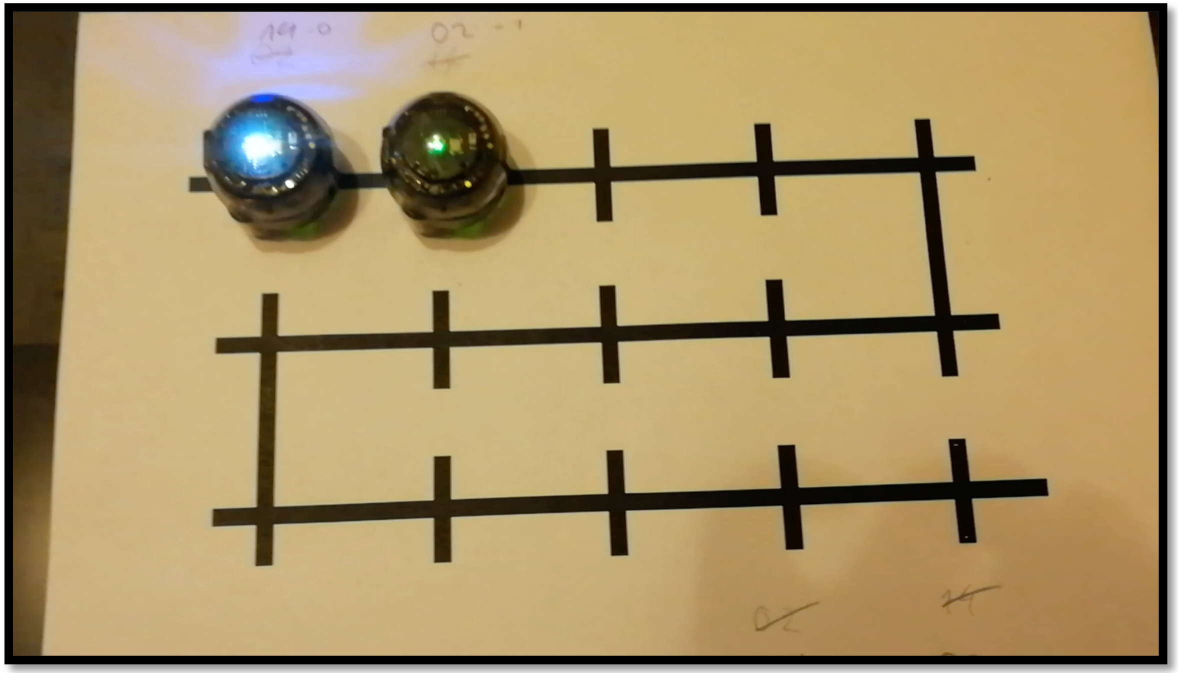
time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	*wait* $\mathbf{v_1}$	move $\mathbf{v_3}$
2	move $\mathbf{v_2}$	*wait* $\mathbf{v_3}$
3	move $\mathbf{v_4}$	*wait* $\mathbf{v_3}$
4	*wait* $\mathbf{v_4}$	move $\mathbf{v_2}$
5	*wait* $\mathbf{v_4}$	move $\mathbf{v_1}$
6	move $\mathbf{v_2}$	*wait* $\mathbf{v_1}$

If any agent is delayed then trains may cause collisions during execution.

To prevent such collisions we may introduce more space between agents.

k-robustness

An agent can visit a node, if that node has not been occupied in recent *k* steps.



1-robustness covers both no-swap and no-train constraints

- No plan (path) has a cycle.
- No two plans (paths) visit the same same location.
- Waiting is not allowed.
- Some specific locations must be visited.
- …

How to measure quality of plans?

Two typical criteria (to minimize):

- **Makespan**
  - distance between the start time of the first agent and the completion time of the last agent
  - maximum of lengths of plans (end times)
- **Sum of costs (SOC)**
  - sum of lengths of plans (end times)

time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	wait $\mathbf{v_1}$	move $\mathbf{v_3}$
2	move $\mathbf{v_3}$	move $\mathbf{v_4}$
3	move $\mathbf{v_4}$	move $\mathbf{v_6}$
4	move $\mathbf{v_5}$	wait $\mathbf{v_6}$

Makespan = 4
SOC = 7

Optimal single agent path finding is tractable.

– e.g. Dijkstra's algorithm

Sub-optimal multi-agent path finding (with two free unoccupied nodes) is tractable.

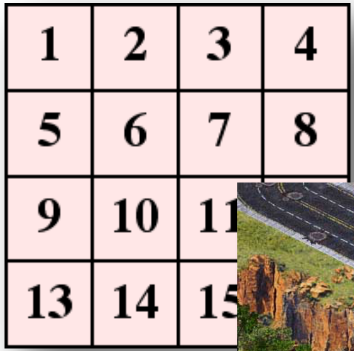
– e.g. algorithm Push and Rotate

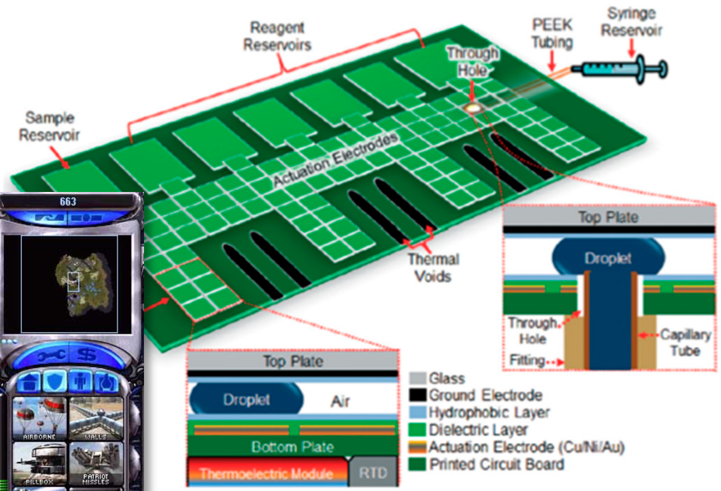
MAPF, where agents have joint goal nodes (it does not matter which agent reaches which goal) is tractable.

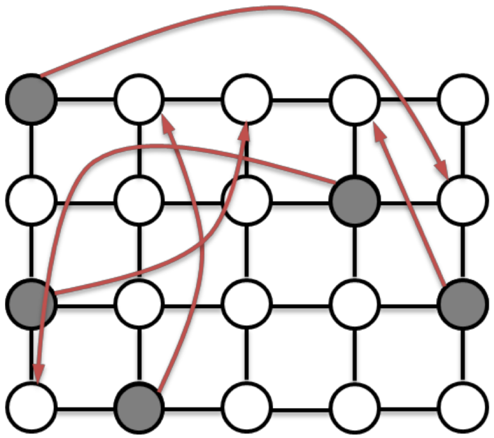
– reduction to min-cost flow problem

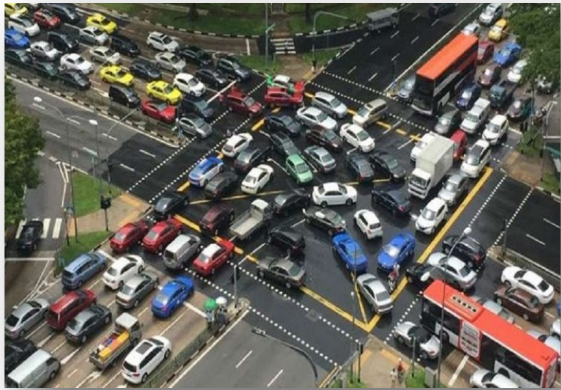
Optimal (makespan, SOC) multi-agent path finding is **NP-hard**.

Applications









Offline MAPF

Online MAPF

	Warehouse	Intersection
Fixed set of agents	Fixed set of agents	Sequence of agents
One task per agent	Sequence of tasks	One task per agent

Search-based techniques

- state-space search (A*)
  - state = location of agents at nodes
  - transition = performing one action for each agent
- conflict-based search

Reduction-based techniques

- translate the problem to another formalism (SAT/CSP/ASP …)

Part II:

SEARCH-BASED SOLVERS

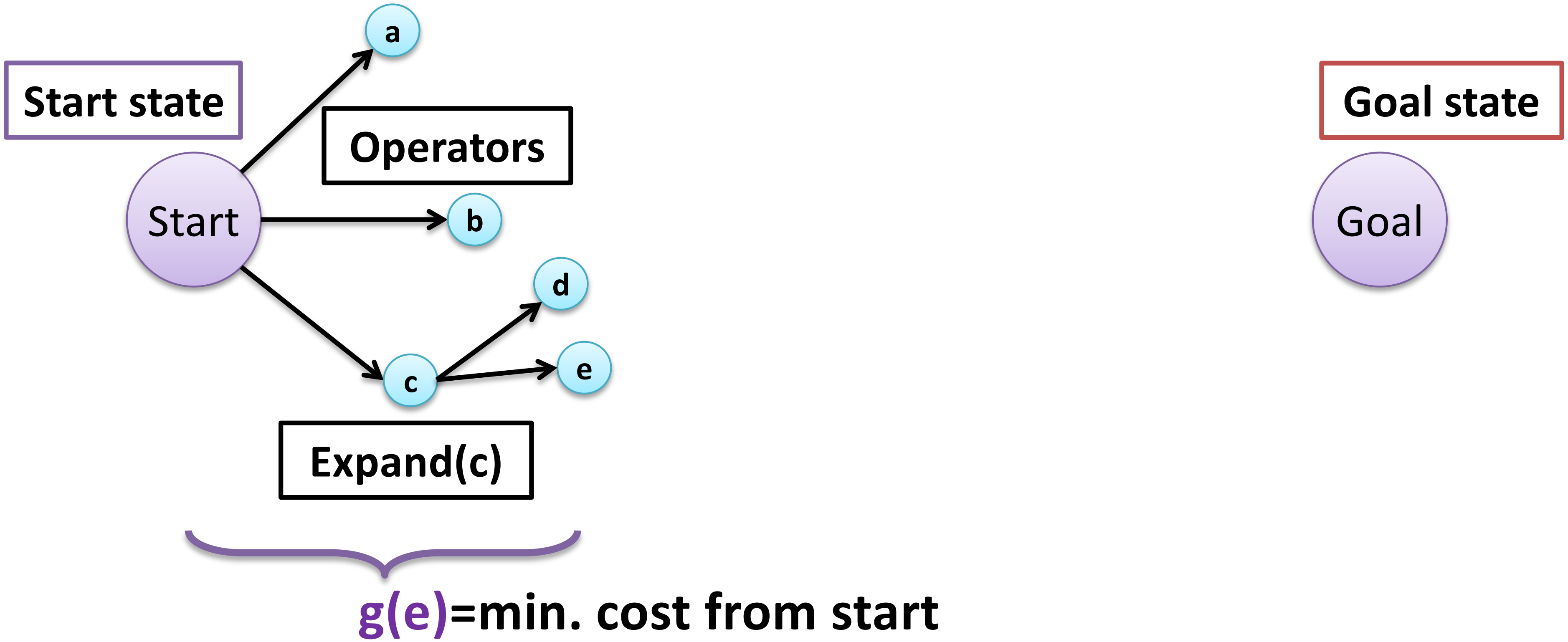
Some slides and animations taken from **Guni Sharon**, **Dor Atzmon**, and **Ariel Felner**

WHAT IS SEARCH?

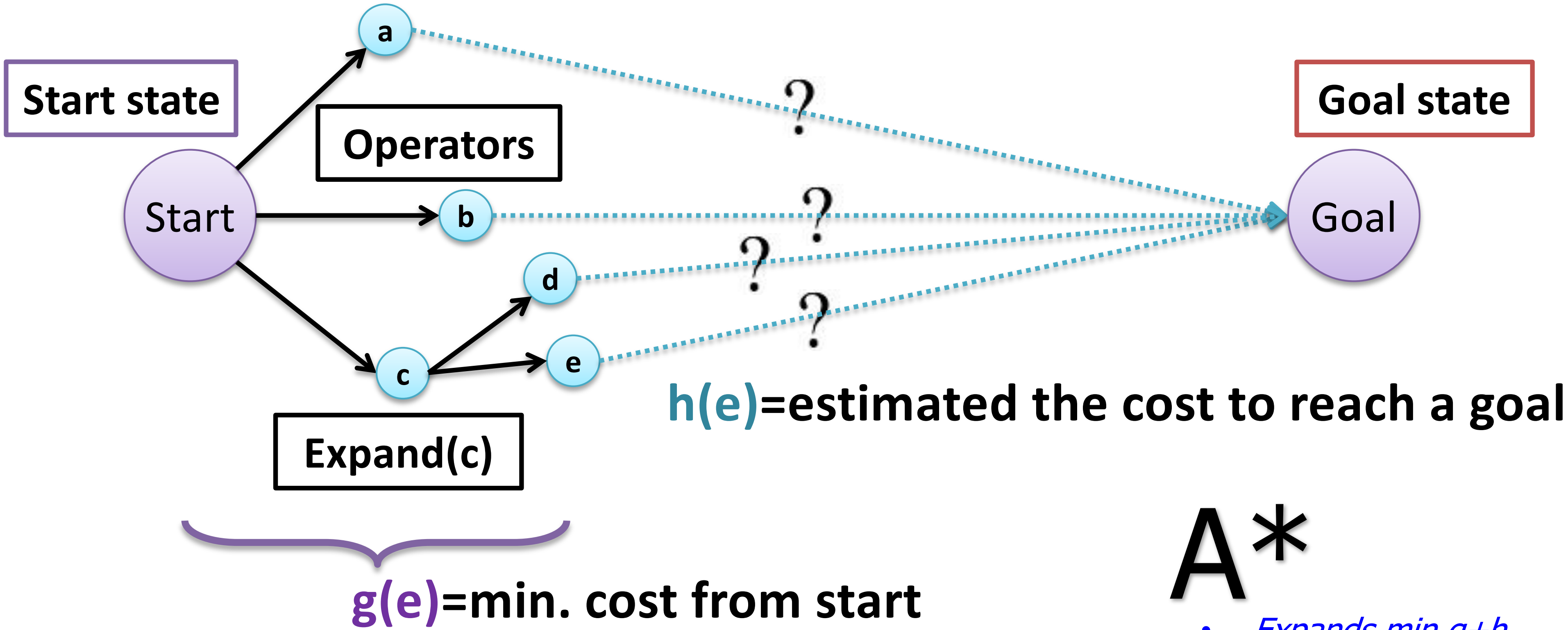
A General Problem Solving technique

Classical Search Setting

To expand or not expand, this is the question



To expand or not expand, this is the question



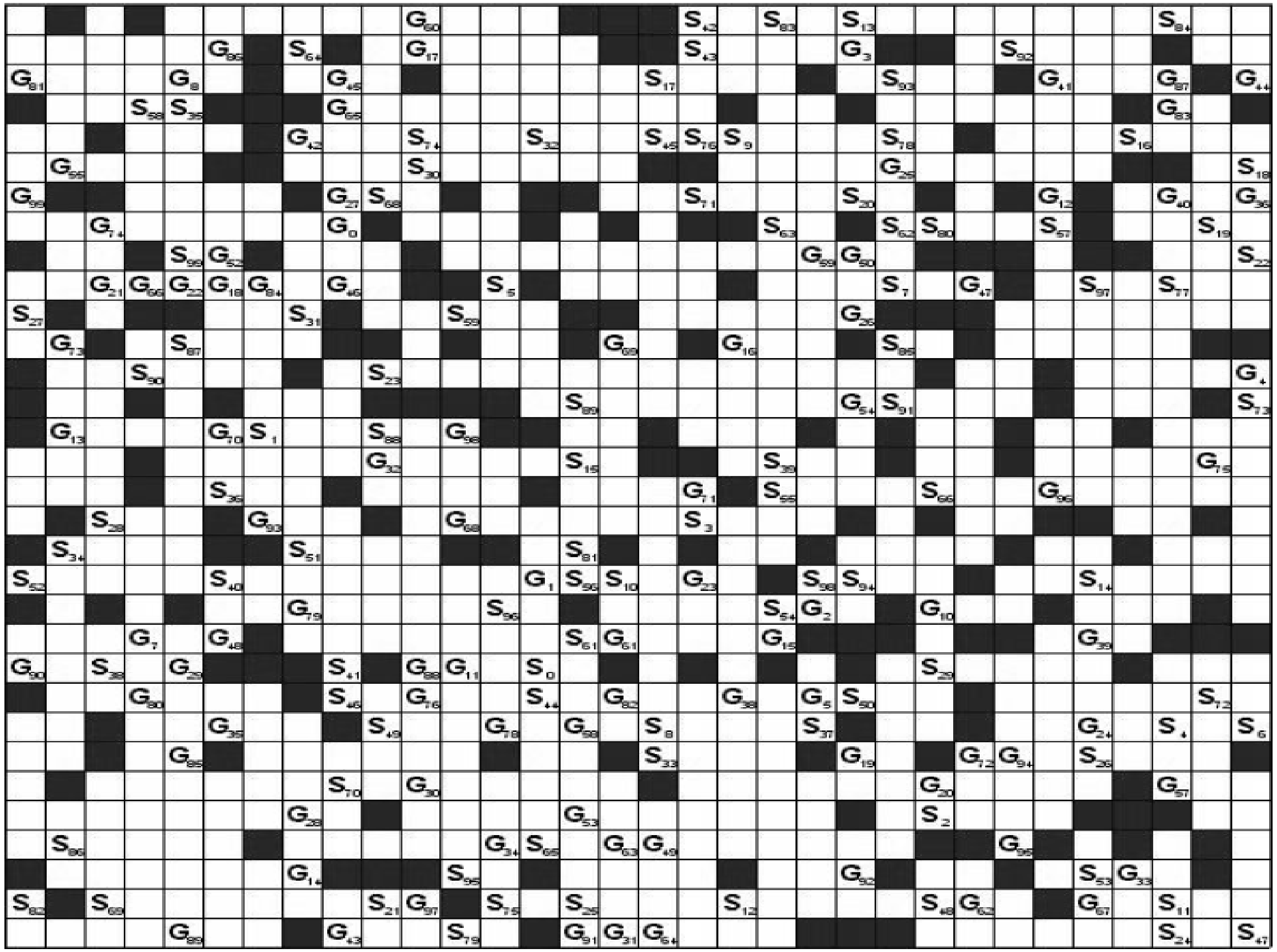
A*

- *Expands min g+h*
- *Returns optimal solutions*
- *"Optimally effective"*

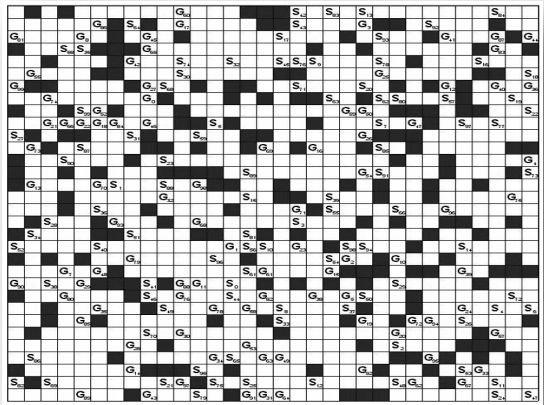
WHY SEARCH FOR MULTI-AGENT PATH FINDING?

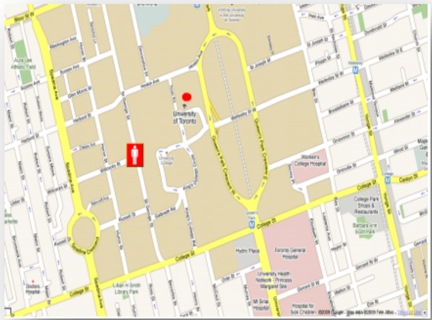
It Works!

Finding an **optimal** solution to **hundreds of agents**

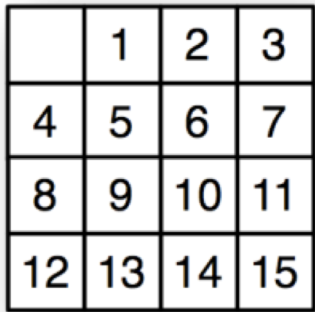


Classical Applications of Search





K (# agents)



K=1 (Navigation in explicit graphs)
Explicit graph

K=N-1 (Tile puzzle)
(Huge) Implicit graph

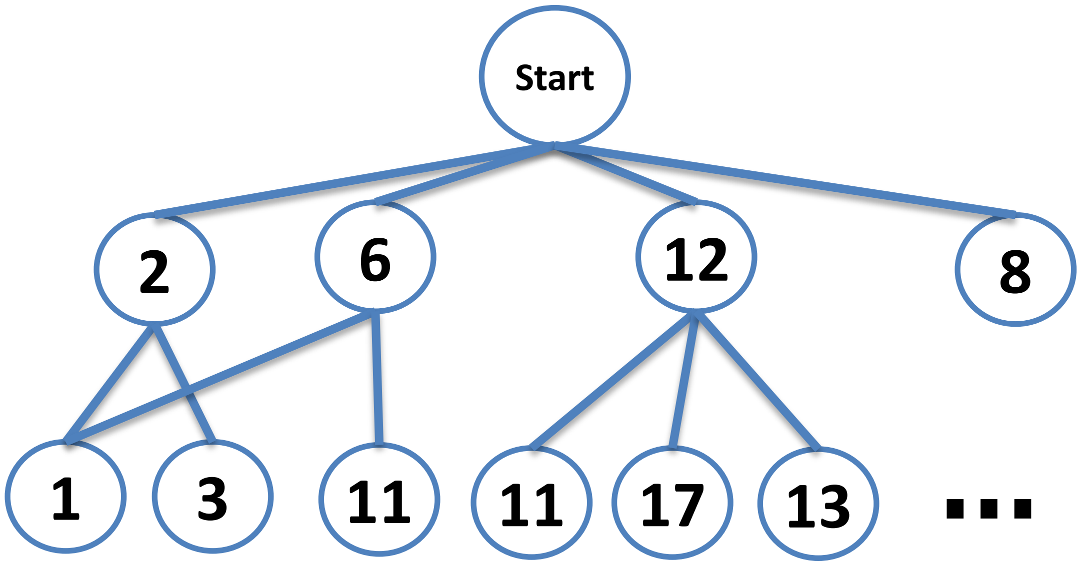
Solving Multi-Agent Path Finding with Search

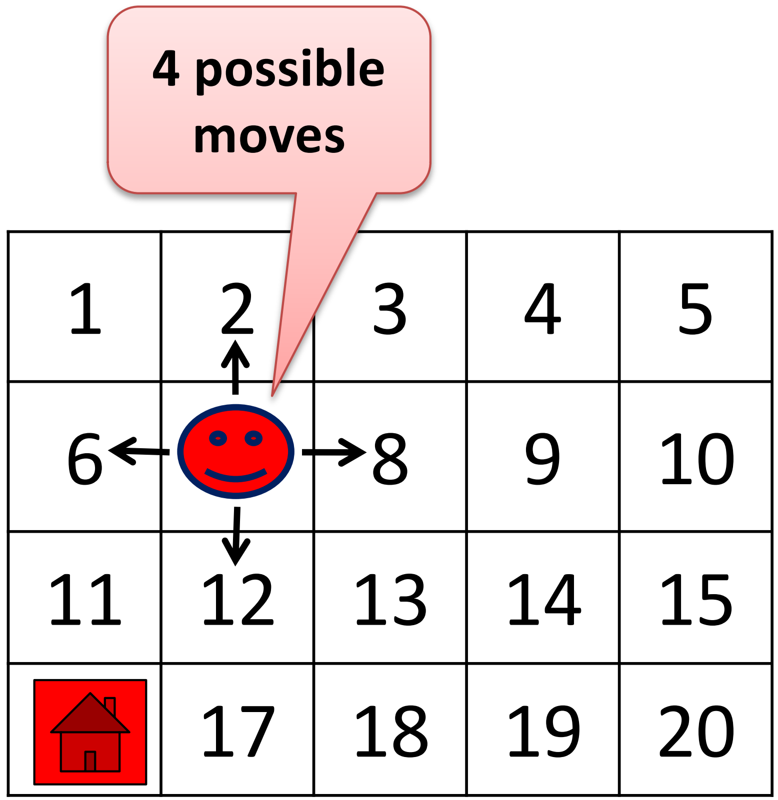
	Suboptimal	Optimal
Incomplete	?	?
Complete	?	?

Solving Multi-Agent Path Finding with Search

– From A* to **prioritized planners**
– From prioritized planners **back to A*** (+ID+OD/M*)
– The **Increasing Cost Tree Search** (ICTS)
– The **Conflict-Based Search** framework (CBS)
– Approximately optimal search-based solvers

Single Agent Pathfinding



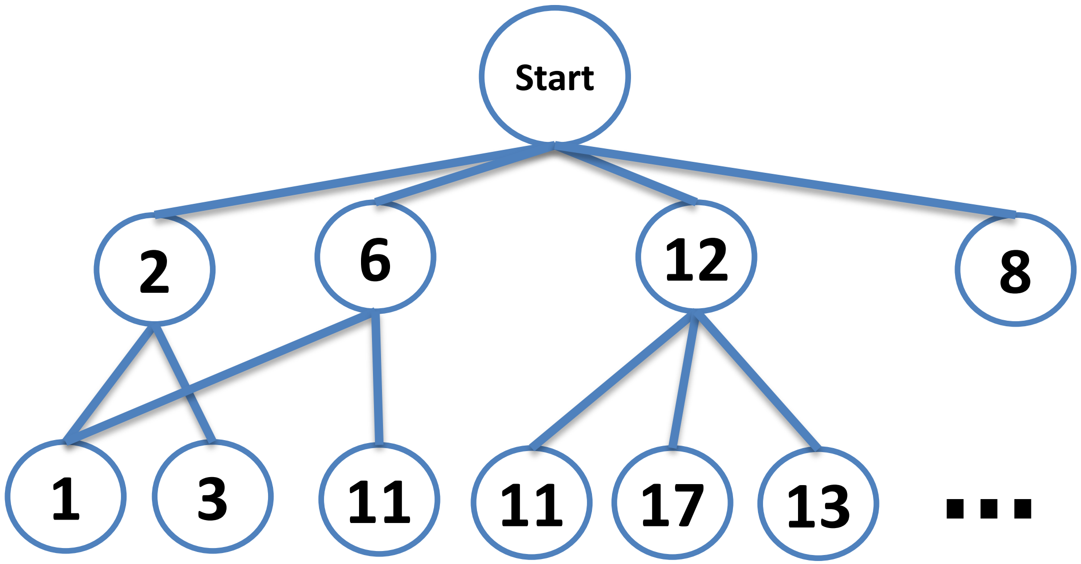


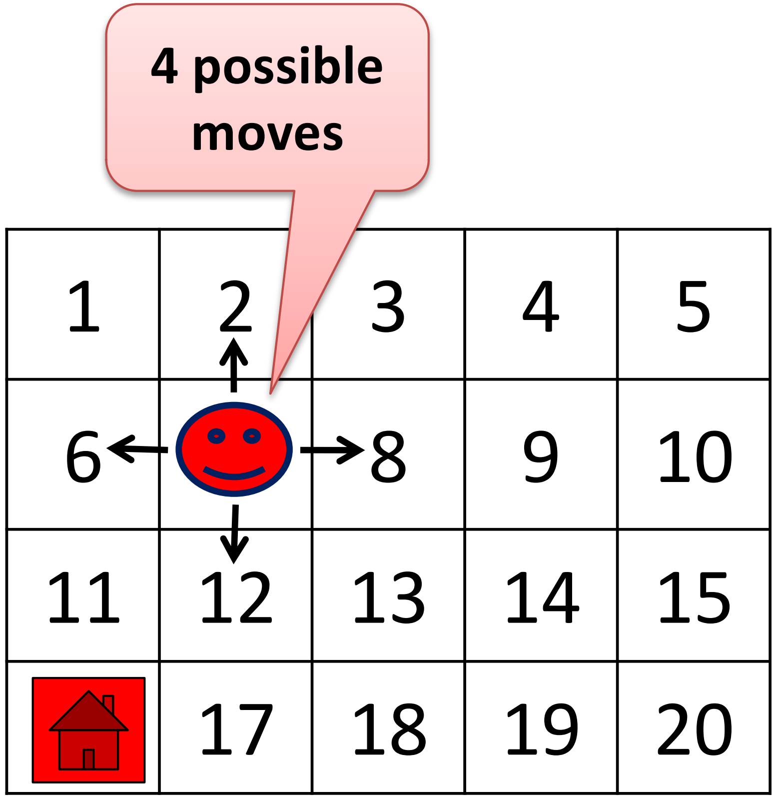
Search problem properties

- Number of states = ?
- Branching factor = ?

Classical search problem!

Single Agent Pathfinding



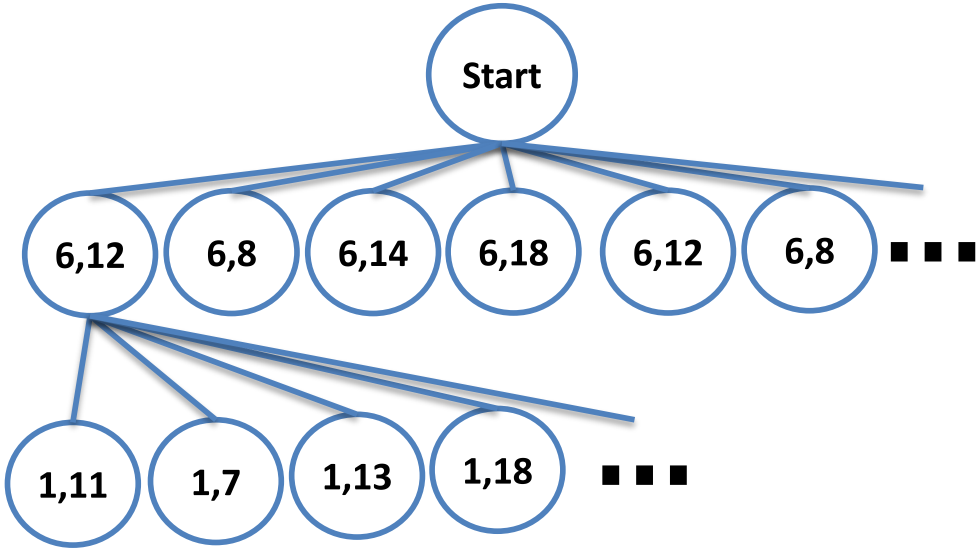


Search problem properties

- Number of states = 20
- Branching factor = 4

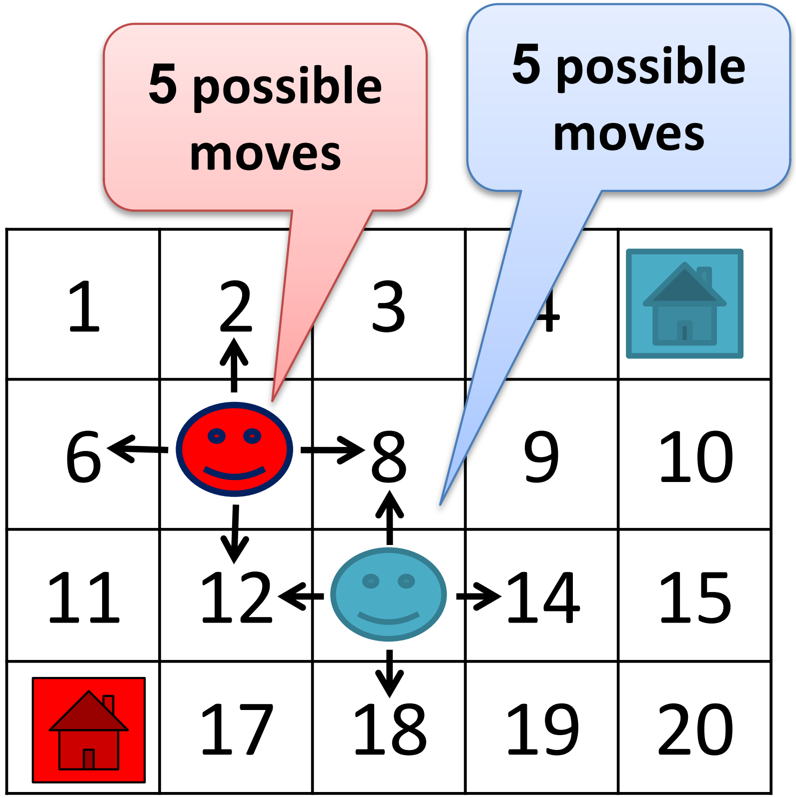
Classical search problem!

25 Possible moves! = 5 x 5

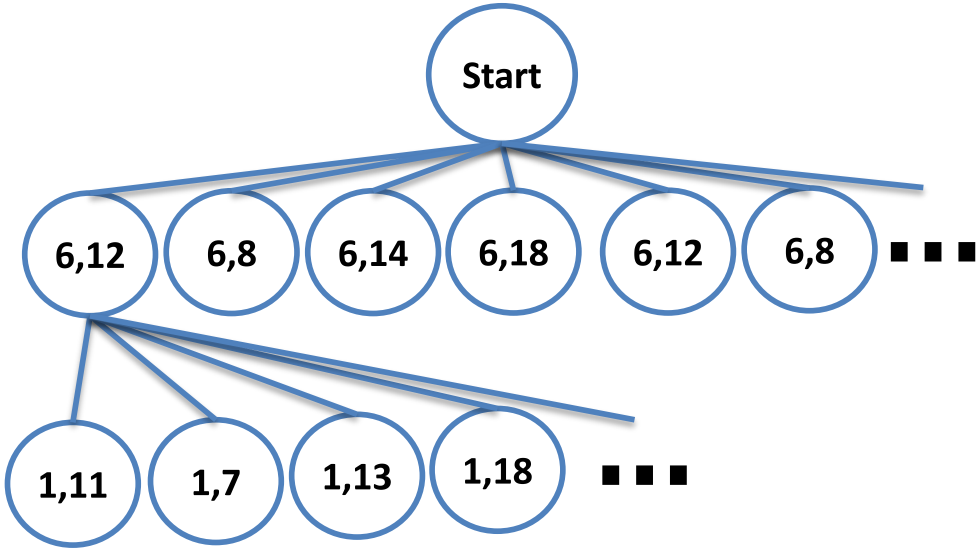


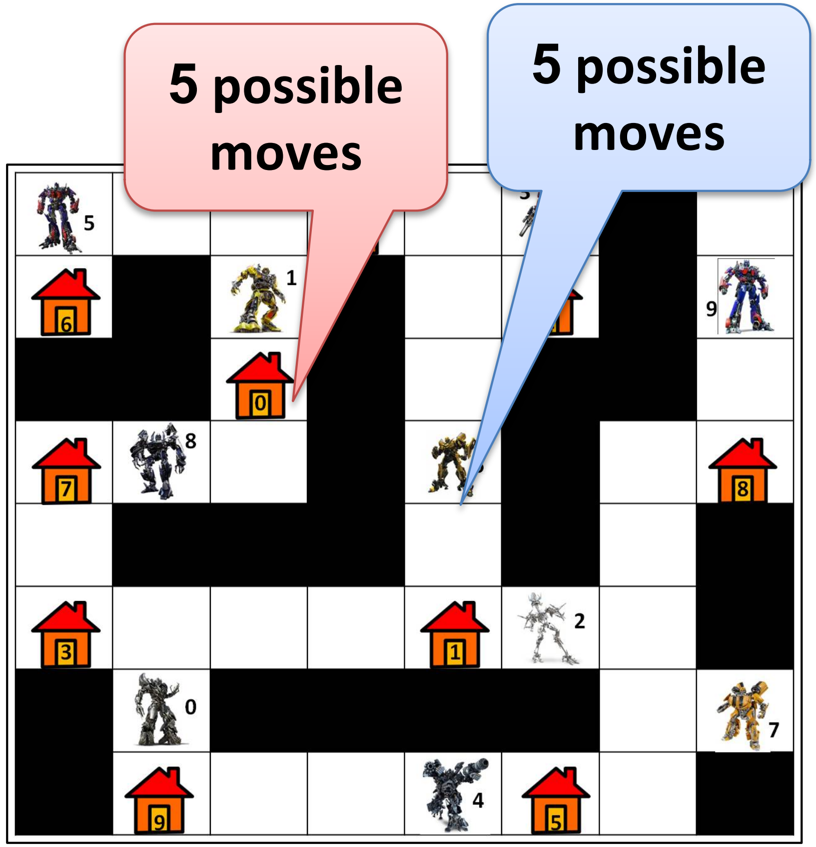
Search problem properties

- Number of states = ?
- Branching factor = ?



25 Possible moves! = 5 x 5



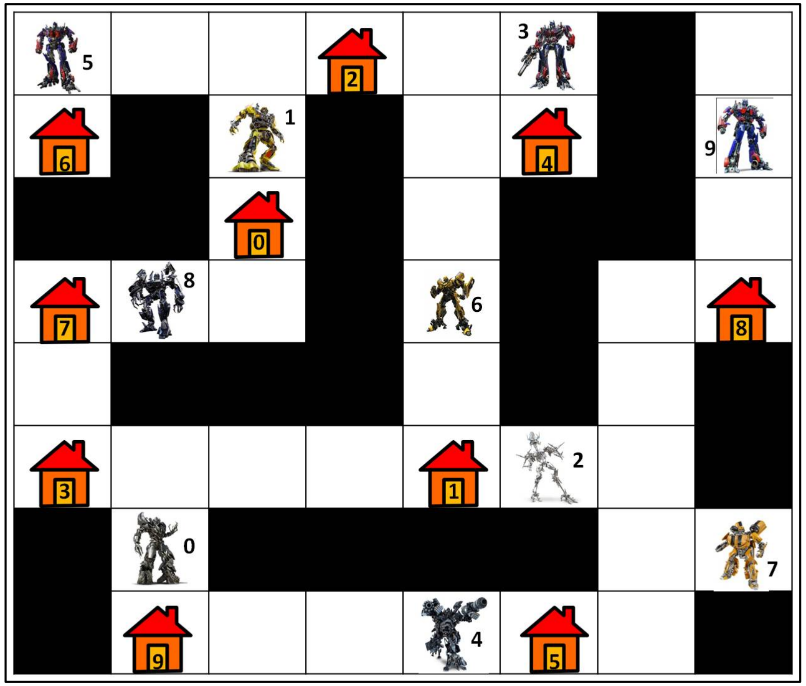


Search problem properties

- Number of states = 20^2
- Branching factor = 5^2

What about k agents?

5^k Possible moves!



Search problem properties

- Number of states = 20^k
- Branching factor = 5^k

Classical search problem?

