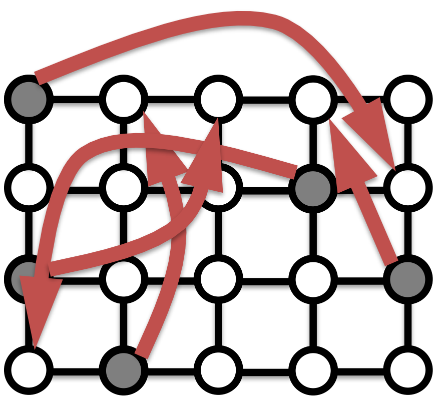
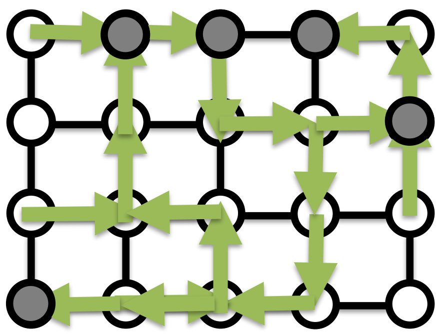
# Multi-Agent Pathfinding

Roman Barták, Roni Stern

*Introduction*

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*

**Part I: Introduction to MAPF**
- *Problem formulation, variants and objectives*
- *Applications*

**Part II. Search-based solvers**
- *Incomplete solvers*
- *Complete suboptimal solvers*
- *Optimal solvers*

**Part III. Reduction-based solvers**
- *SAT encodings*
- *CP encodings*

**Part IV. From planning to execution**
- *Execution policies for MAPF*
- *Execution-aware offline planning*

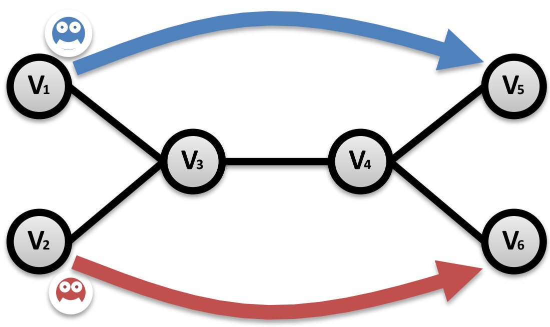
**Part V. Challenges and conclusions**

Part I:

# INTRODUCTION TO MAPF

## MAPF formulation

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




## Plans

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

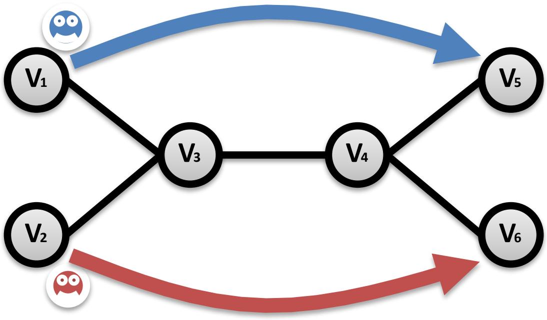
*Typical assumption*:

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).



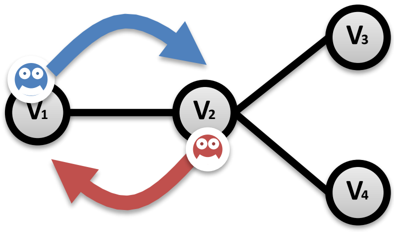

| time | agent 1 | agent 2 |
|---|---|---|
| 0 | **v1** | **v2** |
| 1 | wait **v1** | move **v3** |
| 2 | move **v3** | move **v4** |
| 3 | move **v4** | move **v6** |
| 4 | move **v5** | wait **v6** |

## *Plan existence*

Some trivial **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

## *No-swap constraint*




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

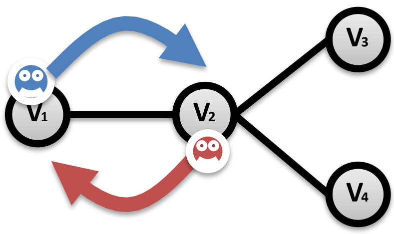
| time | agent 1 | agent 2 |
|---|---|---|
| 0 | $v_1$ | $v_2$ |
| 1 | move $v_2$ | move $v_1$ |

Agents use the same edge at the same time!

**Swap is not allowed.**

| time | agent 1 | agent 2 |
|---|---|---|
| 0 | $v_1$ | $v_2$ |
| 1 | move $v_2$ | move $v_3$ |
| 2 | move $v_4$ | move $v_2$ |
| 3 | move $v_2$ | move $v_1$ |

## *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 | $v_1$ | $v_2$ |
| 1 | move $v_2$ | move $v_3$ |
| 2 | move $v_4$ | move $v_2$ |
| 3 | move $v_2$ | move $v_1$ |

Agents form a **train**.

Trains may be forbidden.

| time | agent 1 | agent 2 |
|---|---|---|
| 0 | $v_1$ | $v_2$ |
| 1 | *wait $v_1$* | move $v_3$ |
| 2 | move $v_2$ | *wait $v_3$* |
| 3 | move $v_4$ | *wait $v_3$* |
| 4 | *wait $v_4$* | move $v_2$ |
| 5 | *wait $v_4$* | move $v_1$ |
| 6 | move $v_2$ | *wait $v_1$* |

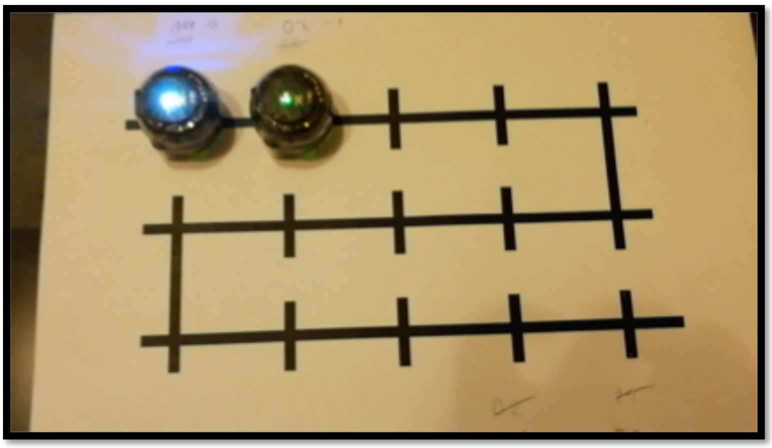
If any agent is delayed then trains may cause collisions during execution.

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*

## Other 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.
- …

## Objectives

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)

Makespan = 4
SOC = 7

| time | agent 1 | agent 2 |
|---|---|---|
| 0 | $v_1$ | $v_2$ |
| 1 | wait $v_1$ | move $v_3$ |
| 2 | move $v_3$ | move $v_4$ |
| 3 | move $v_4$ | move $v_6$ |
| 4 | move $v_5$ | wait $v_6$ |

## Complexity

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.

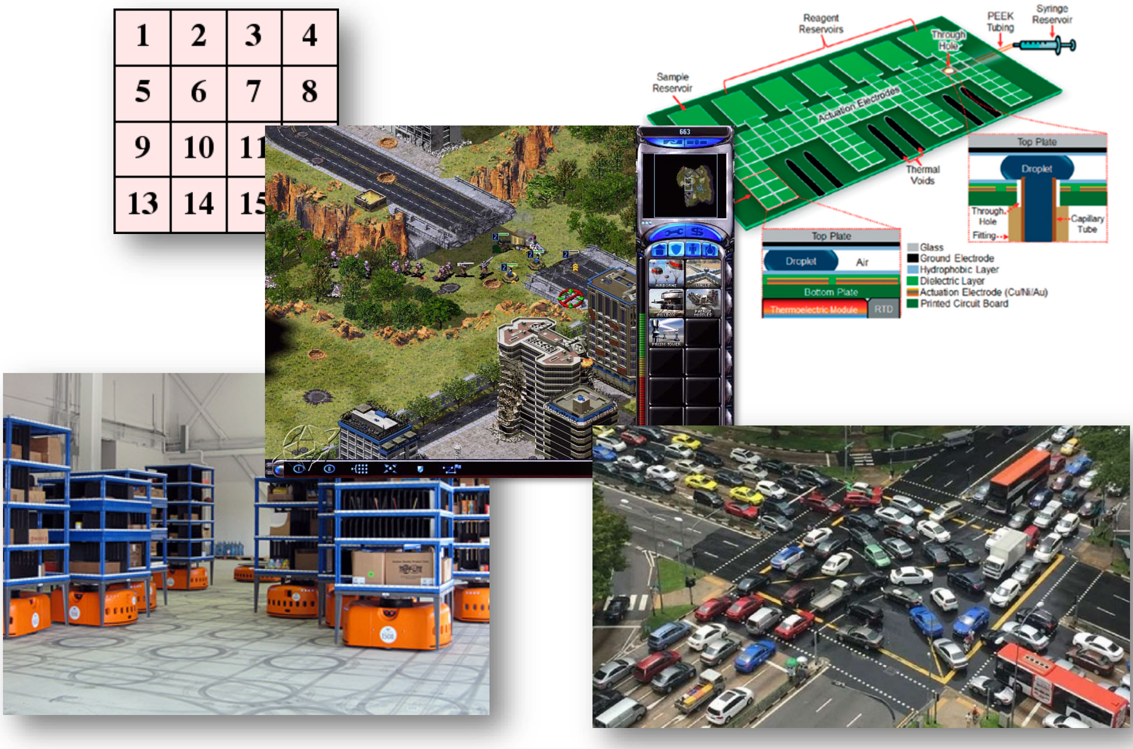
– e.g. algorithm Push and Rotate

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



**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**

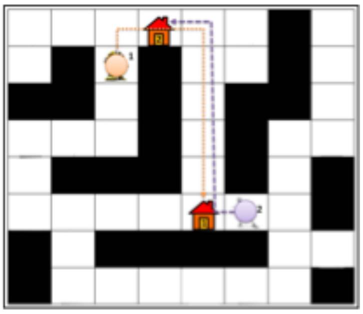
translate the problem to another formalism (SAT/CSP/ASP …)

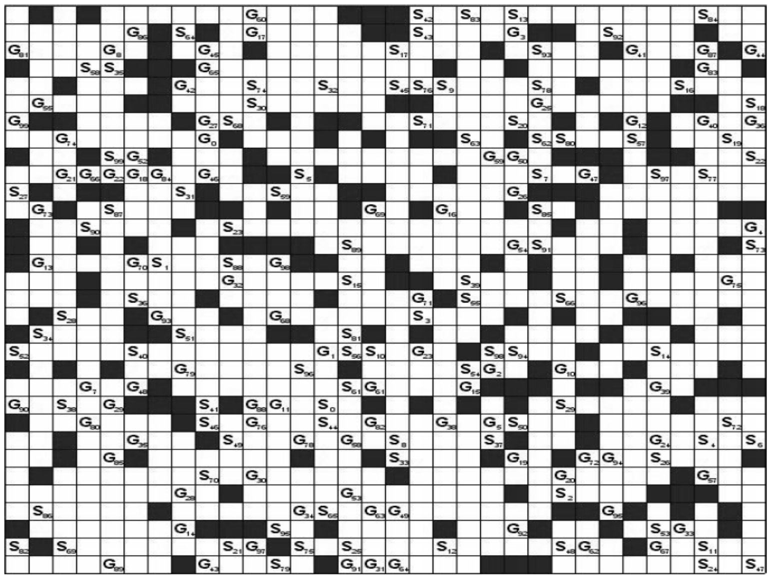
Part II:

# SEARCH-BASED SOLVERS

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

## How to Find Paths for +100 Agents

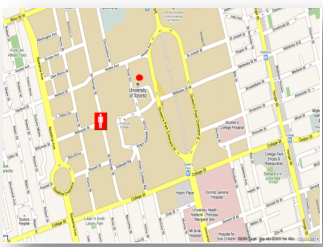




## Why Search-Based MAPF Solvers?

**K=1** **(Navigation in explicit graphs)**
**Explicit graph**

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

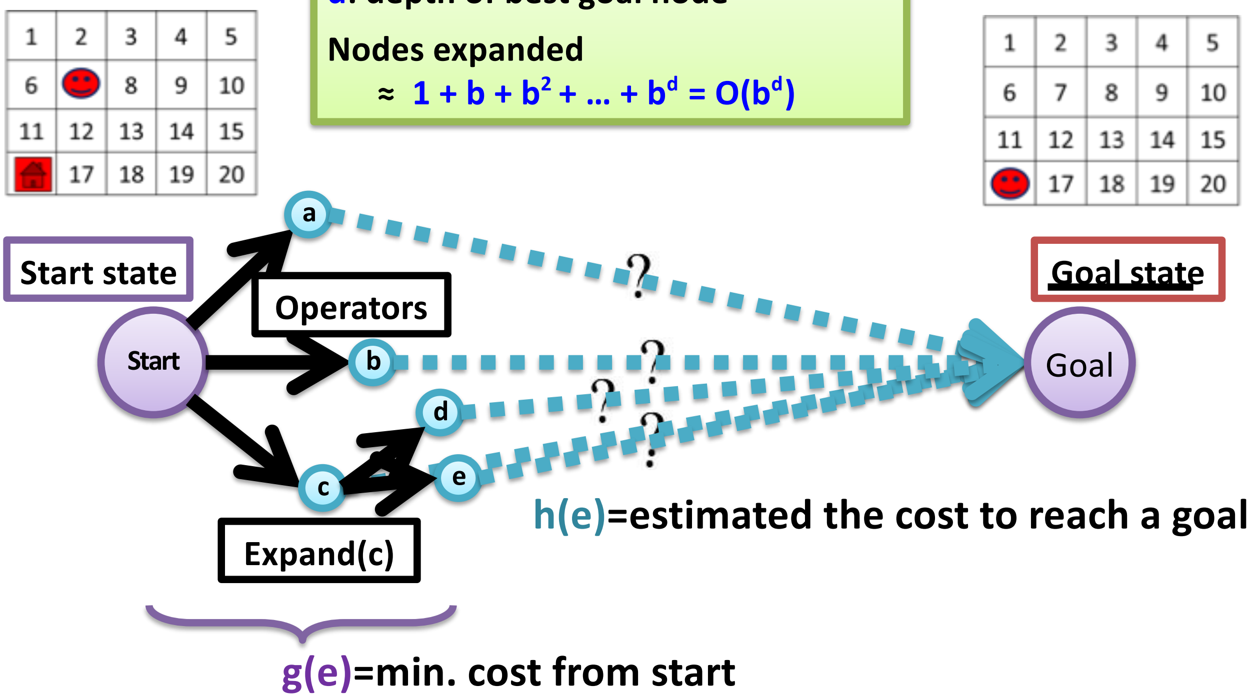



## Classical Search Setting


## Classical Search Setting

b: branching factor = # of operators
d: depth of best goal node

Nodes expanded
$\approx\ 1 + b + b^2 + \ldots + b^d = O(b^d)$



| | **Suboptimal** | **Optimal** |
|---|---|---|
| Incomplete | ? | ? |
| Complete | ? | ? |

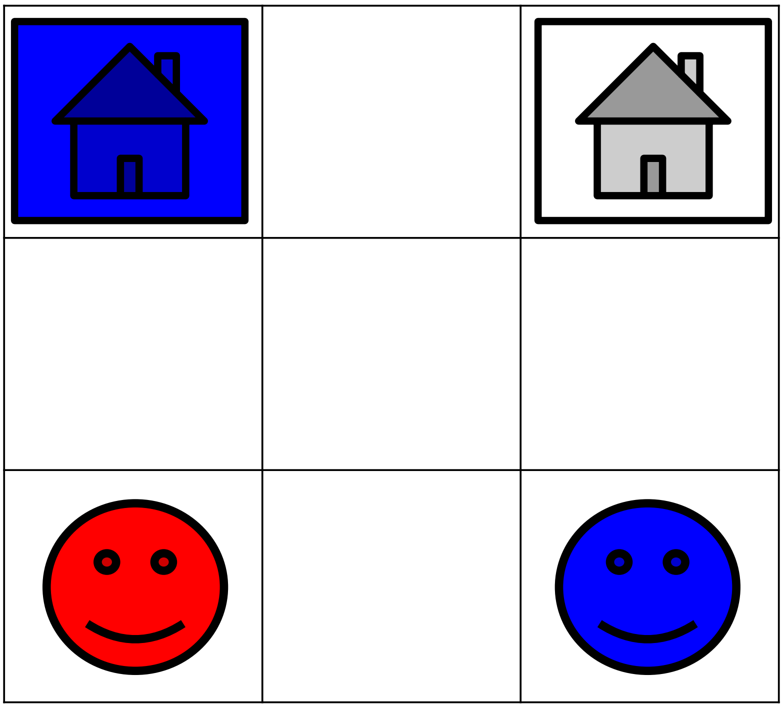


First Attempt: **Cooperative A*** (Silver '05)

- Plan for each agent **separately**
- Avoid collisions with previously planned agents

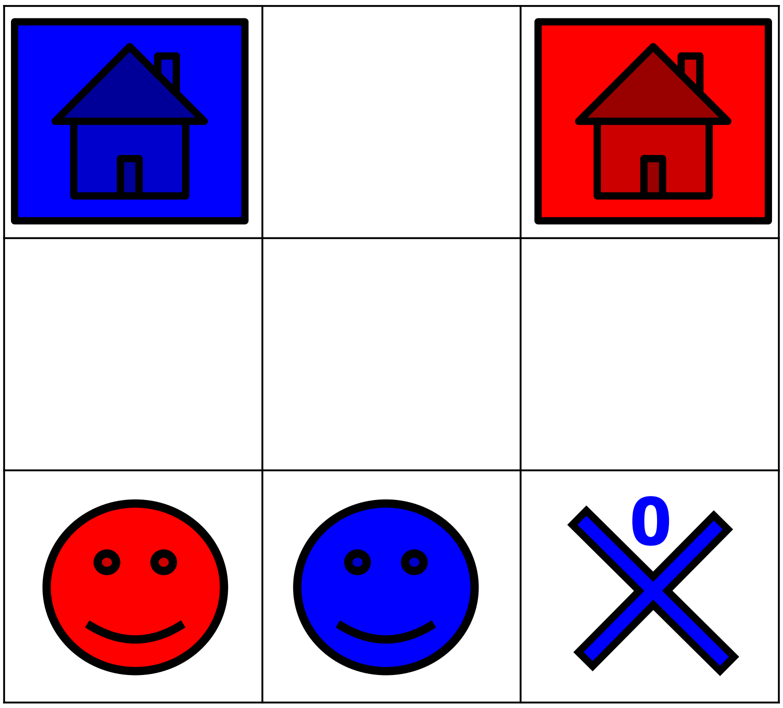
## Cooperative A* (Silver 2005)

- Step 1: Plan **blue**



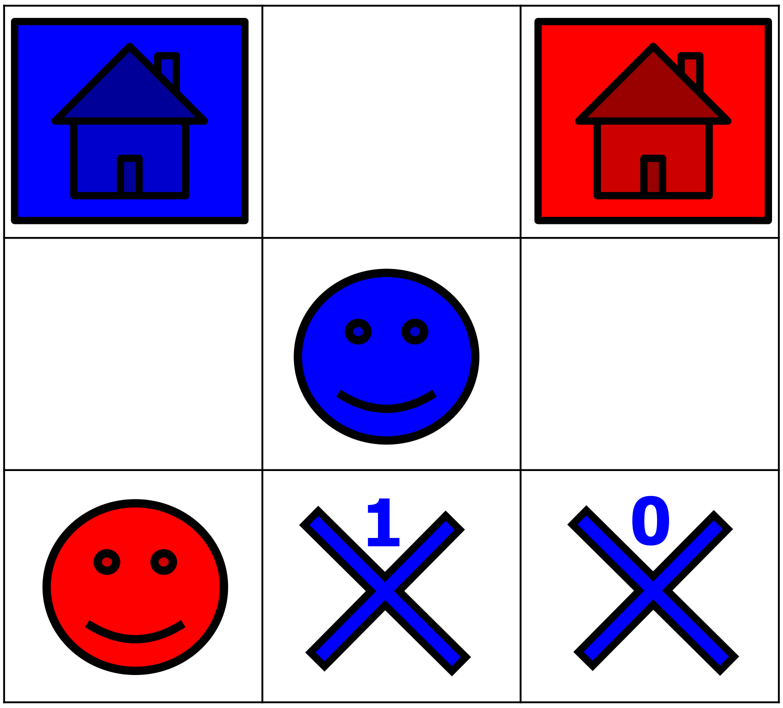
## Cooperative A* - Example

- Step 1: Plan **blue**



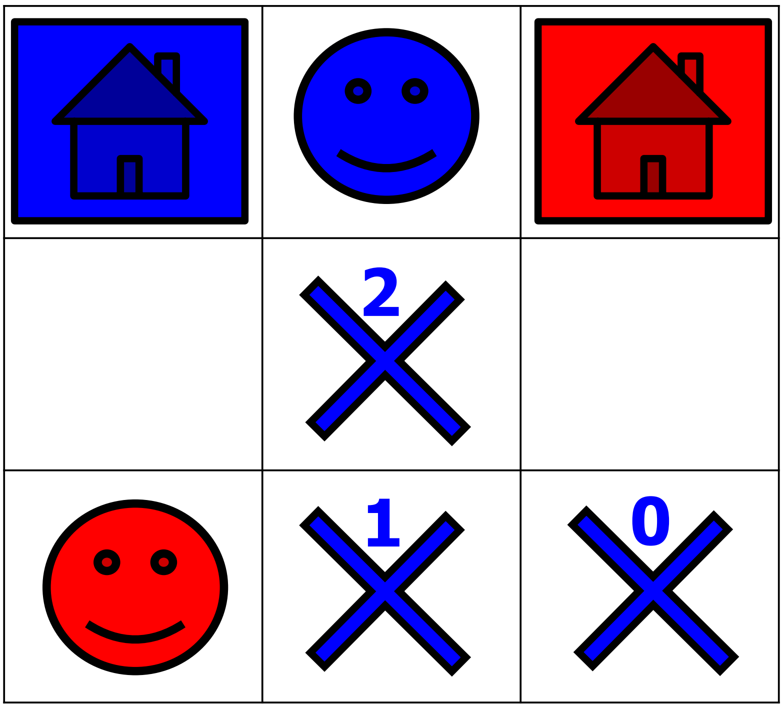
## *Cooperative A* - Example*

- Step 1: Plan **blue**



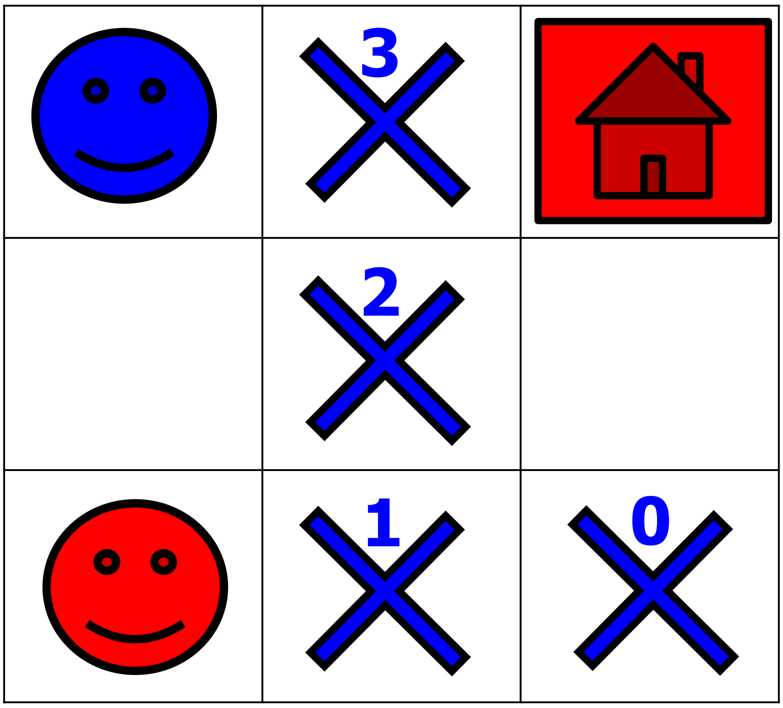

## *Cooperative A* - Example*

- Step 1: Plan **blue**



## *Cooperative A* - Example*

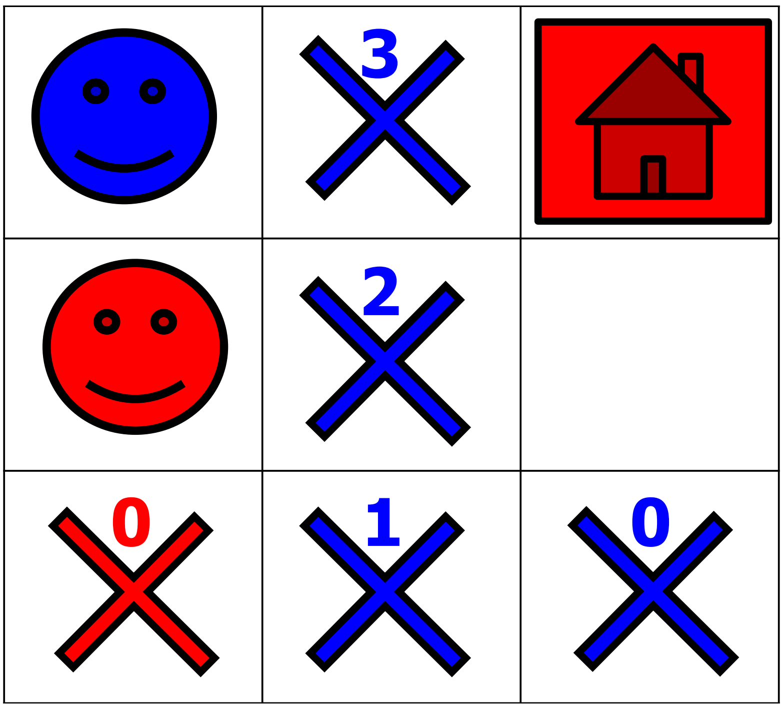
- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**