A very hard search problem

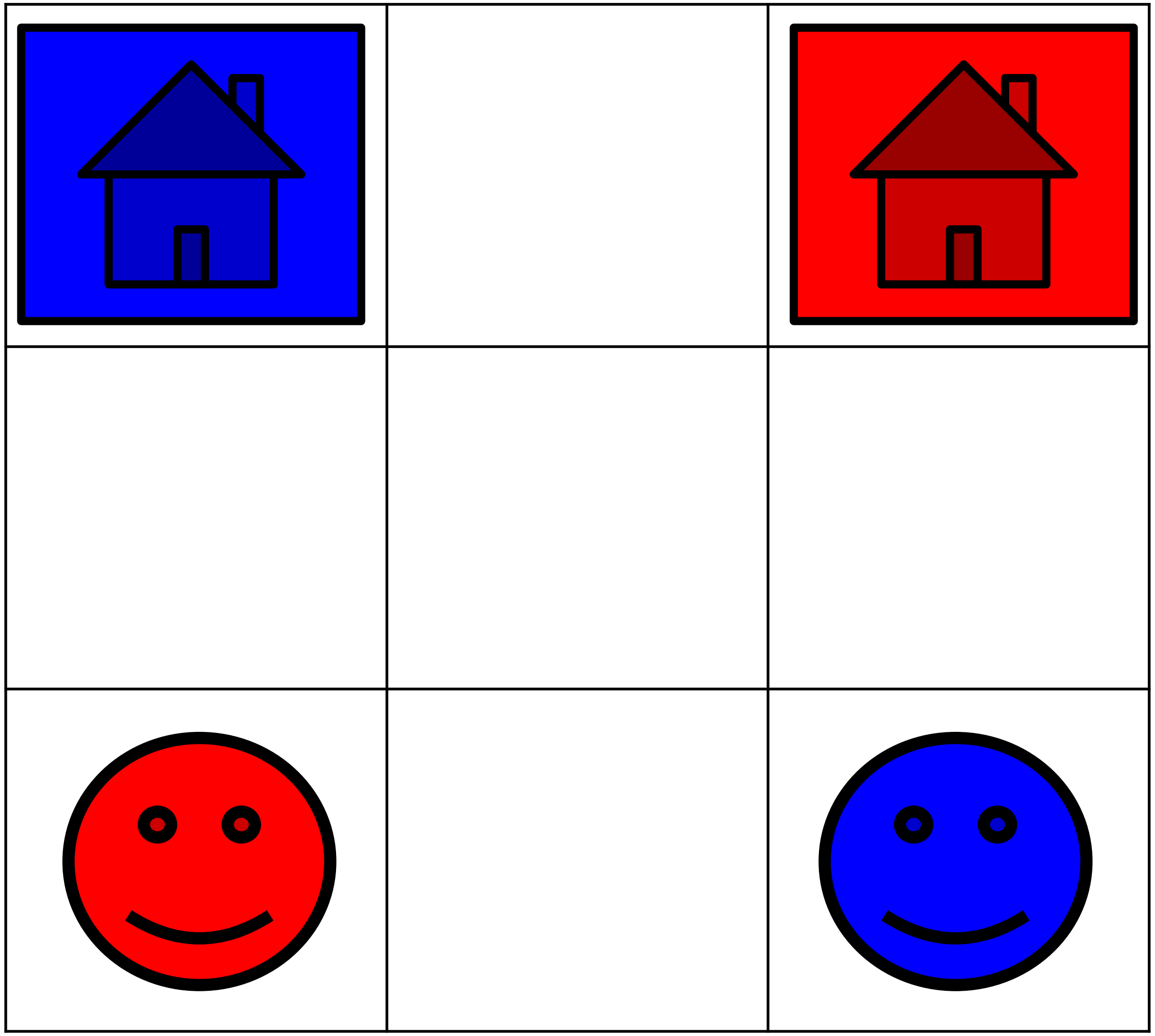
Key idea:

Plan for each agent **separately**

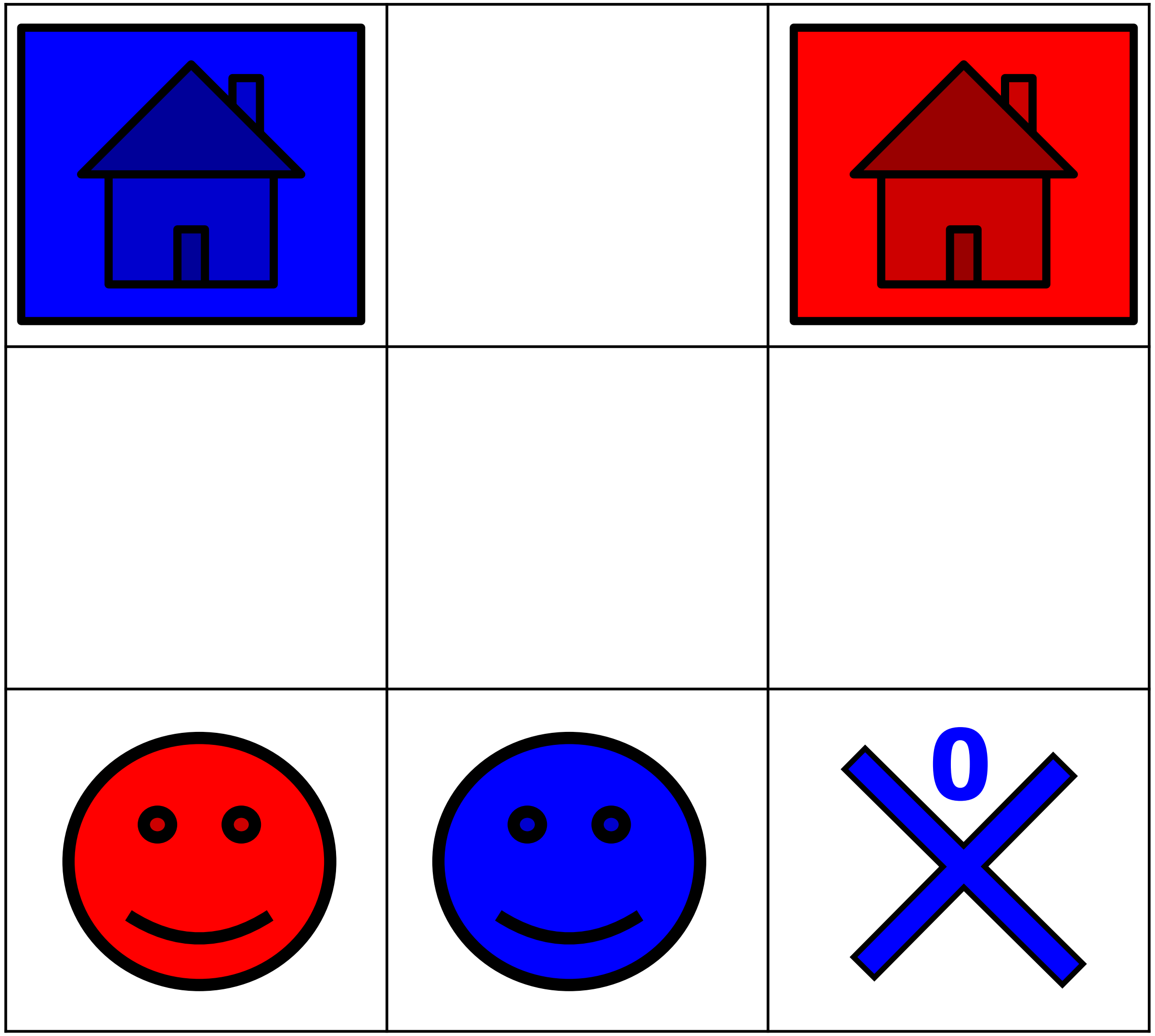
Challenge:

Maintaining **soundness**, **completeness**, and **optimality**

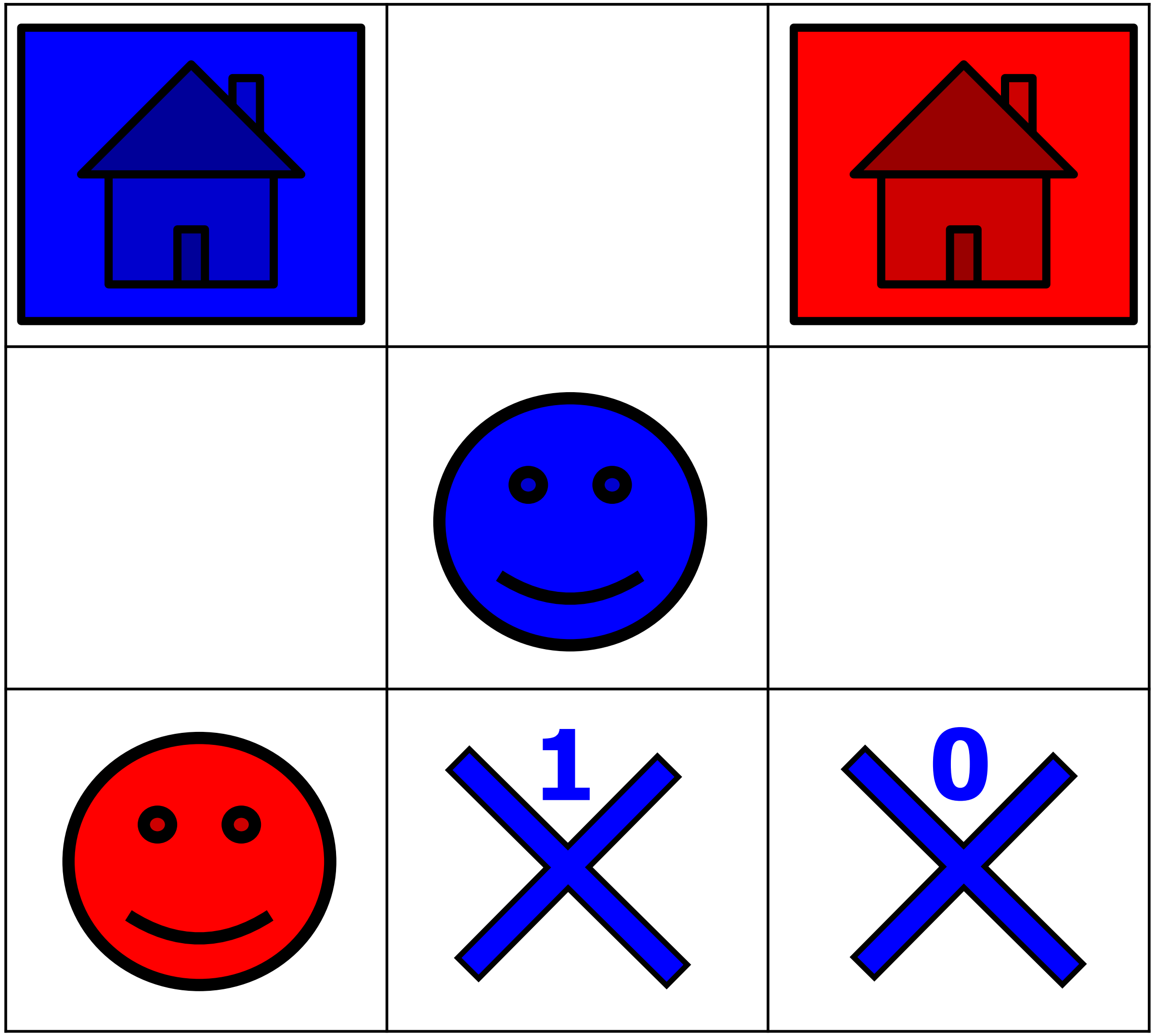
- Step 1: Plan **blue**



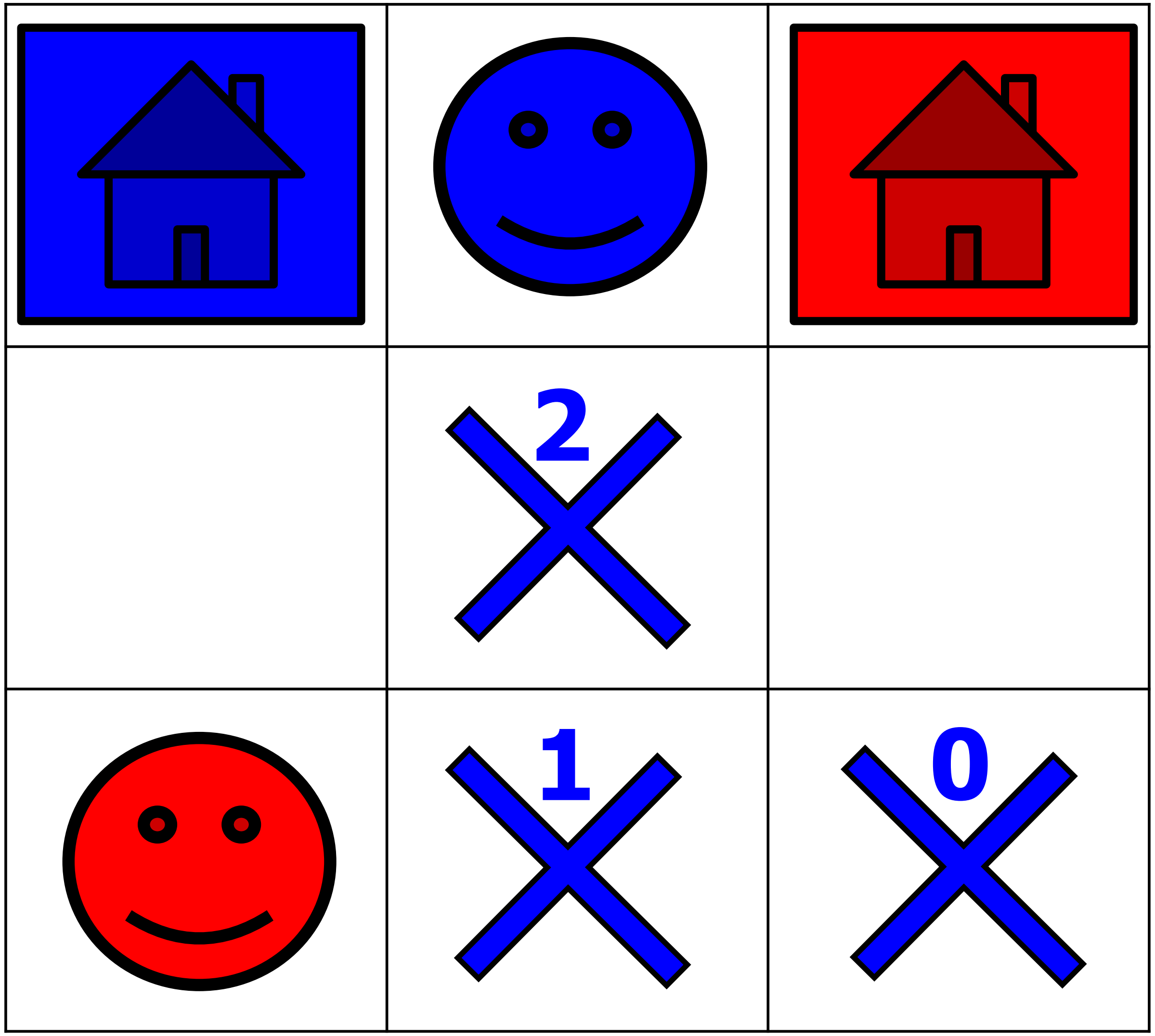
- Step 1: Plan **blue**



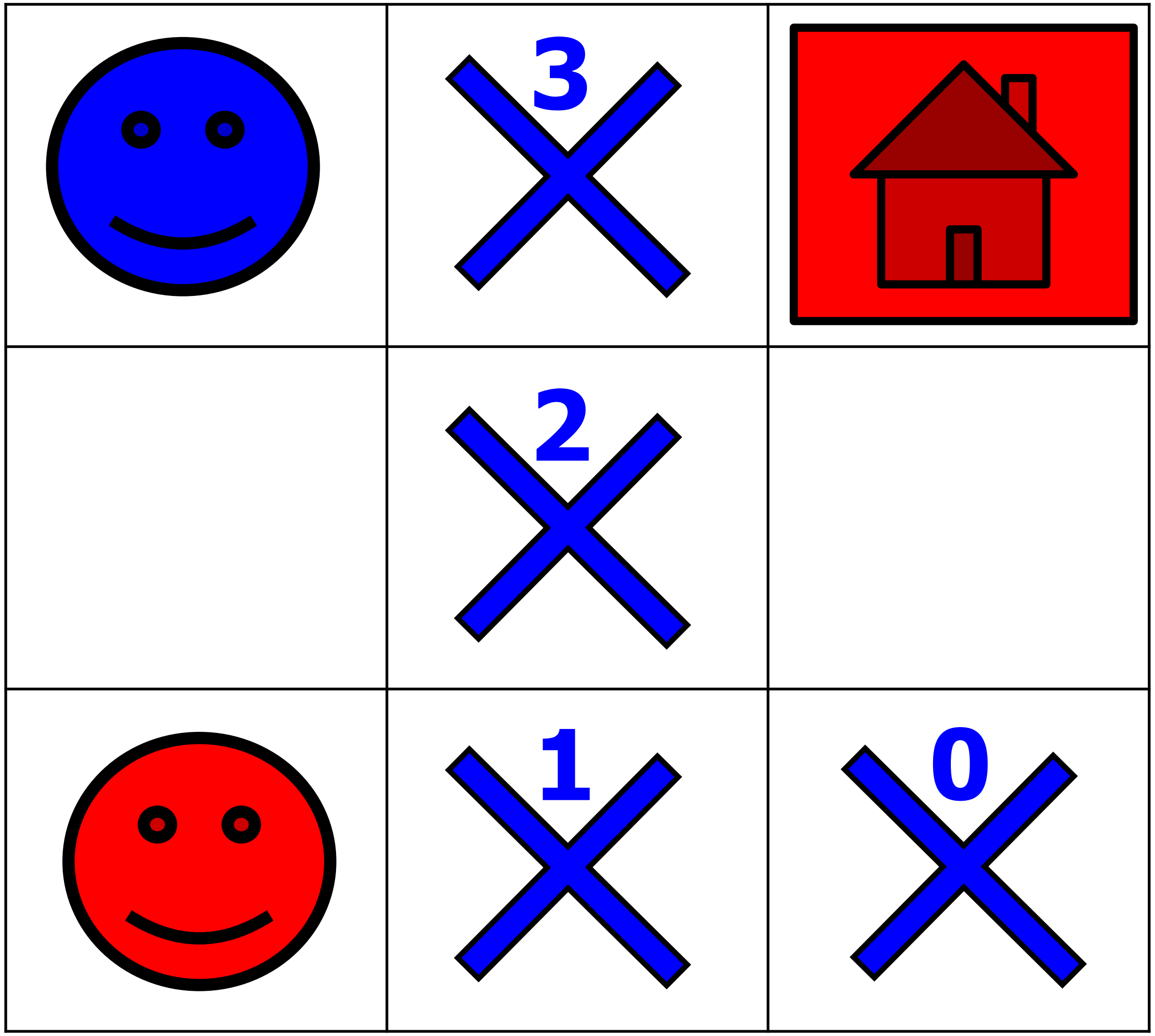
- Step 1: Plan **blue**



- Step 1: Plan **blue**

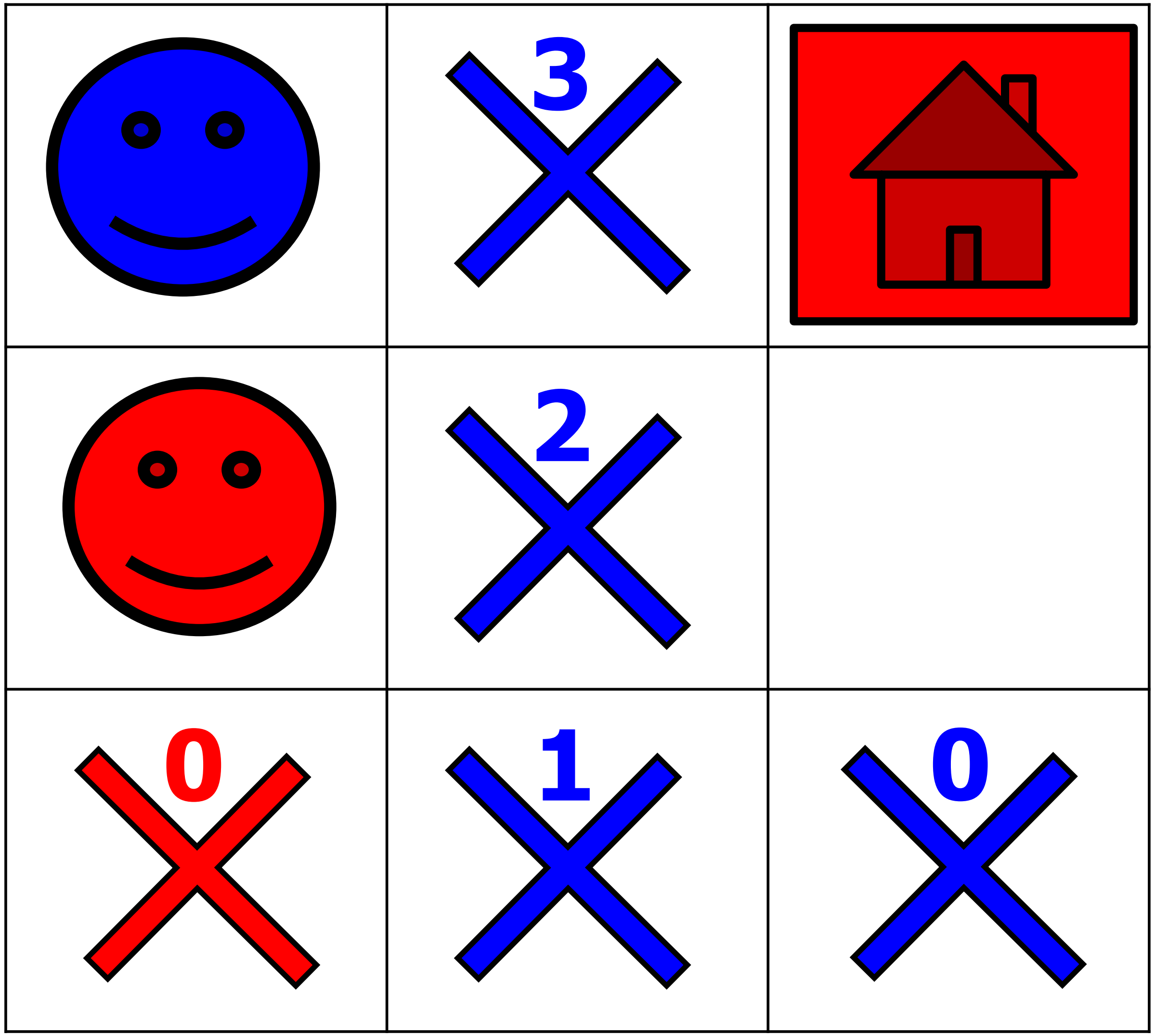


- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**

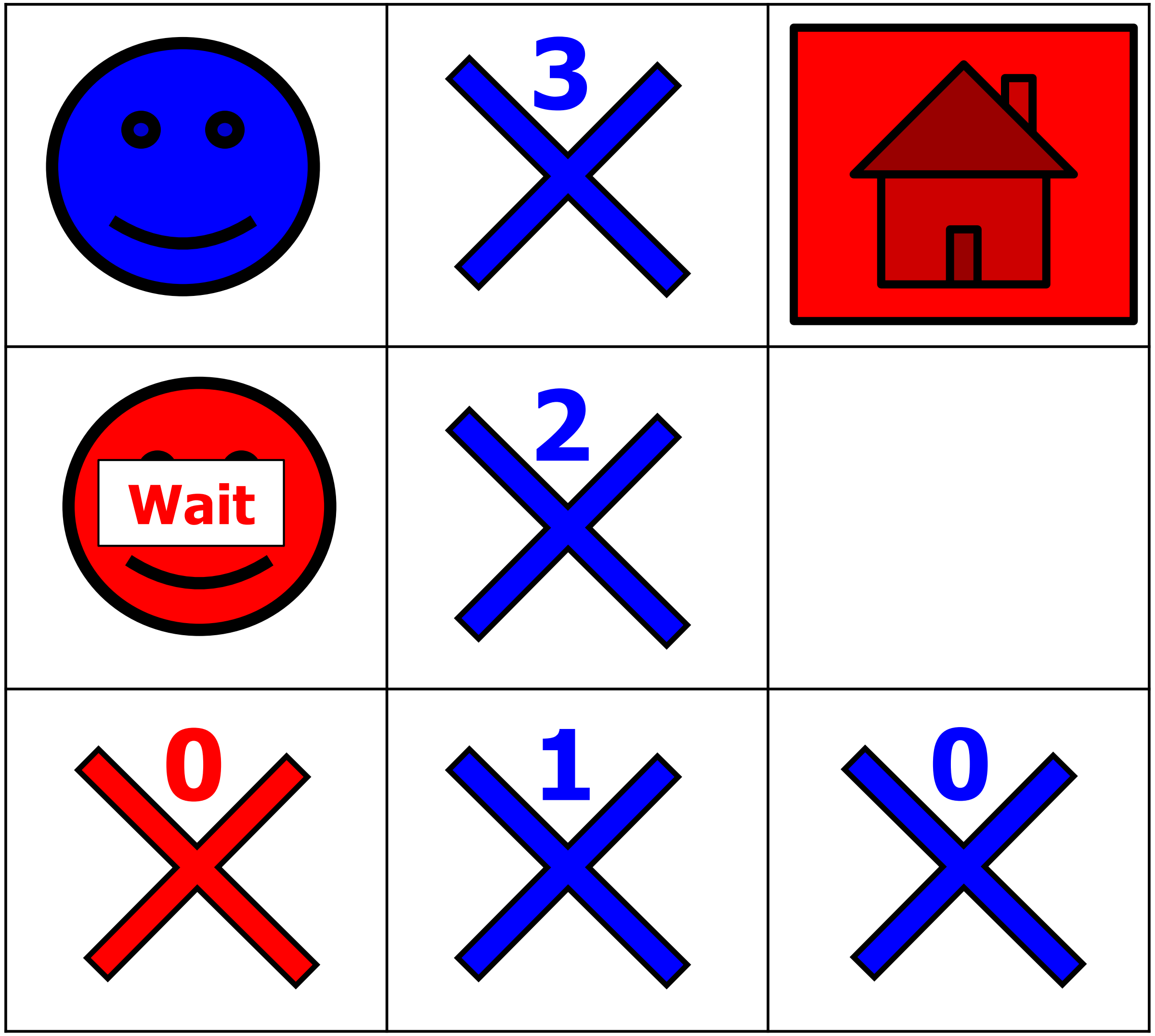


- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**

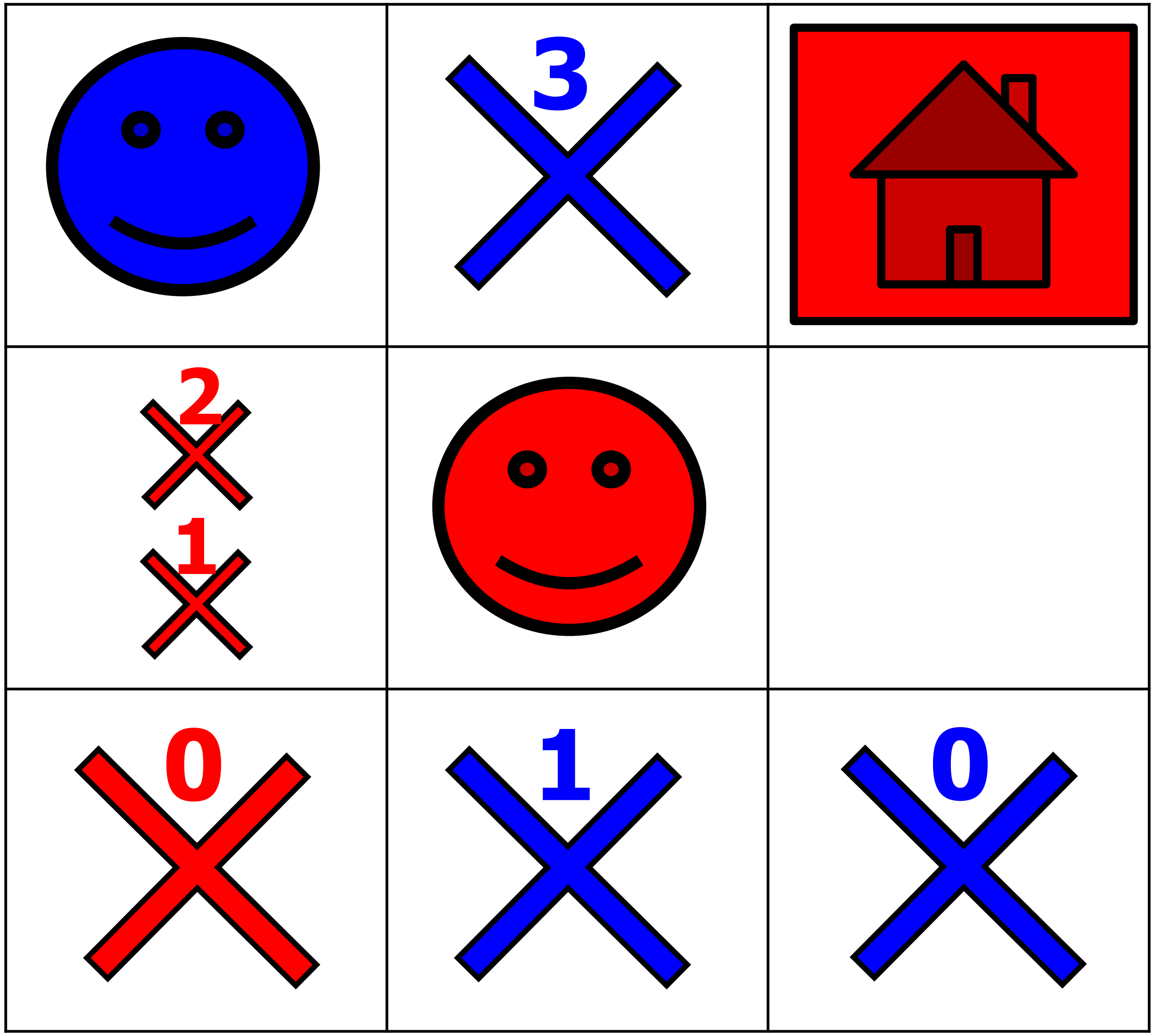
avoid blue's plan



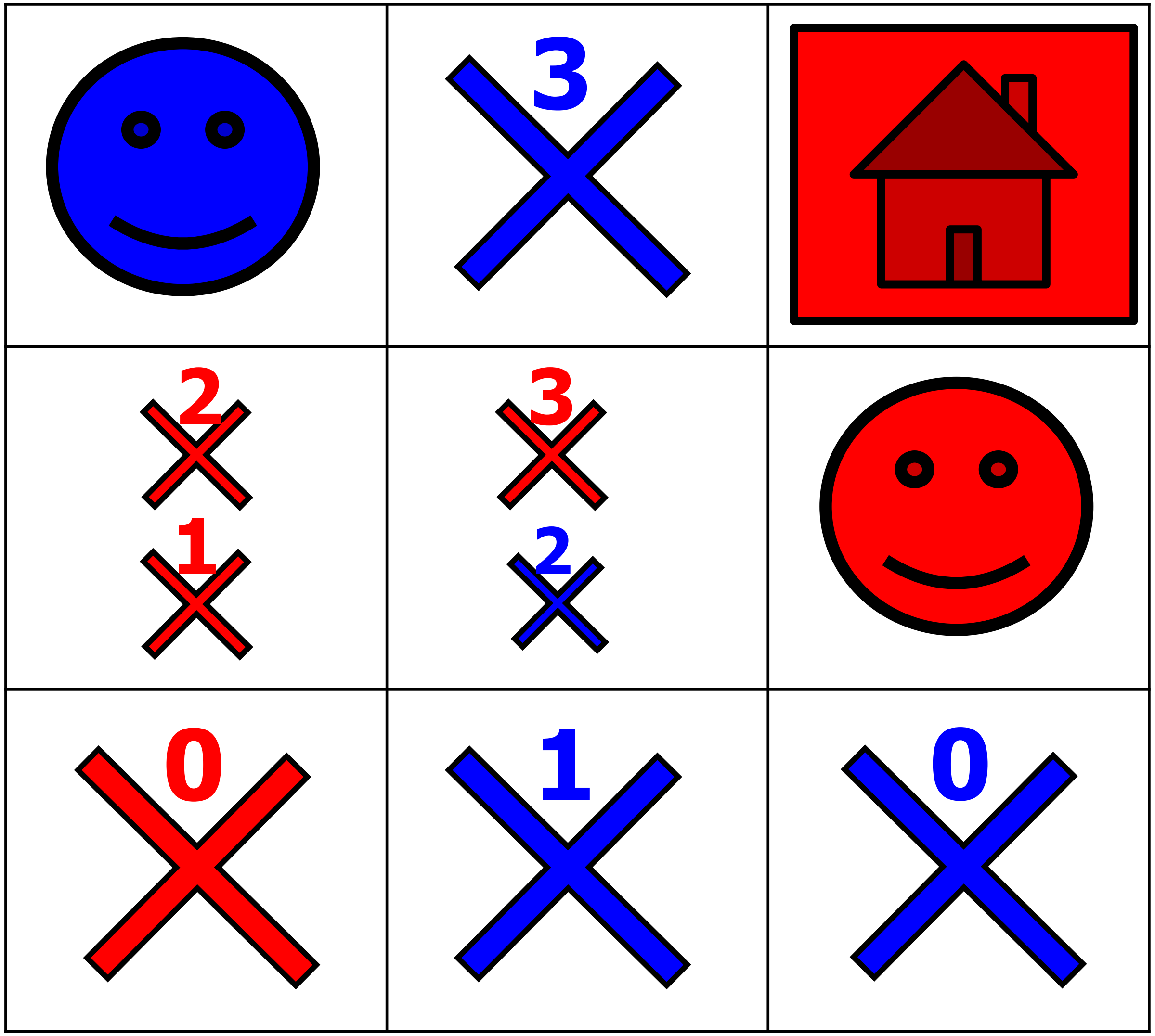
- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**



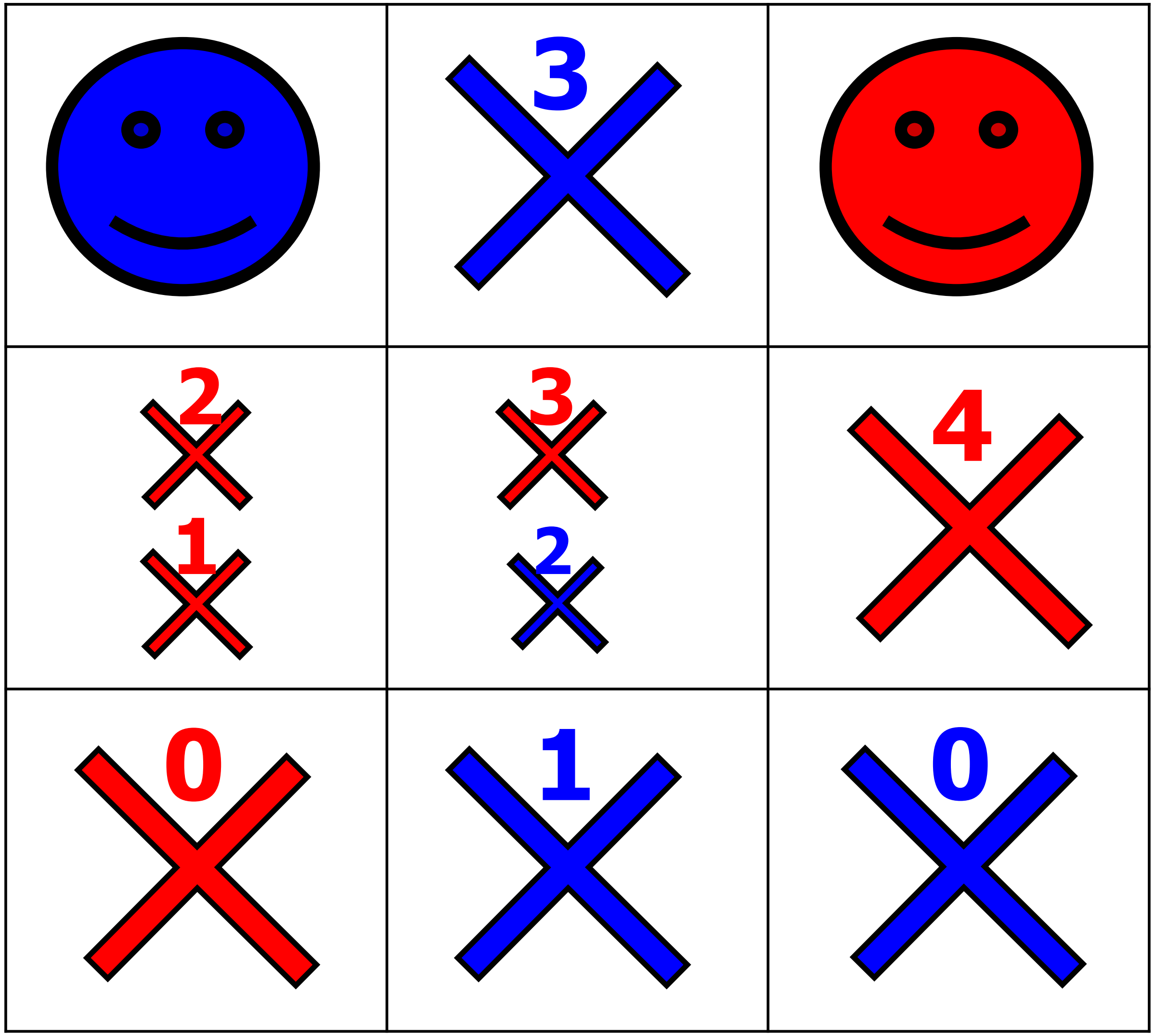
- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**



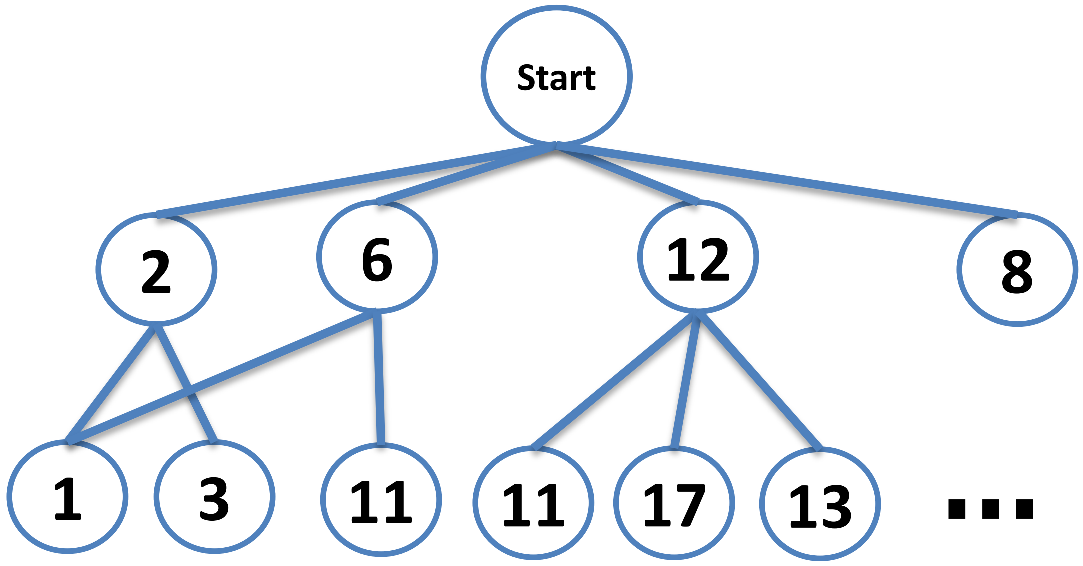
- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**

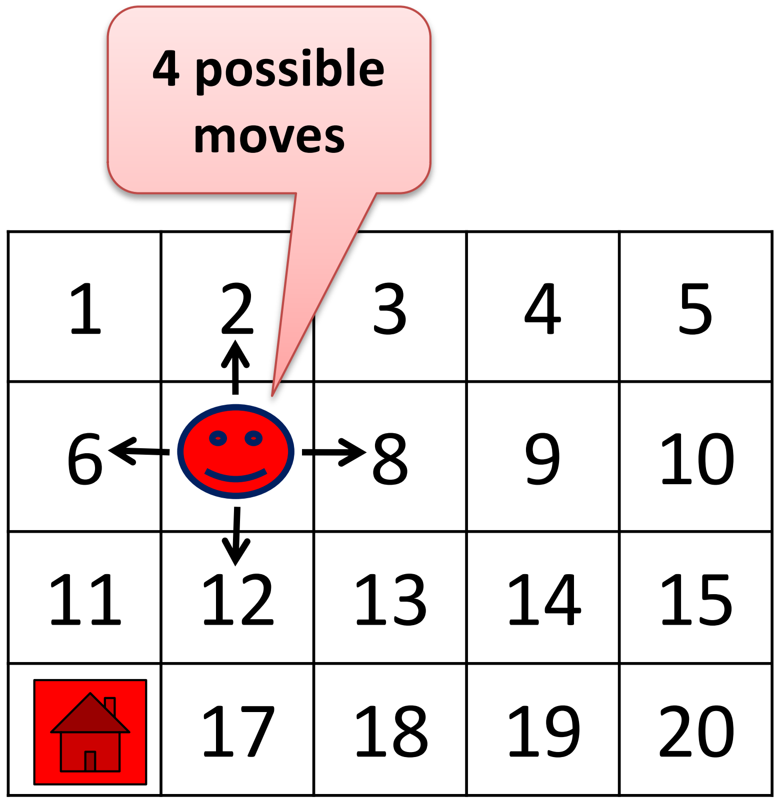


- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**
  - **Done!**
- …
- **Step N: Plan N^th^ agent**

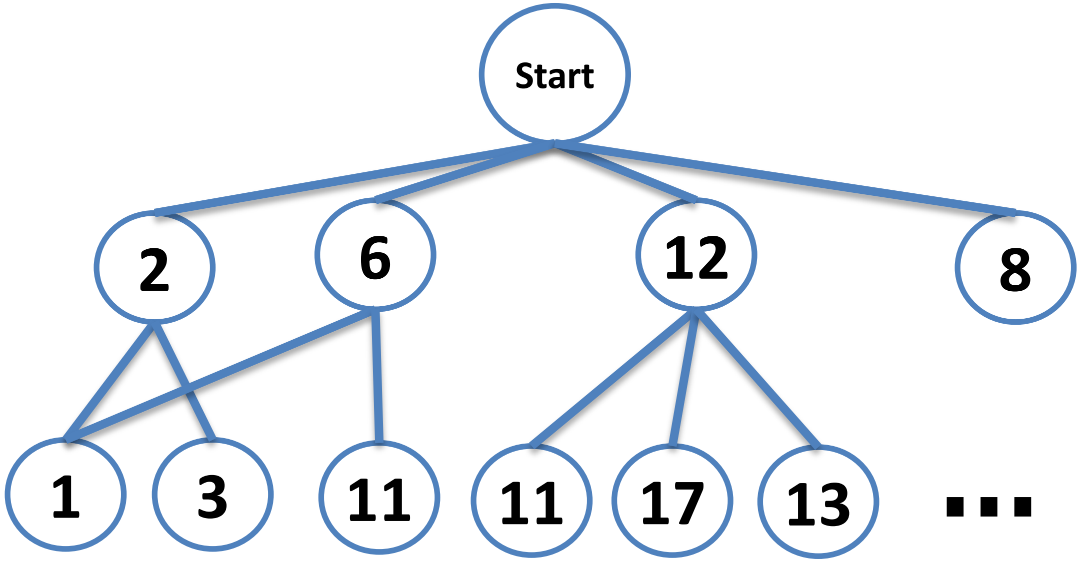


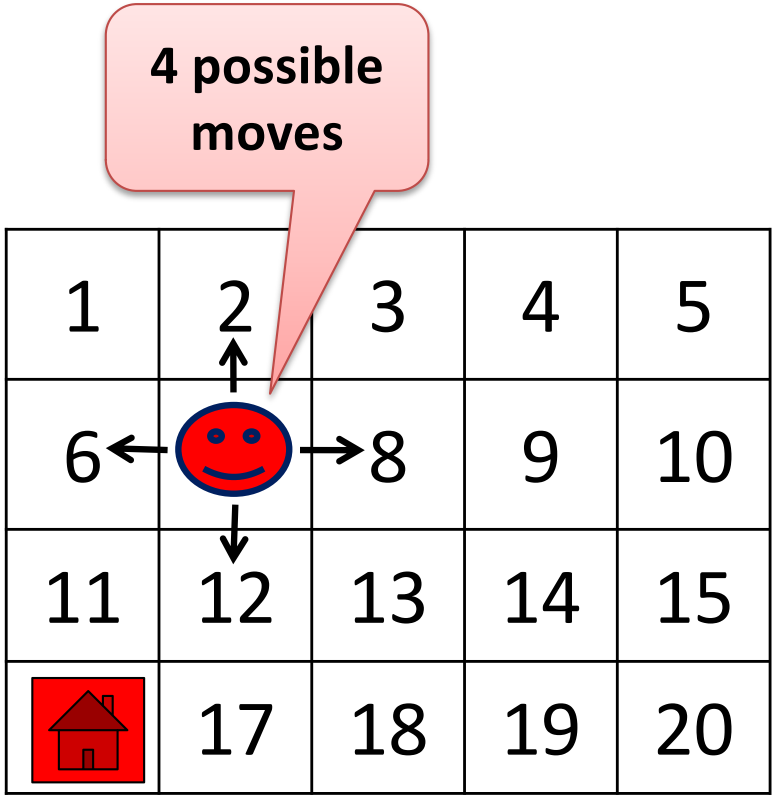
Prioritized Planning (Silver 2005) Analysis: First Agent





Prioritized Planning (Silver 2005) Analysis: First Agent

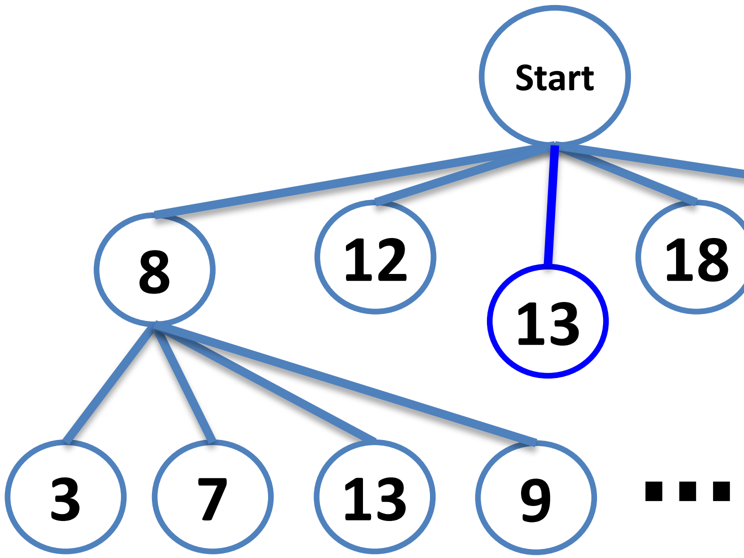


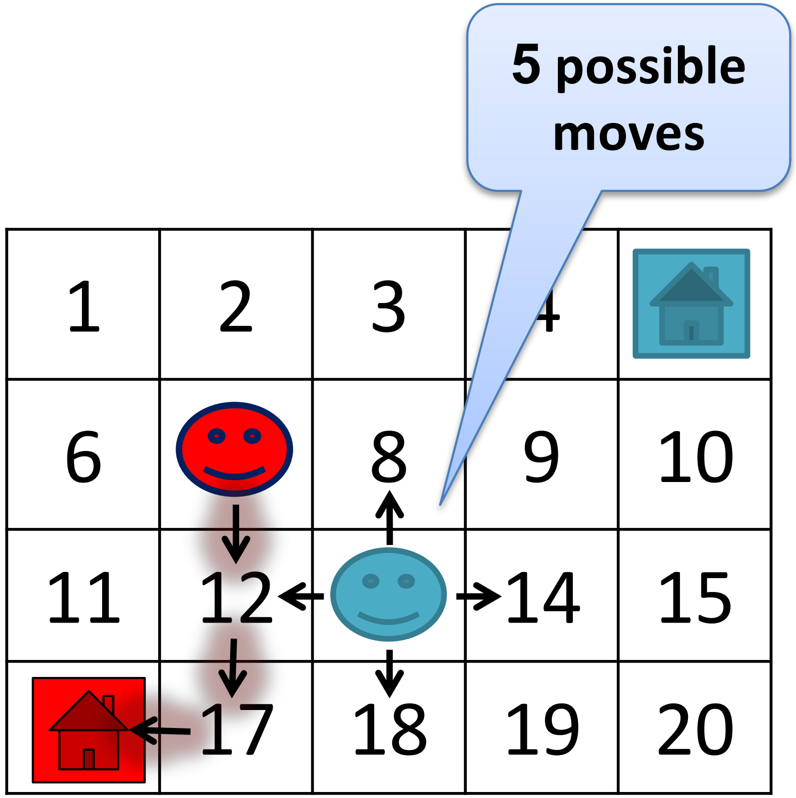


Singe-agent pathfinding

- A state is the agent's location
- Number of states = 4 x 5
- Branching factor = 4

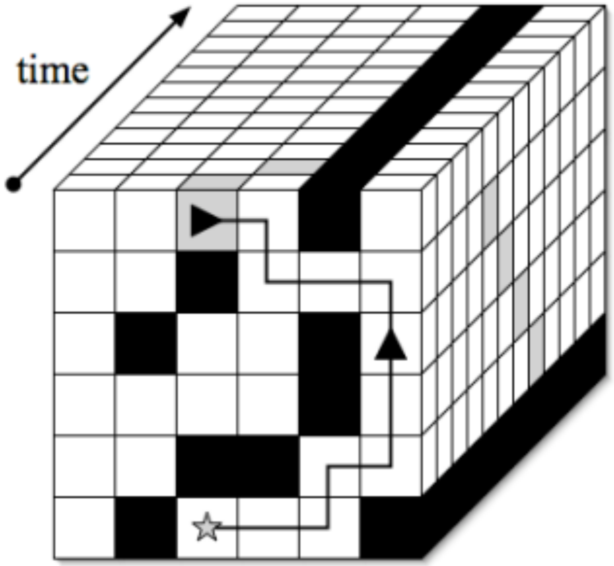
Prioritized Planning (Silver 2005) Analysis: Second Agent

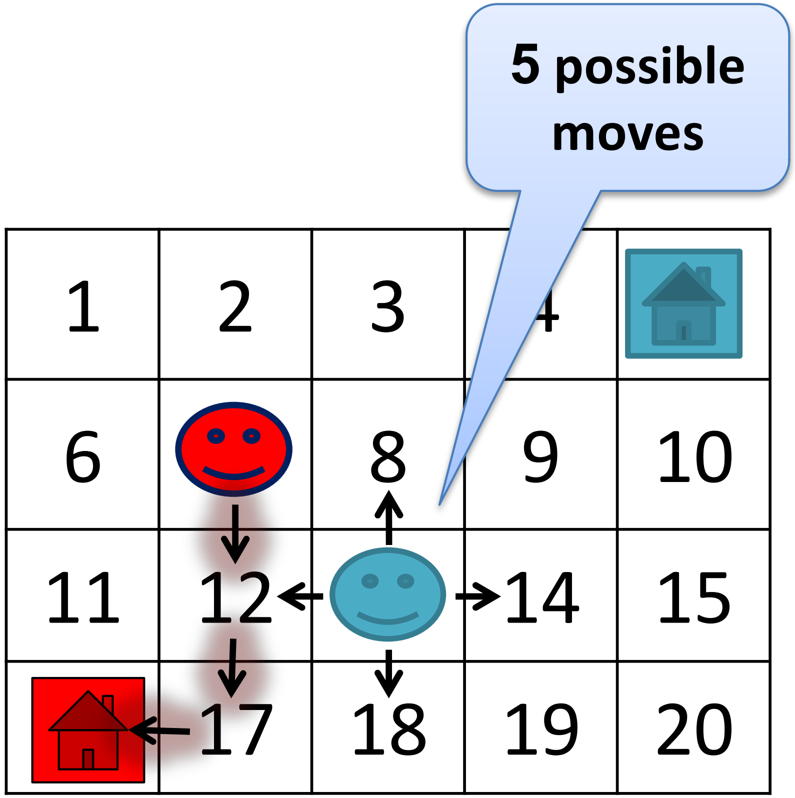




- A state is a (location,time) pair
- Number of states = 4 x 5 x maxTime
- Branching factor = 4+1

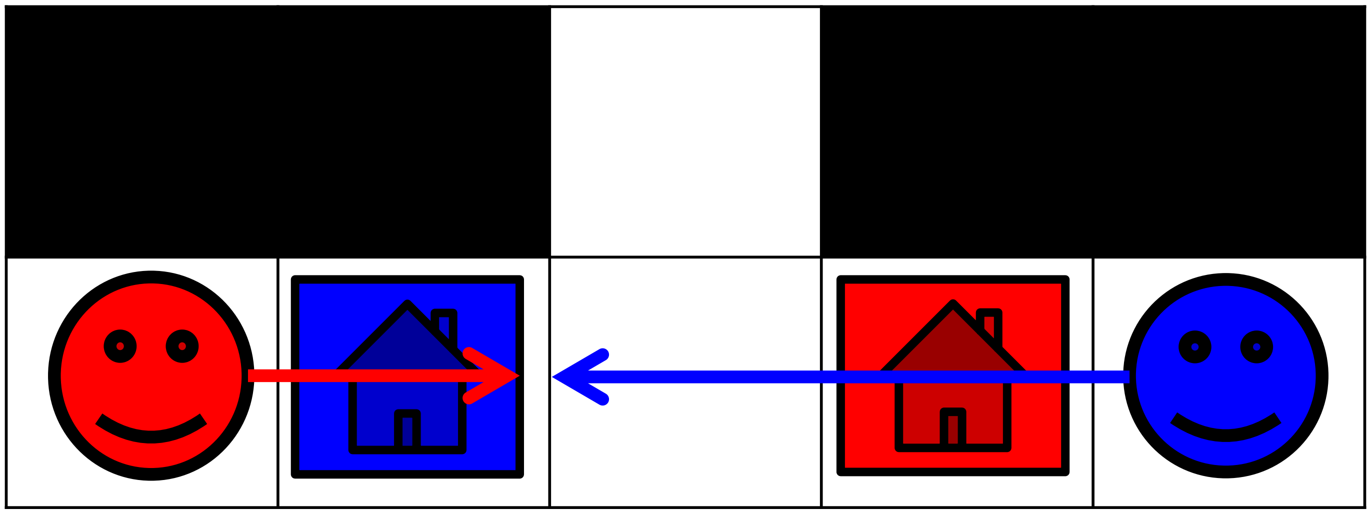
Prioritized Planning (Silver 2005) Analysis: Second Agent





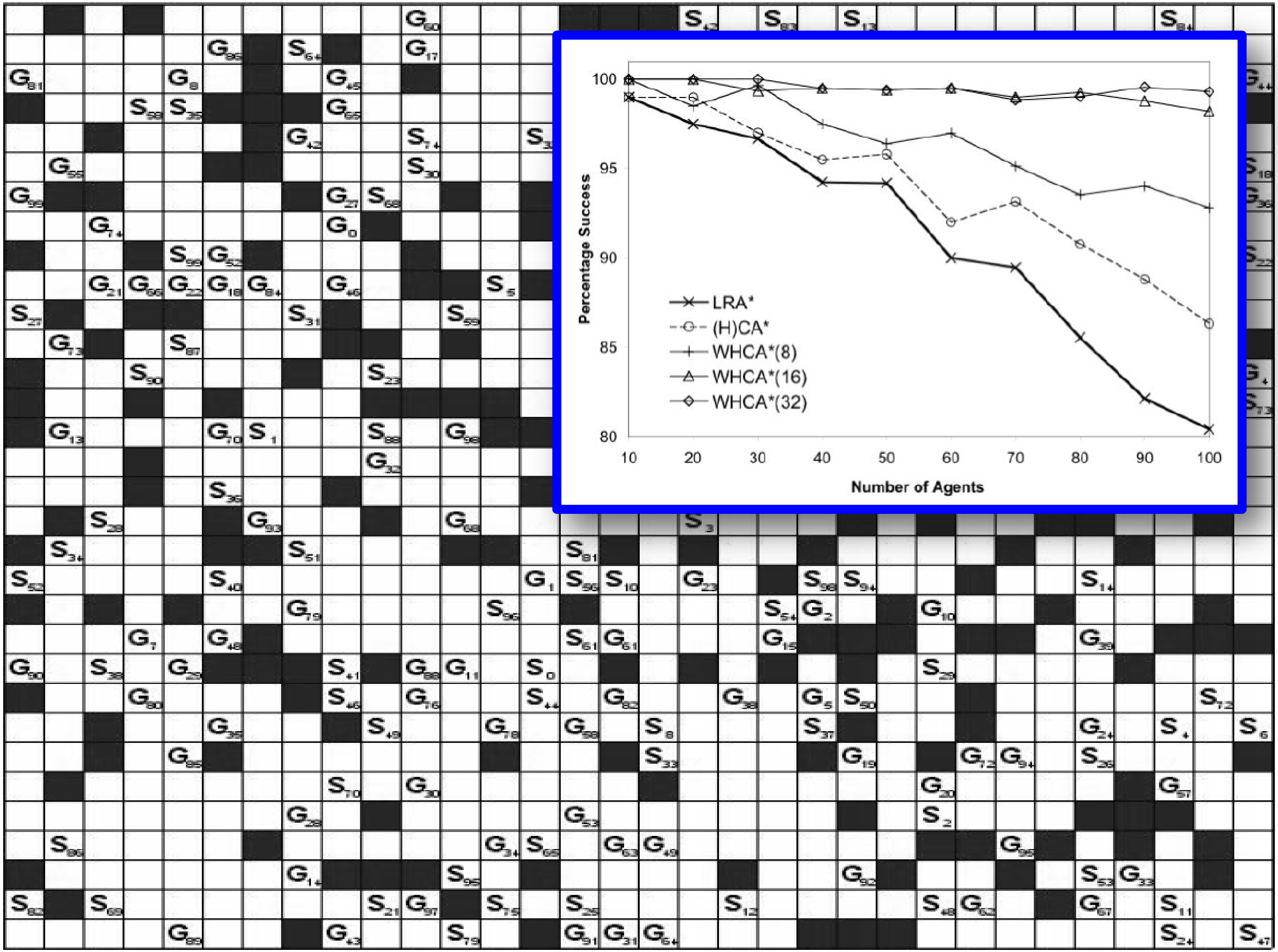
- A state is a (location,time) pair
- Number of states = 4 x 5 x maxTime
- Branching factor = 4+1

- Complexity?
  - **Polynomial** in the grid size and max time
- Soundness?
  - **Yes!**
- Complete? Optimal?
  - **No ☹**

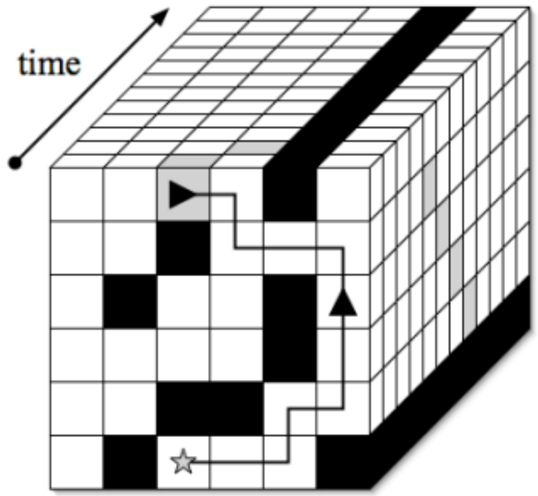


- Smart agent **prioritization**
  - Conflict oriented WHCA* [Bnaya and Felner '14]
  - Re-prioritization and safe intervals [Andreychuk and Yakovlev '18]
- Integrate **planning and execution**
  - Windowed Hierarchical CA* [Silver '06]

Prioritized Planning (Silver 2005) - Results



High-level idea: reservation-based planning

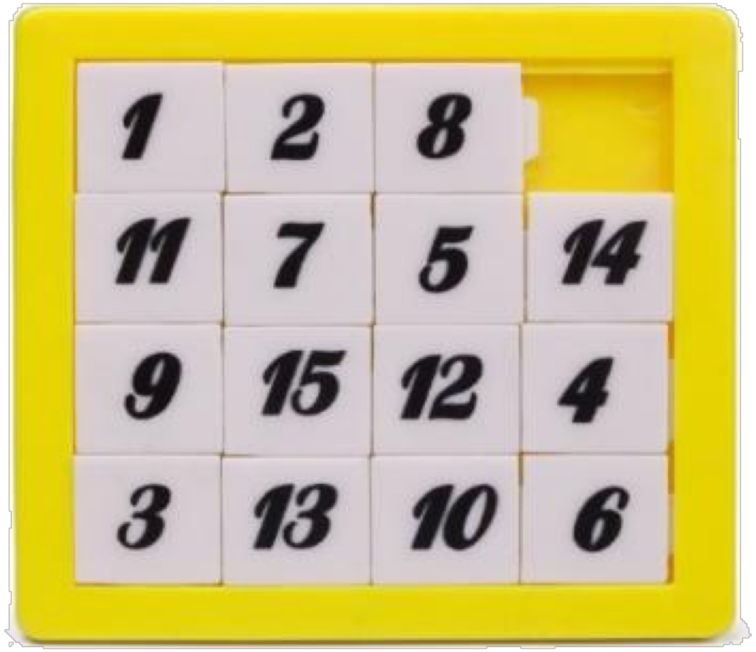


+ **Fast**, requires almost no coordination
- But **incomplete** and **not optimal**

	Suboptimal	Optimal
Incomplete	• Cooperative A* • WHCA*	?
Complete	?	?

Can a MAPF algorithm be

complete and **efficient**?



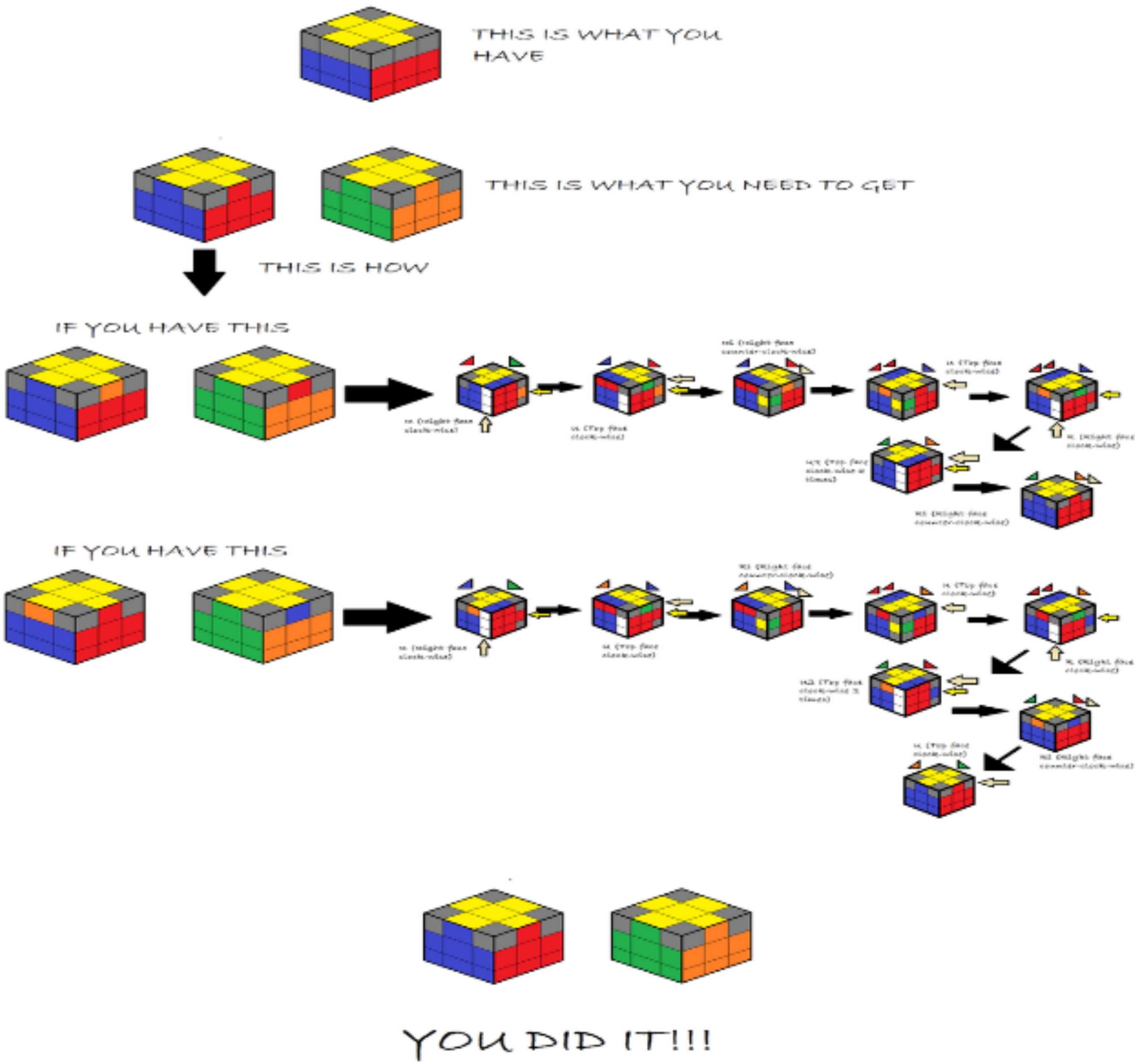
Can a MAPF algorithm be **complete** and **efficient**?

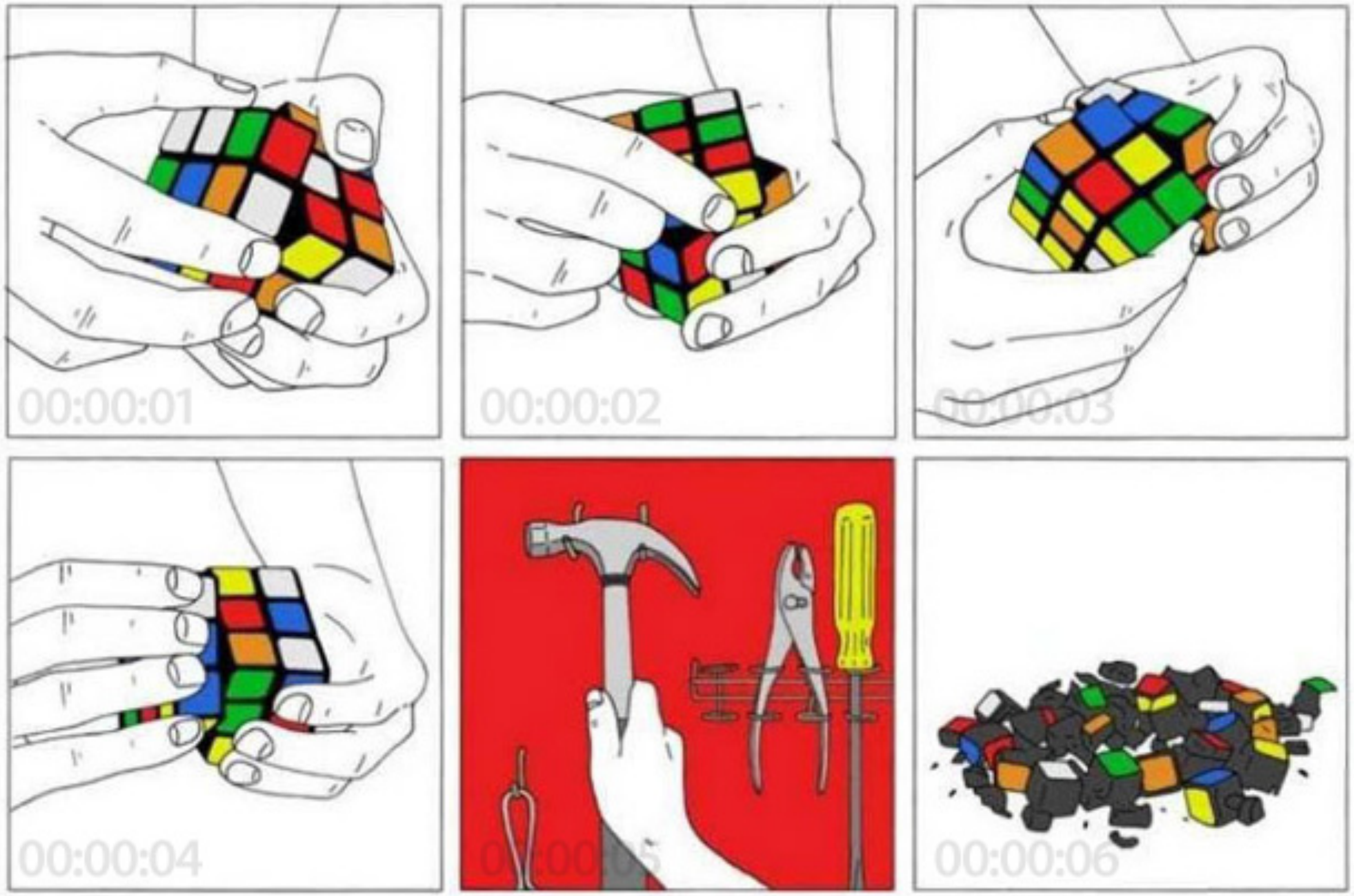
- MAPF is highly related to ***pebble motion problems***
  - Each agent is a pebble
  - Need to move each pebble to its goal
  - Cannot put two pebbles in one hole
- **Pebble motion can be solved polynomially!**
  - **But far from optimally**
  - **Complex formulation**

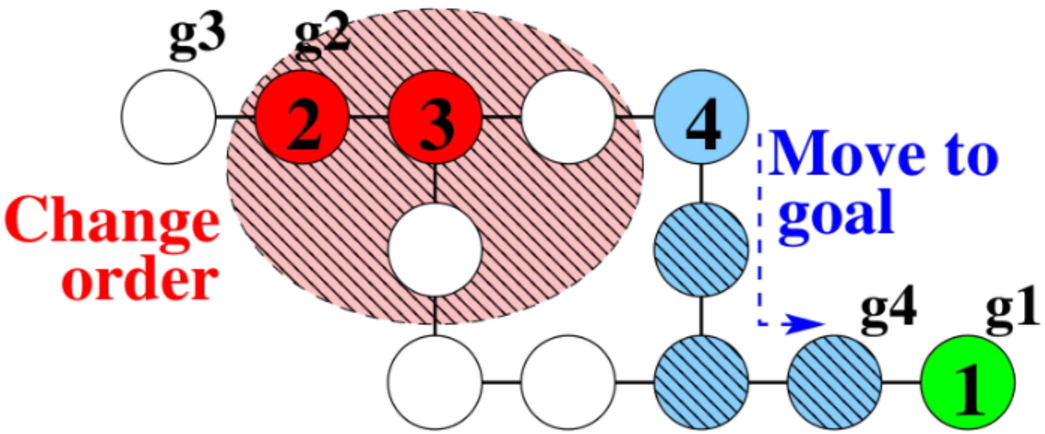
[**Kornhauser et al.**, FOCS 1984]

Similar approaches:

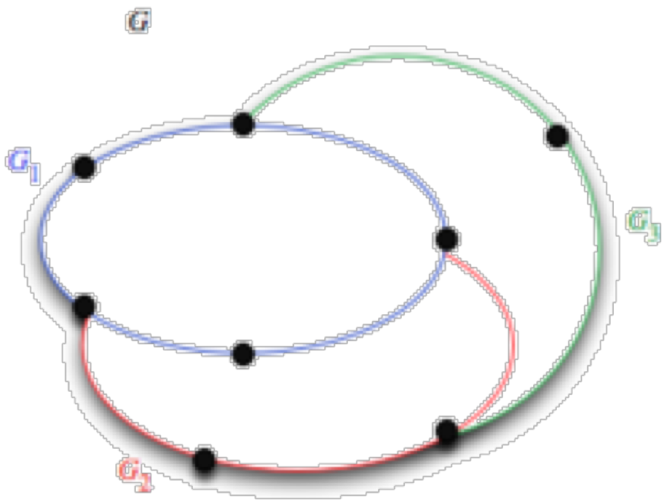
- Slidable Multi-Agent Path Planning [Wang & Botea, IJCAI, 2009]
- Push and Swap [Luna & Bekris, IJCAI, 2011]
  - Parallel push and swap [Sajid, Luna, and Bekris, SoCS 2012]
  - Push and Rotate [de Wilde et al. AAMAS 2013]
- Tree-based agent swapping strategy [Khorshid at el. SOCS, 2011]


THIS IS WHAT YOU HAVE
THIS IS WHAT YOU NEED TO GET
THIS IS HOW
IF YOU HAVE THIS
IF YOU HAVE THIS
YOU DID IT!!!

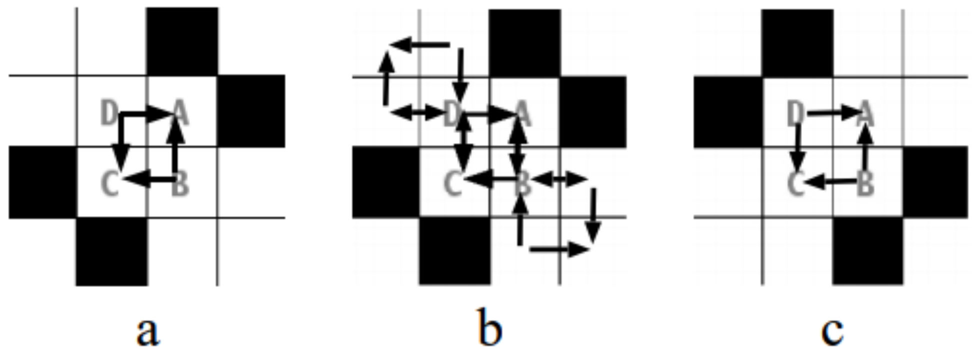

00:00:01
00:00:02
00:00:04
00:00:06



Push and Swap (Luna and Bekris '13)



Bibox (Surynek '09)

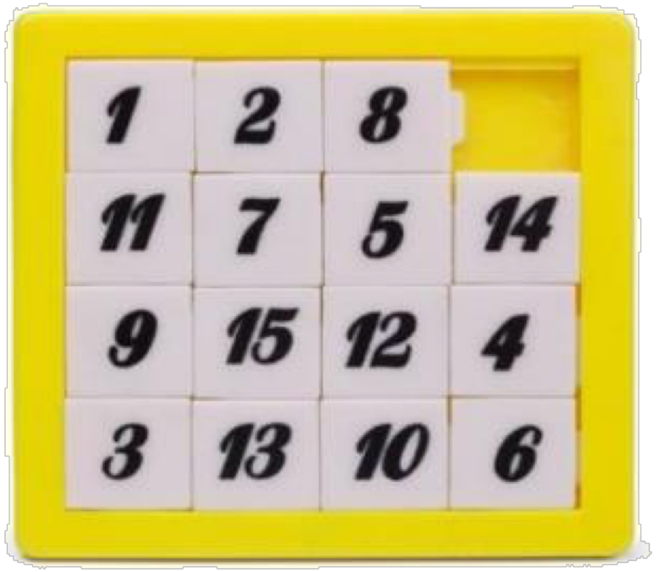


FAR (Wang and Botea '08)

	Suboptimal	**Optimal**
Incomplete	• Cooperative A* • WHCA*	?
Complete	• Kornhauser et al. '84 • Push & Swap (Luna & Bekris) • Bibox (Surynek) …	?

Can a MAPF algorithm be

complete and **efficient** and **optimal**?

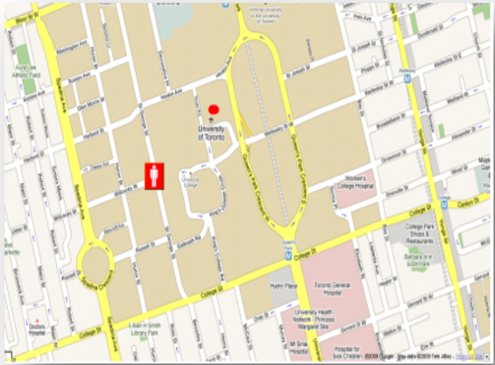


NP-hard

(Surynek '15, '10)
(Yu and LaValle '13)
(Ratner & Warmuth, '86)

On the Complexity of Optimal Parallel Cooperative Path-Finding, Surynek 2015
Planning Optimal Paths for Multiple Robots on Graphs, Yu and LaValle, 2013

K=1 (Navigation in explicit graphs)
Explicit graph



K (# agents)

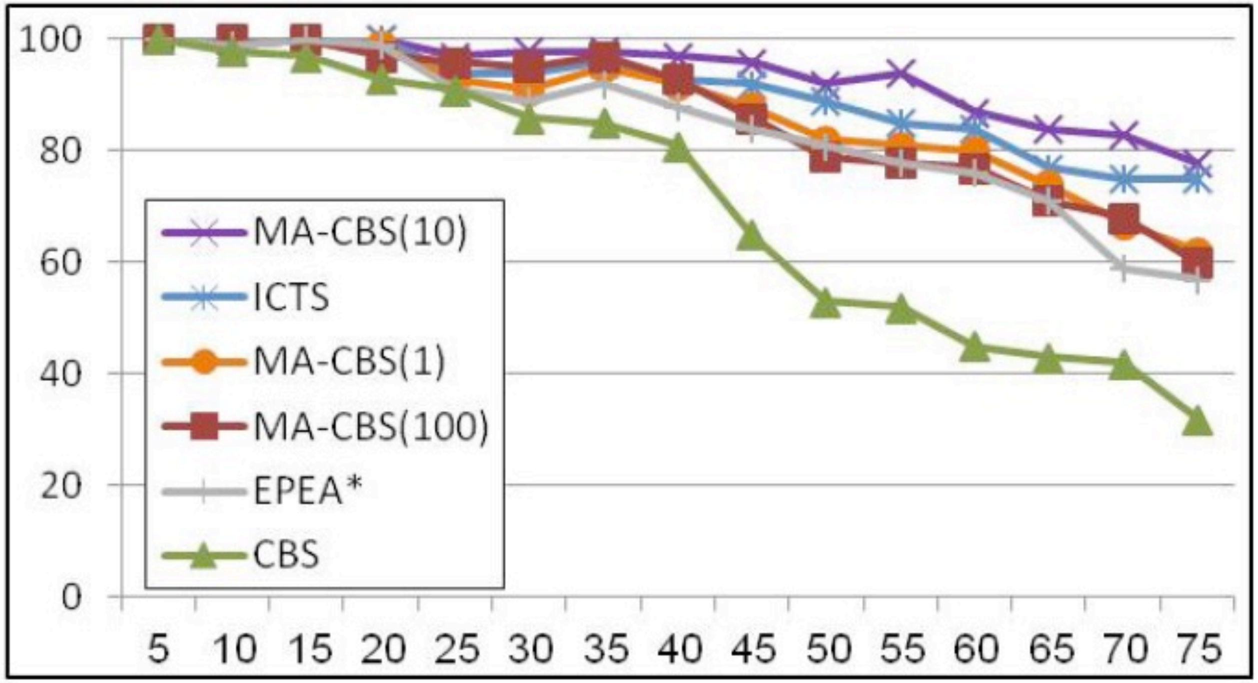
K=N-1 (Tile puzzle)
(Huge) Implicit graph

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Can we adapt techniques from these extreme cases?

Yes!

(and invent some new techniques also)

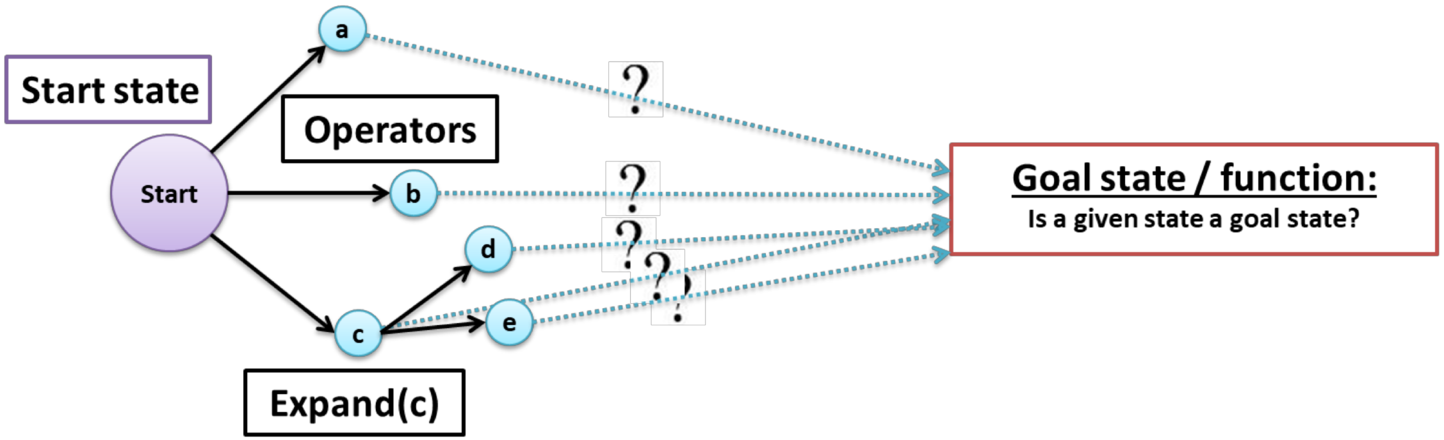


Searching the k-agent search space

- A*+OD+ID [Standley '10]
- EPEA* [Felner 'X, Goldenberg 'Y]
- M* [Wagner & Choset 'Z]

Other search-based approaches

- ICTS [Sharon et al '13]
- CBS [Sharon et al '15]