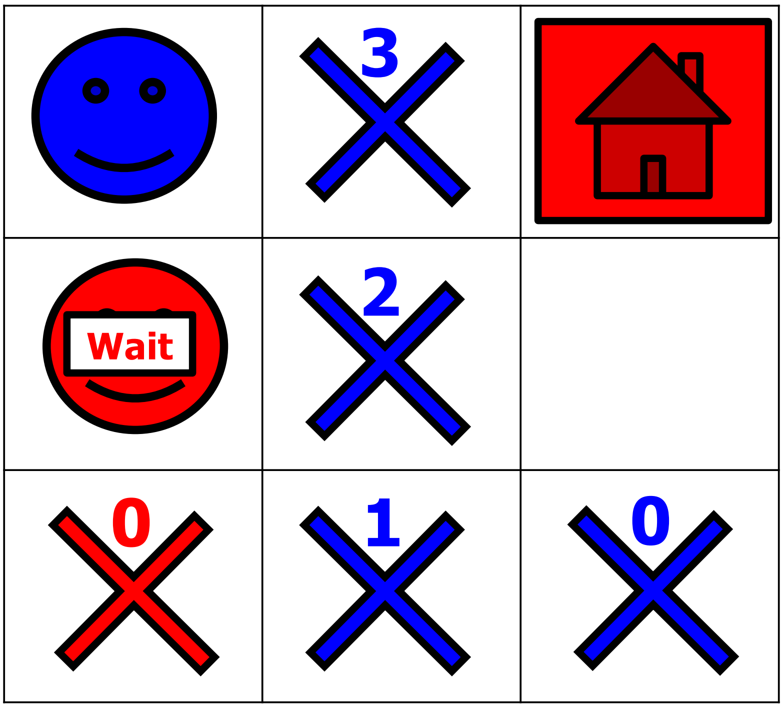

## *Cooperative A* - Example*

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

***avoid blue's plan***

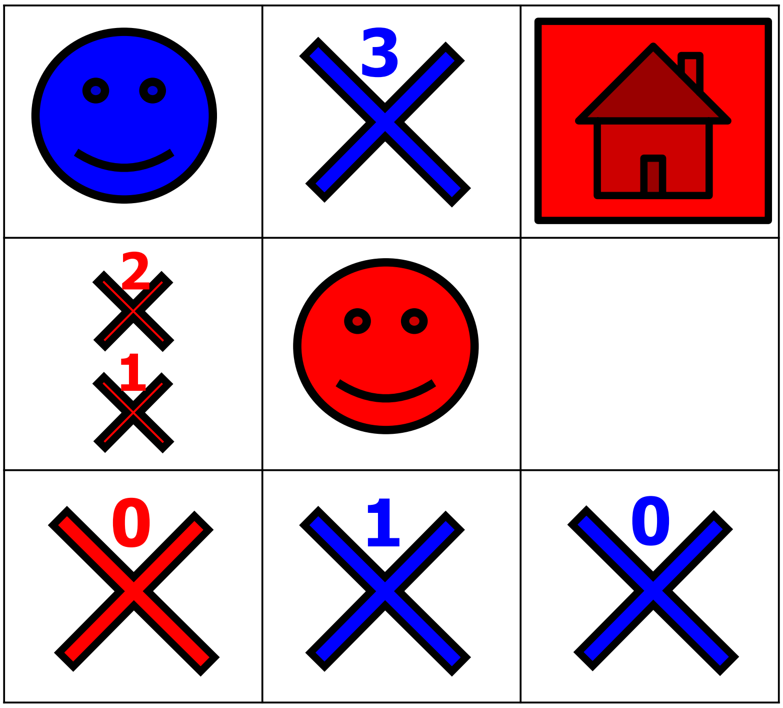


- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**

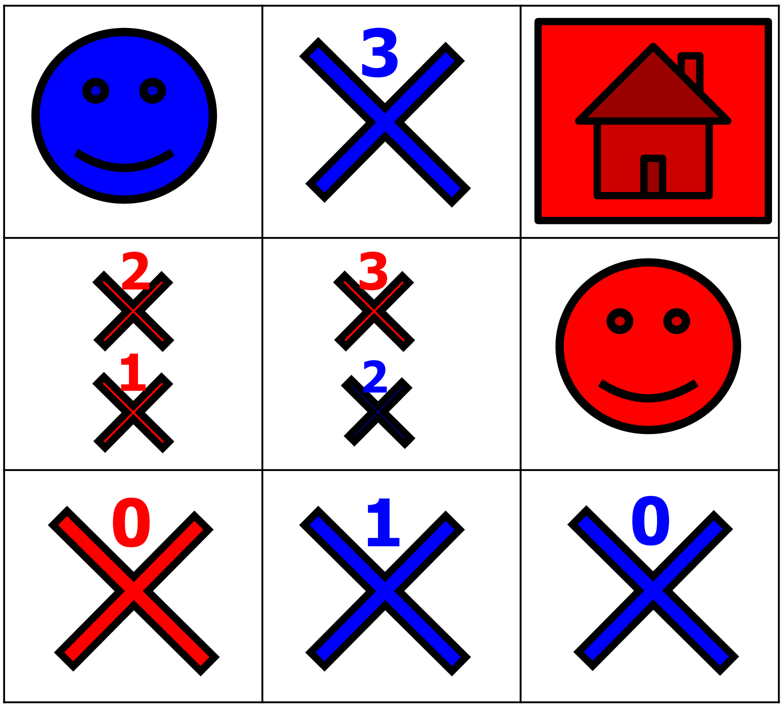



*Cooperative A* - Example*


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

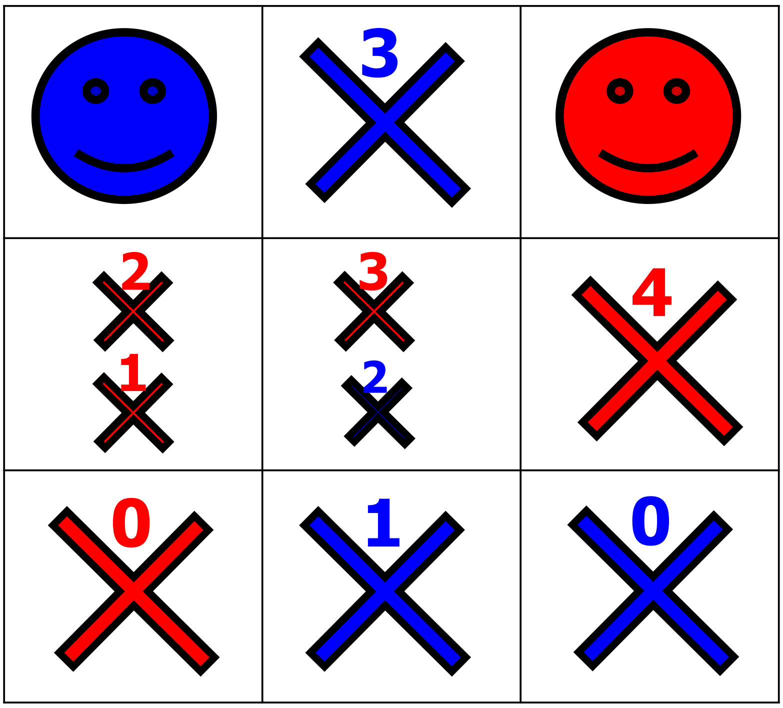



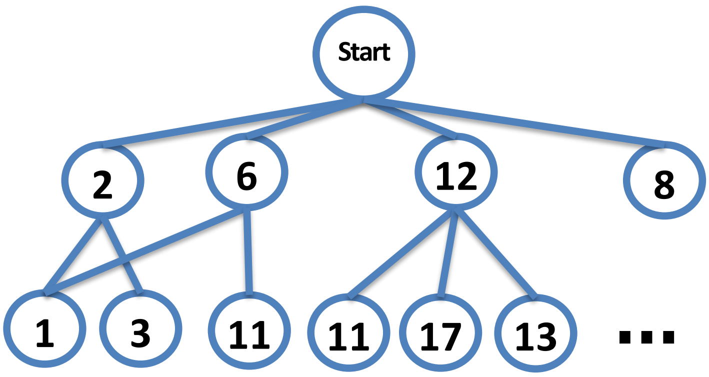
- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**

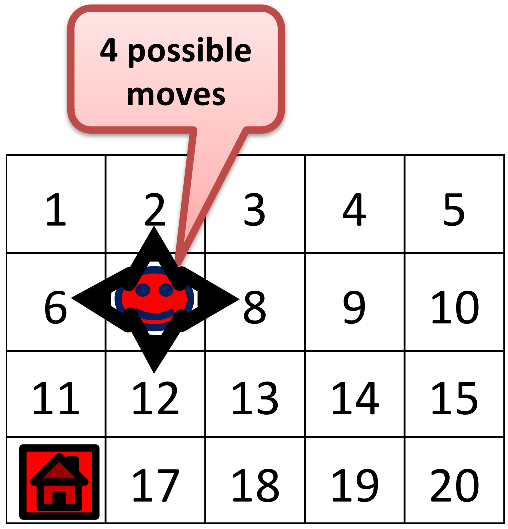



*Cooperative A* - Example*

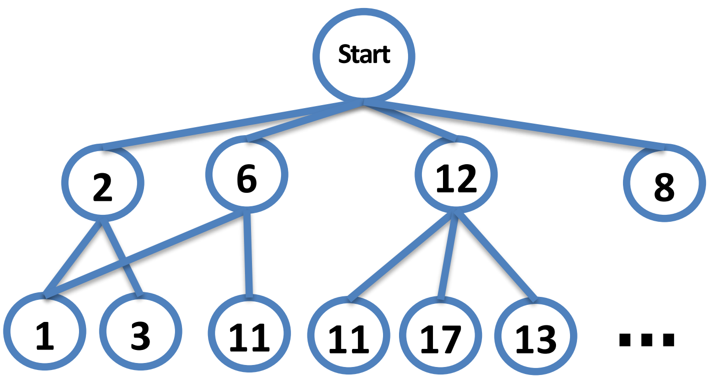
- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**
  - **Done!**
- **…**
- **Step N: Plan N<sup>th</sup> agent**

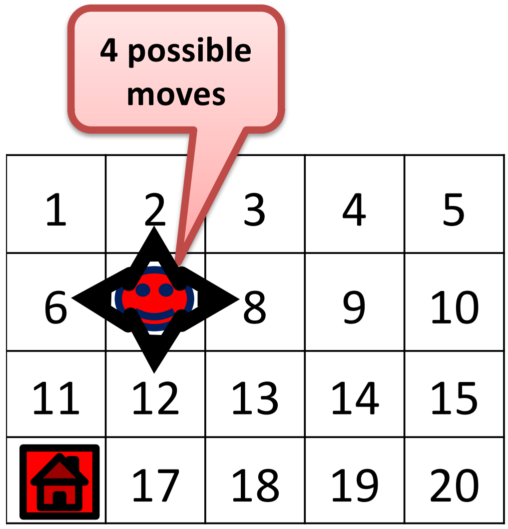





Cooperative A*: Analysis - First Agent



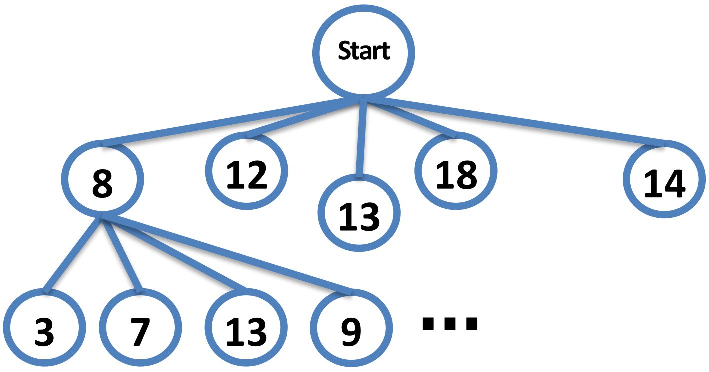



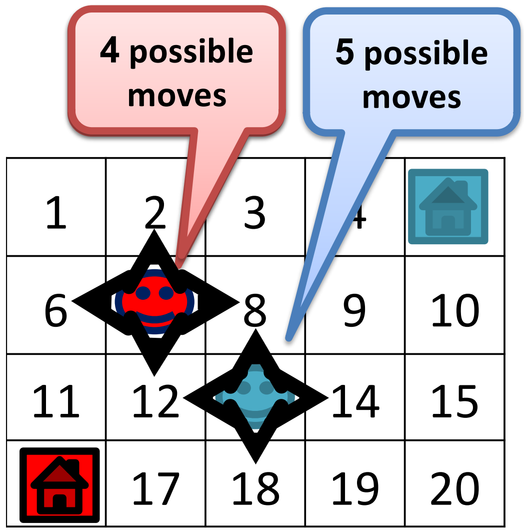

**Singe-agent pathfinding**

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

**Classical search problem!**

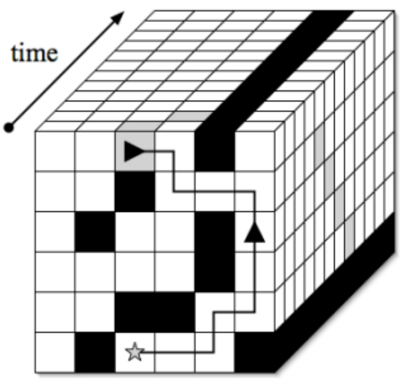
## Cooperative A*: Analysis - Second Agent