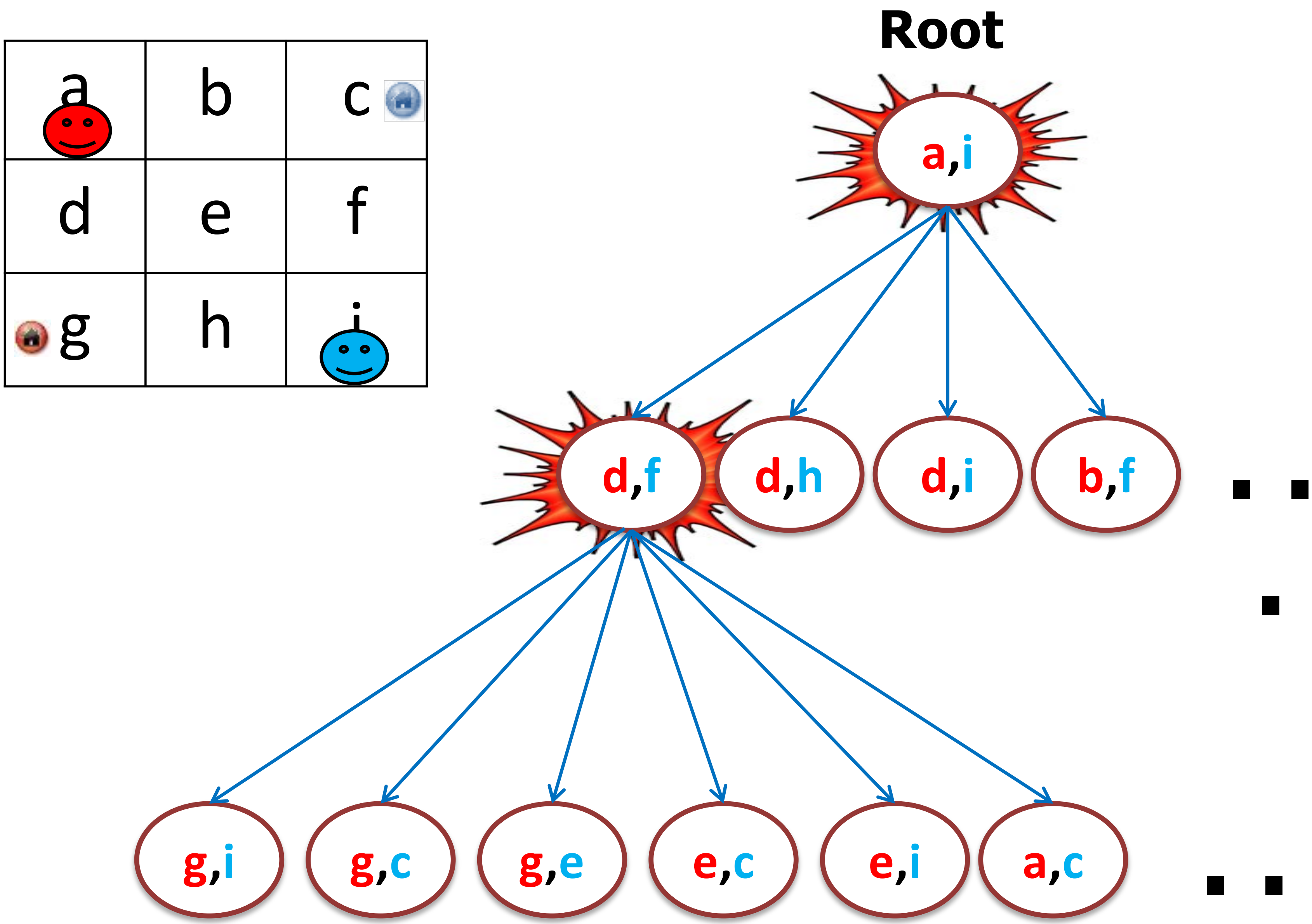


- A* **expands nodes**
- A* gain efficiency by **choosing which node to expand**

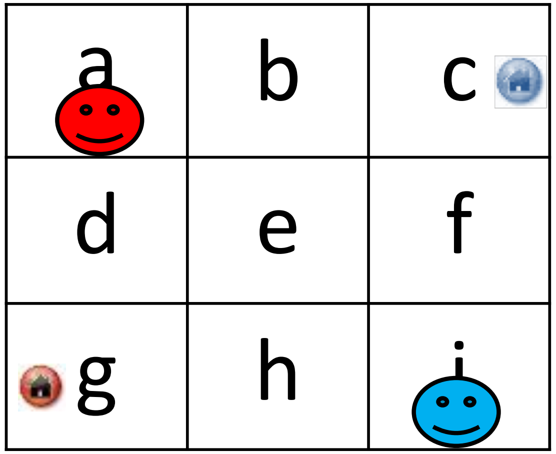
What is the complexity of expanding

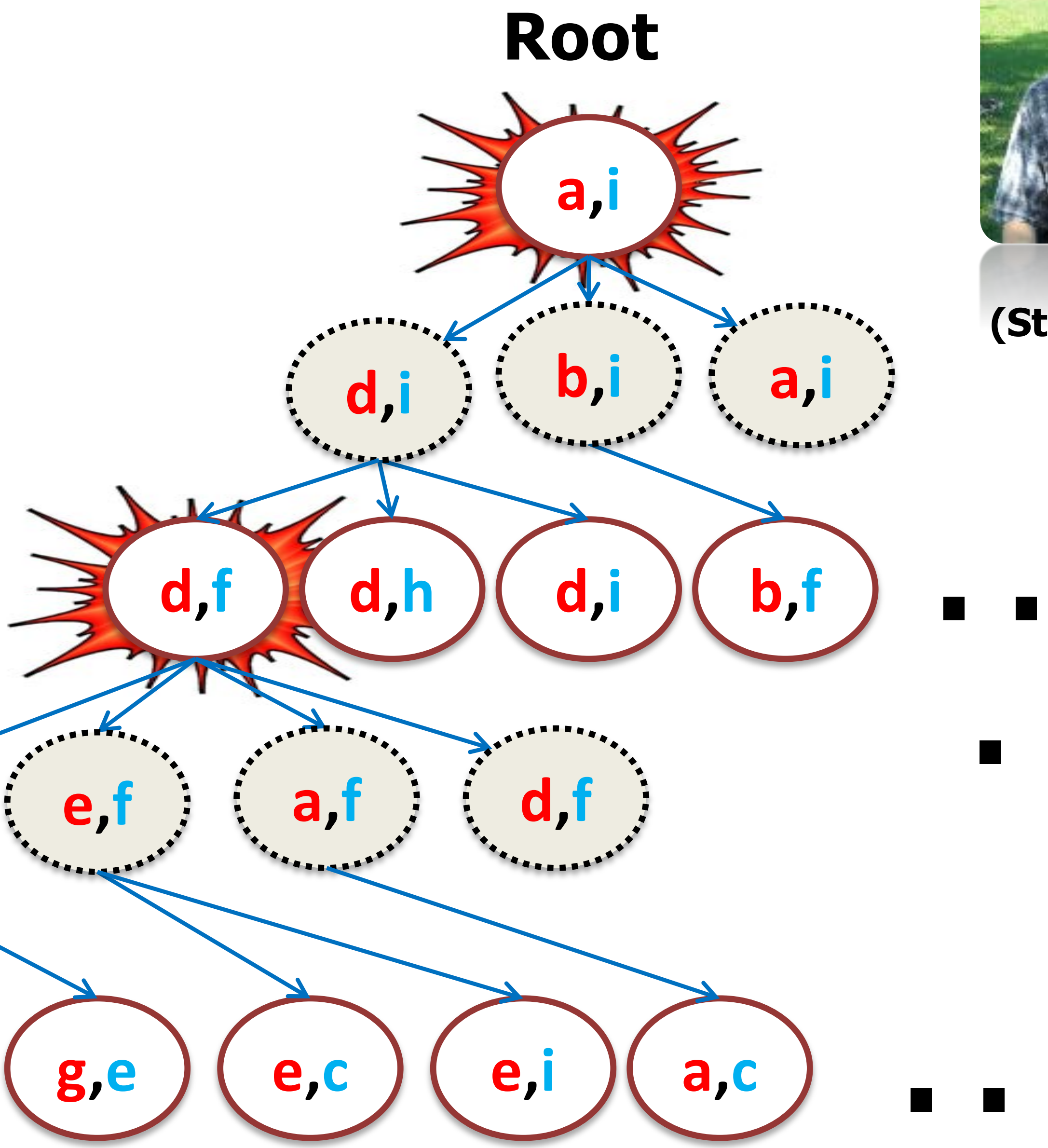
a single node *in MAPF with 20 agents?*

5^{20}= 95,367,431,640,625


a
b
c
d
e
f
g
h
i
Root
a,i
d,f
d,h
d,i
b,f
g,i
g,c
g,e
e,c
e,i
a,c

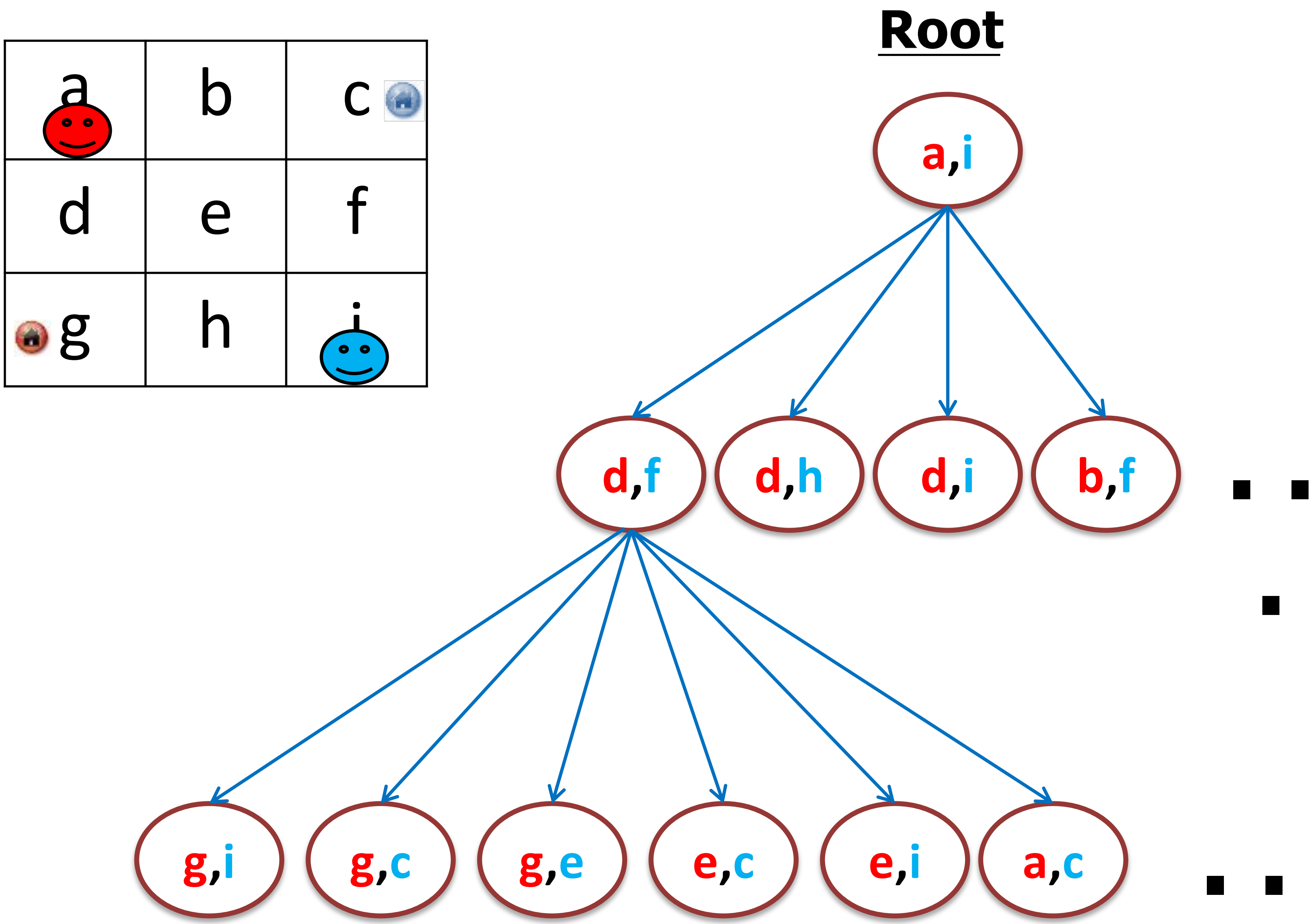
Search Tree Growth with **Operator Decomposition**



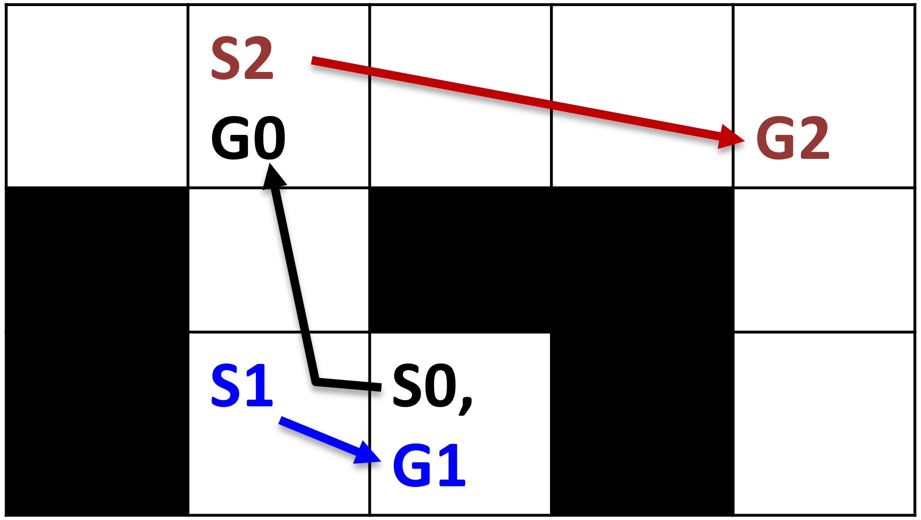


(Standley '10)

- Pros
  - **Branching factor is reduced** to 5 (= single agent)
  - With a perfect heuristic can solve the problem
- Cons
  - **Solution is deeper** by a factor of k
  - **More nodes may be expanded**, due to intermediates


a
b
c
d
e
f
g
h
i
Root
a,i
d,f
d,h
d,i
b,f
g,i
g,c
g,e
e,c
e,i
a,c

Theoretically, a 3 agents problem, but …

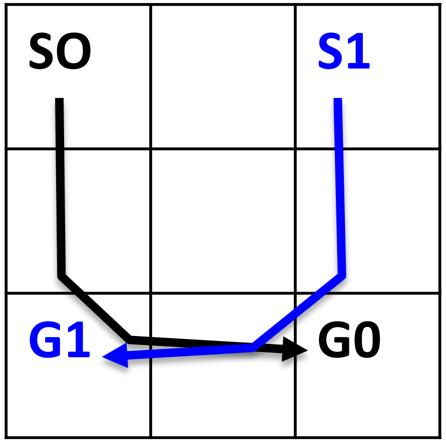


(Standley '10)

Simple Independence Detection

1. Solve optimally each agent separately
2. While some agents conflict
   1. Merge conflicting agents to one group
   2. Solve optimally new group

Theoretically, a 2 agents problem, but …

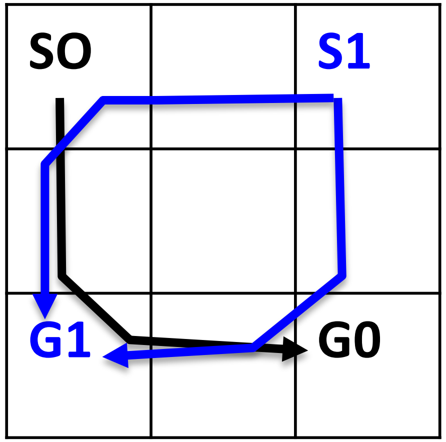


(Standley '10)

Simple Independence Detection

1. Solve optimally each agent separately
2. While some agents conflict
   1. Merge conflicting agents to one group
   2. Solve optimally new group

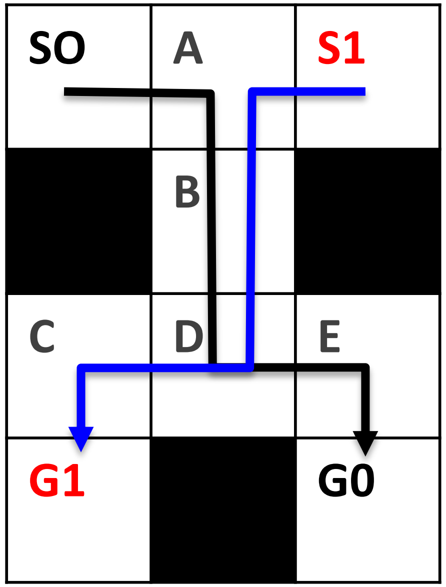
Theoretically, a 2 agents problem, but …



(Standley '10)

Independence Detection

1. Solve optimally each agent separately
2. While some agents conflict
    1. **Try to avoid conflict, with the same cost**
    2. Merge conflicting agents to one group
    3. Solve optimally new group

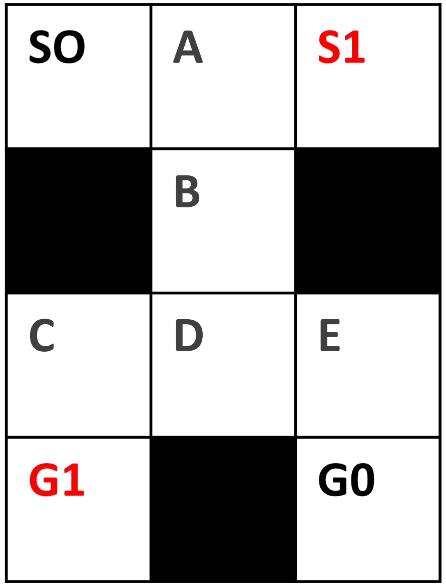


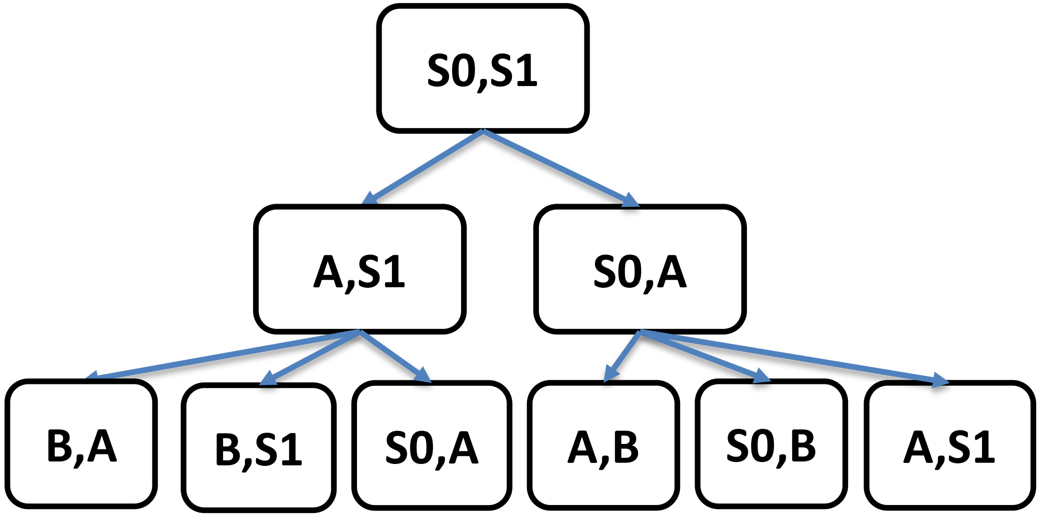
Really a 2 agent problem

But….

Independence Detection

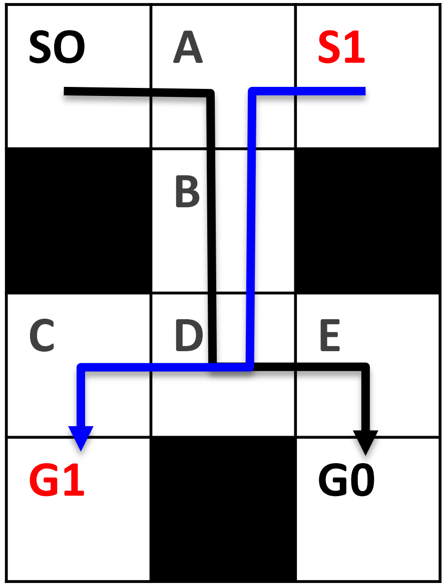
1. Solve optimally each agent separately
2. While some agents conflict
    1. **Try to avoid conflict, with the same cost**
    2. Merge conflicting agents to one group
    3. Solve optimally new group





M*

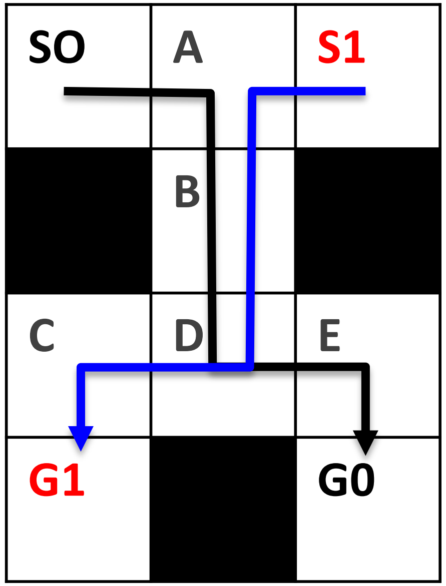
1. Find optimal path for each agent individually
2. Start the search. **Generate only nodes on optimal paths**
3. If **conflict occurs – backtrack** and consider all ignored actions

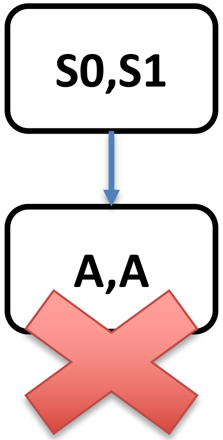


M*

1. Find optimal path for each agent individually
2. Start the search. **Generate only nodes on optimal paths**
3. If **conflict occurs – backtrack** and consider all ignored actions

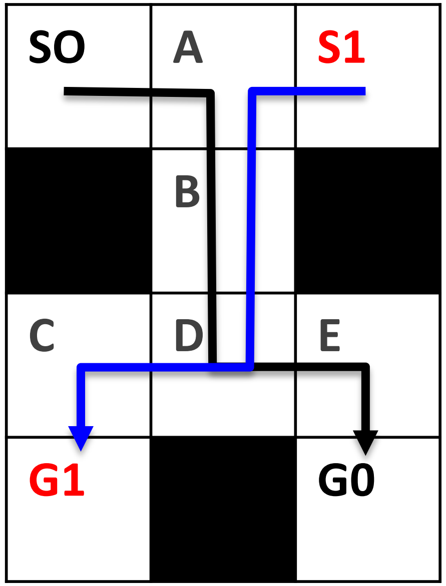
M* (Wagner & Choset '11,'14)

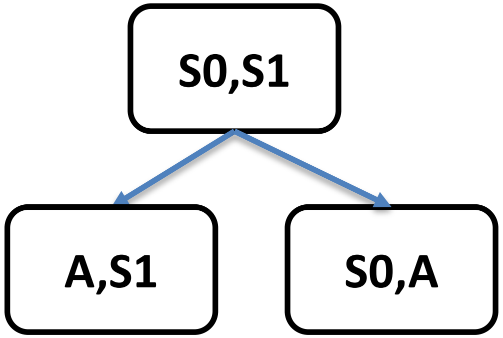




M*

1. Find optimal path for each agent individually
2. Start the search. **Generate only nodes on optimal paths**
3. If **conflict occurs – backtrack** and consider all ignored actions

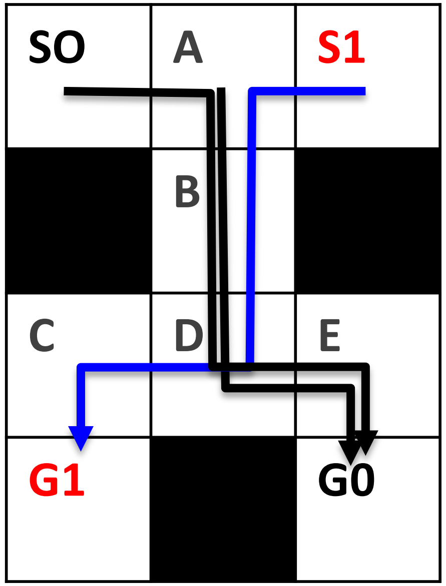


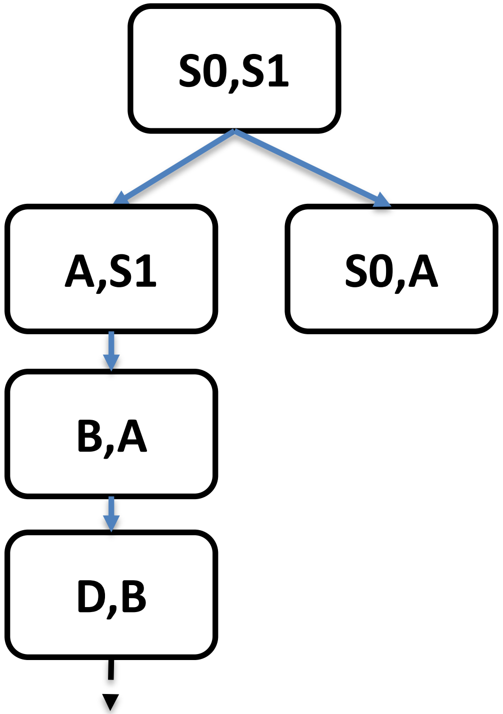


M*

1. Find optimal path for each agent individually
2. Start the search. **Generate only nodes on optimal paths**
3. If **conflict occurs – backtrack** and consider all ignored actions

Recursive M* (Wagner & Choset '11,'14)

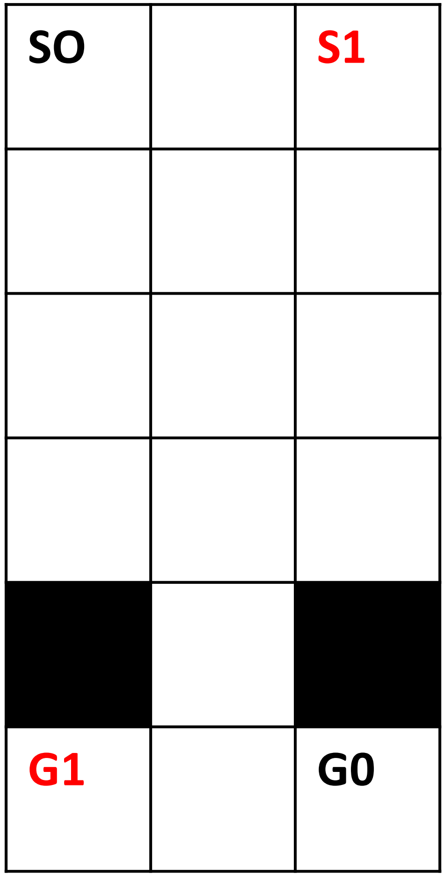


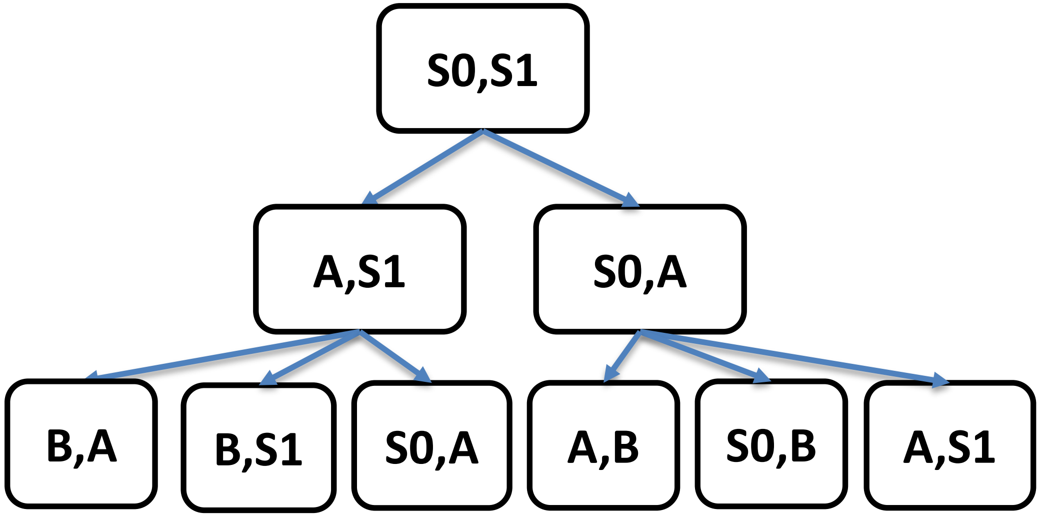


Recursive M*

1. Find optimal path for each agent individually
2. Start the search. **Generate only nodes on optimal paths**
3. If **conflict occurs – backtrack** and consider all ignored actions
   - Apply M* recursively after backtracking

Recursive M* (Wagner & Choset '11,'14)





Joint path up to bottleneck can be long…

Searching the k-agent search space

– A*+OD+ID [Standley '10]

– EPEA* [Felner 'X, Goldenberg 'Y]

– M* [Wagner & Choset 'Z]

Other search-based approaches

– ICTS [Sharon et al '13]

– CBS [Sharon et al '15]

High-level

Is there a solution with costs

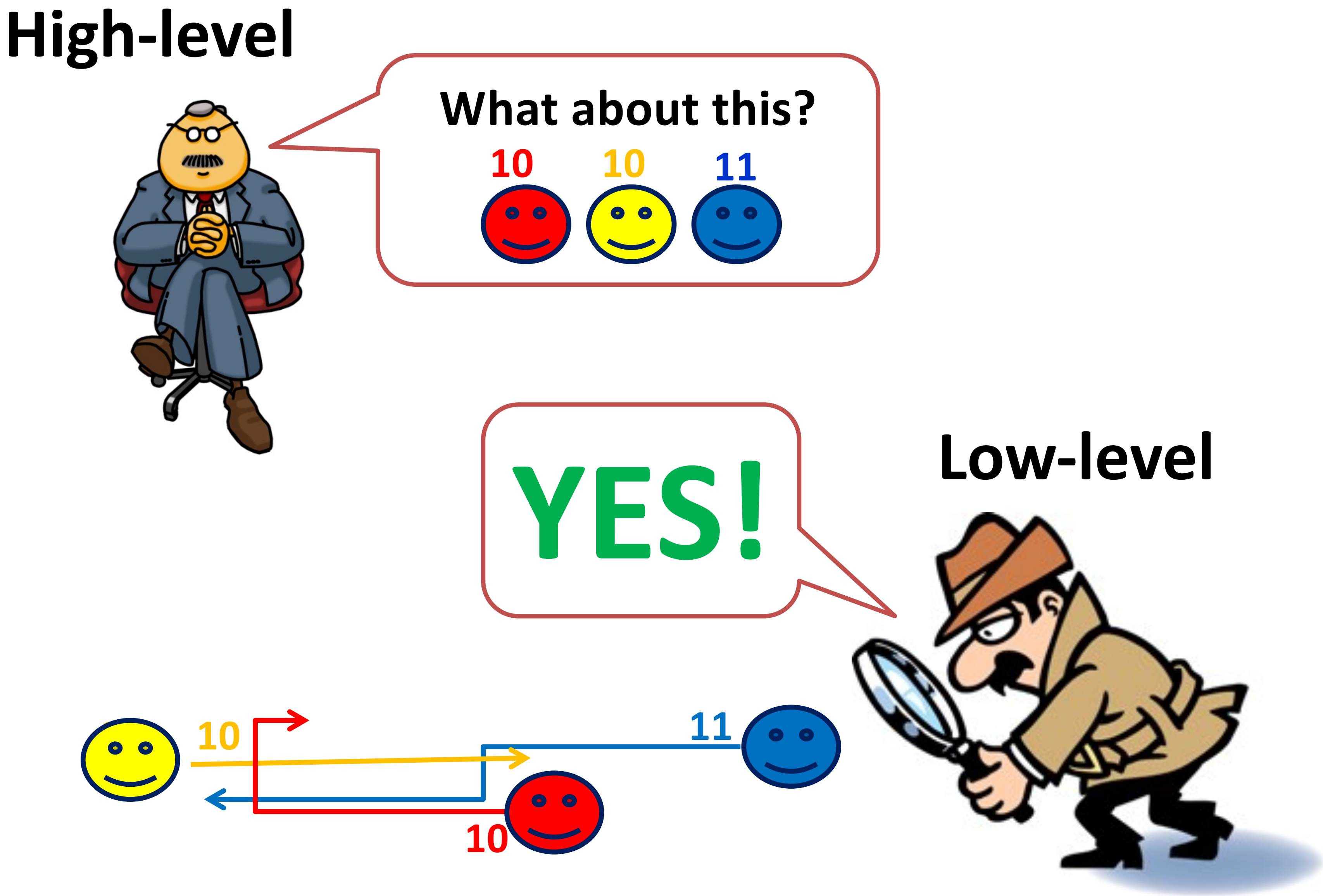
10 10 10

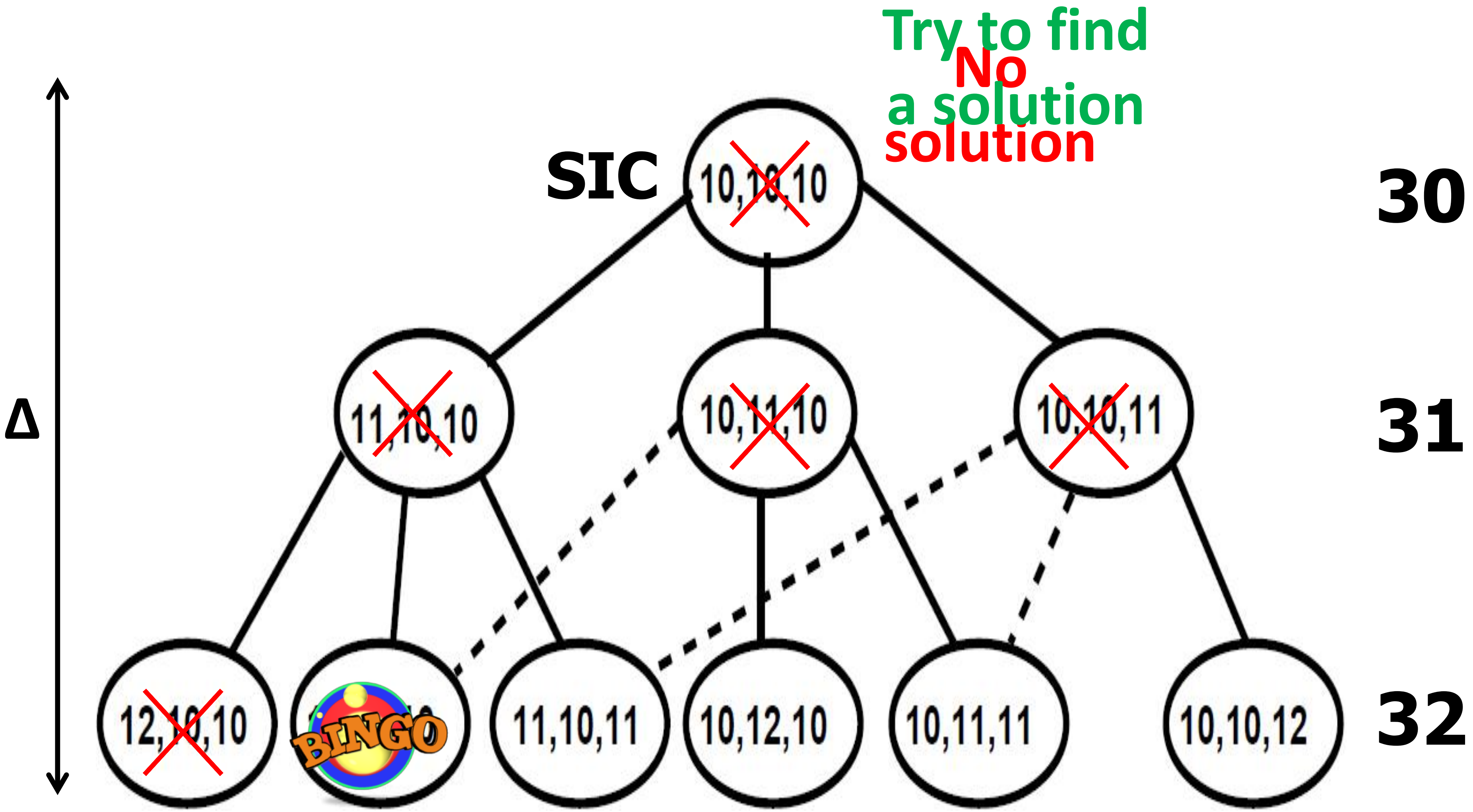
?

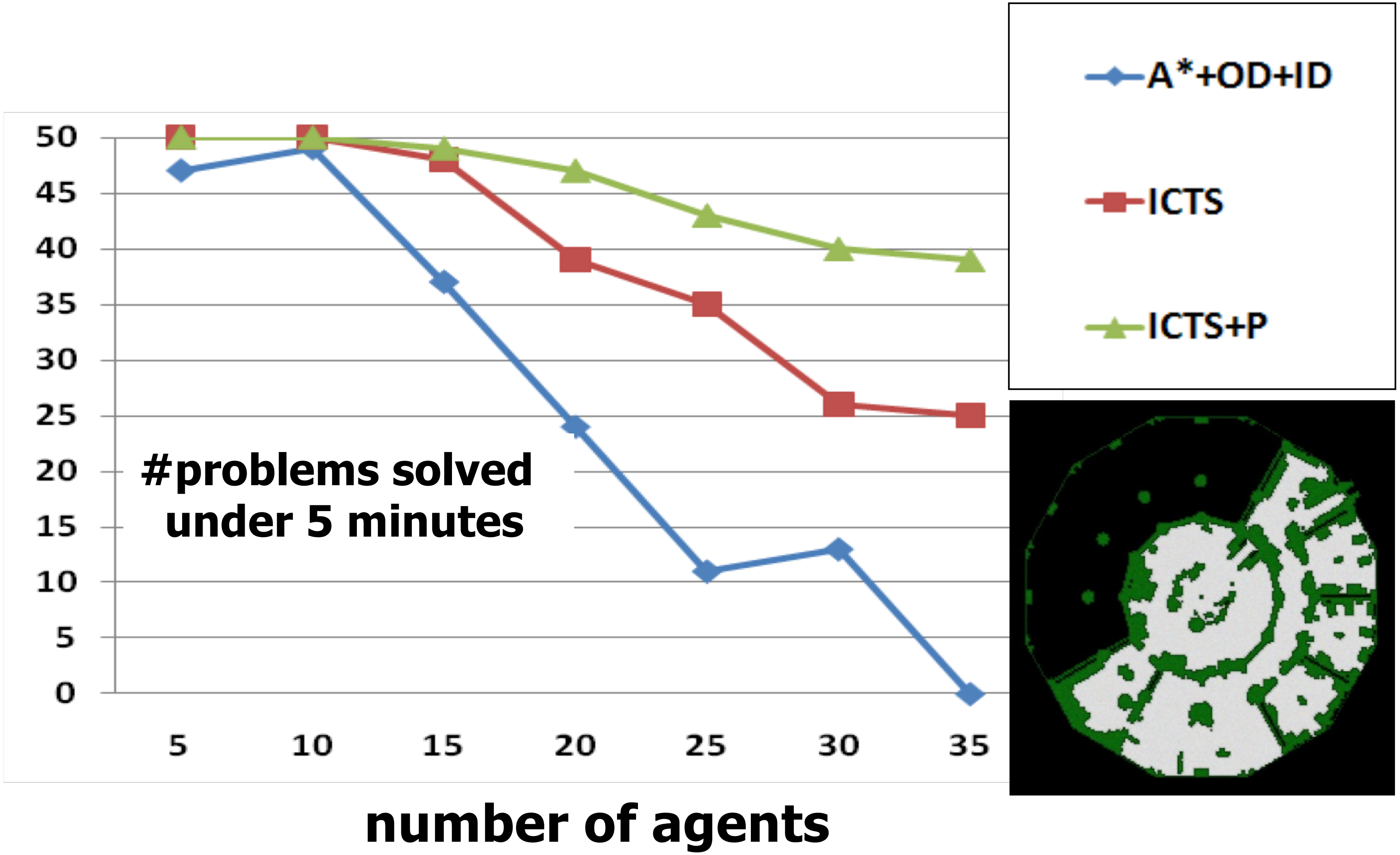
NO!

Low-level

Increasing Cost Tree Search *(Sharon et al. '12)*

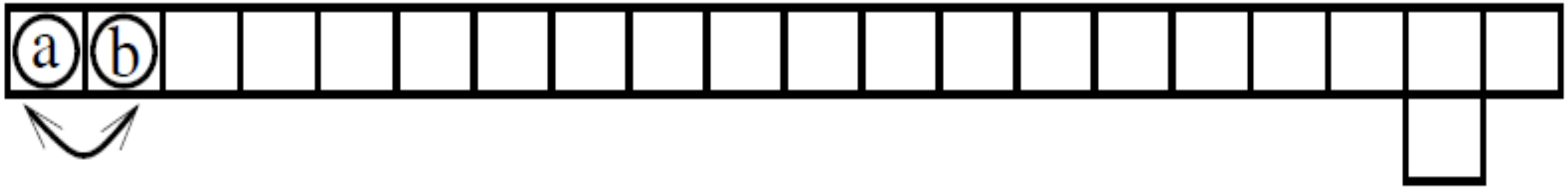



Try to find a solution
No solution
SIC
10,10,10
11,10,10
10,11,10
10,10,11
12,10,10
BINGO
11,10,11
10,12,10
10,11,11
10,10,12
Δ
30
31
32



A*+OD+ID
ICTS
ICTS+P
50
45
40
35
30
25
20
15
10
5
0
#problems solved
under 5 minutes
5
10
15
20
25
30
35
number of agents

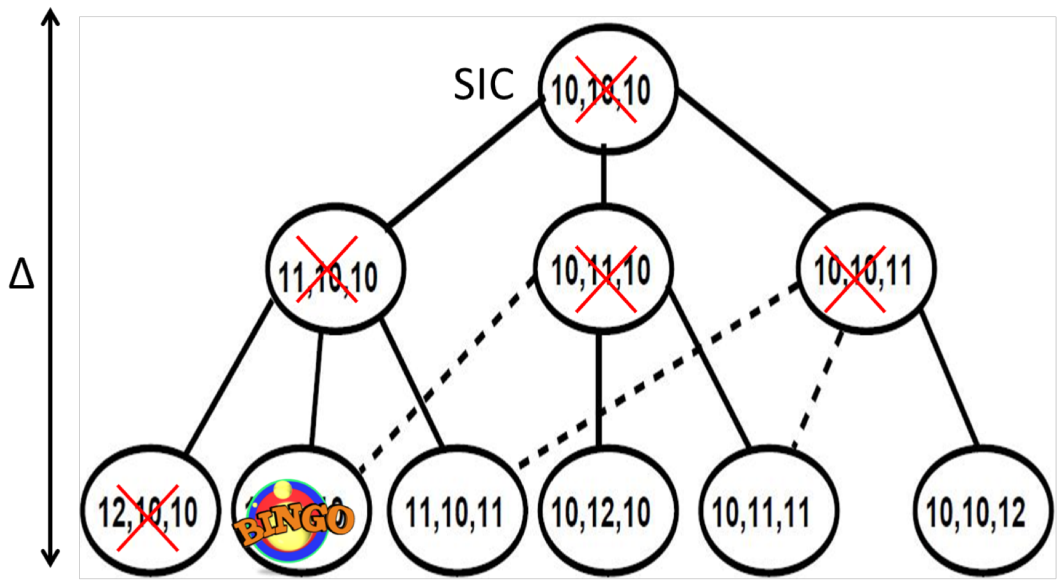
Does it work? – Not Always



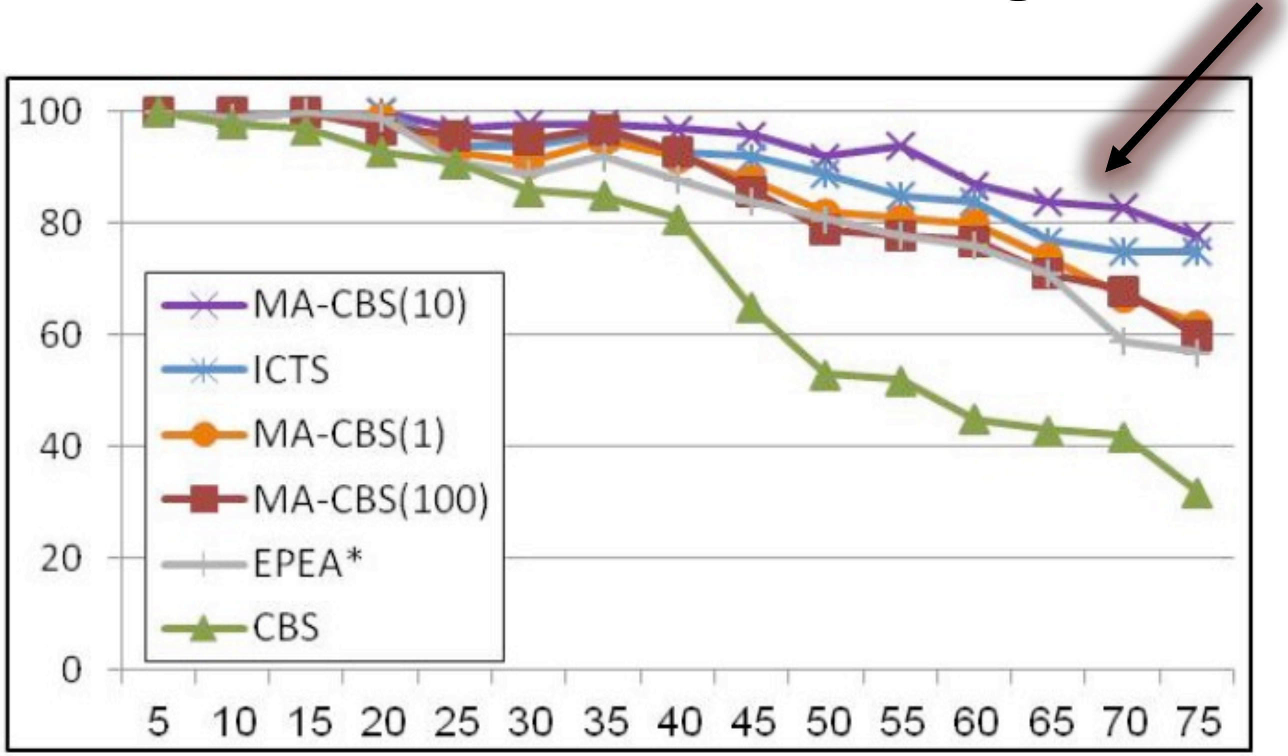
- A*: solved in **51**ms

ICTS Complexity depends on Δ

  - Sum of single agent costs =2 **BUT** optimal solution =74

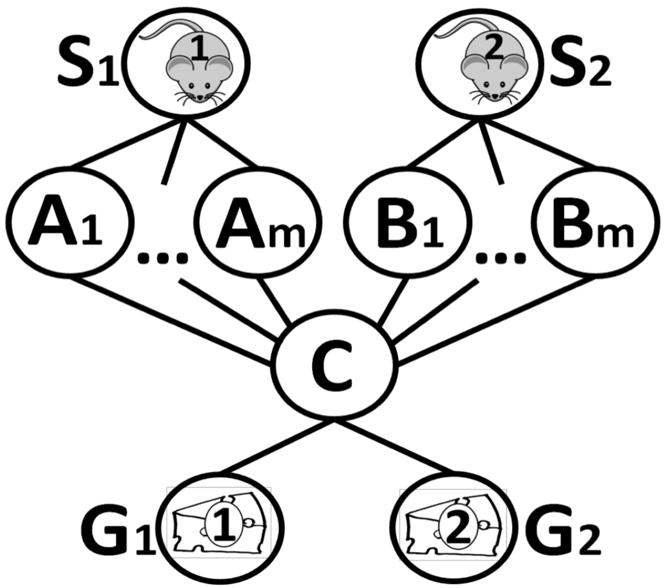


Solving Optimally Problems with more than 75 agents!