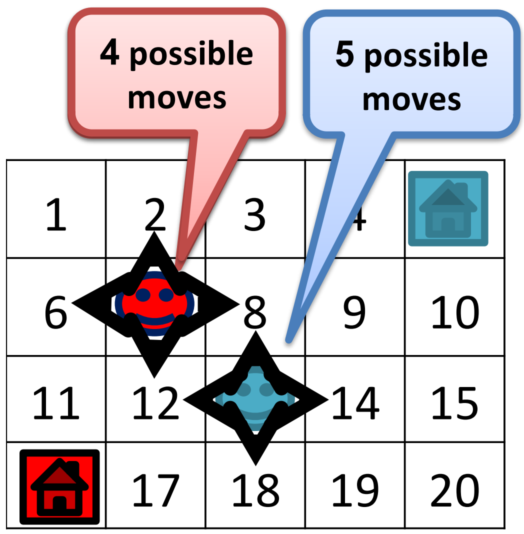



- A state is a (location,time) pair
- Number of states = 4 x 5 x maxTime
- Branching factor = 4+1

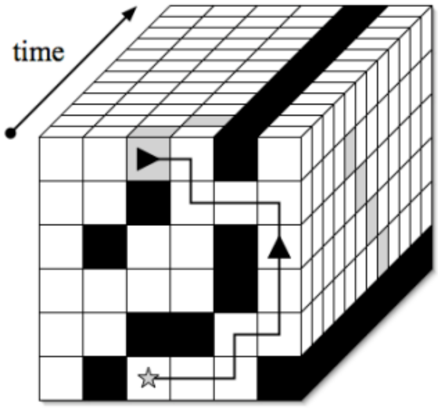
## Cooperative A*: Analysis - Second Agent



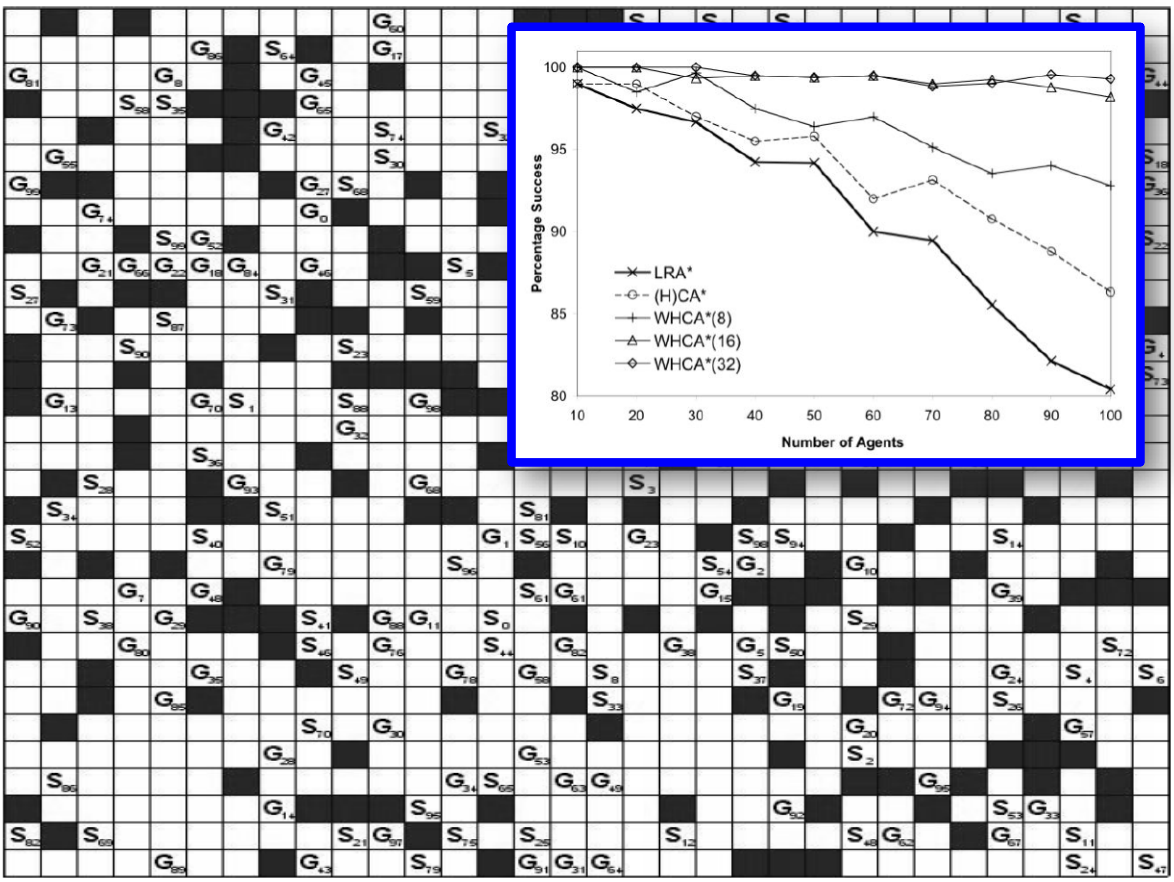

- A state is a (location,time) pair
- Number of states = 4 x 5 x maxTime
- Branching factor = 4+1

## Cooperative A* (Silver 2005)




1. Initialize the reservation table T
2. For each agent do
2.1. Find a path (do not conflict with T)
2.2. Reserve the path in T

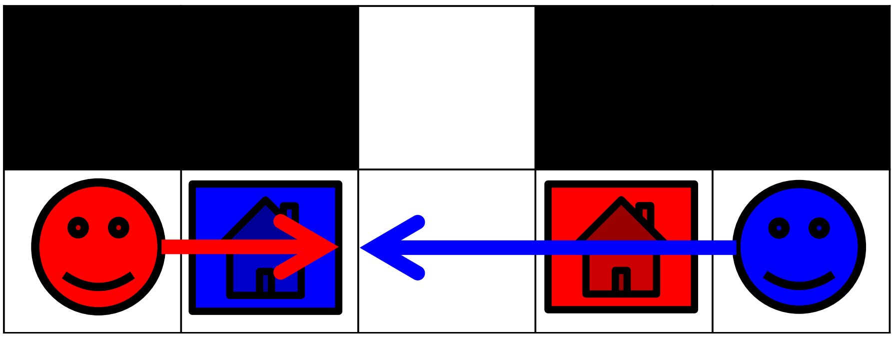
## WHCA* - Results



## Cooperative A* - Analysis

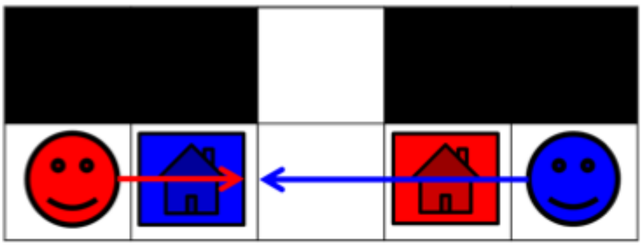
- Complexity?
  - **Polynomial** in the grid size and max time
- Soundness?
  - **Yes!**
- Complete? Optimal?
  - **No ☹**

## Cooperative A* - Limitations



**Not complete** (=may not find a solution)
**Not optimal** (=may find an inefficient solution)

# Cooperative A* - Key Limitations



- A goal location that blocks another agent
- All-or-nothing (can't move until planning is done)
  - *Some relief to this with **WHCA*** (Silver '05)*
- Ordering the agents is key (how to do that?)
  - *Conflict oriented ordering (Byana & Felner '14)*

# Search-based Solvers - Overview

| | **Suboptimal** | **Optimal** |
|---|---|---|
| Incomplete | • Cooperative A*<br>• WHCA* | ? |
| **Complete** | ? | ? |

Can a MAPF algorithm be

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



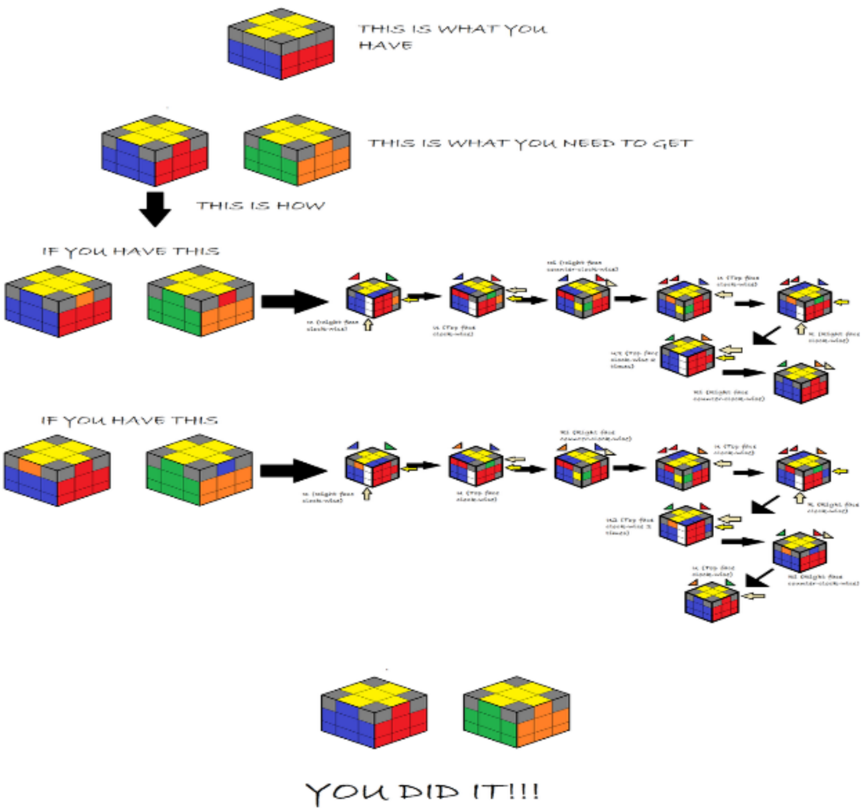

- MAPF is highly related to ***pebble motion problems***
  - Each agent is a pebble
  - Need to move each pebble to its goal
  - Cannot put two pebbles in one hole
- **Pebble motion can be solved polynomially!**
  - **But far from optimally**
  - **Complex formulation**

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

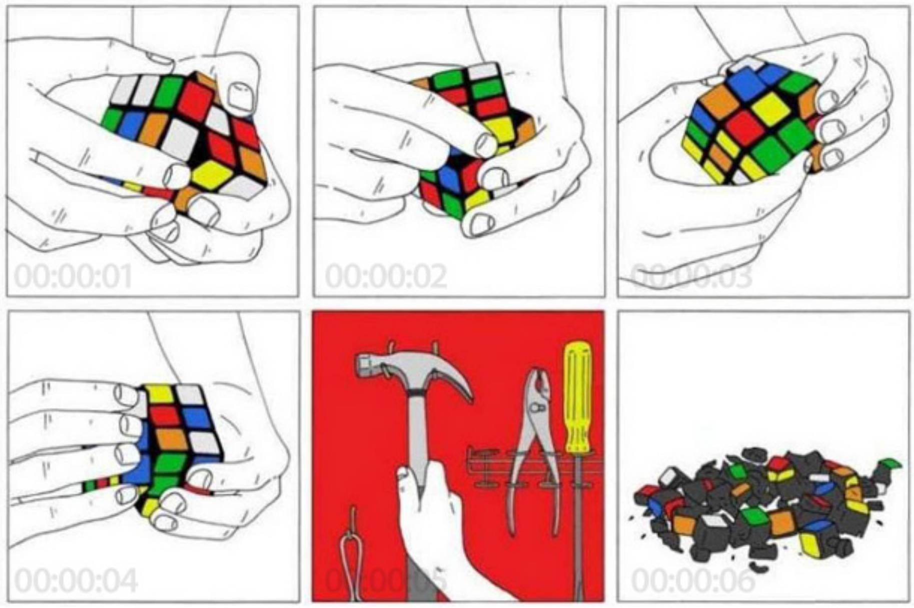
Similar approaches:

- 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]

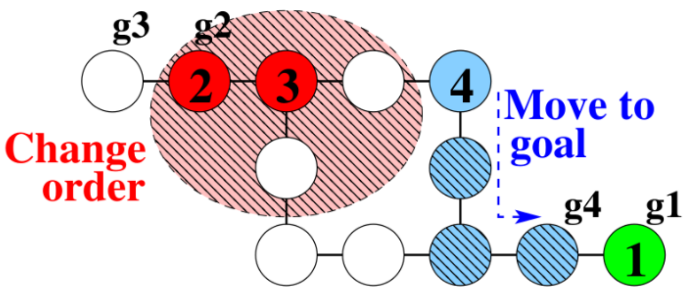




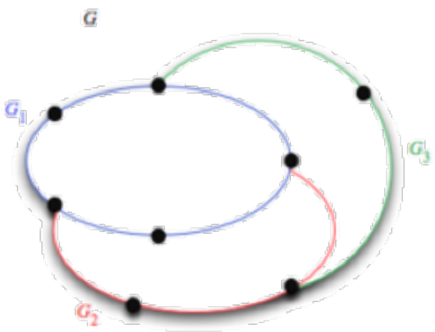
## Procedure-based Solvers #2



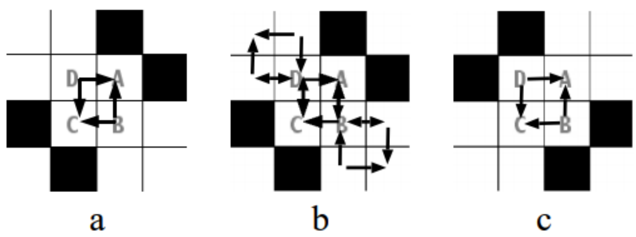

## Examples




Push and Swap (Luna and Bekris ‘13)



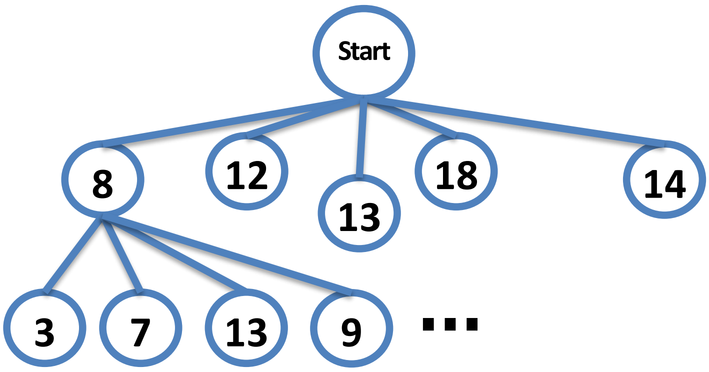
Bibox (Surynek ‘09)

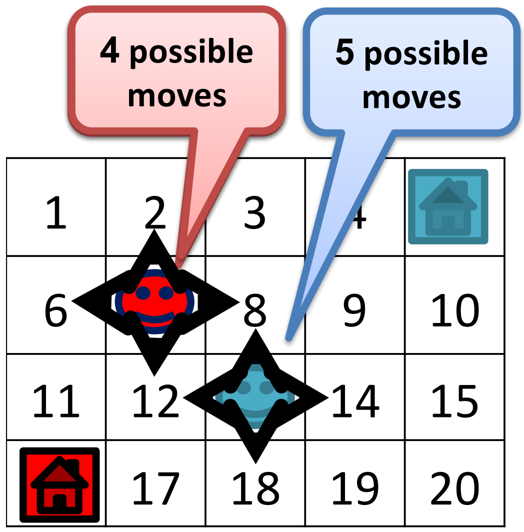



FAR (Wang and Botea ’08)

| | Suboptimal | Optimal |
|---|---|---|
| Incomplete | • Cooperative A*<br>• WHCA* | ? |
| Complete | • Kornhauser et al. ’84<br>• Push & Swap (Luna & Bekris)<br>• Bibox (Surynek)<br>… | ? |

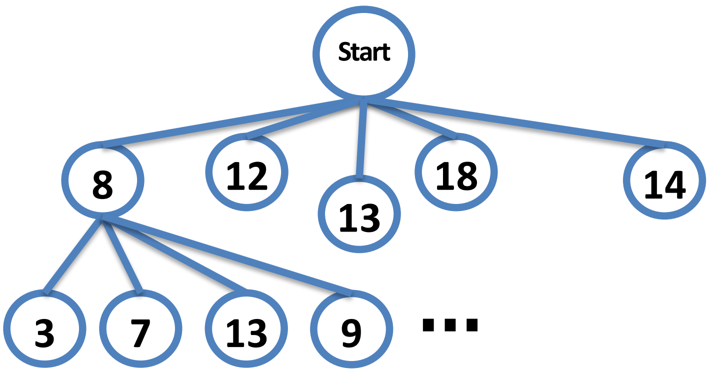


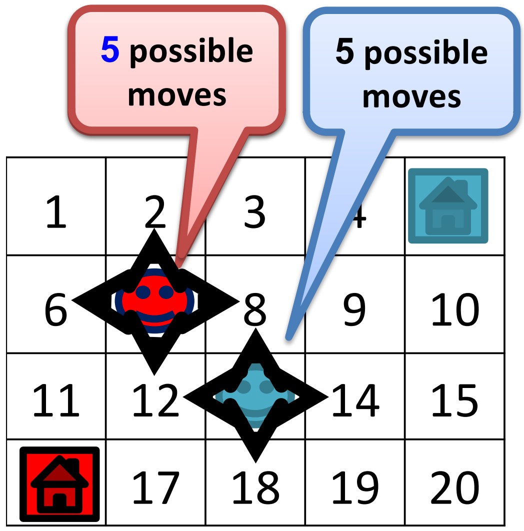



- A state is a (location,time) pair
- Number of states = 4 x 5 x maxTime
- Branching factor = 4+1

## A Two-Agent Search Problem

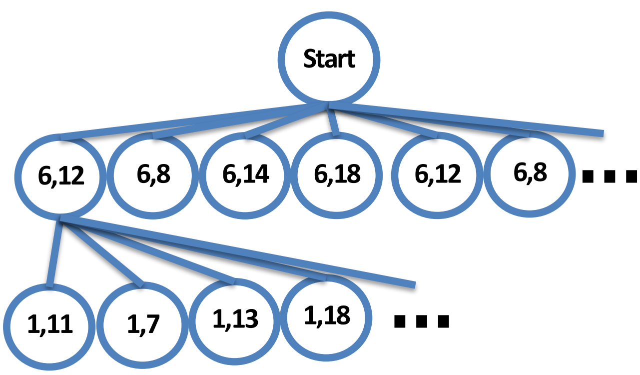


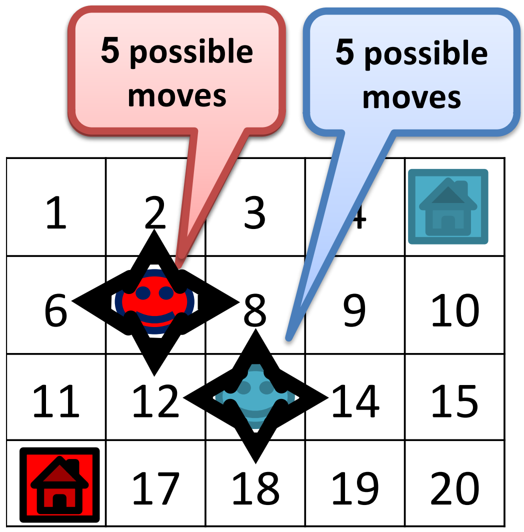




- A state is a pair (location1, location2)
- Number of states = **?**
- Branching factor = **?**

## Optimal Pathfinding for Two Agent

**25 Possible moves! = 5 x 5**




2-agent pathfinding search problem

- Number of states = $(4 \times 5)^2$
- Branching factor = $5^2$

**A very hard search problem**

**Classical search problem?**

Can a MAPF algorithm be

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


**NP-hard**

(Surynek 'X)
(Yu and La Velle 'Y)

Search problem properties
- Number of states = $(4 \times 5)^k$
- Branching factor = $5^k$



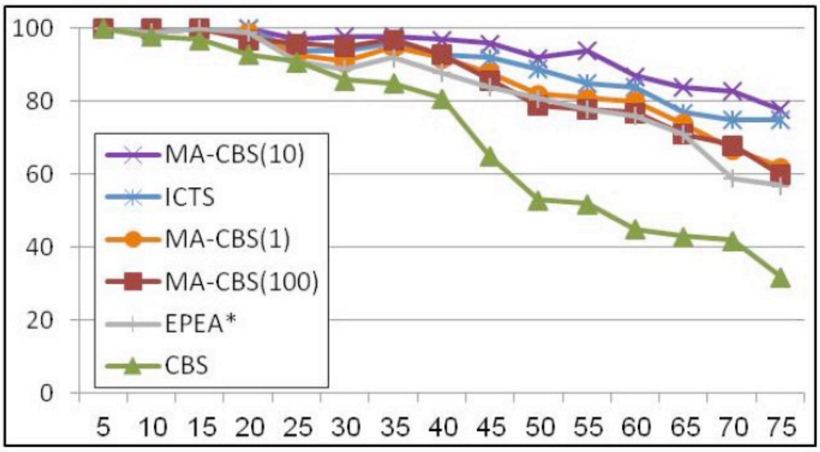
**K=1 (Navigation in explicit graphs)**
**Explicit graph**

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


Can we adapt techniques from these extreme cases?

**Yes!**

**(and invent some new techniques also)**



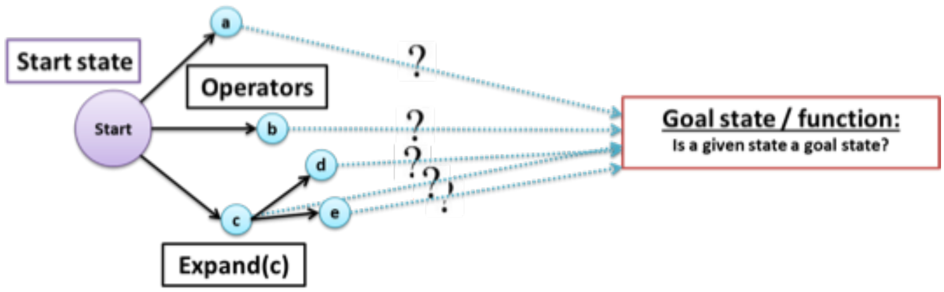



## Searching the k-agent search space

- 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]