CBS only performs single agents

But, many times, and under different constraints



Conflict: agent 1 and agent 2 plan to occupy C at time 2

⬇

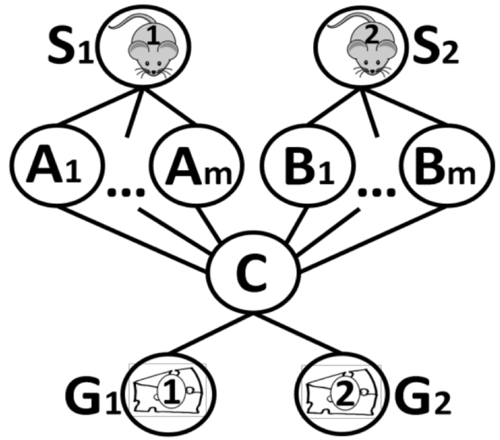
Constrain agent 1, avoid C at time 2
or
Constrain agent 2 to avoid C at time 2

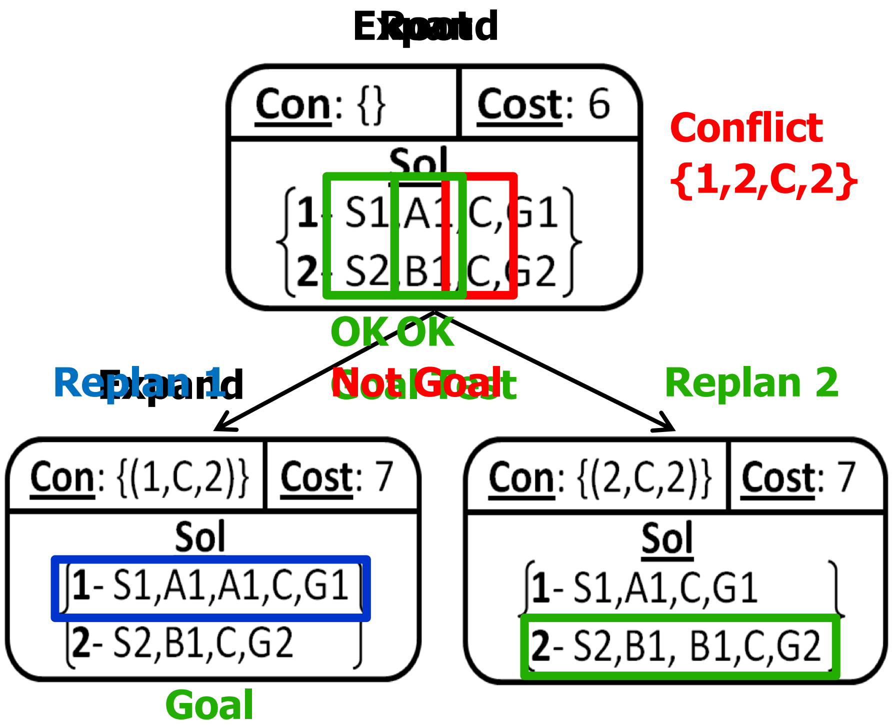
Nodes:

- A set of individual constraints for each agent
- A set of paths consistent with the constraints

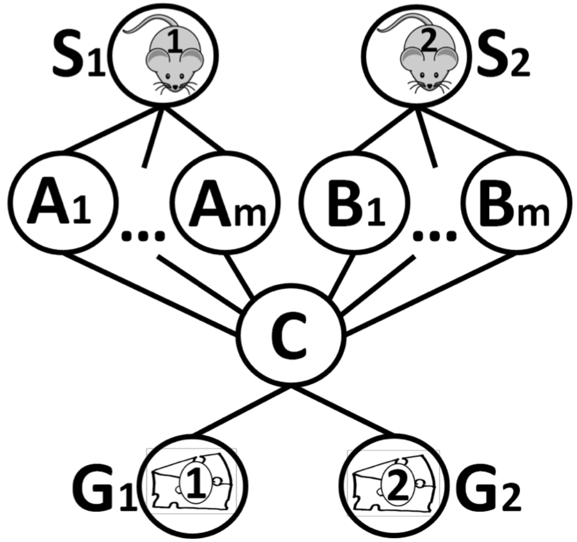
Goal test:

- Are the paths conflict free





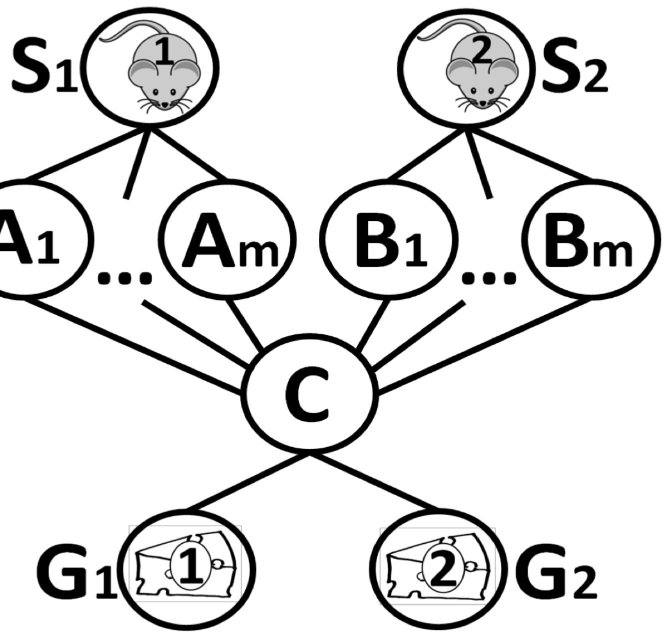
- How many states **A*** will expand?
- How many states **CBS** will?

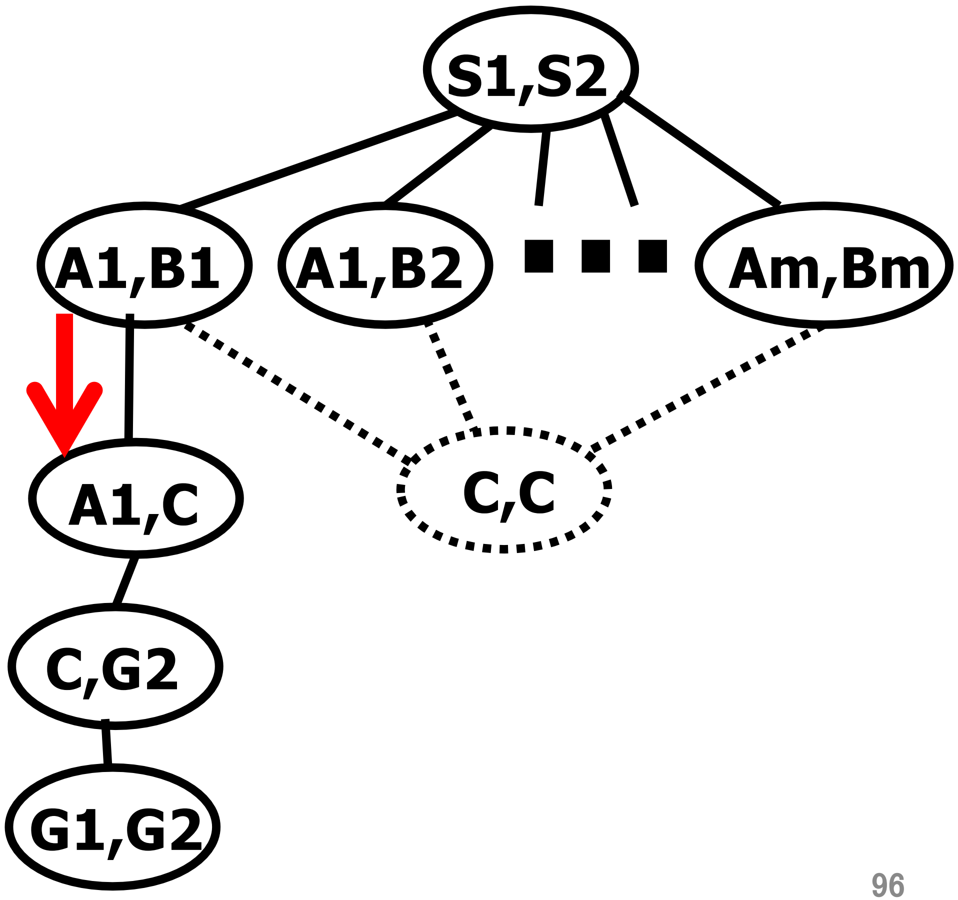


Motivation: cases with bottlenecks:

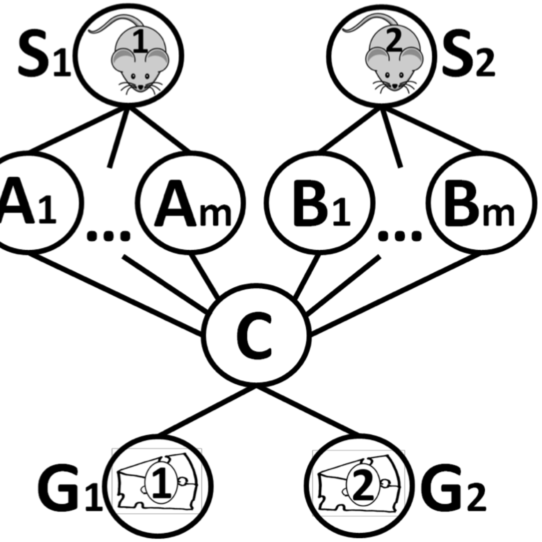
A*

- **g+h=6:** ***All*** m^2 ***combinations*** **of** (A_i, B_j) **will be expanded**
- **f=7:** **3 states are expanded**

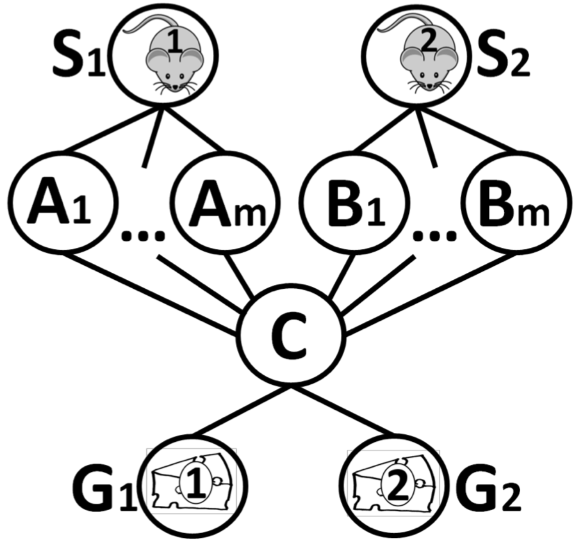




- **A*** **:** $m^2+3 = O(m^2)$ states
- **CBS**: **?**



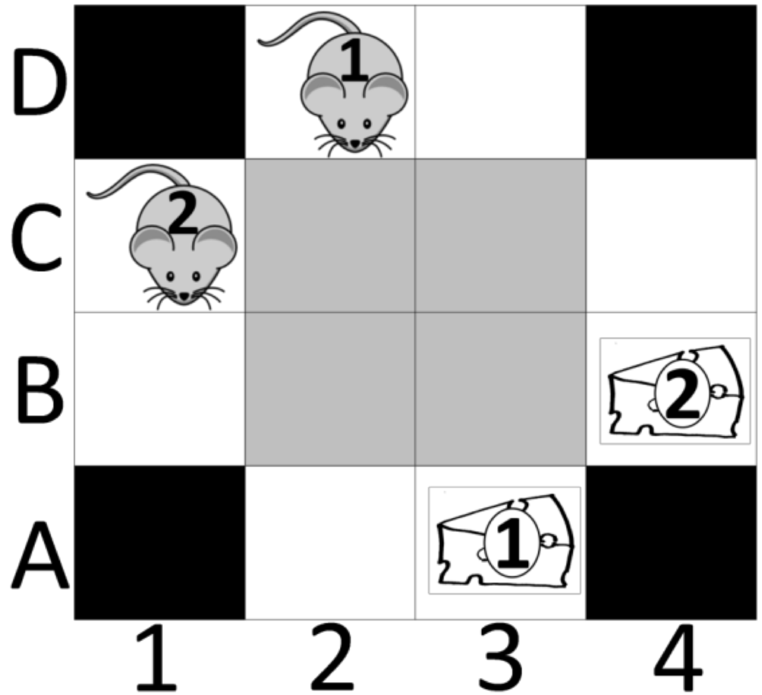
- **A*** : $m^2+3 = O(m^2)$ states
- **CBS:** $2m+14 = O(m)$ states

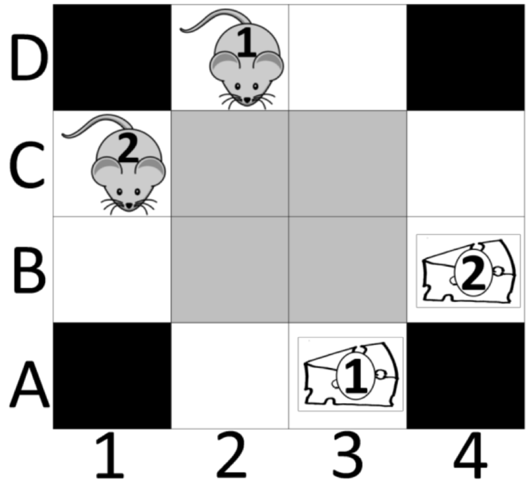


When m > 4 CBS will examine fewer states than A*

Analysis: Example 2

- States expanded by CBS**?**
- States expanded by A***?**





- 4 optimal solutions for each agent
- Each pair of solutions has a conflict

- Rough analysis:
  - **CBS:** exponential in #conflicts = 54 states
  - **A*:** exponential in #agents = 8 states

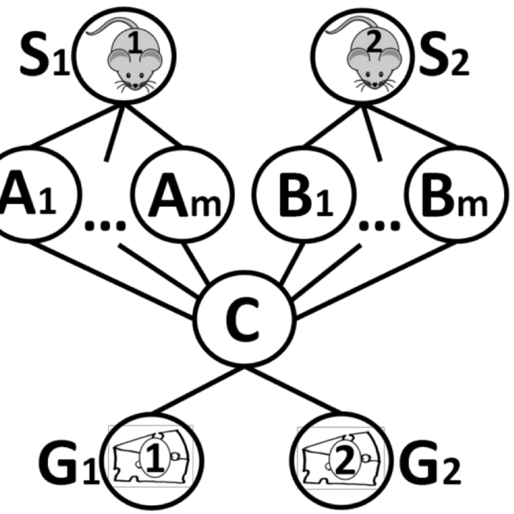
<u>Trends observed</u>
- **In open spaces: use A***
- **In bottlenecks: use CBS**

What if I have both?

1. Plan for each agent individually
2. Validate plans
3. If the plans of agents A and B conflict

4 If (should merge(A,B))
merge A and B into a meta-agent
and solve with A*
Else

5 Constrain A to avoid the conflicts
or
Constrain B to avoid the conflict



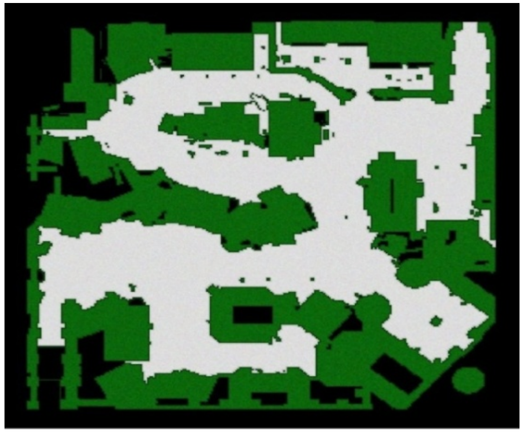
Should merge(A,B) (simple rule):

Merge when observed more than T conflicts between A,B

T=0 (always merge)	MA-CBS	(never merge) T=∞
Standley's ID		basic CBS

Choosing the Right B

brc202d



den520d

Many bottlenecks

	brc202d with EPEA* as a low-level solver						
k	EPEA*	B(1)	B(5)	B(10)	B(100)	B(500)	CBS
5	1,834	2,351	1,286	1,276	1,268	**1,267**	1,664
10	6,034	8,059	4,580	4,530	**4,498**	4,508	5,495
15	12,354	15,389	6,903	6,871	6,820	**6,793**	8,685
20	> 70,003	>73,511	35,095	21,729	**19,846**	31,229	>43,625

Few bottlenecks

	den520d with A* as a low-level solver						
k	A*	B(1)	B(5)	B(10)	B(100)	B(500)	CBS
5	0.223	273	218	220	**219**	222	**219**
10	1,099	1,458	553	552	549	552	**546**
15	1,182	1,620	1,838	1,810	1,829	1,703	**1,672**
20	4,792	4,375	1,996	2,011	2,020	1,857	**1,708**
25	7,633	14,749	**2,193**	2,255	2,320	2,888	3,046
30	> 62,717	> 60,214	8,082	8,055	8,107	8,013	**7,745**
35	> 65,947	> 51,815	13,670	**13,587**	15,981	28,274	> 45,954
40	> 81,487	>82,860	18,473	**18,399**	20,391	31,189	> 45,857

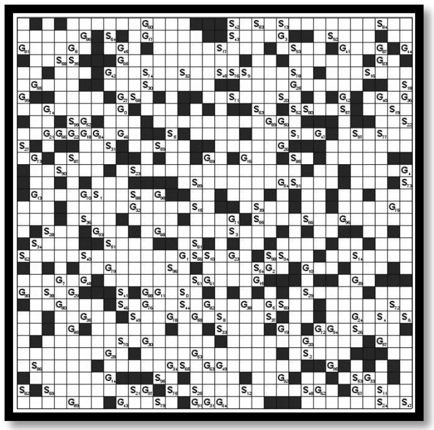
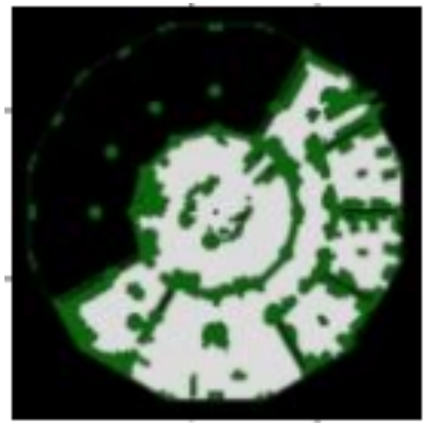
Many bottlenecks → High T (closer to CBS)
More agents → Low T (closer to A*)
Faster single-agent search → lower T (close to A*)

- Which conflict to resolve? [Boyarski et al. ‘16]
- What to do after merging? [Boyarski et al. ‘16]
- Heuristics for the constraint tree search [Felner et al. ‘18]
- Augmenting CBS with human knowledge [Cohen et al.]
- Which low-level solver to use?
- When to merge the agents ?

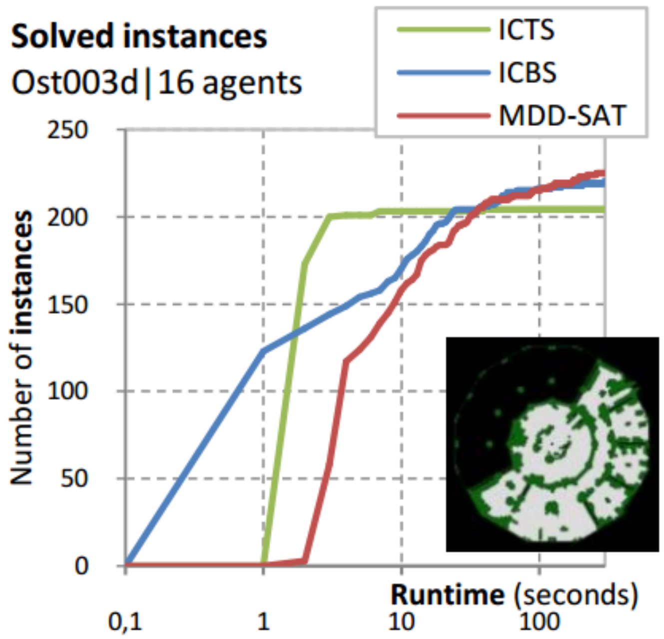
…

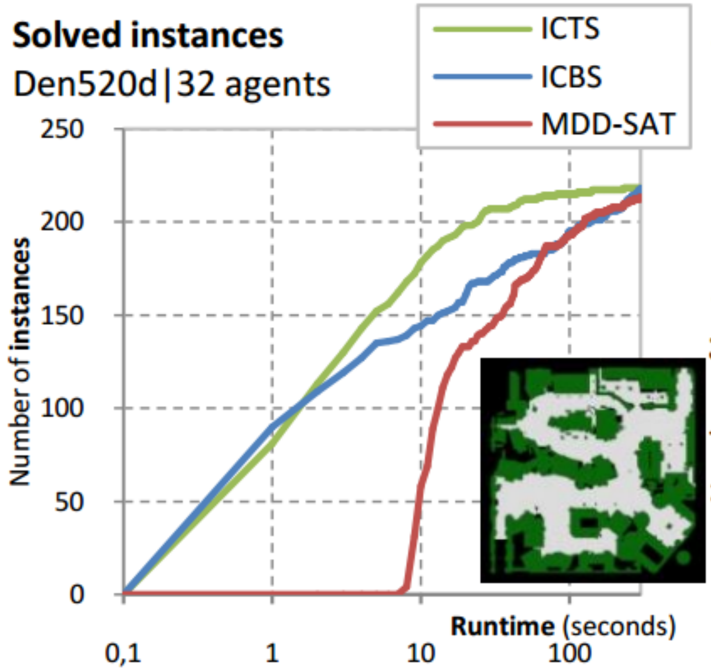
- A* (M*, EPEA*, A*+OD+ID)
  - Main factors: #agents, graph size, heuristic accuracy
- ICTS
  - Main factors: #agents, Δ, graph size
- CBS and its variants
  - Main factors: #conflicts

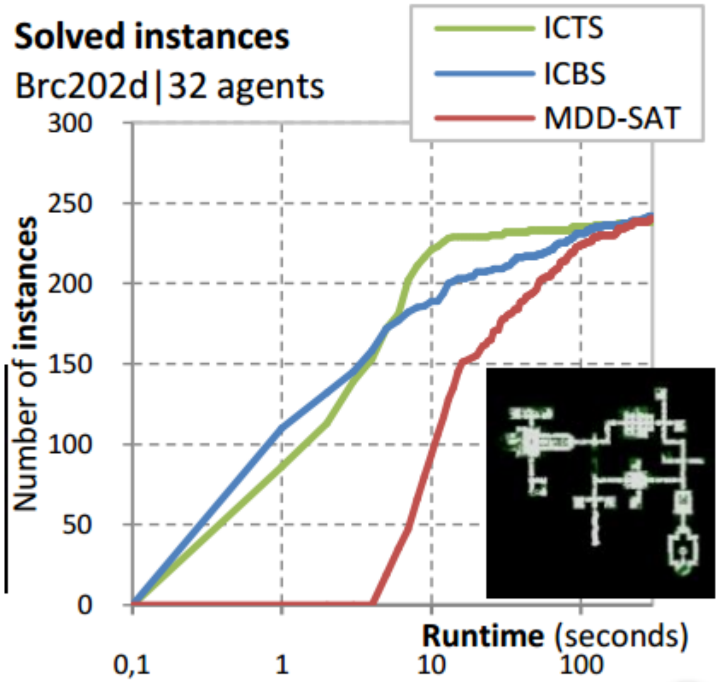
Where to use what?

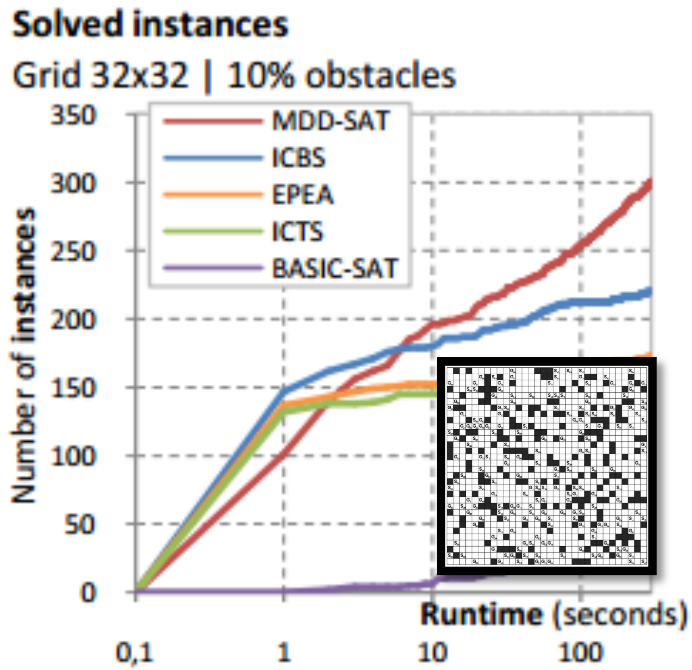



Results...

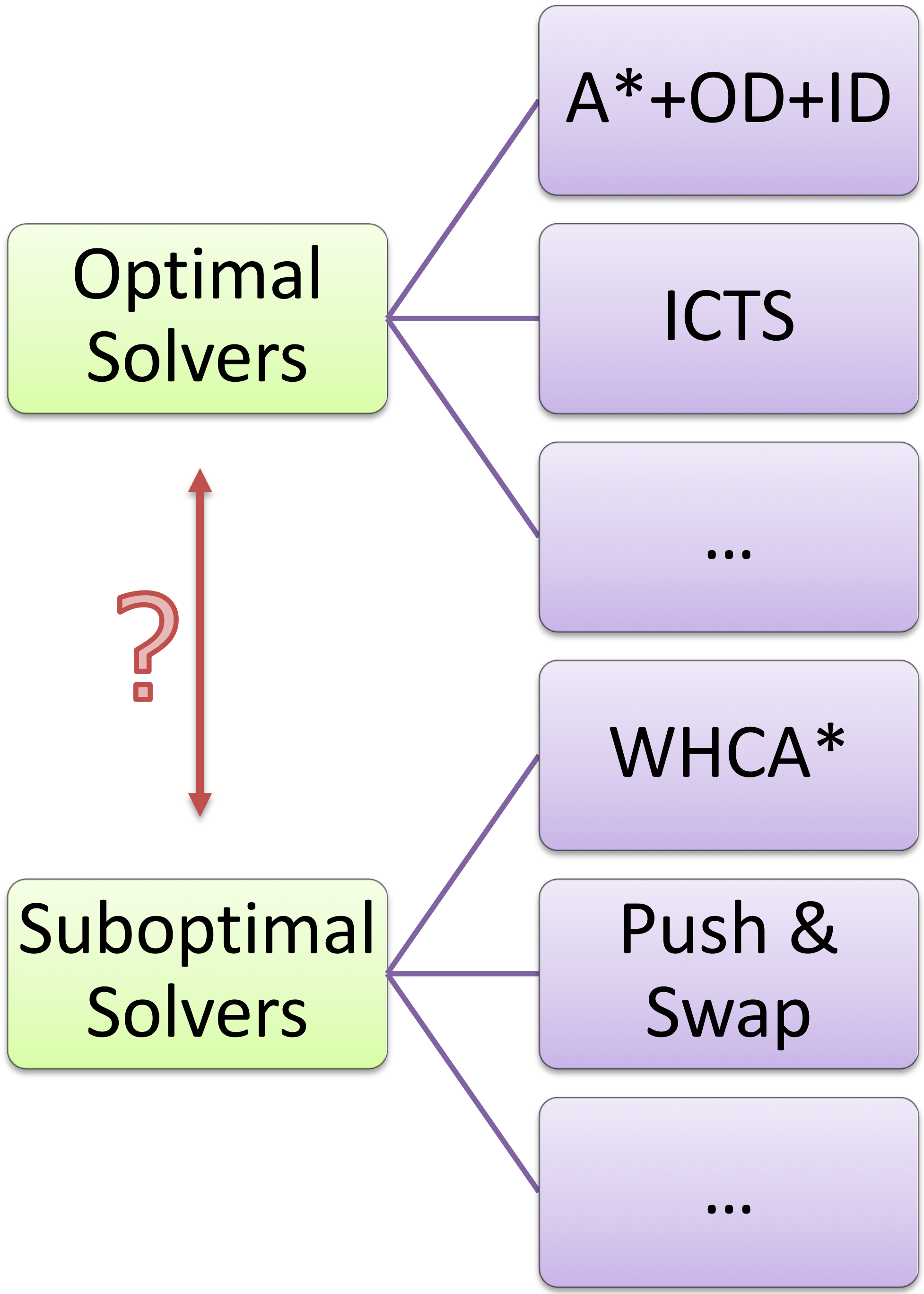

Solved instances
Ost003d|16 agents
ICTS
ICBS
MDD-SAT
250
200
150
100
50
0
Number of instances
0,1
1
10
100
Runtime (seconds)


Solved instances
Den520d|32 agents
ICTS
ICBS
MDD-SAT
250
200
150
100
50
0
Number of instances
0,1
1
10
100
Runtime (seconds)


Solved instances
Brc202d|32 agents
ICTS
ICBS
MDD-SAT
300
250
200
150
100
50
0
Number of instances
0,1
1
10
100
Runtime (seconds)


Solved instances
Grid 32x32 | 10% obstacles
MDD-SAT
ICBS
EPEA
ICTS
BASIC-SAT
350
300
250
200
150
100
50
0
Number of instances
0,1
1
10
100
Runtime (seconds)

	Suboptimal	Optimal
Incomplete	• Cooperative A* • WHCA*	?
Complete	• Kornhauser et al. ’84 • Push & Swap (Luna & Bekris) • Bibox (Surynek) …	• A*+OD+ID (Standley) • ICTS (Sharon et al.) • M* (Wagner & Choset) • CBS (Sharon et al.) …

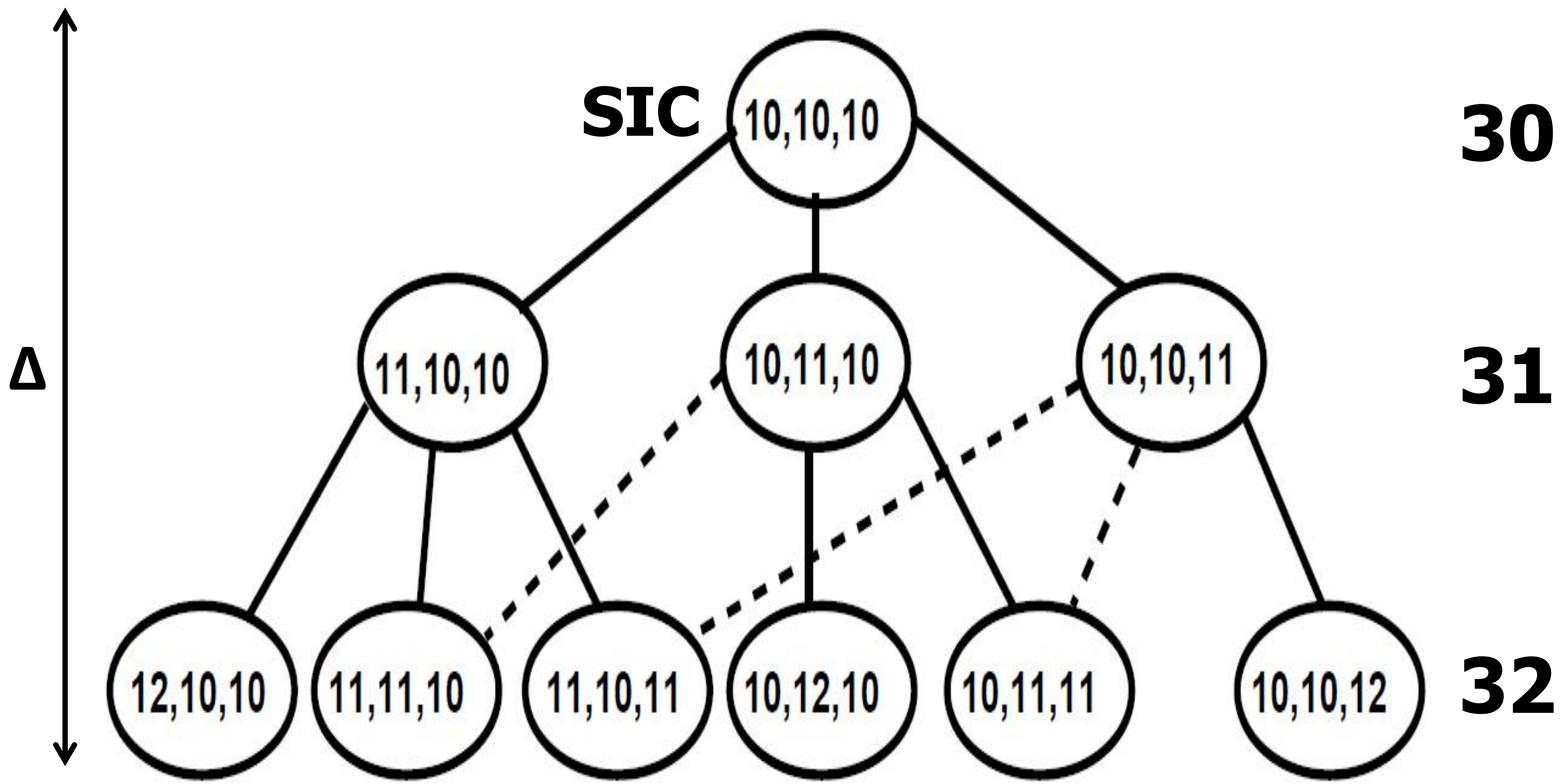

Optimal Solvers
A*+OD+ID
ICTS
...
?
Suboptimal Solvers
WHCA*
Push & Swap
...

An algorithm is bounded suboptimal iff

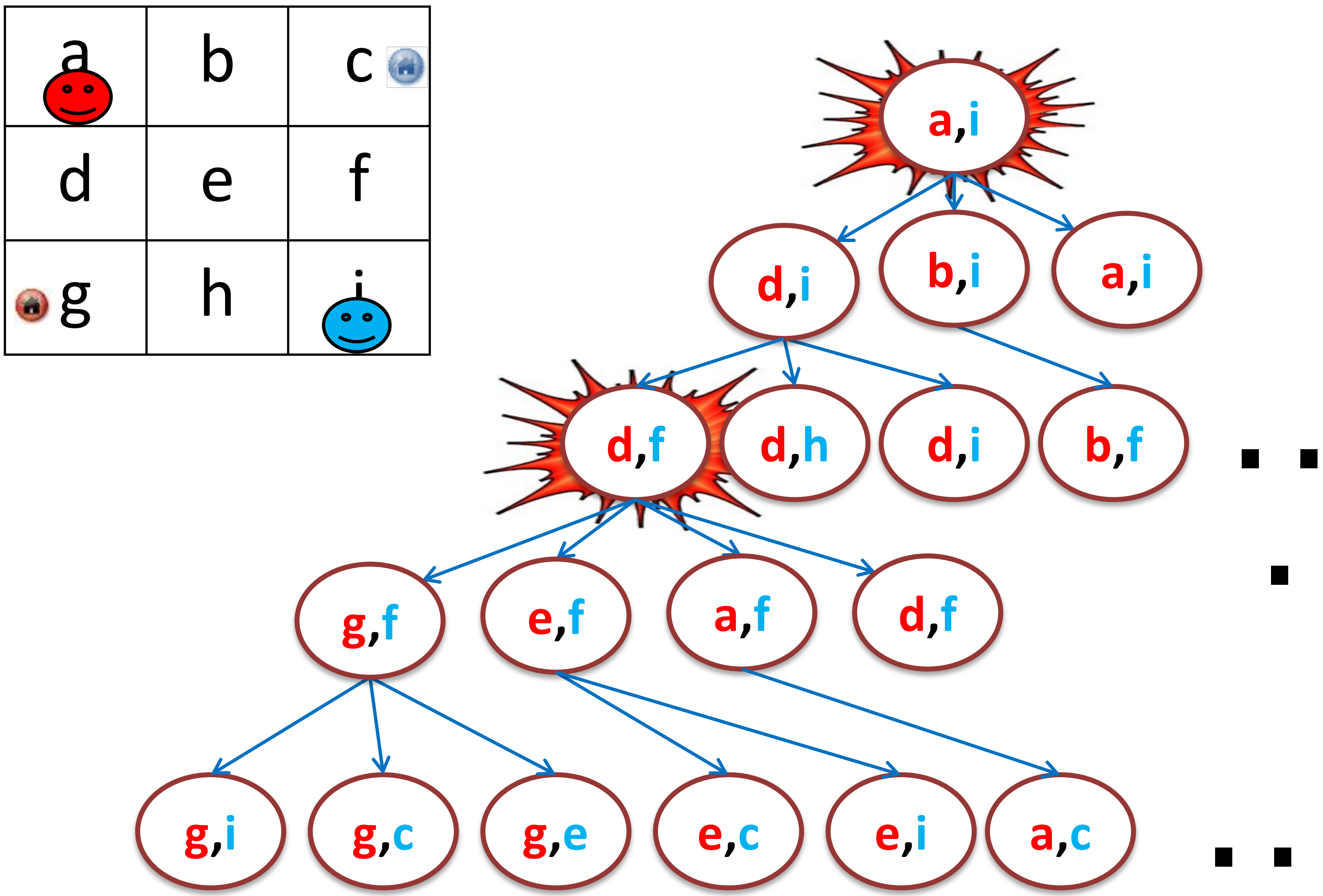
- **It accepts a parameter ϵ**
- **It outputs a solution whose cost is at most $(1+\epsilon)$ ·Optimal**

How to create a bounded suboptimal algorithm?