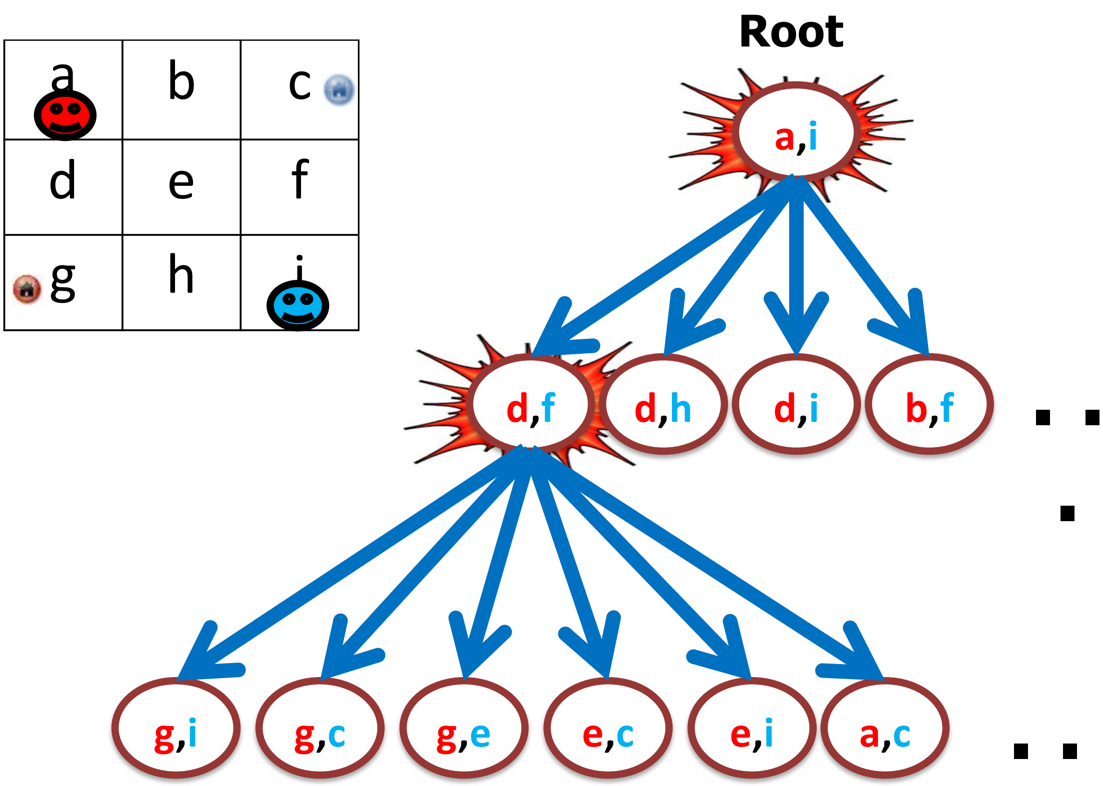



- A* **expands nodes**
- A* gain efficiency by **choosing which node to expand**

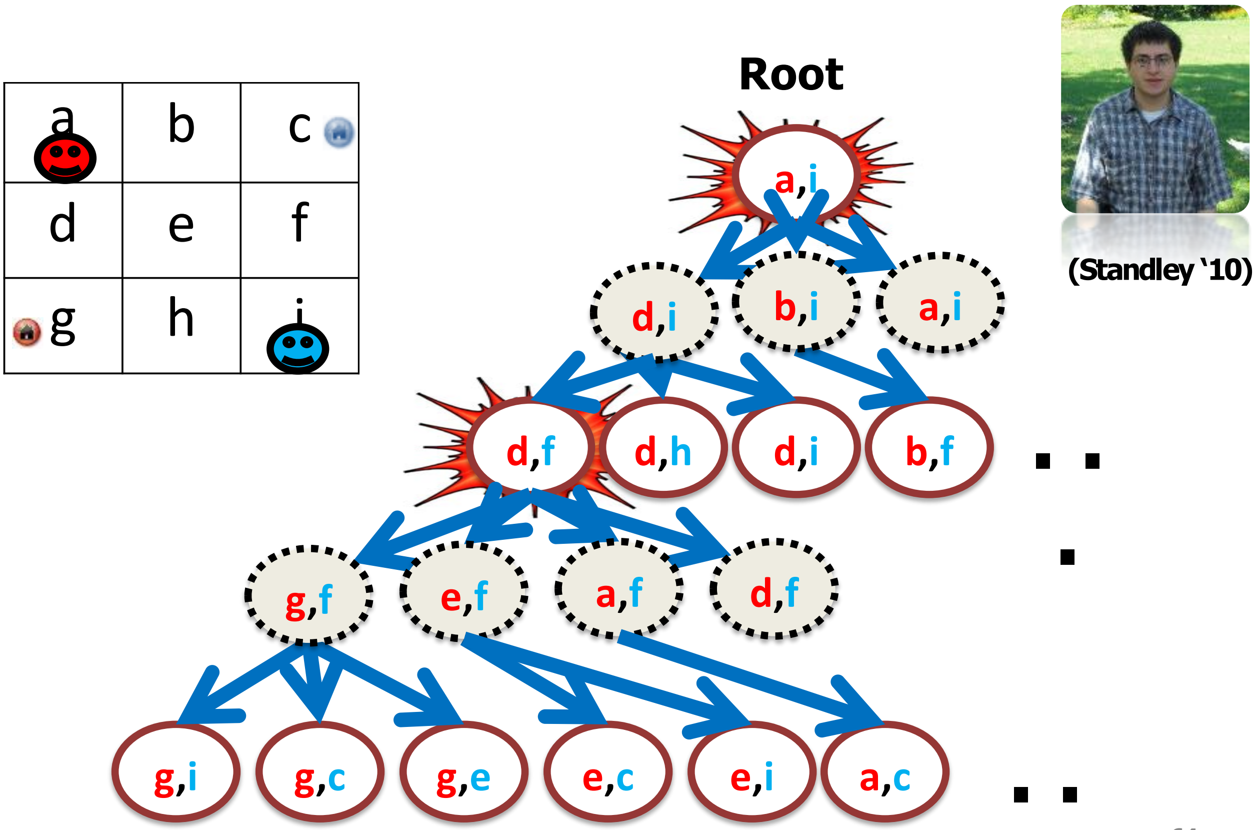
*What is the complexity of expanding*
***a single node*** *in MAPF with 20 agents?*

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





## Search Tree Growth with Operator Decomposition

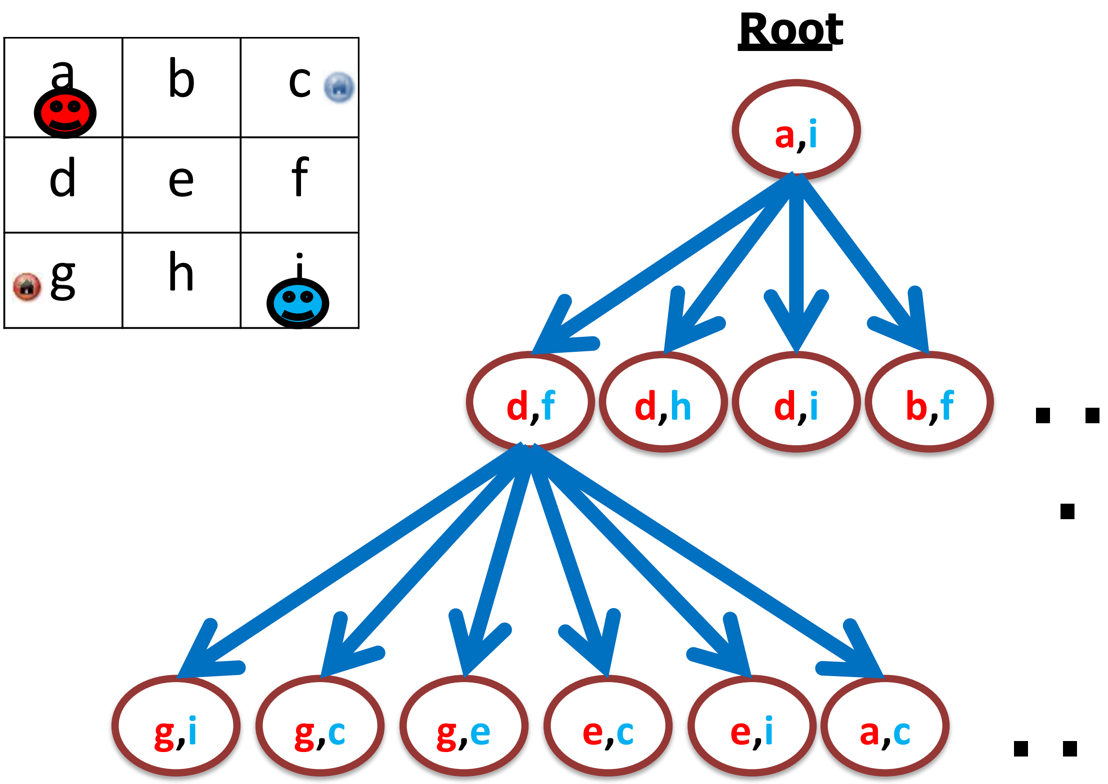





## Analysis of OD

- 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

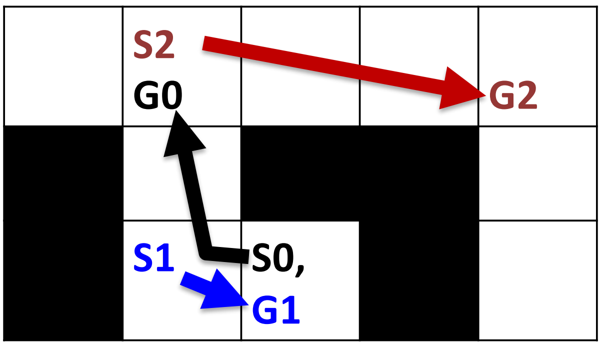
## Enhanced Partial Expansion A* (Felner '12, Goldenberg '14)






## Independence Detection (Standley '10)

Theoretically, a 3 agents problem, but …



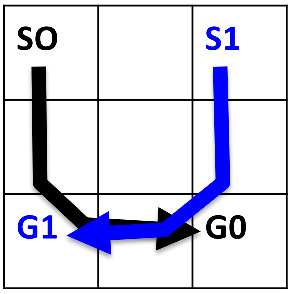


(Standley '10)

**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 …

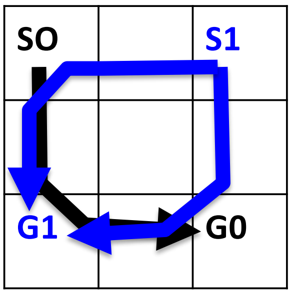



(Standley '10)

**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



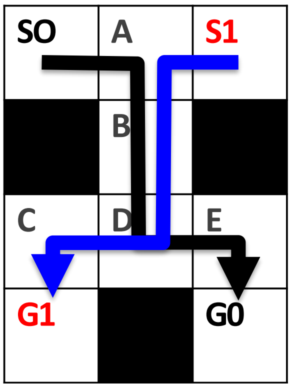
Theoretically, a 2 agents problem, but …




(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

## *Independence Detection (Standley '10)*



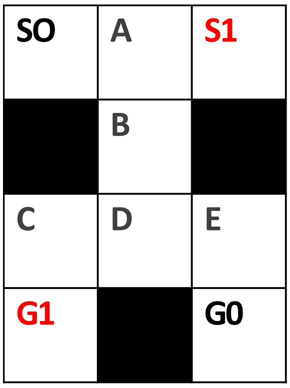

Really a 2 agent problem

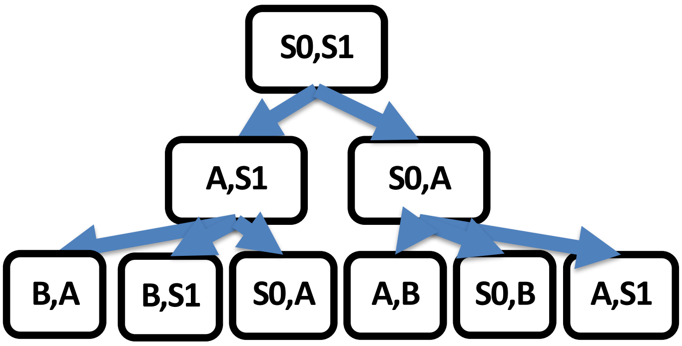
**Independence Detection**

**But….**

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

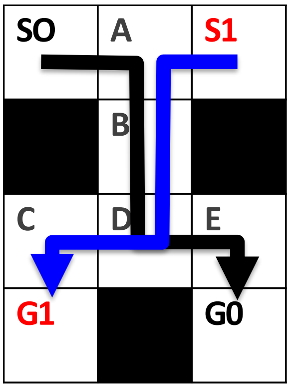
## *M* (Wagner & Choset '11,'14)*



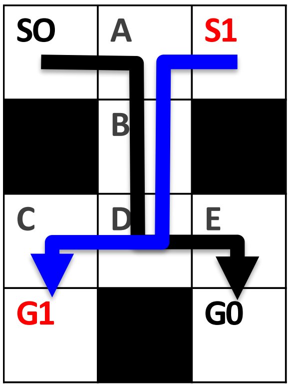

**M***

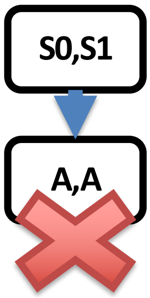
1. Find optimal path for each agent individually
2. Start the search. **Generate only nodes on optimal paths**
3. If **conflict occurs – backtrack** and consider all ignored actions




**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

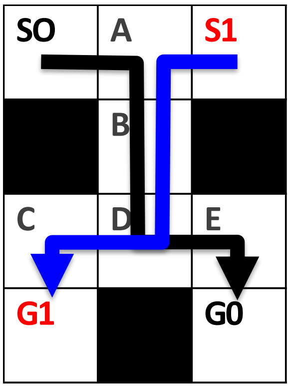


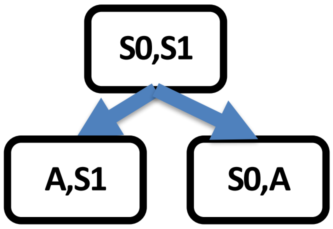



**M***
1. Find optimal path for each agent individually
2. Start the search. **Generate only nodes on optimal paths**
3. If **conflict occurs – backtrack** and consider all ignored actions

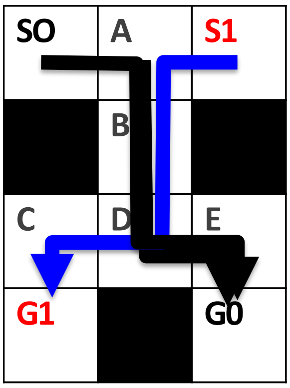
## *M* (Wagner & Choset '11,'14)*

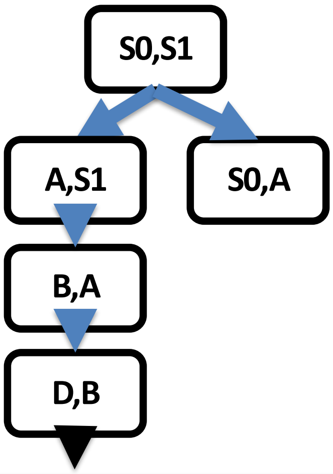



**M***
1. Find optimal path for each agent individually
2. Start the search. **Generate only nodes on optimal paths**
3. If **conflict occurs – backtrack** and consider all ignored actions

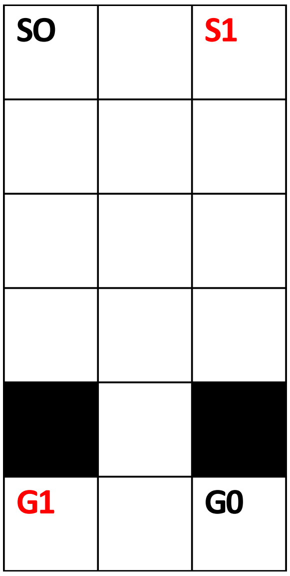
## *Recursive M* (Wagner & Choset '11,'14)*

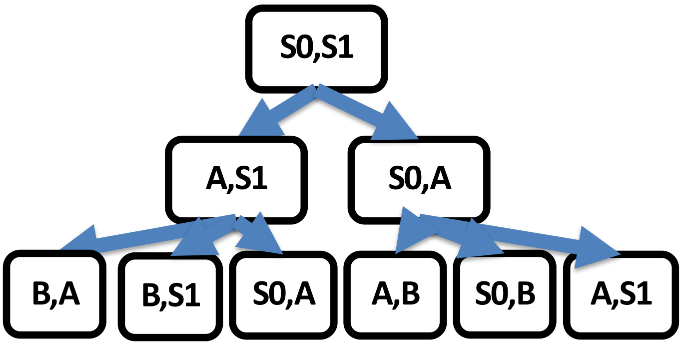



**Recursive M***
1. Find optimal path for each agent individually
2. Start the search. **Generate only nodes on optimal paths**
3. If **conflict occurs – backtrack** and consider all ignored actions
   - Apply M* recursively after backtracking

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




Joint path up to bottleneck can be long…

## *Search-based Approaches to Optimal MAPF*

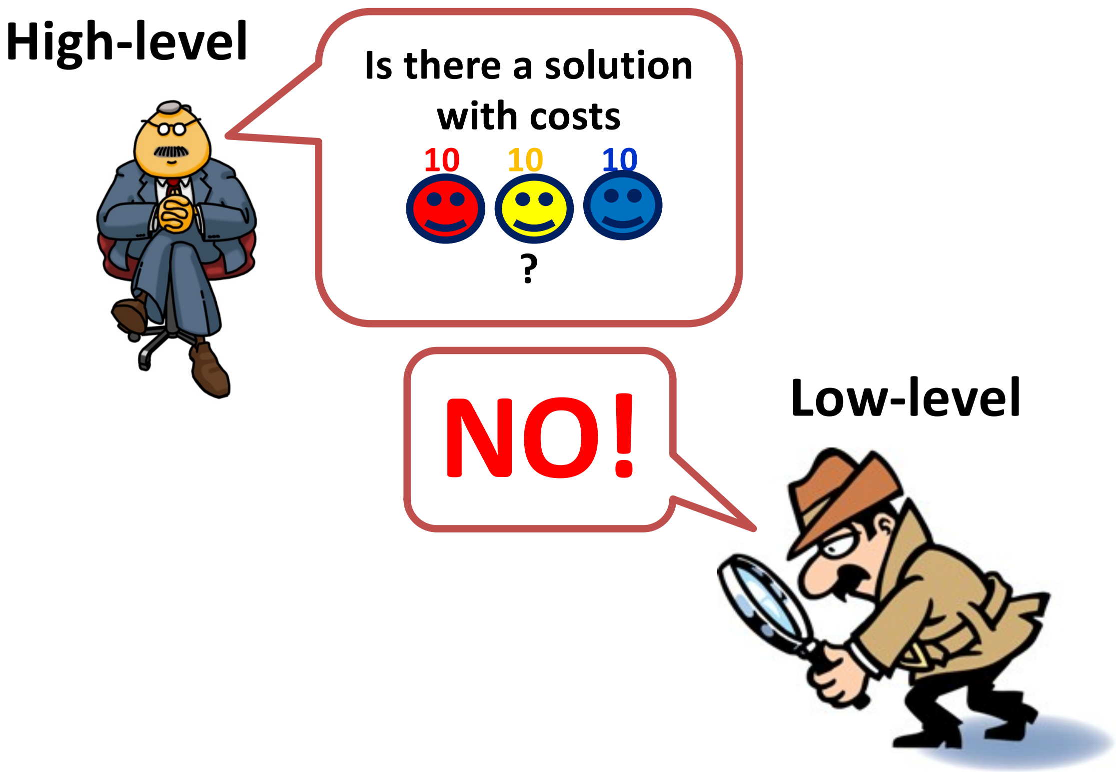
**Searching the k-agent search space**

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

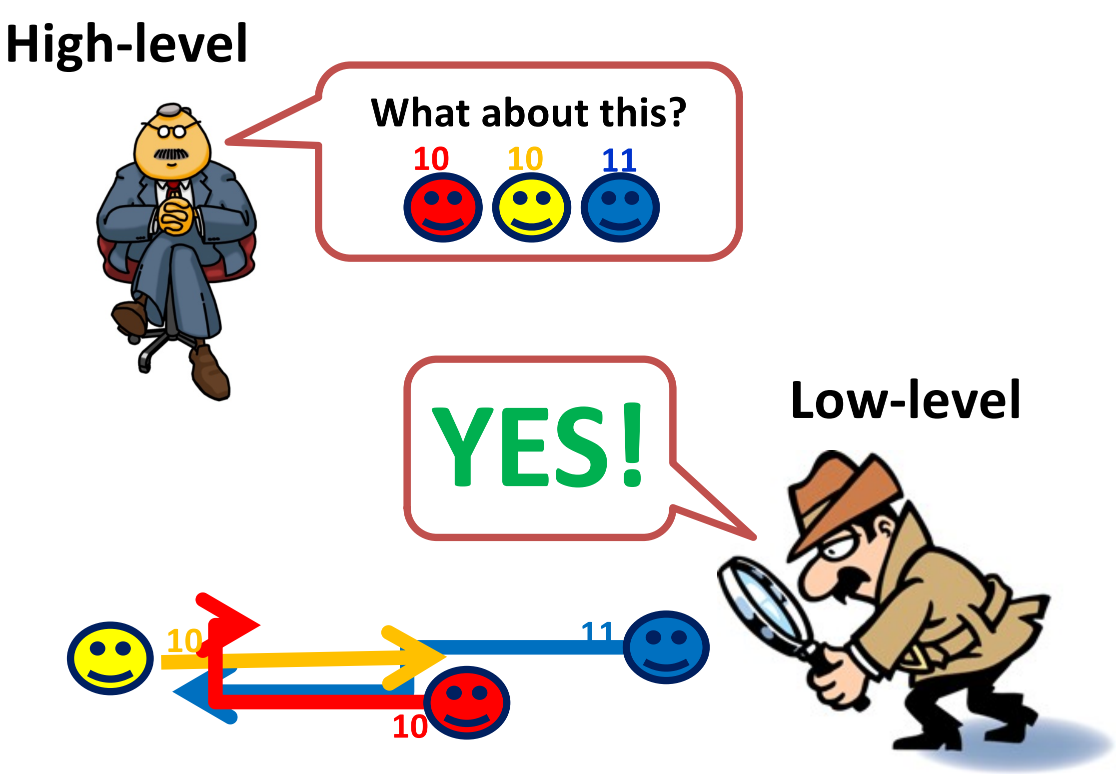
**Other search-based approaches**

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

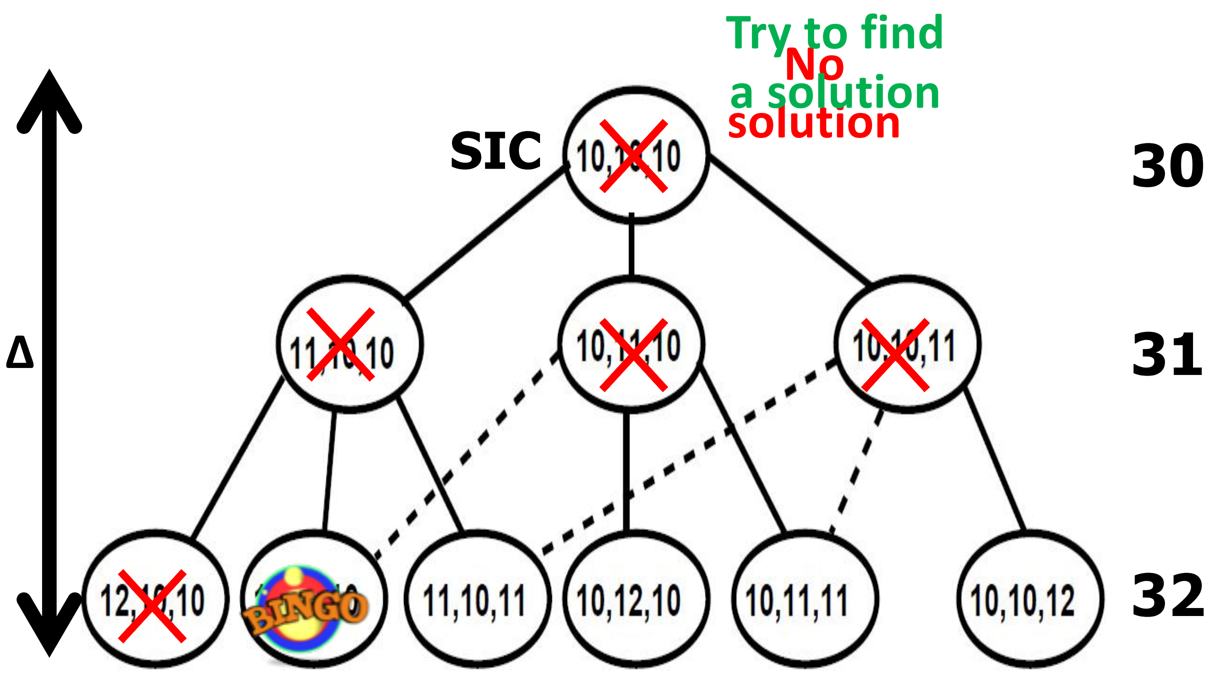
## *Increasing Cost Tree Search* (Sharon et al. '12)



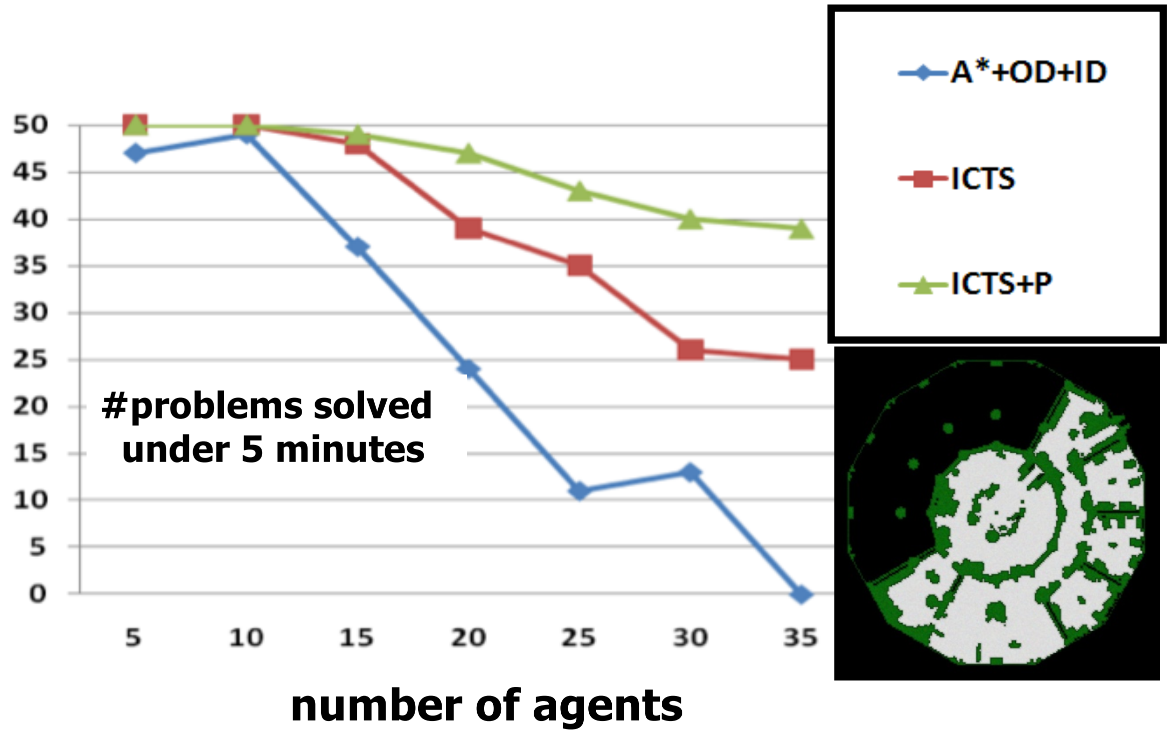

## *Increasing Cost Tree Search* (Sharon et al. '12)



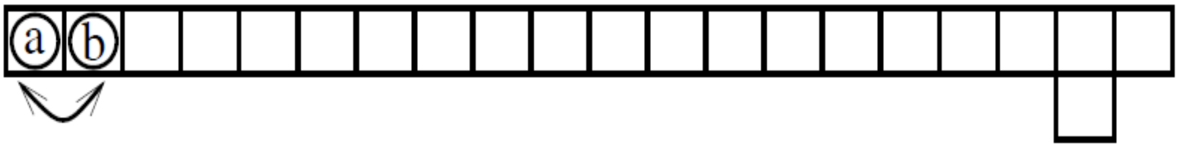
## *Increasing Cost Tree Search (Sharon et al. '12)*