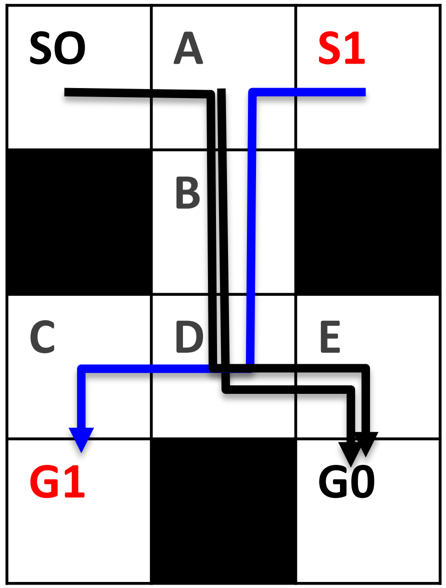
- Different search algorithms
- Inadmissible heuristics

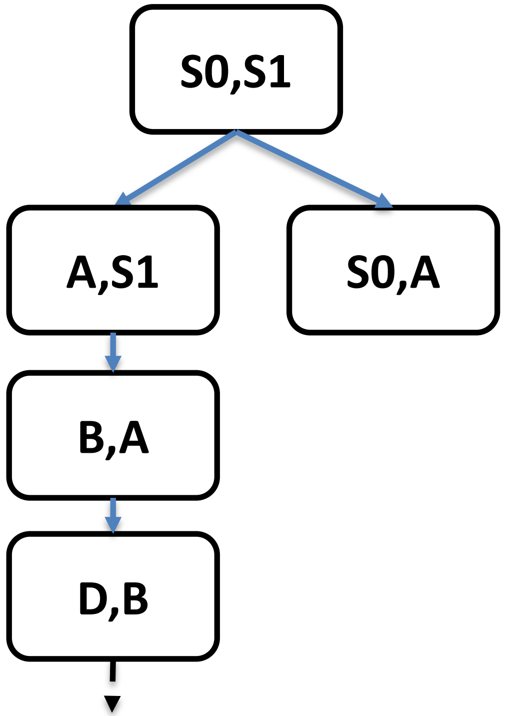


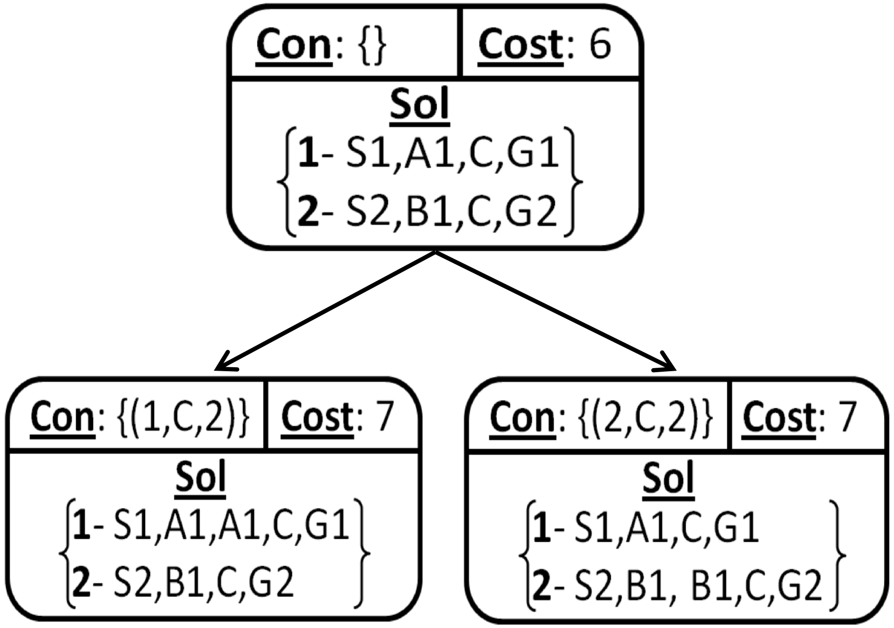
Open Question!


a
b
c
d
e
f
g
h
i
a,i
d,i
b,i
a,i
d,f
d,h
d,i
b,f
g,f
e,f
a,f
d,f
g,i
g,c
g,e
e,c
e,i
a,c

*Suboptimal rM**





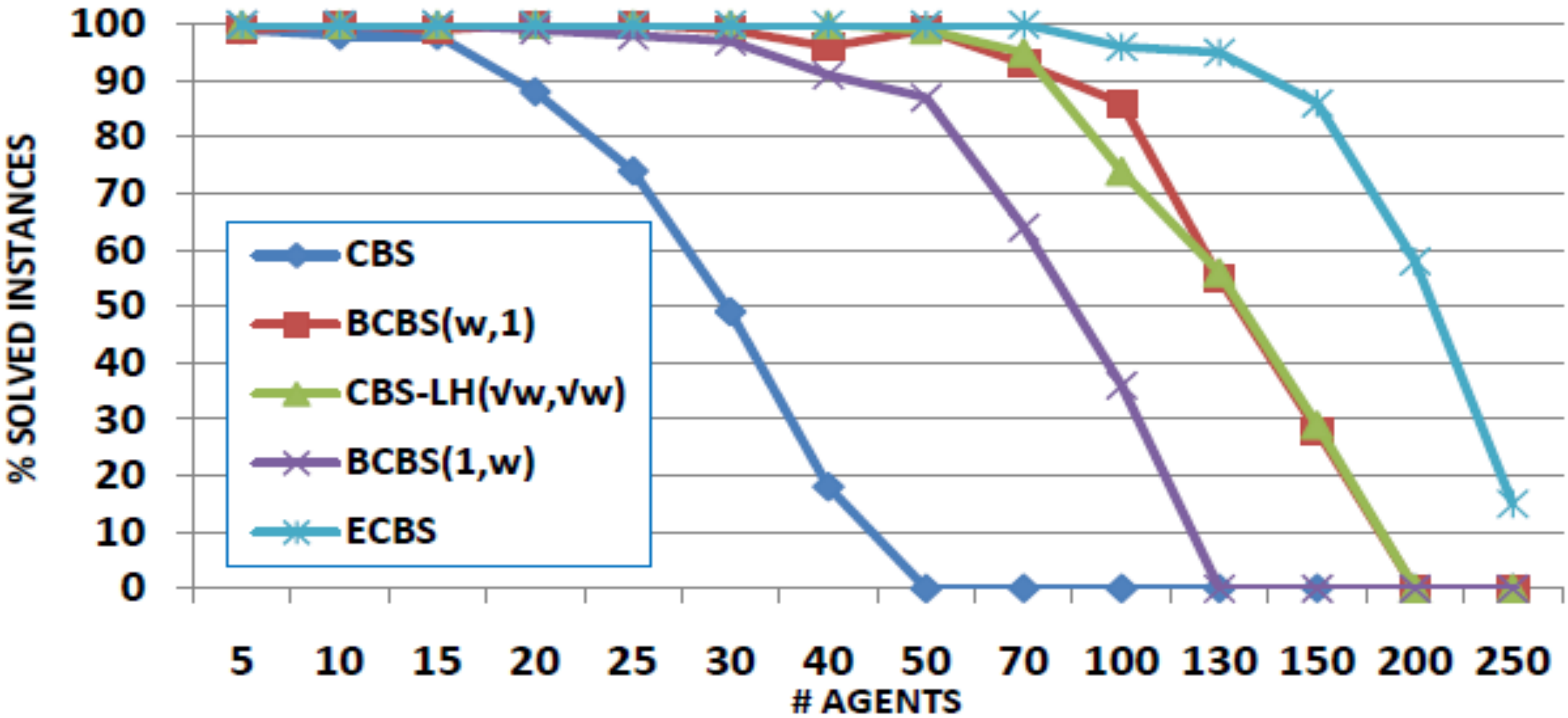


Observation:

Suboptimality can be introduced in both levels

– ECBS (Barer et al. ‘14)

– ECBS+Highways (Cohen et al. ’15, ‘16)



100
90
80
70
60
50
40
30
20
10
0
% SOLVED INSTANCES
CBS
BCBS(w,1)
CBS-LH(√w,√w)
BCBS(1,w)
ECBS
5
10
15
20
25
30
40
50
70
100
130
150
200
250
AGENTS

- When to use which algorithm? Ensembles?
- Using **knowledge** about past plans (Cohen et al. ’15)
- Stronger **heuristics** for all algorithms
- Deeper **analysis** of algorithms’ complexity
- Beyond grid worlds
  - Kinematic constraints (Ma et al. ‘16)
  - Any-angle planning (Yakovlev et al. ‘17)
  - Hierarchical environments (Walker et al. ’17)
  - Large agents (Li et al. ‘19)
- Priorized planning based on CBS (Ma et al. ‘19)
- Planning & **execution** (see later today ☺)

Part III:

REDUCTION-BASED SOLVERS

How to **exploit knowledge of others** for solving own problems?

- by translating the problem P to another problem Q

Why is it useful?

- If anybody improves the solver for Q then we get an improved solver for P for free.
- Staying on the shoulders of giants.

Reduction, compilation, re-formulation techniques

Boolean satisfiability

– fast SAT solvers

Constraint programming

– global constraints for pruning search space

Answer set programming

– declarative framework

Combinatorial auctions

…

Express (model) the problem as a **SAT formula** in a conjunctive normal form (CNF)

Boolean *variables* (true/false values)

clause = a disjunction of literals (variables and negated variables)

formula = a conjunction of clauses

solution = an instantiation of variables such that the formula is satisfied

Example:

(X *or* Y) *and* (*not* X *or not* Y)

[exactly one of X and Y is true]

SAT model is expressed as a CNF formula

We can go beyond CNF and use **abstract expressions** that are translated to CNF.

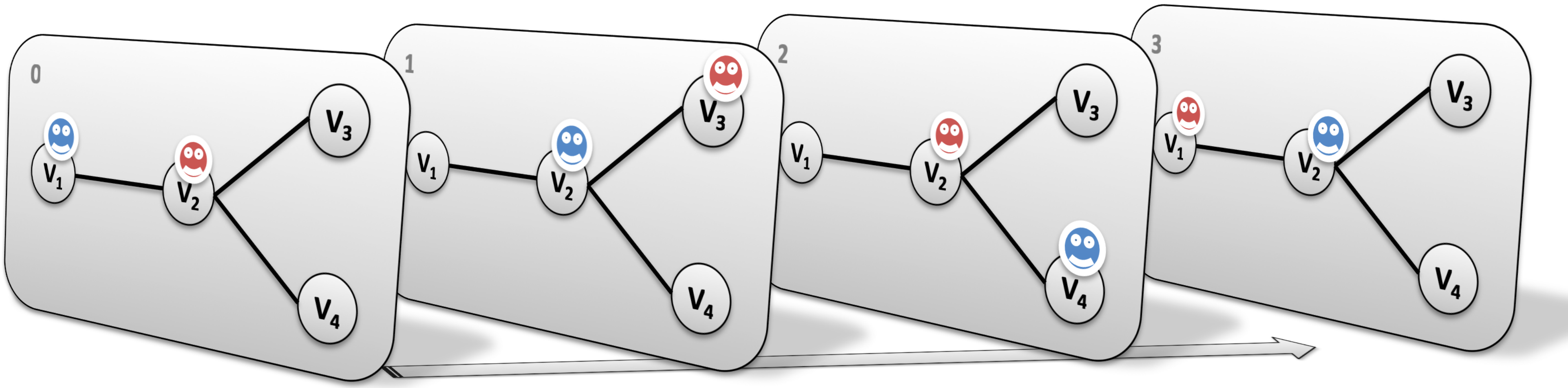
A => B	B *or not* A
sum(Bs) >= 1 (at-least-one(Bs))	disj(Bs)
sum(Bs) = 1	at-most-one(B) *and* at-least-one(B)

We can even use **numerical variables** (and constraints).

In MAPF, we do not know the lengths of plans (due to possible re-visits of nodes)!

We can encode plans of a known length using a **layered graph** (temporally extended graph).

- Each layer corresponds to one time slice and indicates positions of agents at that time.



Uses multi-valued state variables (logarithmic encoding) encoding position of agents in layers.

- Agent waits or moves to a neighbor

$$\mathcal{L}_i^a = l \Rightarrow \mathcal{L}_{i+1}^a = l \vee \bigvee_{\ell \in \{1,\dots,n\} \mid \{v_l, v_\ell\} \in E} \mathcal{L}_{i+1}^a = \ell$$

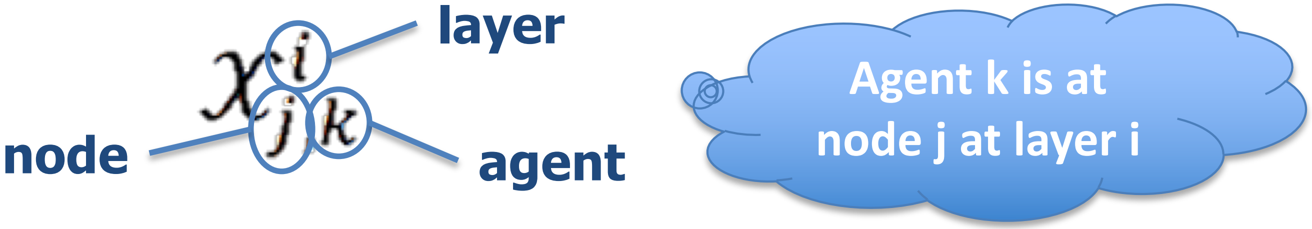
- No-train constraint

$$\bigwedge_{b \in A \mid b \neq a} \mathcal{L}_{i+1}^a \neq \mathcal{L}_i^b$$

- Agents are not at the same nodes

$$\mathrm{AllDifferent}(\mathcal{L}_i^{a_1}, \mathcal{L}_i^{a_2}, \dots, \mathcal{L}_i^{a_\mu})$$

Directly encodes positions of agents in layers



- Agent is placed at exactly one node in each layer

$$\bigwedge_{j,l=1,j<l}^{n} \neg \mathcal{X}_{j,k}^{i} \vee \neg \mathcal{X}_{l,k}^{i} \qquad \bigvee_{j=1}^{n} \mathcal{X}_{j,k}^{i}$$

- No two agents are placed at the same node in each layer

$$\bigwedge_{k,h=1,k<h}^{\mu} \neg \mathcal{X}_{j,k}^{i} \vee \neg \mathcal{X}_{j,h}^{i}$$

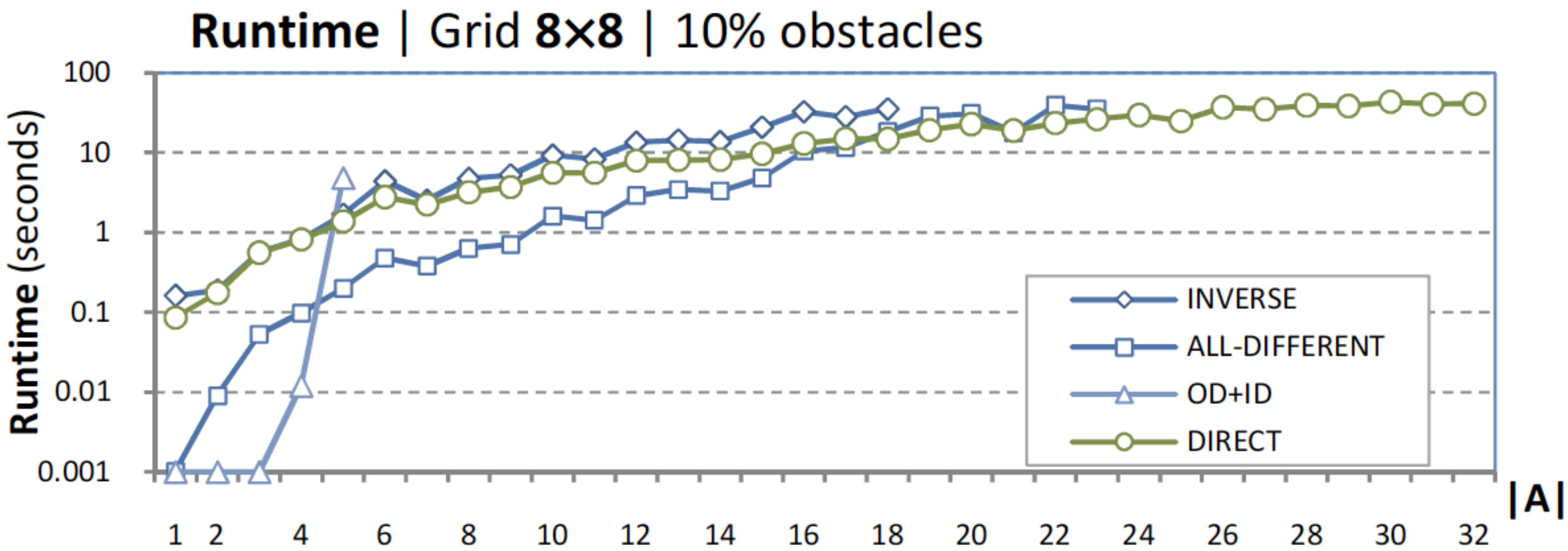
- Agent waits or moves to a neighbor

$$\mathcal{X}_{j,k}^{i} \Rightarrow \mathcal{X}_{j,k}^{i+1} \vee \bigvee_{l:\{v_j,v_l\}\in E} \mathcal{X}_{l,k}^{i+1} \qquad \mathcal{X}_{j,k}^{i+1} \Rightarrow \mathcal{X}_{j,k}^{i} \vee \bigvee_{l:\{v_j,v_l\}\in E} \mathcal{X}_{l,k}^{i}$$

- No-swap and no-train (nodes before and after move are empty)

$$\mathcal{X}_{j,k}^{i} \wedge \mathcal{X}_{l,k}^{i+1} \Rightarrow \bigwedge_{h=1}^{\mu} \neg \mathcal{X}_{l,h}^{i} \wedge \bigwedge_{h=1}^{\mu} \neg \mathcal{X}_{j,h}^{i+1}$$

Finding makespan optimal solutions



Using **layered graph** describing agent positions at each time step

B_{tav} : agent *a* occupies vertex *v* at time *t*

Constraints:

- each agent occupies exactly one vertex at each time.

$$\Sigma_{v=1}^{n} B_{tav} = 1 \text{ for } t = 0, \ldots, m, \text{ and } a = 1, \ldots, k.$$

- no two agents occupy the same vertex at any time.

$$\Sigma_{a=1}^{k} B_{tav} \leq 1 \text{ for } t = 0, \ldots, m, \text{ and } v = 1, \ldots, n.$$

- if agent *a* occupies vertex *v* at time *t*, then *a* occupies a neighboring vertex or stay at *v* at time *t* + 1.

$$B_{tav} = 1 \Rightarrow \Sigma_{u \in neibs(v)} (B_{(t+1)au}) \geq 1$$

Preprocessing:

B_{tav} = 0 if agent *a* cannot reach vertex *v* at time *t* or *a* cannot reach the destination being at *v* at time *t*

Picat code

```
import sat.

path(N,As) =>
    K = len(As),
    lower_upper_bounds(As,LB,UB),
    between(LB,UB,M),
    B = new_array(M+1,K,N),
    B :: 0..1,

    % Initialize the first and last states
    foreach (A in 1..K)
        (V,FV) = As[A],
        B[1,A,V] = 1,
        B[M+1,A,FV] = 1
    end,

    % Each agent occupies exactly one vertex
    foreach (T in 1..M+1, A in 1..K)
        sum([B[T,A,V] : V in 1..N]) #= 1
    end,

    % No two agents occupy the same vertex
    foreach (T in 1..M+1, V in 1..N)
        sum([B[T,A,V] : A in 1..K]) #=< 1
    end,

    % Every transition is valid
    foreach (T in 1..M, A in 1..K, V in 1..N)
        neibs(V,Neibs),
        B[T,A,V] #=>
        sum([B[T+1,A,U] : U in Neibs]) #>= 1
    end,

    solve(B),
    output_plan(B).
```

Incremental generation of layers

Setting the initial and destination locations

Agent occupies one vertex at any time

No conflict between agents

Agent moves to a neighboring vertex

```
foreach(T in 1..M1, A in 1..K, V in 1..N)
    B[T,A,V] #=> sum([B[Prev,A2,V] :
                      A2 in 1..K, A2!=A,
                      Prev in max(1,T-L)..T]) #= 0
end
```

K-robustness

Instance	Makespan			Sum of costs		
	Picat	MDD	ASP	Picat	MDD	ICBS
g16_p10_a05	0.27	**0.02**	10.86	5.68	**0.01**	**0.01**
g16_p10_a10	1.37	**0.14**	9.58	35.82	**0.01**	**0.01**
g16_p10_a20	2.76	**0.76**	26.06	143.35	**0.01**	**0.01**
g16_p10_a30	3.11	**0.79**	>600	495.04	0.52	**0.02**
g16_p10_a40	8.25	**4.71**	>600	>600	**107.95**	>600
g16_p20_a05	1.01	**0.16**	5.96	16.2	**0.01**	**0.01**
g16_p20_a10	1.5	**0.31**	18.59	92.16	1.58	**0.16**
g16_p20_a20	2.12	**0.46**	20.71	209.74	0.6	**0.05**
g16_p20_a30	4.37	**1.45**	>600	>600	>600	>600
g16_p20_a40	3.48	**1.15**	>600	>600	>600	>600
g32_p10_a05	1.98	**0.53**	12.93	29.91	**0.01**	**0.01**
g32_p10_a10	3.08	**1.21**	31.34	84.92	**0.01**	**0.01**
g32_p10_a20	8.71	**6.8**	105.47	586.71	0.03	**0.01**
g32_p10_a30	**34.48**	40.13	274.11	>600	0.22	**0.02**
g32_p10_a40	34.95	**24.87**	>600	>600	1.81	**0.34**
g32_p20_a05	5.75	**2.77**	11.99	58.27	**0.01**	**0.01**
g32_p20_a10	2.97	**1.11**	33.22	112.2	0.09	**0.01**
g32_p20_a20	16.93	**13.73**	101.84	>600	2.5	**0.22**
g32_p20_a30	12.98	**4.54**	199.69	>600	1.78	**0.05**
g32_p20_a40	16.51	**8.17**	418.56	>600	3.24	**0.13**
Total solved	20	20	15	12	18	17

Runtime in seconds

Makespan (minimize the maximum end time)

incrementally add layers until a solution found

Sum of cost (minimize the sum of end times)

incrementally add layers and look for the SOC optimal solution in each iteration (makespan+SOC optimal)

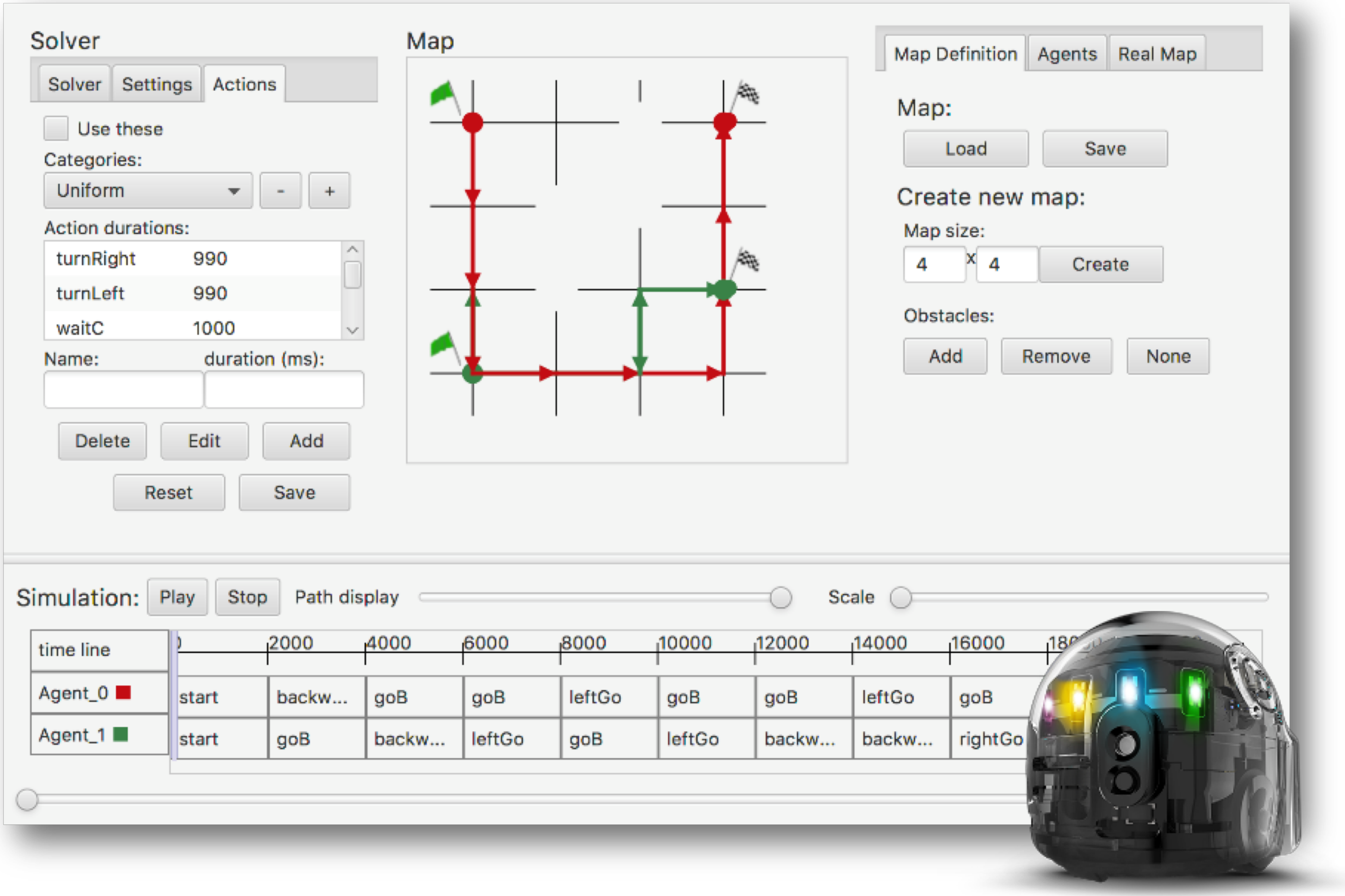
generate more layers (upper bound) and then optimize SOC (naïve)

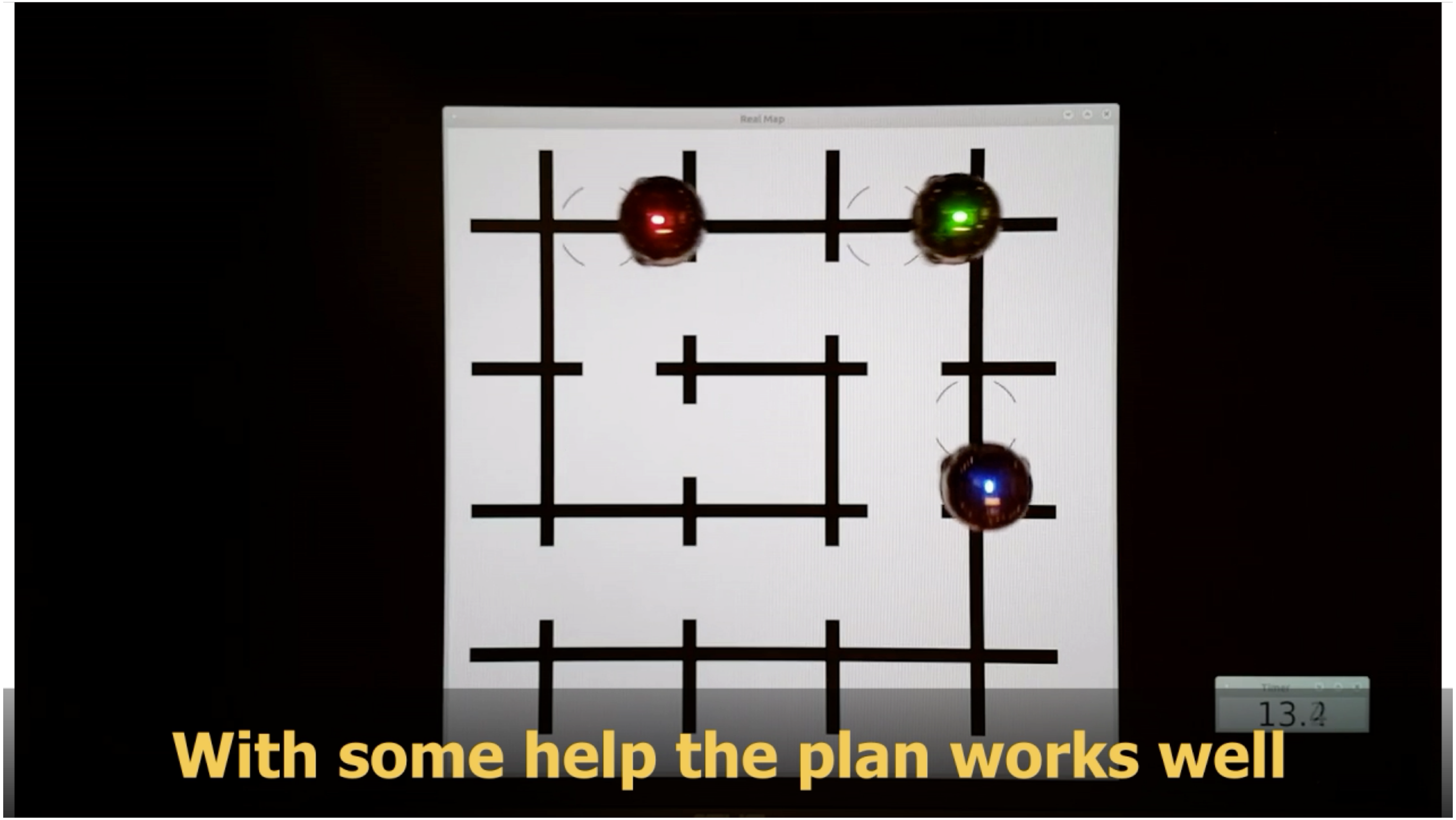
incrementally add layers and increase the cost limit until a solution is found [Surynek et al, ECAI 2016]

Part IV:

DEMOS

MAPF Scenario




Real Map
13.4
With some help the plan works well

ASPRILO, a world beyond MAPF

Philipp Obermeier[2,3] Torsten Schaub[2,3] Tran Cao Son[1,3]

[1]New Mexico State University [2]University of Potsdam [3]Potassco Solutions

Potassco

Outline

Potassco

Outline

Potassco

Motivation

- Objective How to develop robust and scalable AI technology for dealing with complex dynamic application scenarios?

- What's needed? — a fruit fly!

  Robotic intra-logistics

- Why?
  - rich multi-faceted, full of variations
  - scalable layout, objects, granularity
  - measurable makespan, energy, quality of service
  - integrative mapf, data, constraints, decisions
  - relevant industry 4.0

- What for? — enabling research and teaching

Potassco

Motivation

- Objective How to develop robust and scalable **KRR** technology for dealing with complex dynamic application scenarios?

- What's needed? — a fruit fly!

  Robotic intra-logistics

- Why?
  - rich multi-faceted, full of variations
  - scalable layout, objects, granularity
  - measurable makespan, energy, quality of service
  - integrative mapf, data, constraints, decisions
  - relevant industry 4.0

- What for? — enabling research and teaching

Potassco

Motivation

- Objective How to develop robust and scalable KRR technology for dealing with complex dynamic application scenarios?
- What's needed? — a fruit fly!

  Robotic intra-logistics

- Why?
  - rich multi-faceted, full of variations
  - scalable layout, objects, granularity
  - measurable makespan, energy, quality of service
  - integrative mapf, data, constraints, decisions
  - relevant industry 4.0
- What for? — enabling research and teaching

Potassco

Motivation

- Objective How to develop robust and scalable KRR technology for dealing with complex dynamic application scenarios?

- What's needed? — a model scenario

  Robotic intra-logistics

- Why?
  - rich multi-faceted, full of variations
  - scalable layout, objects, granularity
  - measurable makespan, energy, quality of service
  - integrative mapf, data, constraints, decisions
  - relevant industry 4.0

- What for? — enabling research and teaching

Potassco

Motivation

- Objective How to develop robust and scalable KRR technology for dealing with complex dynamic application scenarios?
- What's needed? — a model scenario

  Robotic intra-logistics
- Why?
  - rich multi-faceted, full of variations
  - scalable layout, objects, granularity
  - measurable makespan, energy, quality of service
  - integrative mapf, data, constraints, decisions
  - relevant industry 4.0
- What for? — enabling research and teaching

Potassco

Motivation

- Objective How to develop robust and scalable KRR technology for dealing with complex dynamic application scenarios?
- What's needed? — a model scenario

  Robotic intra-logistics
- Why?
  - rich multi-faceted, full of variations
  - scalable layout, objects, granularity
  - measurable makespan, energy, quality of service
  - integrative mapf, data, constraints, decisions
  - relevant industry 4.0
- What for? — enabling research and teaching

Robotic intra-logistics

- Robotics systems for logistics and warehouse automation based on many
  - mobile robots
  - movable shelves
- Main tasks: order fulfillment, i.e.
  - routing
  - order picking
  - replenishment
- Many competing industry solutions:
  - Amazon, Dematic, Genzebach, Gray Orange, Swisslog

Potassco

Robotic intra-logistics at Amazon

Robotic intra-logistics at Swisslog

What's (not) in the picture?

- Objects
  floor, robots, shelves, products, people, etc.
- Relations
  positions, carries/d, capacity, orientation, durations, etc.
- Actions
  move, pickup, putdown, pick, charge, restock, etc.
- Objectives
  deadlines, throughput, exploitation, energy management, human machine interaction, etc.

Potassco

Outline

*APF

Classified by objects, measurability, constraints, decisions

- MAPF
- TAPF
- GTAPF
- Others

A good overview of many extensions can be found in [1].

MAPF: Multi-Agent Path Finding

Most simple, straightforwards extension of APF

Objects: only robots and the map

- *anonymous*: n agents, n targets, any agent can be assigned to any target
- *non-anonymous*: n agents, n targets, each agent is assigned a (pre-defined) target

TAPF: Combined Target Assignment and Path Finding

Proposed in [2]: teams of robots

- Multiple teams of robots (objects: only robots and the map)
- Targets assigned to teams (constraint: one robot - one target)
- Collision free paths for robots to targets (no swapping), with minimal maxspan

Generalized-TAPF

Proposed in [3], inspired by online store order fulfilling requirements

- Order #1
  - "Vintage LEGO Kit" and "Programming LEGO"
  - Rush order: 2/1/2019
- Order #2
  - "Vintage LEGO Kit" and "Dancing with the Stars video"
  - International shipping

Requirements

- *Group*: an order might contain many items
- *Deadline*: each order needs to be accomplished before a timestamp
- *Checkpoint*: to fulfill certain item, some checkpoint needs to be visited

Generalized-TAPF

- Multiple teams of robots (same as TAPF).
- Sets of orders (multiple targets for an order, #robots $\neq$ #orders possible).
- Checkpoints for robots/teams (certain locations must be visited before targets).
- Deadlines for orders.
- Group completion (one order at a time).
- Collision free paths for robots to targets, with minimal maxspan.
- ASP-based solutions.

Others

Inspired by *real-word applications*, different considerations:

- Continuous vs. discrete movement
- Online vs. offline
- Checkpoints not to be (can be) revisited
- Suboptimal solutions vs. scalability
- Complex actions: transfers of items/targets between robots when pickup/putdown actions are considered
- Multi-dimensional G-TAPF: on the ground (two dimensions, cars) vs. in the air (three dimensions, drones)

Outline

Outline

Components

- Main Components
  - Standardized benchmark domains
  - Formal specification
  - Versatile instance generator
  - Visualizer for problems and (candidate) solutions
  - Solution checker with error feedback
  - Reference ASP encodings
- Resources
  - Web `potassco.org/asprilo`
  - Paper `arxiv.org/abs/1804.10247`

Components

- Main Components
  - Standardized benchmark domains
    - Concise problem specification
    - Domains ranging from MAPF to full order fulfillment
  - Formal specification
  - Versatile instance generator
  - Visualizer for problems and (candidate) solutions
  - Solution checker with error feedback
  - Reference ASP encodings
- Resources
  - Web potassco.org/asprilo
  - Paper arxiv.org/abs/1804.10247

Potassco

Components

- Main Components
  - Standardized benchmark domains
  - Formal specification
    - Formal elaboration
    - Correctness, completeness, optimality
  - Versatile instance generator
  - Visualizer for problems and (candidate) solutions
  - Solution checker with error feedback
  - Reference ASP encodings
- Resources
  - Web `potassco.org/asprilo`
  - Paper `arxiv.org/abs/1804.10247`

Potassco

Components

- Main Components
  - Standardized benchmark domains
  - Formal specification
  - Versatile instance generator
    - Rich set of customization options
    - Leverages multi-shot ASP for generation
  - Visualizer for problems and (candidate) solutions
  - Solution checker with error feedback
  - Reference ASP encodings
- Resources
  - Web `potassco.org/asprilo`
  - Paper `arxiv.org/abs/1804.10247`

Potassco

Components

- Main Components
  - Standardized benchmark domains
  - Formal specification
  - Versatile instance generator
  - Visualizer for problems and (candidate) solutions
    - Animated playback of plans
    - Graphical editor for instances
  - Solution checker with error feedback
  - Reference ASP encodings
- Resources
  - Web potassco.org/asprilo
  - Paper arxiv.org/abs/1804.10247

Potassco

Components

- Main Components
  - Standardized benchmark domains
  - Formal specification
  - Versatile instance generator
  - Visualizer for problems and (candidate) solutions
  - Solution checker with error feedback
    - Specific error descriptions
    - Modular design, easily extensible
  - Reference ASP encodings
- Resources
  - Web `potassco.org/asprilo`
  - Paper `arxiv.org/abs/1804.10247`

Potassco

Components

- Main Components
  - Standardized benchmark domains
  - Formal specification
  - Versatile instance generator
  - Visualizer for problems and (candidate) solutions
  - Solution checker with error feedback
  - Reference ASP encodings
    - High-level, elaboration-tolerant
    - Test bed for ASP and KRR technology
- Resources
  - Web `potassco.org/asprilo`
  - Paper `arxiv.org/abs/1804.10247`

Installation

1. Install the conda package manager
2. Create a custom conda environment that provides python 3.6
3. Activate your conda environment
4. Install clingo in your environment
5. Install asprilo's instance generator and visualizer in your environment

Installation

1. Install the conda package manager

   ⇝ available at `https://www.anaconda.com`

   - anaconda: many python packages including conda
   - miniconda: only the conda package manager

2. Create a custom conda environment that provides python 3.6
3. Activate your conda environment
4. Install clingo in your environment
5. Install asprilo's instance generator and visualizer in your environment

Potassco

Installation

1. Install the conda package manager

   ⇝ available at `https://www.anaconda.com`

   - anaconda: many python packages including conda
   - miniconda: only the conda package manager

2. Create a custom conda environment that provides python 3.6
3. Activate your conda environment
4. Install clingo in your environment
5. Install asprilo's instance generator and visualizer in your environment

Potassco

Installation

1. Install the conda package manager

   ⇝ available at `https://www.anaconda.com`

   - anaconda: many python packages including conda
   - miniconda: only the conda package manager

2. Create a custom conda environment that provides python 3.6
3. Activate your conda environment
4. Install clingo in your environment
5. Install asprilo's instance generator and visualizer in your environment

Installation

1. Install the conda package manager
2. Create a custom conda environment that provides python 3.6
3. Activate your conda environment
4. Install clingo in your environment
5. Install asprilo's instance generator and visualizer in your environment

Potassco

Installation

1. Install the conda package manager
2. Create a custom conda environment that provides python 3.6

```
$ conda create -n myenv python=3.6
```

3. Activate your conda environment
4. Install clingo in your environment
5. Install asprilo's instance generator and visualizer in your environment

Installation

1. Install the conda package manager
2. Create a custom conda environment that provides python 3.6

```
$ conda create -n myenv python=3.6
```

3. Activate your conda environment
4. Install clingo in your environment
5. Install asprilo's instance generator and visualizer in your environment

Installation

1. Install the conda package manager
2. Create a custom conda environment that provides python 3.6

```
$ conda create -n myenv python=3.6
```

3. Activate your conda environment

```
$ conda activate myenv
```

4. Install clingo in your environment
5. Install asprilo's instance generator and visualizer in your environment

Potassco

Installation

1. Install the conda package manager
2. Create a custom conda environment that provides python 3.6

```
$ conda create -n myenv python=3.6
```

3. Activate your conda environment

```
$ conda activate myenv
```

4. Install clingo in your environment
5. Install asprilo's instance generator and visualizer in your environment

Installation

1. Install the conda package manager
2. Create a custom conda environment that provides python 3.6

```
$ conda create -n myenv python=3.6
```

3. Activate your conda environment

```
$ conda activate myenv
```

4. Install clingo in your environment

```
$ conda install -c potassco clingo
```

5. Install asprilo's instance generator and visualizer in your environment

Installation

1. Install the conda package manager
2. Create a custom conda environment that provides python 3.6

```
$ conda create -n myenv python=3.6
```

3. Activate your conda environment

```
$ conda activate myenv
```

4. Install clingo in your environment

```
$ conda install -c potassco clingo
```

5. Install asprilo's instance generator and visualizer in your environment

Potassco

Installation

1. Install the conda package manager
2. Create a custom conda environment that provides python 3.6

```
$ conda create -n myenv python=3.6
```

3. Activate your conda environment

```
$ conda activate myenv
```

4. Install clingo in your environment

```
$ conda install -c potassco clingo
```

5. Install asprilo's instance generator and visualizer in your environment

```
$ conda install -c asprilo generator visualizer
```

Installation

1. Install the conda package manager
2. Create a custom conda environment that provides python 3.6

```
$ conda create -n myenv python=3.6
```

3. Activate your conda environment

```
$ conda activate myenv
```

4. Install clingo in your environment

```
$ conda install -c potassco clingo
```

5. Install asprilo's instance generator and visualizer in your environment

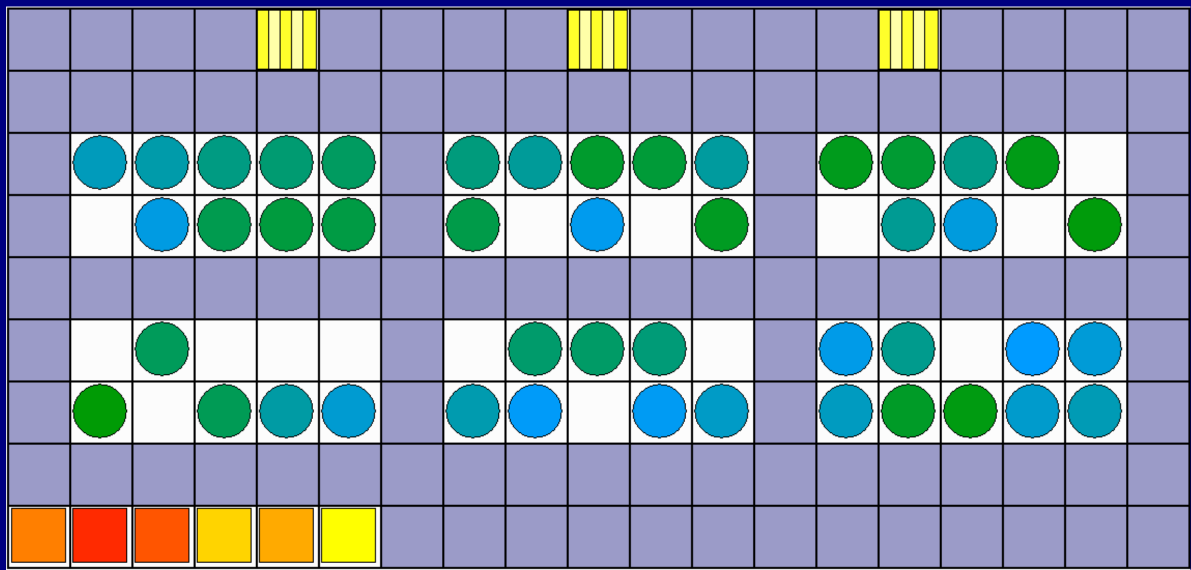
```
$ conda install -c asprilo generator visualizer
```

DONE!

Potassco

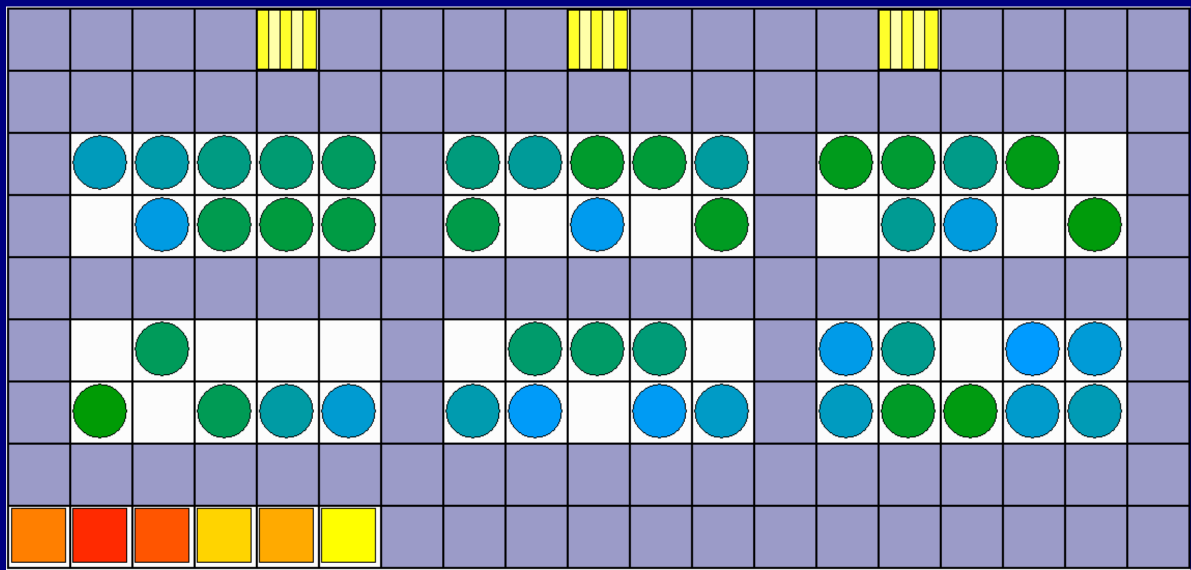
General Domain A

- The warehouse is laid out as a (partial) 2-dimensional grid
- Shelves store products in a certain quantity, each shelf occupies a single grid node
- Mobile robots move and navigate through the warehouse along the grid, can carry shelves and deliver product units to picking stations



General Domain A

- Highway nodes are special grid nodes where robots must never put down a shelf
- A set of orders is initially provided, an order is fulfilled if all its requested product units are delivered to its assigned picking station
- Main Goal: plan robot actions such that all orders will be fulfilled



Domain A Demo

Domains A, B, C, M

Domain A	most general domain
Domain B	ignores product quantities
Domain C	ignores product quantities delivery actions at once
Domain M	only move actions singleton orders and shelves reach shelves with ordered products

Domains A, B, C, M

Domain A most general domain

Domain B ignores product quantities

Domain C ignores product quantities
delivery actions at once

Domain M only move actions
singleton orders and shelves
reach shelves with ordered products

Potassco

Domains A, B, C, M

Domain A most general domain

Domain B ignores product quantities

Domain C ignores product quantities
delivery actions at once

Domain M only move actions
singleton orders and shelves
reach shelves with ordered products

Potassco

Domains A, B, C, M

Domain A	most general domain
Domain B	ignores product quantities
Domain C	ignores product quantities delivery actions at once
Domain M	only move actions singleton orders and shelves reach shelves with ordered products

Potassco

Domain M Demo

Instance format

- Fact format

`init(object(T,I),value(A,V)).`

  where
  - *T* is an object type
  - *I* a (relative) object identifier
  - *A* an attribute
  - *V* its value
- Object types and their attributes

node	at/2
highway	at/2
robot	at/2, carries/1
shelf	at/2
pickingStation	at/2
product	on/2
order	line/2, pickingStation/1

Potassco

Instance format

- Fact format

  `init(object(`T`,`I`),value(`A`,`V`)).`

- Object types and their attributes

`node`	`at/2`
`highway`	`at/2`
`robot`	`at/2`, `carries/1`
`shelf`	`at/2`
`pickingStation`	`at/2`
`product`	`on/2`
`order`	`line/2`, `pickingStation/1`

- Example robot 34 is at position (2,3).

  `init(object(robot,34),value(at,(2,3))).`

Potassco

Instance format

- Fact format

  `init(object(`T`,`I`),value(`A`,`V`)).`

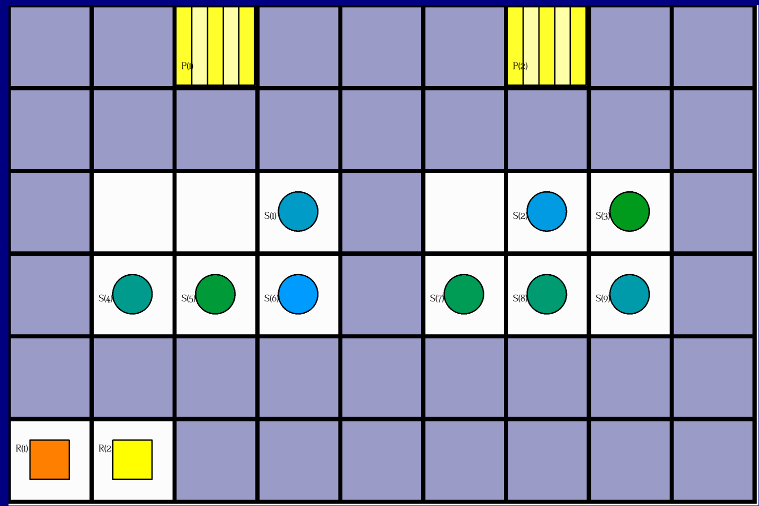
- Object types and their attributes

`node`	`at/2`
`highway`	`at/2`
`robot`	`at/2`, `carries/1`
`shelf`	`at/2`
`pickingStation`	`at/2`
`product`	`on/2`
`order`	`line/2`, `pickingStation/1`

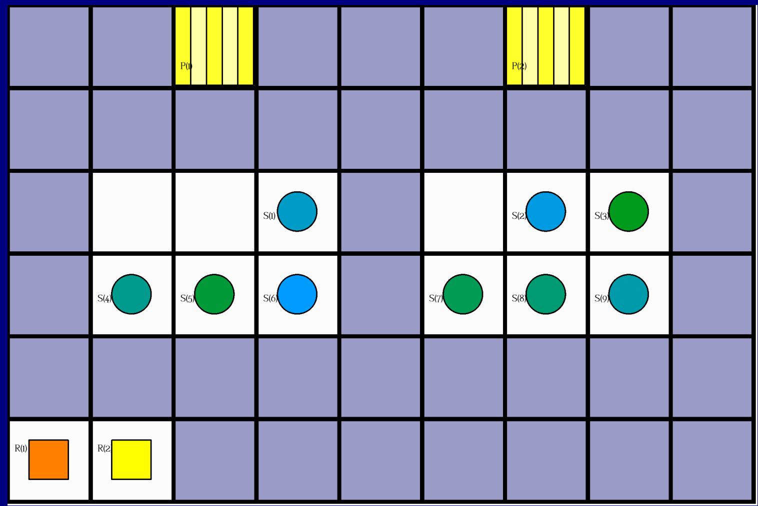
- Example robot 34 is at position (2,3).

  `init(object(robot,34),value(at,(2,3))).`

Example



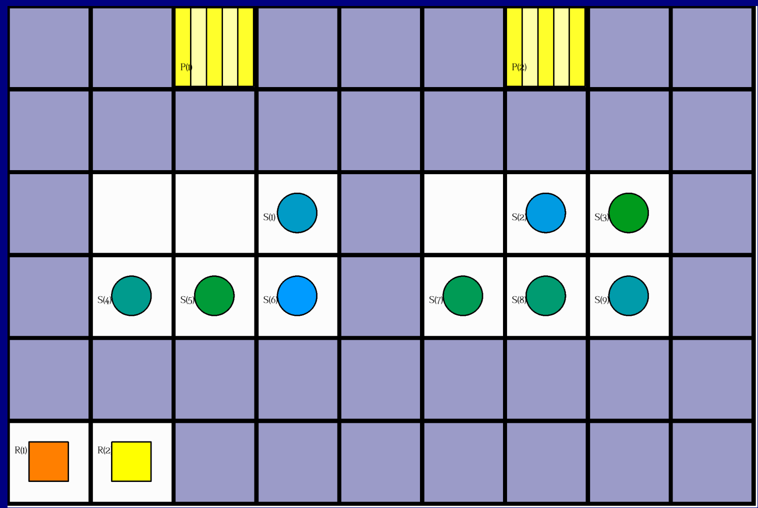
Example




```
init(object(node, 1), value(at, (1,1))).
init(object(highway, 1), value(at, (1,1))).
```

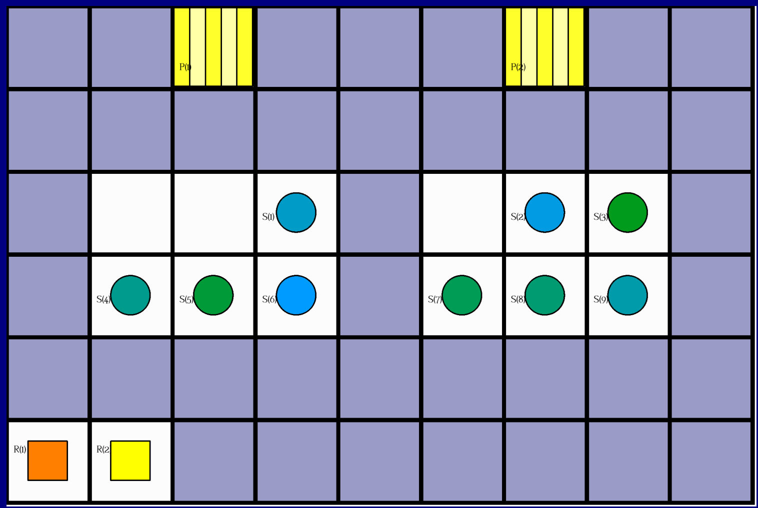
Potassco

Example



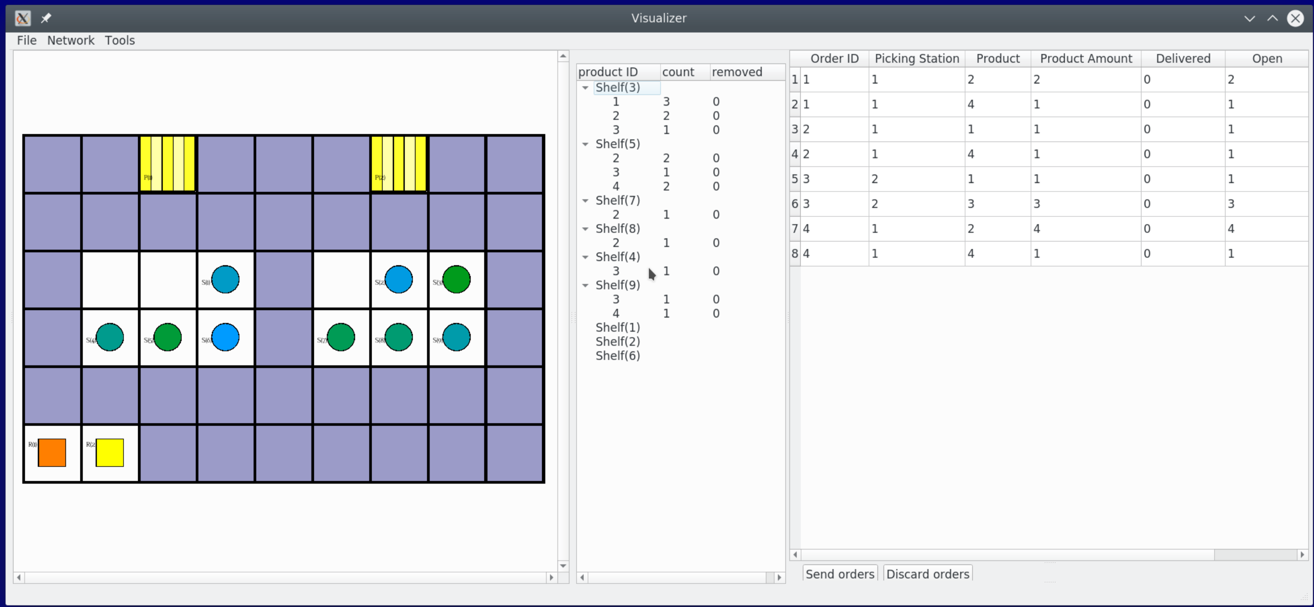
```
init(object(node, 7), value(at, (7,1))).
init(object(pickingStation,2), value(at,(7,1))).
```

Example

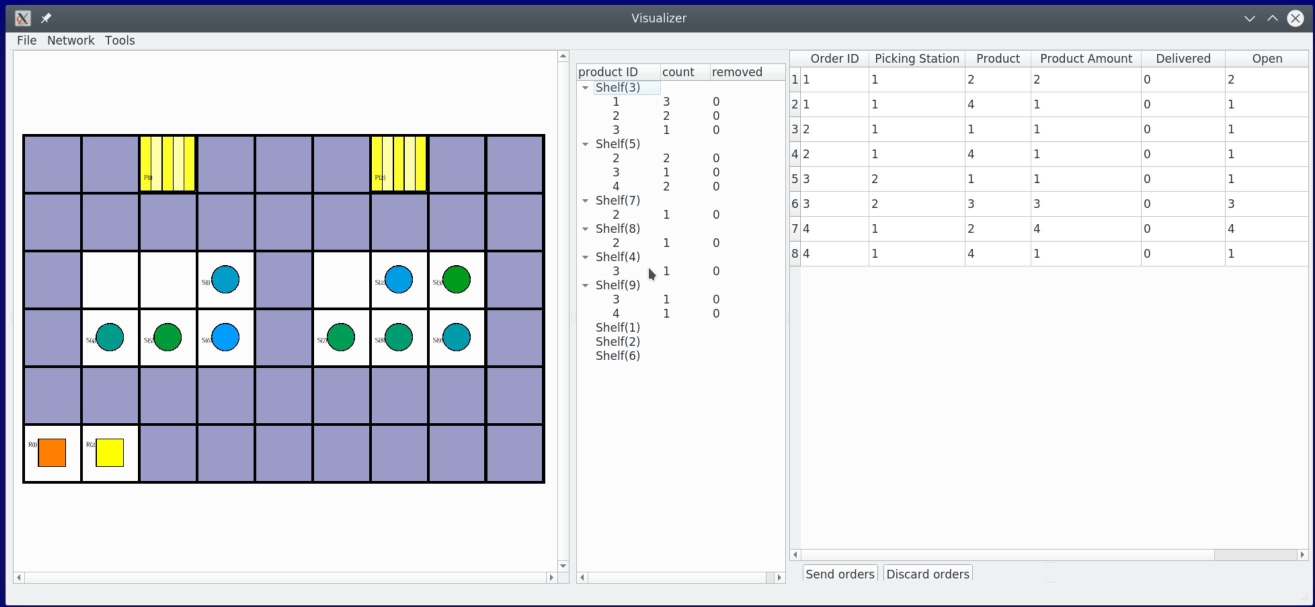


```
init(object(node, 47), value(at, (2,6))).
init(object(robot,2), value(at, (2,6))).
init(object(robot,2), value(max_energy,0)).
init(object(robot,2), value(energy,0)).
```

Example

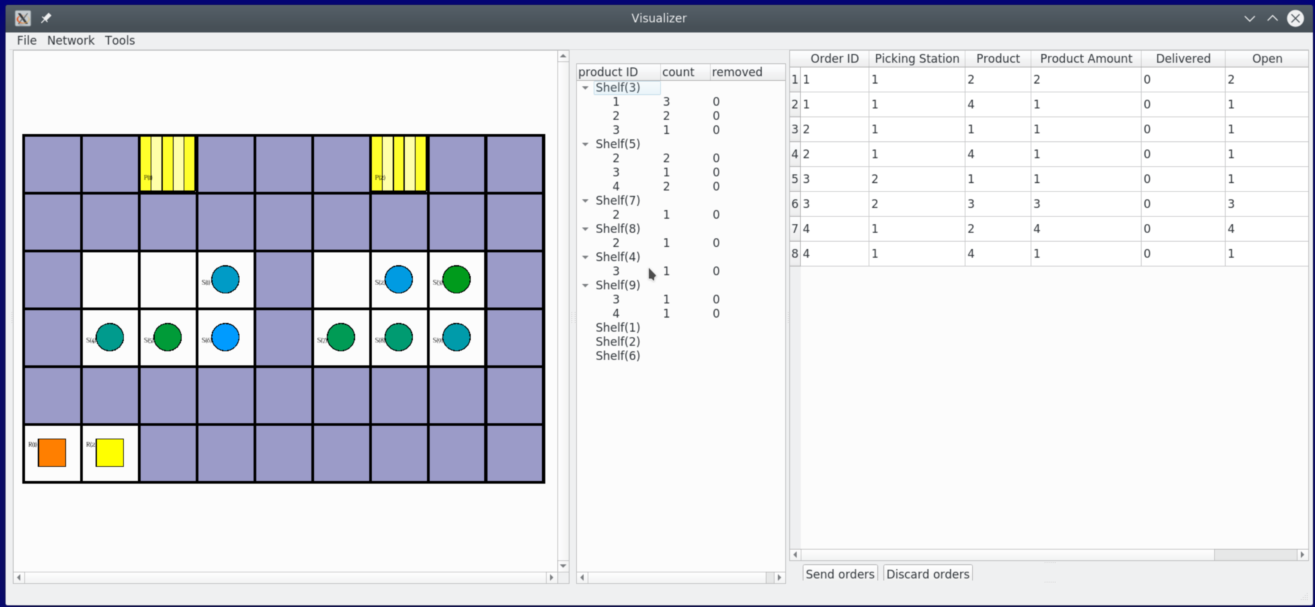

Visualizer
File Network Tools
product ID count removed
Shelf(3)
1 3 0
2 2 0
3 1 0
Shelf(5)
2 2 0
3 1 0
4 2 0
Shelf(7)
2 1 0
Shelf(8)
2 1 0
Shelf(4)
3 1 0
Shelf(9)
3 1 0
4 1 0
Shelf(1)
Shelf(2)
Shelf(6)
Order ID Picking Station Product Product Amount Delivered Open
1 1 1 2 2 0 2
2 1 1 4 1 0 1
3 2 1 1 1 0 1
4 2 1 4 1 0 1
5 3 2 1 1 0 1
6 3 2 3 3 0 3
7 4 1 2 4 0 4
8 4 1 4 1 0 1
Send orders
Discard orders

Example




```
init(object(node, 26), value(at, (8,3))).
init(object(shelf,3), value(at,(8,3))).
```

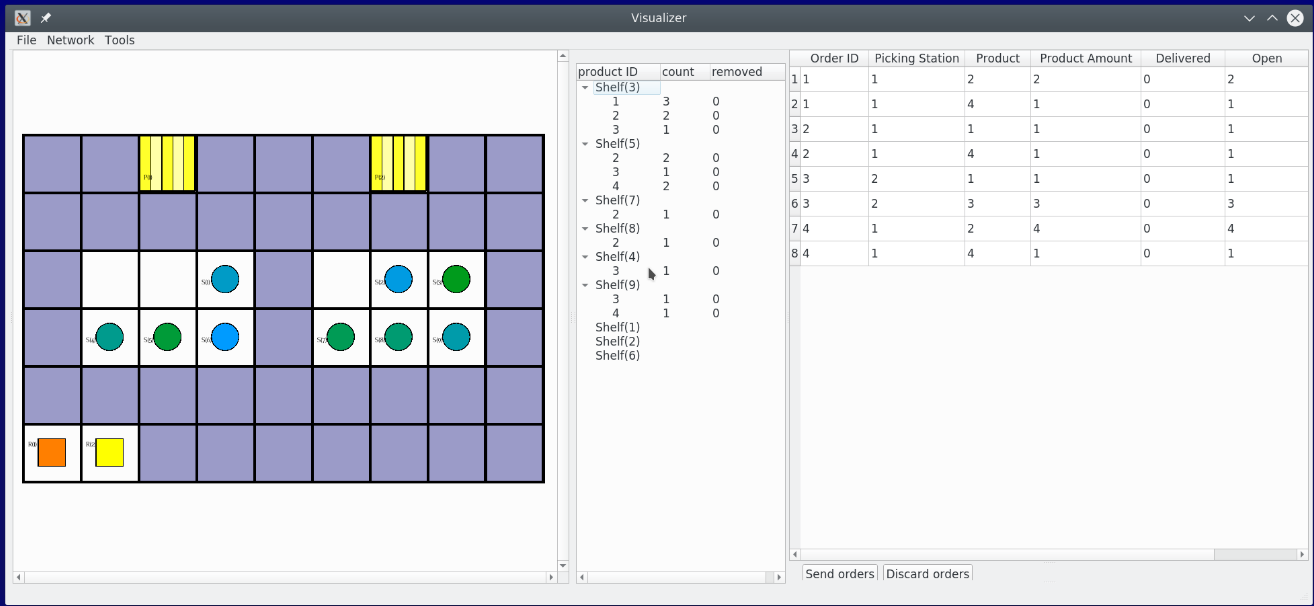
Example




```
init(object(node, 26), value(at, (8,3))).
init(object(shelf,3), value(at,(8,3))).
init(object(product,1),value(on,(3,3))).
init(object(product,2),value(on,(3,2))).
init(object(product,3),value(on,(3,1))).
```

Potassco

Example




```
init(object(order,3),value(pickingStation,2)).
init(object(order,3),value(line,(1,1))).
init(object(order,3),value(line,(3,3))).
```

Solution format

- Solution (candidate) is a parallel plan for multiple robots
- Fact format

  `occurs(object(`T`,`I`),action(`A`,`V`)).`

- Actions
  - `move` takes cardinal points `(0,1)`, `(1,0)`, `(0,-1)`, and `(-1,0)`
  - `pickup` has no arguments, viz. `()`.
  - `putdown` has no arguments, viz. `()`.
  - `deliver` takes a triple `(`O`,`A`,`N`)` where O is an order, A a product, and N its quantity

Solution format

- Solution (candidate) is a parallel plan for multiple robots
- Fact format

```
occurs(object(T,I),action(A,V)).
```

where

- T is an object type
- I an object identifier
- A an action name, and
- V a tuple of terms capturing the action's arguments

- Actions
  - `move` takes cardinal points `(0,1)`, `(1,0)`, `(0,-1)`, and `(-1,0)`
  - `pickup` has no arguments, viz. `()`.
  - `putdown` has no arguments, viz. `()`.
  - `deliver` takes a triple `(`O`,`A`,`N`)` where O is an order, A a product, and N its quantity

Potassco

Solution format

- Solution (candidate) is a parallel plan for multiple robots
- Fact format

```
occurs(object(T,I),action(A,V)).
```

- Actions
  - `move` takes cardinal points `(0,1)`, `(1,0)`, `(0,-1)`, and `(-1,0)`
  - `pickup` has no arguments, viz. `()`.
  - `putdown` has no arguments, viz. `()`.
  - `deliver` takes a triple `(O,A,N)` where
    O is an order, A a product, and N its quantity

Example

Outline

Preliminaries

- Multi sets
  - multi-set operations $\dot\cup, \dot\cap, \dot\subseteq, \dot| \cdot \dot| \dots$
  - $(t_1, \dots, t_n, n)$ stands for n occurrence of $(t_1, \dots, t_n)$ in a multi-set
- Relational algebra
  - projection $\pi_i R$
  - selection $\sigma_{i=t} R$
- Planning
  - Actions
  - Fluents

Warehouse layout

- Layout A quadruple (G, R, S, F) where
  - $G = (V, E)$ is a labeled directed graph where
    - $\nu : V \rightarrow \{high, pick, star, park\}$
    - $\epsilon : E \rightarrow \mathbb{N}$
  - R is a set of robots
  - S is a set of shelves
  - $F = (position_0, position_0, carries_0)$ is a triple of functions
    - $position_0 : R \rightarrow V$ is an injective function
    - $position_0 : S \rightarrow V$ is an injective function
    - $carries_0 : R \rightarrow \{\{s\} \mid s \in S\} \cup \{\emptyset\}$ is a function injective on all non-empty functional values

Actions

- Layout $((V, E), R, S, F)$
- Action atoms
  - $\mathit{move}(r, e)$ for each $(r, e) \in R \times E$
  - $\mathit{pickup}(r, s)$ for each $(r, s) \in R \times S$
  - $\mathit{putdown}(r, s)$ for each $(r, s) \in R \times S$
- Notation
  - $M(R, E) = \{ \mathit{move}(r, e) \mid (r, e) \in R \times E \}$
  - $P(R, S) = \{ \mathit{pickup}(r, s), \mathit{putdown}(r, s) \mid (r, s) \in R \times S \}$
- Abuse We write $M(r, E)$ instead of $M(\{r\}, E)$

Actions

- Layout $((V, E), R, S, F)$
- Action atoms
  - $\mathit{move}(r, e)$ for each $(r, e) \in R \times E$
  - $\mathit{pickup}(r, s)$ for each $(r, s) \in R \times S$
  - $\mathit{putdown}(r, s)$ for each $(r, s) \in R \times S$
- Notation
  - $M(R, E) = \{ \mathit{move}(r, e) \mid (r, e) \in R \times E \}$
  - $P(R, S) = \{ \mathit{pickup}(r, s), \ \mathit{putdown}(r, s) \mid (r, s) \in R \times S \}$
- Abuse We write $M(r, E)$ instead of $M(\{r\}, E)$

Fluents

- Layout $((V,E),R,S,F)$
- Time points T
- Fluent functions for each $t \in T$
  - $position_t : R \to V$ is an injective function
  - $position_t : S \to V$ is an injective function
  - $carries_t : R \to \{\{s\} \mid s \in S\} \cup \{\emptyset\}$ is a function
- Evolution A sequence $(position_t, position_t, carries_t)_{t \in T}$ is called a (warehouse) evolution

Trajectory

- Layout $((V, E), R, S, (position_0, position_0, carries_0))$
- Trajectory A sequence

$$\langle A_t \rangle_{t \in T} \text{ of sets } A_t \subseteq M(R, E) \cup P(R, S)$$

along with an evolution $(position_t, position_t, carries_t)_{t \in T}$
such that for all $t \in T, r \in R, s \in S, (u, v) \in E$:

Trajectory

- Layout $((V, E), R, S, (position_0, position_0, carries_0))$
- Trajectory A sequence

$$\langle A_t \rangle_{t \in T} \text{ of sets } A_t \subseteq M(R, E) \cup P(R, S)$$

along with an evolution $(position_t, position_t, carries_t)_{t \in T}$ such that for all $t \in T, r \in R, s \in S, (u, v) \in E$:

Trajectory

- Layout $((V, E), R, S, (position_0, position_0, carries_0))$
- Trajectory A sequence

$$\langle A_t \rangle_{t \in T} \text{ of sets } A_t \subseteq M(R, E) \cup P(R, S)$$

along with an evolution $(position_t, position_t, carries_t)_{t \in T}$ such that for all $t \in T, r \in R, s \in S, (u, v) \in E$:

1. $|A_t \cap (M(r, E) \cup P(r, S))| \leq 1$

Trajectory

- Layout $((V,E),R,S,(position_0,position_0,carries_0))$
- Trajectory A sequence

$$\langle A_t\rangle_{t\in T} \text{ of sets } A_t \subseteq M(R,E)\cup P(R,S)$$

along with an evolution $(position_t,position_t,carries_t)_{t\in T}$
such that for all $t\in T, r\in R, s\in S, (u,v)\in E$:

1. $|A_t \cap (M(r,E)\cup P(r,S))| \leq 1$
2. $move(r,(u,v)) \in A_t$ only if $position_{t-1}(r)=u$

Potassco

Trajectory

- Layout $((V, E), R, S, (position_0, position_0, carries_0))$
- Trajectory A sequence

$$\langle A_t \rangle_{t \in T} \text{ of sets } A_t \subseteq M(R, E) \cup P(R, S)$$

along with an evolution $(position_t, position_t, carries_t)_{t \in T}$
such that for all $t \in T, r \in R, s \in S, (u, v) \in E$:

1. $|A_t \cap (M(r, E) \cup P(r, S))| \leq 1$
2. $move(r, (u, v)) \in A_t$ only if $position_{t-1}(r) = u$
3. $\{move(r, (u, v)), move(r', (v, u)) \mid r \neq r' \in R, (v, u) \in E\} \not\subseteq A_t$

Potassco

Trajectory

- Layout $((V,E),R,S,(position_0,position_0,carries_0))$
- Trajectory A sequence

$$\langle A_t \rangle_{t\in T} \text{ of sets } A_t \subseteq M(R,E)\cup P(R,S)$$

along with an evolution $(position_t, position_t, carries_t)_{t\in T}$ such that for all $t\in T, r\in R, s\in S, (u,v)\in E$:

1. $|A_t \cap (M(r,E)\cup P(r,S))| \leq 1$
2. $move(r,(u,v)) \in A_t$ only if $position_{t-1}(r) = u$
3. $\{move(r,(u,v)), move(r',(v,u)) \mid r\neq r' \in R, (v,u)\in E\} \not\subseteq A_t$
4. $pickup(r,s)\in A_t$ only if
   1. $position_{t-1}(r) = position_{t-1}(s)$
   2. $carries_{t-1}(r) = \emptyset$
   3. $s \notin carries_{t-1}(r')$ for all $r'\in R$

Trajectory

- Layout $((V,E),R,S,(position_0,position_0,carries_0))$
- Trajectory A sequence

$$\langle A_t\rangle_{t\in T} \text{ of sets } A_t \subseteq M(R,E)\cup P(R,S)$$

along with an evolution $(position_t, position_t, carries_t)_{t\in T}$
such that for all $t\in T, r\in R, s\in S, (u,v)\in E$:

1. $|A_t\cap(M(r,E)\cup P(r,S))|\leq 1$
2. $move(r,(u,v))\in A_t$ only if $position_{t-1}(r)=u$
3. $\{move(r,(u,v)), move(r',(v,u)) \mid r\neq r'\in R, (v,u)\in E\}\not\subseteq A_t$
4. $pickup(r,s)\in A_t$ only if
    1. $position_{t-1}(r)=position_{t-1}(s)$
    2. $carries_{t-1}(r)=\emptyset$
    3. $s\notin carries_{t-1}(r')$ for all $r'\in R$
5. $putdown(r,s)\in A_t$ only if $s\in carries_{t-1}(r)$

Trajectory

- Layout $((V, E), R, S, (position_0, position_0, carries_0))$
- Trajectory A sequence

$$\langle A_t \rangle_{t \in T} \text{ of sets } A_t \subseteq M(R, E) \cup P(R, S)$$

along with an evolution $(position_t, position_t, carries_t)_{t \in T}$
such that for all $t \in T, r \in R, s \in S, (u, v) \in E$:

6 $$position_t(r) = \begin{cases} v & \text{if } move(r, (u, v)) \in A_t \\ position_{t-1}(r) & \text{otherwise} \end{cases}$$

7 $$position_t(s) = \begin{cases} position_t(r) & \text{if } s \in carries_t(r) \\ position_{t-1}(s) & \text{otherwise} \end{cases}$$

8 $$carries_t(r) = \begin{cases} carries_{t-1}(r) \cup \{s\} & \text{if } pickup(r, s) \in A_t \\ carries_{t-1}(r) \setminus \{s\} & \text{if } putdown(r, s) \in A_t \\ carries_{t-1}(r) & \text{otherwise} \end{cases}$$

Potassco

Orders, products, inventories

- Products P
- Orders O
  $destination : O \rightarrow \{v \in V \mid \nu(v) = pick\}$
- Order line $(o, p, n) \in O \times P \times \mathbb{N}$,
  the request of $n \in \mathbb{N}$ products $p \in P$ by order $o \in O$

  $I \subseteq S \times P \times \mathbb{N}$,
  a relation reflecting the in-stock quantity per shelf and product

Order line sets and inventories are functional in their last argument

Order line sets and inventories are manipulated with multi-set operations

Orders, products, inventories

- Products P
- Orders O
  $destination : O \to \{v \in V \mid \nu(v) = pick\}$
- Order line $(o, p, n) \in O \times P \times \mathbb{N}$,
  the request of $n \in \mathbb{N}$ products $p \in P$ by order $o \in O$
- Inventory $I \subseteq S \times P \times \mathbb{N}$,
  a relation reflecting the in-stock quantity per shelf and product
- Requirement
  Order line sets and inventories are functional in their last argument
- Note
  Order line sets and inventories are manipulated with multi-set operations

Potassco

Orders, products, inventories

- Products P
- Orders O
  $destination : O \to \{v \in V \mid \nu(v) = pick\}$
- Order line $(o, p, n) \in O \times P \times \mathbb{N}$,
  the request of $n \in \mathbb{N}$ products $p \in P$ by order $o \in O$
- Inventory $I \subseteq S \times P \times \mathbb{N}$,
  a relation reflecting the in-stock quantity per shelf and product
- Requirement
  Order line sets and inventories are functional in their last argument
- Note
  Order line sets and inventories are manipulated with multi-set operations

Orders, products, inventories

- Products P
- Orders O
  $destination : O \to \{v \in V \mid \nu(v) = pick\}$
- Order line $(o, p, n) \in O \times P \times \mathbb{N}$,
  the request of $n \in \mathbb{N}$ products $p \in P$ by order $o \in O$
- Inventory $I \subseteq S \times P \times \mathbb{N}$,
  a relation reflecting the in-stock quantity per shelf and product
- Requirement
  Order line sets and inventories are functional in their last argument
- Note
  Order line sets and inventories are manipulated with multi-set operations

Potassco

Orders, products, inventories

- Products P
- Orders O
  $destination : O \to \{v \in V \mid \nu(v) = pick\}$
- Order line $(o, p, n) \in O \times P \times \mathbb{N}$,
  the request of $n \in \mathbb{N}$ products $p \in P$ by order $o \in O$
- Inventory $I \subseteq S \times P \times \mathbb{N}$,
  a relation reflecting the in-stock quantity per shelf and product
- Requirement
  Order line sets and inventories are functional in their last argument
- Note
  Order line sets and inventories are manipulated with multi-set operations

Pick set

- Situation
  A set L of order lines, an inventory I, a shelf $s \in S$, and a picking rate n
- Pick set A set Q of order lines such that
  1. $Q \dot{\subseteq} L$
  2. $\pi_{2,3} Q \dot{\subseteq} \pi_{2,3} \sigma_{1=s} I$
  3. $\dot{|} Q \dot{|} \leq n$
  4. $\{destination(o) \mid o \in \pi_1 Q\} = \{v\}$ for some $v \in V$
- Note A pick set may be empty
- Notation $destination(Q) = v$ denotes the unique destination v of all order lines in pick set Q (cf. Item 4 above)

Pick set

- Situation
  A set L of order lines, an inventory I, a shelf $s \in S$, and a picking rate n
- Pick set A set Q of order lines such that
  1. $Q \dot{\subseteq} L$
  2. $\pi_{2,3} Q \dot{\subseteq} \pi_{2,3} \sigma_{1=s} I$
  3. $|\dot{Q}| \leq n$
  4. $\{destination(o) \mid o \in \pi_1 Q\} = \{v\}$ for some $v \in V$
- Note A pick set may be empty
- Notation $destination(Q) = v$ denotes the unique destination v of all order lines in pick set Q (cf. Item 4 above)

Pick set

- Situation
  A set L of order lines, an inventory I, a shelf $s \in S$, and a picking rate n
- Pick set A set Q of order lines such that
  1. $Q \dot{\subseteq} L$
  2. $\pi_{2,3} Q \dot{\subseteq} \pi_{2,3} \sigma_{1=s} I$
  3. $\dot{|} Q \dot{|} \leq n$
  4. $\{destination(o) \mid o \in \pi_1 Q\} = \{v\}$ for some $v \in V$
- Note A pick set may be empty
- Notation $destination(Q) = v$ denotes the unique destination v of all order lines in pick set Q (cf. Item 4 above)

Pick set

- Situation
  A set L of order lines, an inventory I, a shelf $s \in S$, and a picking rate n
- Pick set A set Q of order lines such that
  1. $Q \dot{\subseteq} L$
  2. $\pi_{2,3} Q \dot{\subseteq} \pi_{2,3} \sigma_{1=s} I$
  3. $\dot{|} Q \dot{|} \leq n$
  4. $\{destination(o) \mid o \in \pi_1 Q\} = \{v\}$ for some $v \in V$
- Note A pick set may be empty
- Notation $destination(Q) = v$ denotes the unique destination v of all order lines in pick set Q (cf. Item 4 above)

Picking sequence

- Situation
  A set L of order lines, an inventory I, a shelf $s \in S$, and a picking rate n
- Picking sequence $\langle Q_t \rangle_{t \in T}$ consists of
  pick sets Q_t for (L_{t-1}, I_{t-1}) at shelf s where
  1. $\langle L_0, I_0 \rangle = \langle L, I \rangle$
  2. $\langle L_t, I_t \rangle = \langle L_{t-1} \mathbin{\dot\setminus} Q_t, I_{t-1} \mathbin{\dot\setminus} (\{s\} \times \pi_{2,3} Q_t) \rangle$
- Note A picking sequence may contain empty pick sets
- Success A picking sequence $\langle L_t, I_t \rangle_{t \in T}$ is successful,
  if there is some $i \geq 0$ such that $L_k = \emptyset$ for all $k \geq i$;
  we refer to I_i as the inventory resulting from applying $\langle L_t, I_t \rangle_{t \in T}$ to I

Picking sequence

- Situation
  A set L of order lines, an inventory I, a shelf $s \in S$, and a picking rate n
- Picking sequence $\langle Q_t \rangle_{t \in T}$ consists of
  pick sets Q_t for (L_{t-1}, I_{t-1}) at shelf s where
  1. $\langle L_0, I_0 \rangle = \langle L, I \rangle$
  2. $\langle L_t, I_t \rangle = \langle L_{t-1} \dot{\setminus} Q_t, I_{t-1} \dot{\setminus} (\{s\} \times \pi_{2,3} Q_t) \rangle$
- Note A picking sequence may contain empty pick sets
- Success A picking sequence $\langle L_t, I_t \rangle_{t \in T}$ is successful,
  if there is some $i \geq 0$ such that $L_k = \emptyset$ for all $k \geq i$;
  we refer to I_i as the inventory resulting from applying $\langle L_t, I_t \rangle_{t \in T}$ to I

Potassco

Picking sequence

- Situation
  A set L of order lines, an inventory I, a shelf $s \in S$, and a picking rate n
- Picking sequence $\langle Q_t \rangle_{t \in T}$ consists of
  pick sets Q_t for (L_{t-1}, I_{t-1}) at shelf s where
  1. $\langle L_0, I_0 \rangle = \langle L, I \rangle$
  2. $\langle L_t, I_t \rangle = \langle L_{t-1} \mathbin{\dot{\setminus}} Q_t, I_{t-1} \mathbin{\dot{\setminus}} (\{s\} \times \pi_{2,3} Q_t) \rangle$
- Note A picking sequence may contain empty pick sets
- Success A picking sequence $\langle L_t, I_t \rangle_{t \in T}$ is successful,
  if there is some $i \geq 0$ such that $L_k = \emptyset$ for all $k \geq i$;
  we refer to I_i as the inventory resulting from applying $\langle L_t, I_t \rangle_{t \in T}$ to I

Potassco

Fulfillment

- Fulfilling trajectory

  A trajectory $\langle A_t \rangle_{t \in T}$ valid in layout $((V, E), R, S, F)$ fulfills a set L of order lines at rate n given inventory I, whenever there is
  - a (shelf-wise) partition $(L^s)_{s \in S}$ of L,
  - a collection of successful picking sequences $(Q^s_t)_{t \in T}$ for each $(L^s, \sigma_{1=s} I)$ at $s \in S$ and rate n, and
  - a robot $r \in R$

  such that

  $$Q^s_t \neq \emptyset$$

  only if
  1. $A_t \cap (M(r, E) \cup P(r, S)) = \emptyset$
  2. $destination(Q^s_t) = position_t(r)$
  3. $s \in carries_t(r)$

Potassco

Outline

Potassco

Motivation, Features

- Tool to automatically generate asprilo instances
- Rich set of options to configure key instance characteristics
  - grid dimensions; number of shelves, robots, orders, products, etc.
  - grid type: random, structured, custom (via template)
  - supported domain: singleton product sets per shelf and order for M-domain; or A,B,C-compatible
- Implemented via multi-shot ASP
  - modular ASP program design: key attributes separately controlled by dedicated program parts
  - easily extensible with new program parts and, hence, features

Generating a Structured Instance

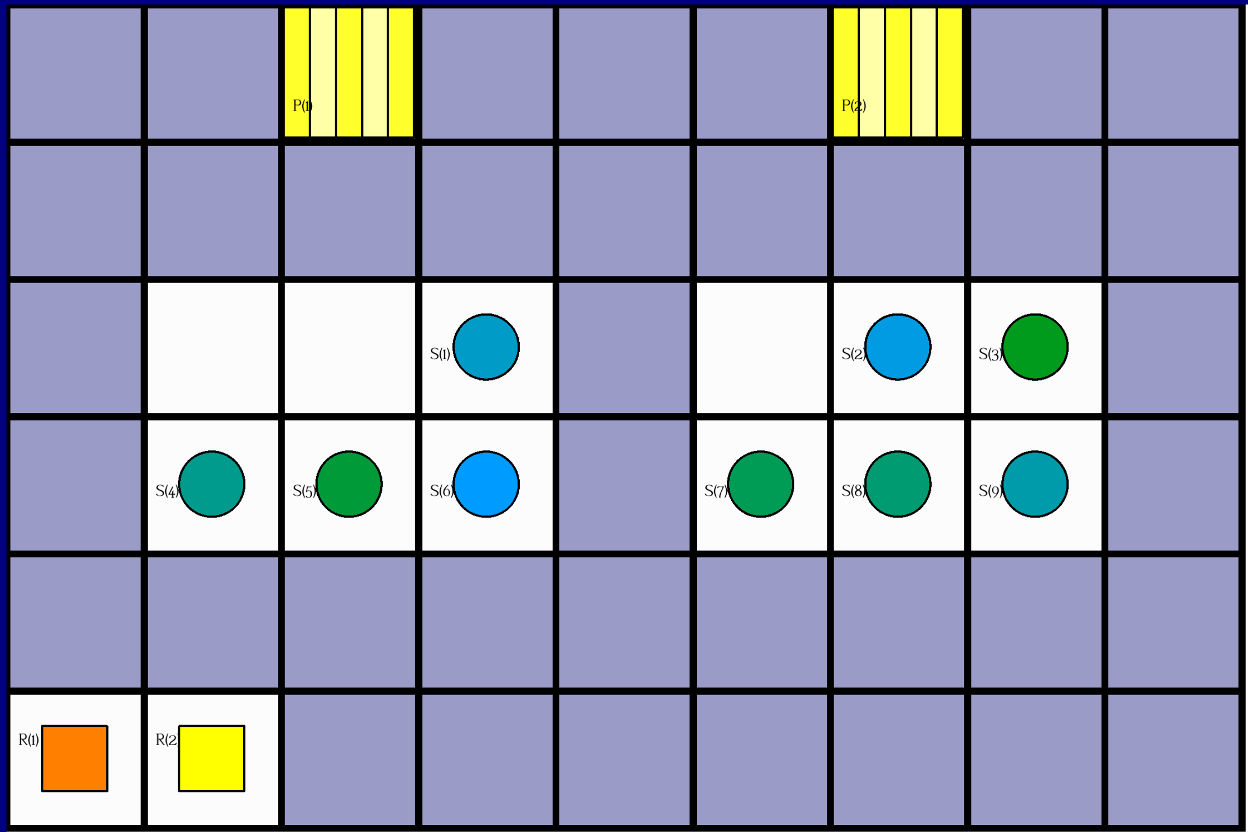
Structured instance with:

- 9x6 grid: `-x 9 -y 6`
- 3x2 storage zones: `-X 3 -Y 2`
- 9 shelves, 2 pick stations, 2 robots: `-s 9 -p 2 -r2`
- 4 products and 16 product units: `-P 4 -u 16`
- 4 orders, 2 order lines each, all products requested at least once: `-o 4 --ol 2 --oap`
- structured layout: `-H`

via

```
gen -x 9 -y 6 -X 3 -Y 2 -s 9 -p 2 -r 2 -P 4 -u 16 -o 4 \
    --ol 2 --oap  -H
```

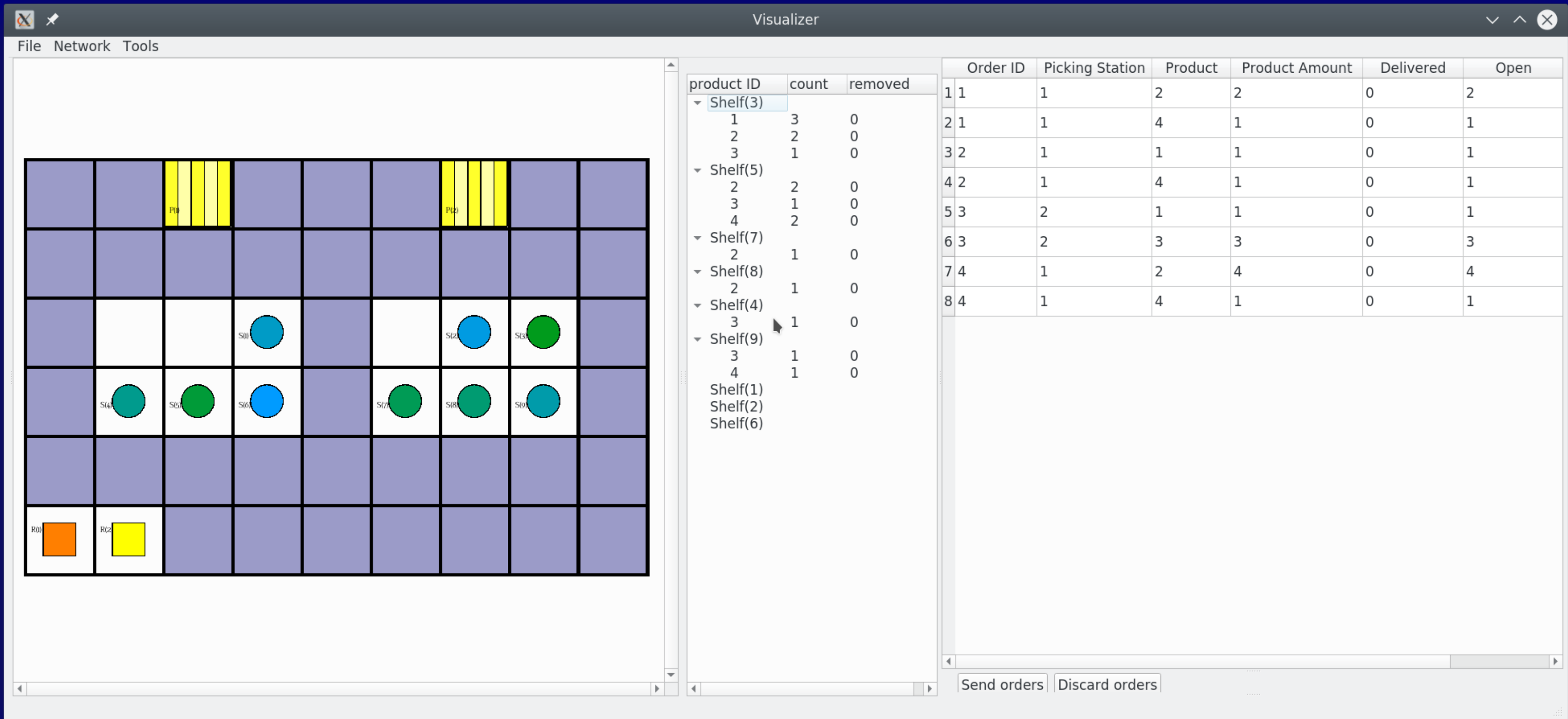
Generating a Structured Instance




```
gen -x 9 -y 6 -X 3 -Y 2 -s 9 -p 2 -r 2 -P 4 -u 16 -o 4 \
    --ol 2 --oap  -H
```

Potassco

Generating a Structured Instance




```
gen -x 9 -y 6 -X 3 -Y 2 -s 9 -p 2 -r 2 -P 4 -u 16 -o 4 \
    --ol 2 --oap  -H
```

Generating a Random Instance

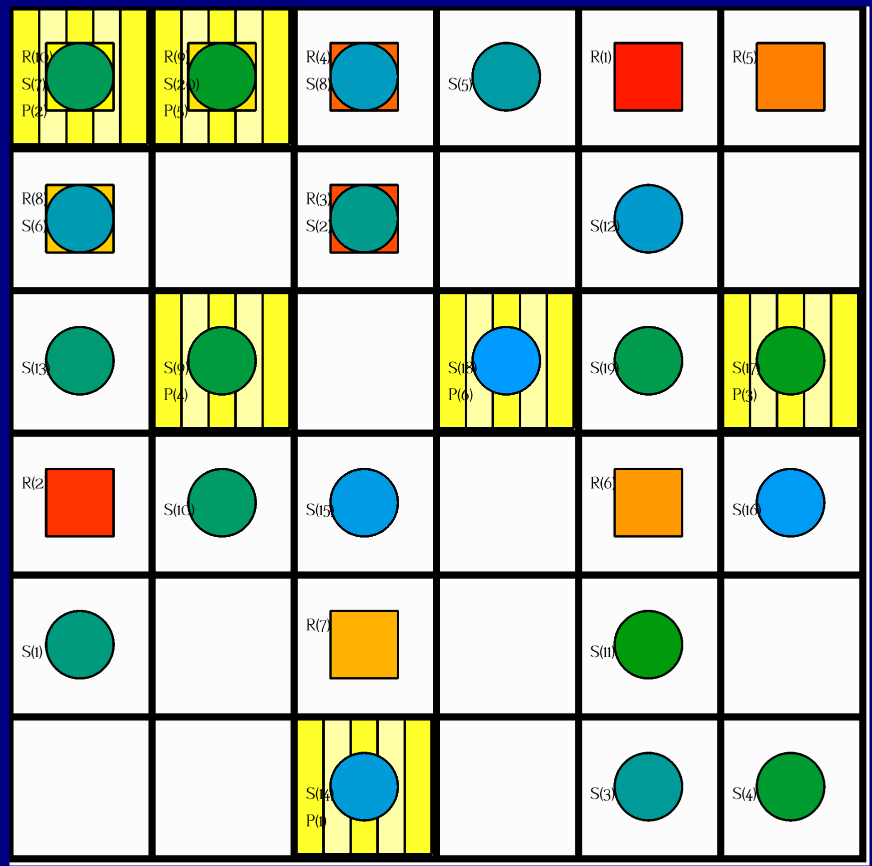
Random instance with:

- 6x6 grid: `-x 6 -y 6`
- 20 shelves, 6 pick stations, 10 robots: `-s 20 -p 6 -r 10`
- 8 products and 30 product units, 2 products per shelf: `-P 8 -u 30 --prs 2`
- 4 orders, 2 order lines each: `-o 4 --ol 2`
- random layout: default, no explicit option required
- for underlying clingo process
  - randomize model enumeration: `--random`
  - use 8 threads, multishot-solving: `-t 8 -I`

via

```
gen -x 6 -y 6 -s 20 -p 6 -r 10 -P 8 -u 30 --prs 2 \
    -o 4 --ol 2 --random -t 8 -I
```

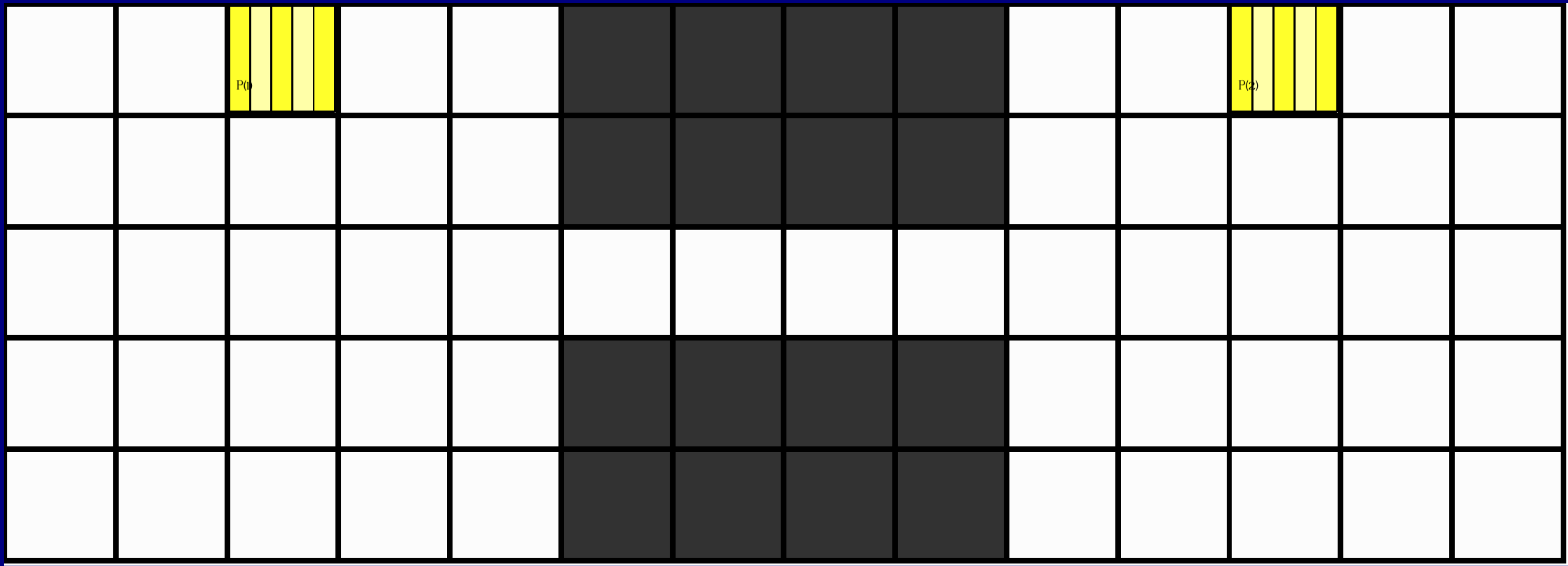
Generating a Random Instance



```
gen -x 6 -y 6 -s 20 -p 6 -r 10 -P 8 -u 30 --prs 2 \
    -o 4 --ol 2 --random -t 8 -I
```

Extending a Custom Layout

Handcrafted Layout: 14x5 grid with “corridor”, 2 pick stations



Extending a Custom Layout

Extend Instance with:

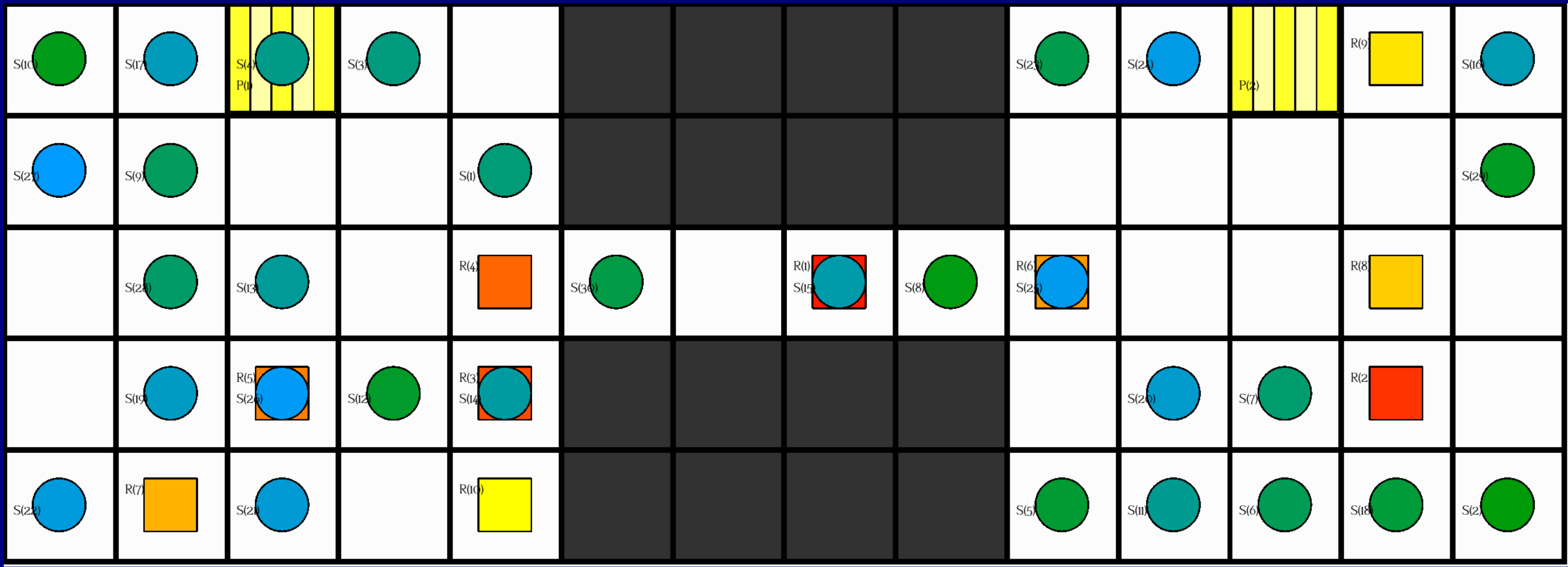
- 30 shelves, 10 robots: `-s 20 -p 6 -r 10`
- 10 products and 100 product units: `-P 10 -u 100`
- 4 orders, 2 order lines each: `-o 4 --ol 2`
- random layout: default, no explicit option required
- custom layout as template: `-T custom_layout.lp`

via

```
gen -s 30 -r 10 -P 10 -u 100 -o 4 --ol 2 \
    -T custom_layout.lp
```

Extending a Custom Layout

Extended Instance:



```
gen -s 30 -r 10 -P 10 -u 100 -o 4 --ol 2 \
    -T custom_layout.lp
```

Outline

Potassco

Motivation, Features

- Visualizer for instances and plans
- Renders ASP instance as warehouse diagram
- Animated execution of plans upon an instances including effects
- Graphical creation and editing of instances

$\rightsquigarrow$ Essential for an intuitive perception of the problem domain and for the verification of plans

Visualizing an Instance

Visualizing a Plan

Editing an Instance

Potassco

Outline

Motivation, Features

- Checks correctness of plans
- Specific description of error causes
- Implemented in ASP

Erroneous Plan

Plan Verification

```
clingo $ASPRILO-ROOT/checker/encodings/a/checker.lp \
       instance.lp plan-err.lp out-ifs="\n"|grep err
```

which yields

```
err(static,highwayPutdown,(2,4,2,29))
err(static,collNode,(robot,5,3,8))
err(deliver,shelfAmount,(3,25))
err(deliver,orderAmount,(3,25))
err(putdown,noShelf,(3,30))
err(move,domain,(1,10))
```

Outline

Potassco

ASP Encoding for Domain M

routing

```
time(1..horizon).

direction((X,Y)) :- X=-1..1, Y=-1..1, |X+Y|=1.
nextto((X,Y),(X',Y'),(X+X',Y+Y')) :- position((X,Y)), direction((X',Y')), position((X+X',Y+Y')).

{ move(R,D,T) : direction(D) } 1 :- isRobot(R), time(T).

position(R,C,T) :- move(R,D,T), position(R,C',T-1),     nextto(C',D,C).
                :- move(R,D,T), position(R,C ,T-1), not nextto(C ,D,_).

position(R,C,T) :- position(R,C,T-1), not move(R,_,T), isRobot(R), time(T).

moveto(C',C,T) :- nextto(C',D,C), position(R,C',T-1), move(R,D,T).
:- moveto(C',C,T), moveto(C,C',T), C < C'.

:- { position(R,C,T) : isRobot(R) }  > 1, position(C), time(T).


processed(O,A) :- ordered(O,A), shelved(S,A), position(S,C,0), position(R,C,horizon), isRobot(R).

processed(O) :- isOrder(O), processed(O,A) : ordered(O,A).

:- not processed(O), isOrder(O).
```

ASP Encoding for Domain M

routing to shelves

```
time(1..horizon).

direction((X,Y)) :- X=-1..1, Y=-1..1, |X+Y|=1.
nextto((X,Y),(X',Y'),(X+X',Y+Y')) :- position((X,Y)), direction((X',Y')), position((X+X',Y+Y')).

{ move(R,D,T) : direction(D) } 1 :- isRobot(R), time(T).

position(R,C,T) :- move(R,D,T), position(R,C',T-1),     nextto(C',D,C).
                :- move(R,D,T), position(R,C ,T-1), not nextto(C ,D,_).

position(R,C,T) :- position(R,C,T-1), not move(R,_,T), isRobot(R), time(T).

moveto(C',C,T) :- nextto(C',D,C), position(R,C',T-1), move(R,D,T).
:- moveto(C',C,T), moveto(C,C',T), C < C'.

:- { position(R,C,T) : isRobot(R) }  > 1, position(C), time(T).


processed(O,A) :- ordered(O,A), shelved(S,A), position(S,C,0), position(R,C,horizon), isRobot(R).

processed(O) :- isOrder(O), processed(O,A) : ordered(O,A).

:- not processed(O), isOrder(O).
```

ASP Encoding for Domain A

routing + transport + delivery

```
time(1..horizon).

direction((X,Y)) :- X=-1..1, Y=-1..1, |X+Y|=1.
nextto((X,Y),(X',Y'),(X+X',Y+Y')) :- position((X,Y)), direction((X',Y')), position((X+X',Y+Y')).

{    move(R,D,T) : direction(D) ;
   pickup(R,S,T) : isShelf(S)   ;
  putdown(R,S,T) : isShelf(S)   } 1 :- isRobot(R), time(T).

waits(R,T) :- not pickup(R,_,T), not putdown(R,_,T), not move(R,_,T), isRobot(R), time(T).

position(R,C,T) :- move(R,D,T),    position(R,C',T-1),     nextto(C',D,C).
                :- move(R,D,T),    position(R,C ,T-1), not nextto(C, D,_).

 carries(R,S,T) :- pickup(R,S,T),  position(R,C,T-1), position(S,C ,T-1).
                :- pickup(R,S,T),   carries(R,_,T-1).
                :- pickup(R,S,T),   carries(_,S,T-1).
                :- pickup(R,S,T),  position(R,C,T-1), position(S,C',T-1), C != C'.

                :- putdown(R,S,T), not carries(R,S,T-1).

serves(R,S,P,T) :- position(R,C,T), carries(R,S,T),  position(P,C), isStation(P).

position(R,C,T) :- position(R,C,T-1), not    move(R,_,T), isRobot(R), time(T).
 carries(R,S,T) :-  carries(R,S,T-1), not putdown(R,_,T),             time(T).

position(S,C,T) :- position(R,C,T  ),     carries(R,S,T).
position(S,C,T) :- position(S,C,T-1), not carries(_,S,T), isShelf(S), time(T).

moveto(C',C,T) :- nextto(C',D,C), position(R,C',T-1), move(R,D,T).
:- moveto(C',C,T), moveto(C,C',T), C < C'.

 :- { position(R,C,T) : isRobot(R) }  > 1, position(C), time(T).
 :- { position(S,C,T) : isShelf(S) }  > 1, position(C), time(T).
```

Potassco

Outline

Summary

- ASPRILO aims at enabling **research** and **teaching** of complex dynamic scenarios occurring in the rich model scenario of **robotic intra-logistics**
- ASPRILO offers
  - Standardized benchmark domains
  - Formal specification
  - Versatile instance generator
  - Visualizer for problems and (candidate) solutions
  - Solution checker with error feedback
  - Reference ASP encodings
- Join us at `potassco.org/asprilo` !

[1] H. Ma, S. Koenig, N. Ayanian, L. Cohen, W. Hoenig, S. Kumar, T. Uras, H. Xu, C. Tovey, and G. Sharon.
Overview: Generalizations of Multi-Agent Path Finding to Real-World Scenarios.
In *Proceedings of the IJCAI-16 Workshop on Multi-Agent Path Finding*, 2016.

[2] Hang Ma and Sven Koenig.
Optimal target assignment and path finding for teams of agents.
In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 1144–1152, 2016.

[3] Van Duc Nguyen, Philipp Obermeier, Tran Cao Son, Torsten Schaub, and William Yeoh.
Generalized target assignment and path finding using answer set programming.
In *IJCAI*, 2017.

Part V:

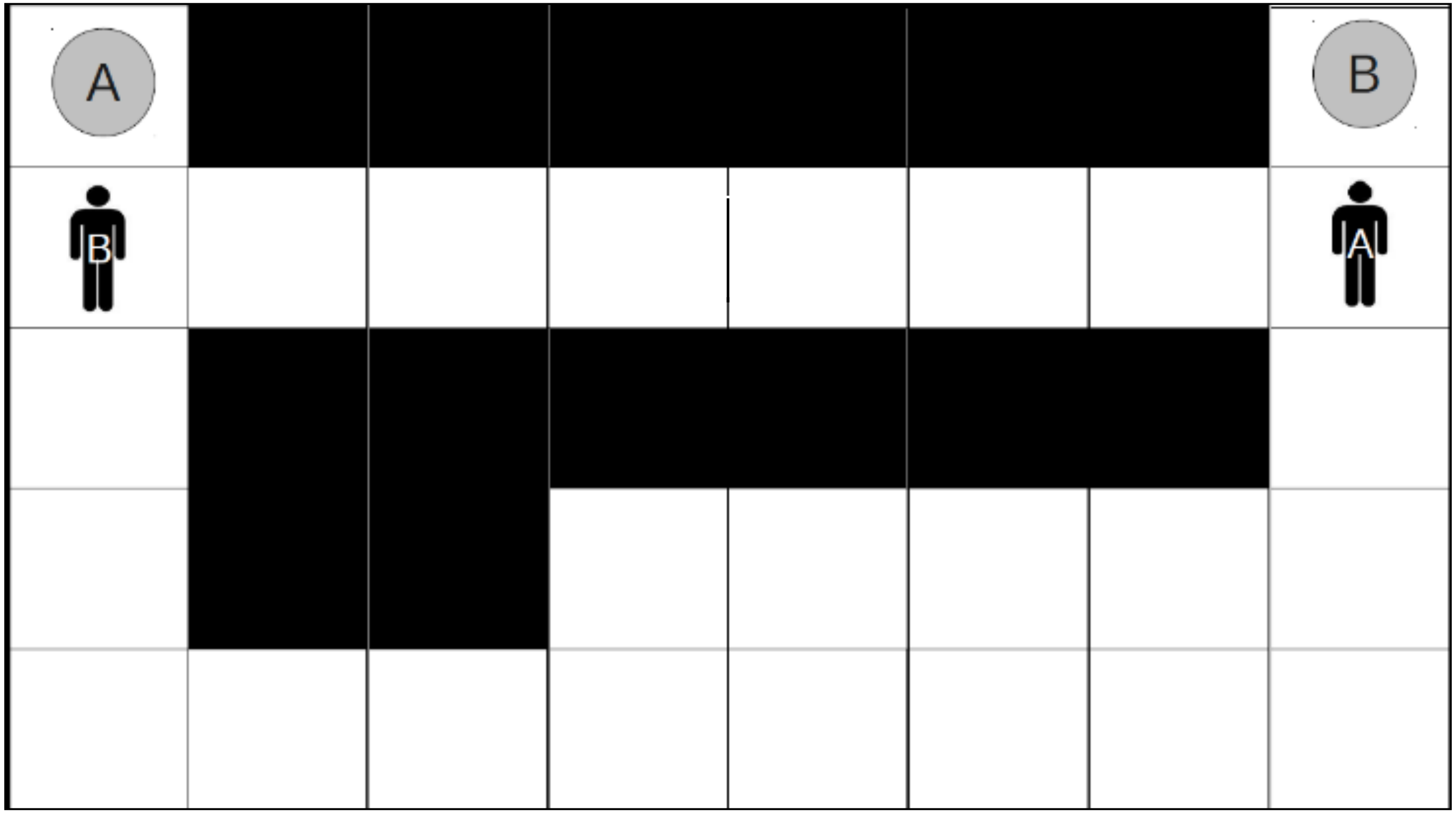
CHALLENGES AND CONCLUSIONS

Multi-Agent Pathfinding in the "Real" World

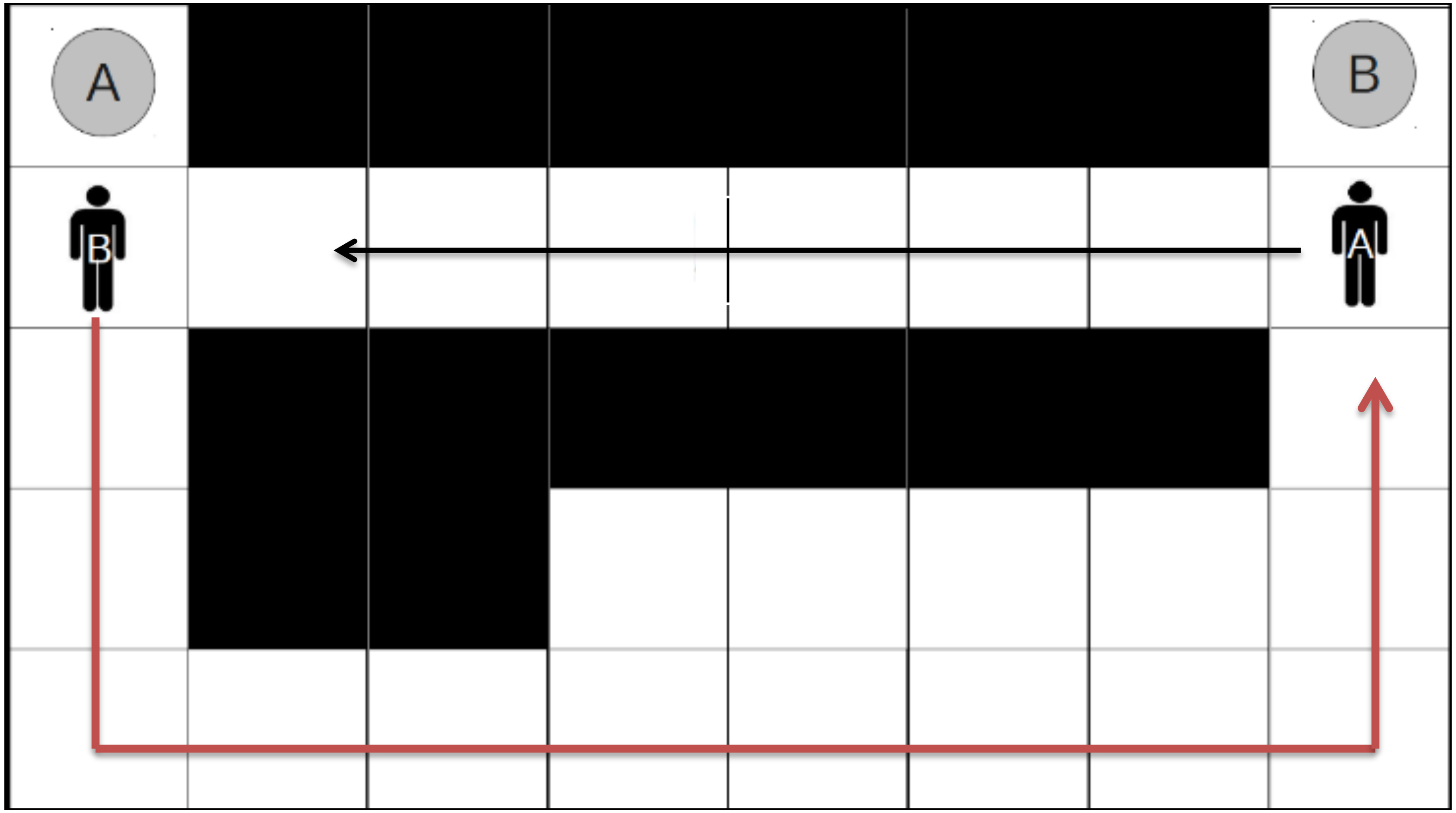
Why I like to work on Multi-Agent Pathfinding

– A **real-world** multi-agent application
– A **very challenging** multi-agent planning problem
– No clear dominant approach (yet)
  • Search-based vs. constraints programming vs. SAT vs. …
– Execution is bound to differ from the plan (integration…)
– So much **left to do**…

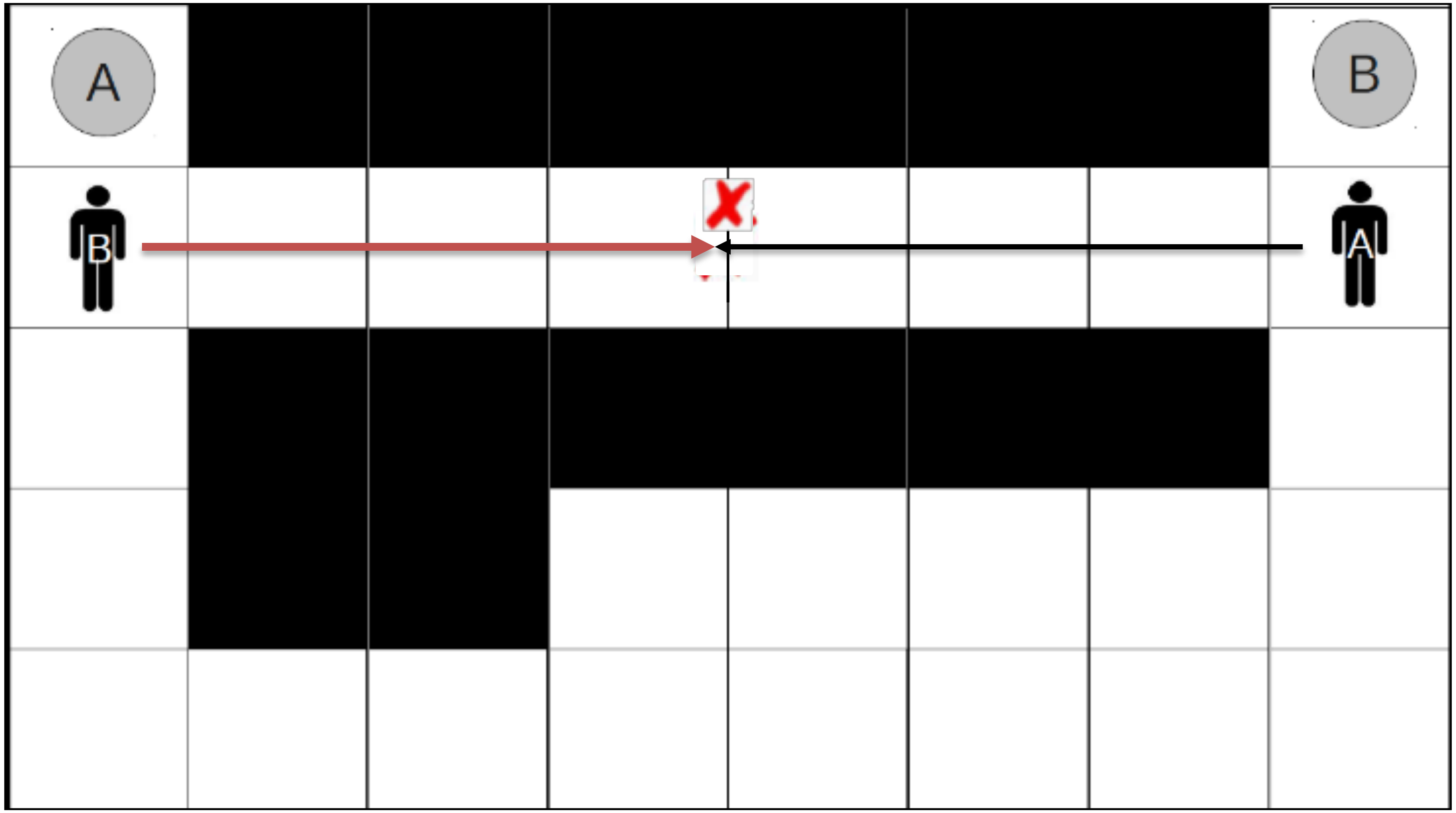
Challenge: MAPF with Self-Interested Agents



Challenge: MAPF with Self-Interested Agents

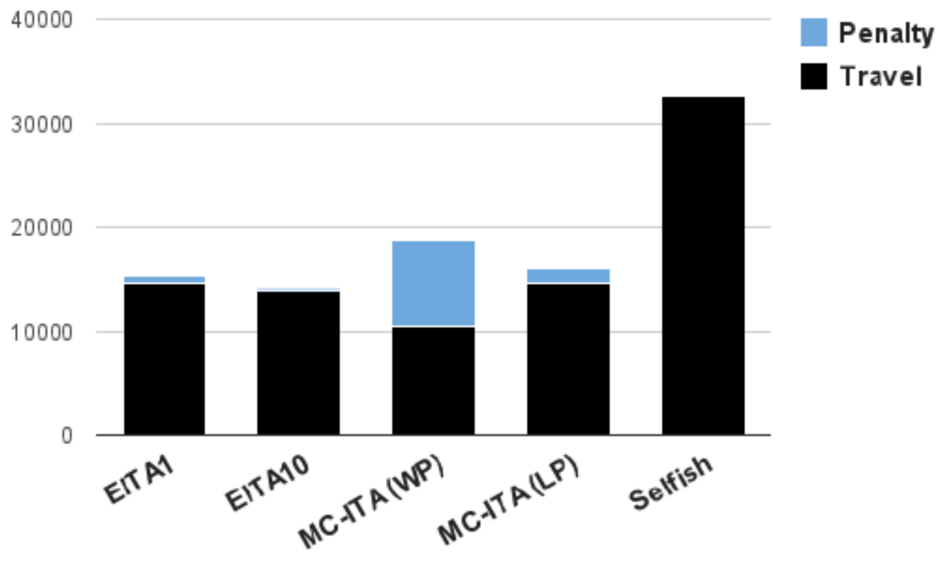


Incentives and mechanism designs [Bnaya et al. '13, Amir '15]

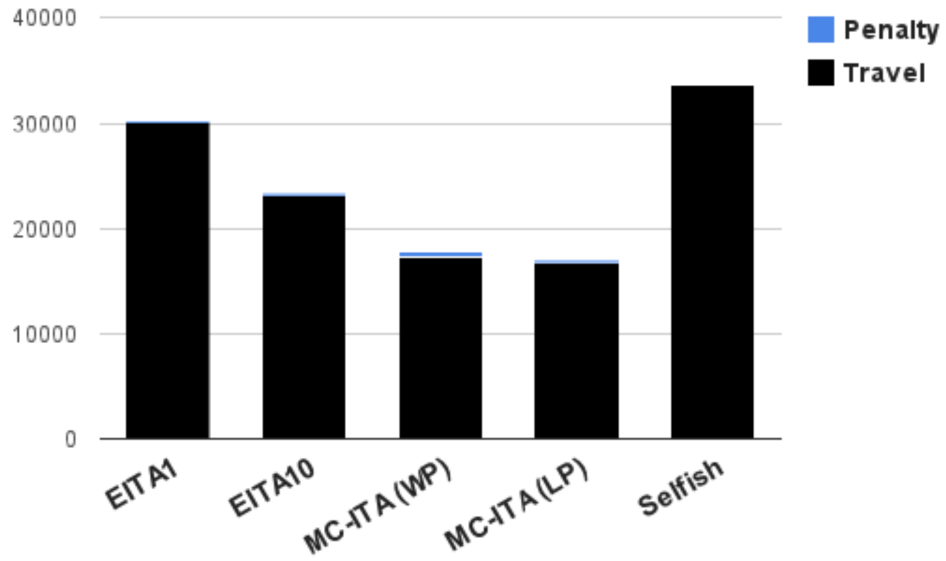


What if the other agent is **adversarial**?
or even worse, a **human**?

Challenge: MAPF with Self-Interested Agents



(a) 50x50 grid with 20% for 20 agents



(b) Dragon age's den520 for 10 agents

Challenges: Applying MAPF for Real Problems

- Robotics
  - Kinematic constraints (Ma et al. '16)
  - Uncertainty is a first-class citizen
  - Continuous configuration space
  - Any-angle motion [Yakovlav et al. '17]
- Traffic management
  - Flow-based approaches
  - No collisions, only traffic jams
  - Scale

Challenge: MAPF as Part of a System

- Task allocation
  - See Ma et al. '16 for combining, flow-based and CBS
- Pick up and delivery tasks
  - See Ma et al. '16, '17, '19 and others
- Online settings

Cross fertilization seems natural

MAPF is a special case of MAP

- MAP
  - Many models, rich literature
  - Much work on uncertainty
  - Poor scaling
- MAPF
  - Fewer models, growing literature
  - Not much work on uncertainty
  - Scales well


066
119
058
116
148
48
064

Some MAPF papers at AAAI-19

- **Multi-Agent Path Finding for Large Agents**
  Wednesday 10:25am-11:25am
- **Searching with Consistent Prioritization for Multi-Agent Path Finding**
  Wednesday 11:30am-12:30pm
- **Online Multi-Agent Pathfinding**
  Wednesday 11:30am-12:30pm
- **Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery**
  Wednesday 2:00pm-3:30pm
- **Symmetry Breaking Constraints for Grid-based Multi-Agent Path Finding**
  Wednesday 3:35pm-4:35pm

https://tinyurl.com/mapf-aaai-2019

Thanks!

Roman Barták, Philipp Obermeier, Torsten Schaub, Tran Cao Son, Roni Stern