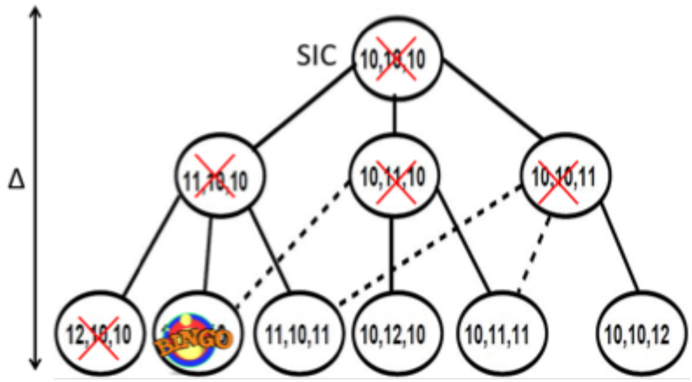

## *Does it work? – YES!*



## *Does it work? – Not Always*

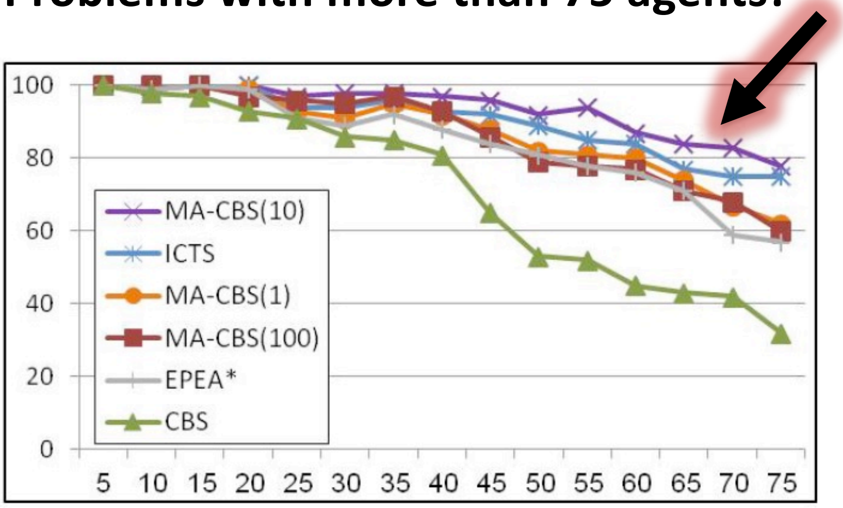



- A*: solved in **51**ms
- **ICTS Complexity depends on Δ**
  - Sum of single agent costs =2 **BUT** optimal solution =74



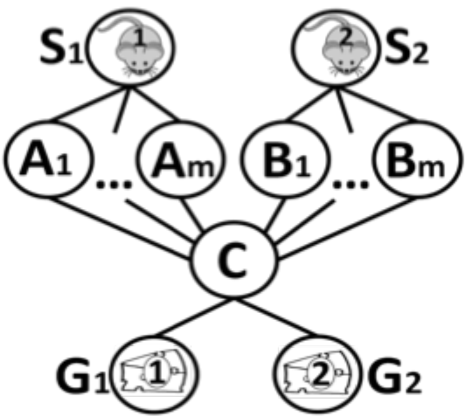

---

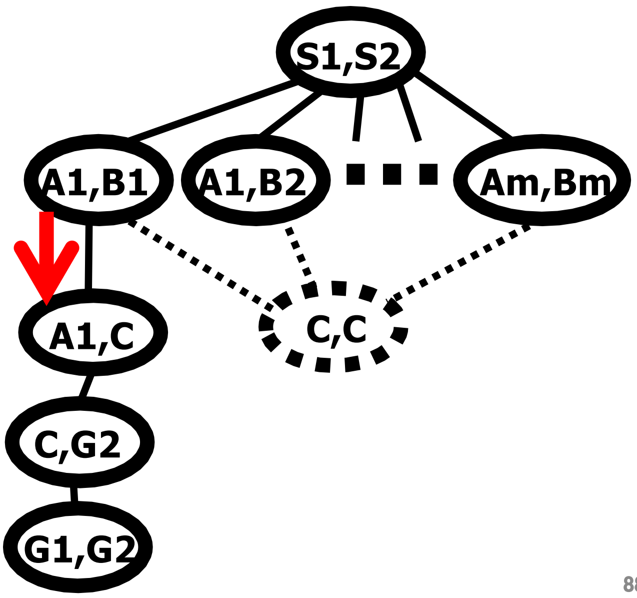
**Solving Optimally Problems with more than 75 agents!**



## 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**





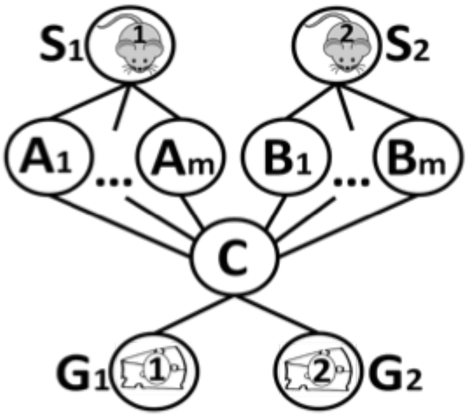



**A* and ICTS work in a K-agent search space**

**CBS plans for single agents but under constraints**

- **Conflict**: [agent A, agent B, location X, time T ]
- **Constraint**: [agent A, location X, time T]

Resolve conflict by imposing [**S1,C,2**] or [**S2,C,2**]








- **Conflict**: [agent A, agent B, location X, time T ]
- **Constraint**: [agent A, location X, time T]

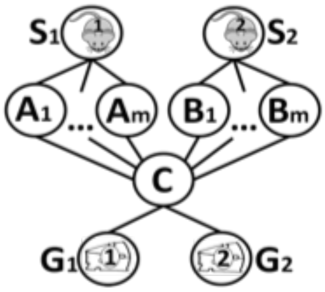
Resolve conflict by adding [**A,X,T**] or [**B,X,T**]

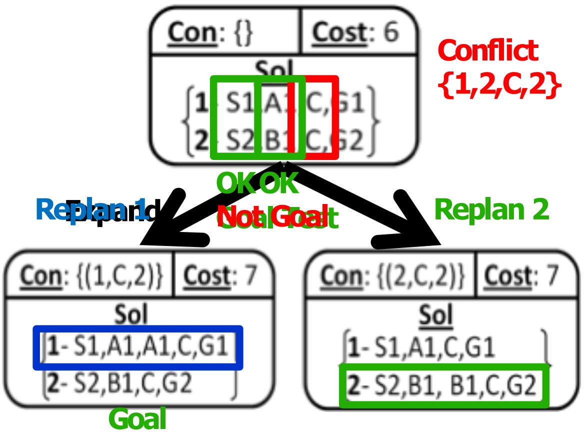
**CBS: general idea**

1. **Plan for each agent individually**
2. **Validate plans**
3. **If the plans of agents A and B conflict**
   **Constrain A to avoid the conflict**
   ***or***
   **Constrain B to avoid the conflict**

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

**Goal test:**
- Are the paths conflict free

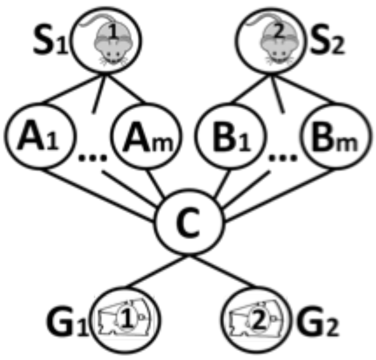




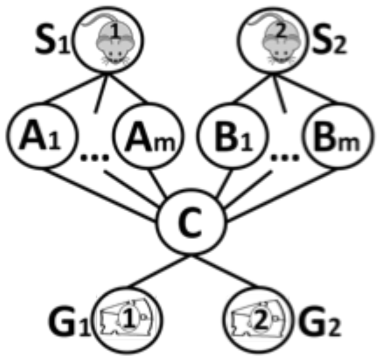



- How many states **A*** will expand?
- How many states **CBS** will?



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

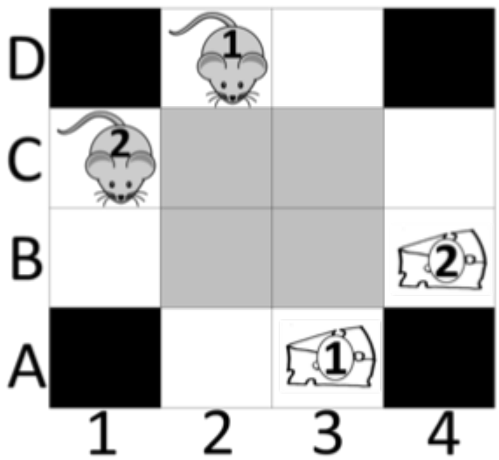
**When m > 4 CBS will examine fewer states than A***



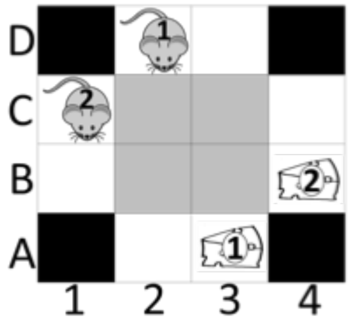





- States expanded by CBS**?**
- States expanded by A***?**



## Analysis: Example 2




- 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

**Trends observed**
- **In open spaces: use A***
- **In bottlenecks: use CBS**

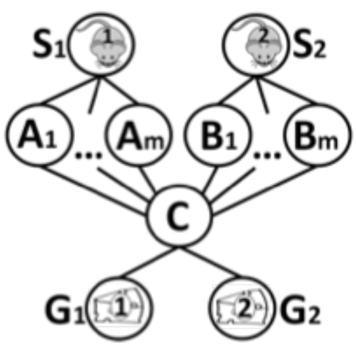
**What if I have both?**

## Meta-Agent CBS (MA-CBS)

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)
Standley's ID

**MA-CBS**

(never merge) T=∞
basic CBS

**Many bottlenecks**

brc202d

| k | 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

| | 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 |

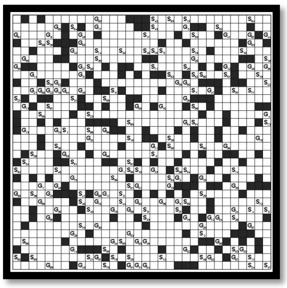
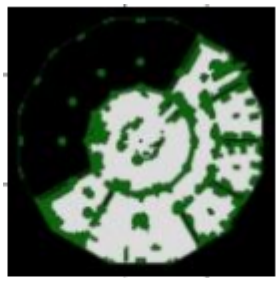
**Many bottlenecks → High T (closer to CBS)**
**More agents → Low T (closer to A*)**
**Faster single-agent search → lower T (close to A*)**



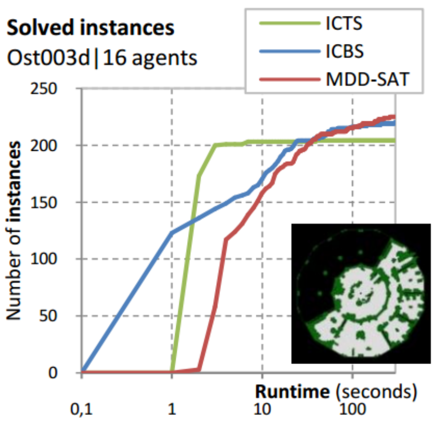
- When to merge agents?
- What to do after merging? [Boyarski et al. ‘16]
- Which conflict to resolve? [Boyarski et al. ‘16]
- How to resolve it?
- Which low-level solver to use?
- Heuristics for the constraint tree search [Ma et al. ‘18]
- …

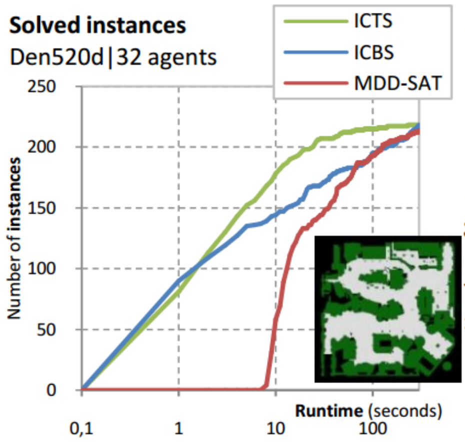
- A* (M*, EPEA*, A*+OD+ID)
  - Main factors: #agents, graph size, heuristic accuracy
- ICTS
  - Main factors: #agents, Δ, graph size
- CBS and its variants
  - Main factors: #conflicts

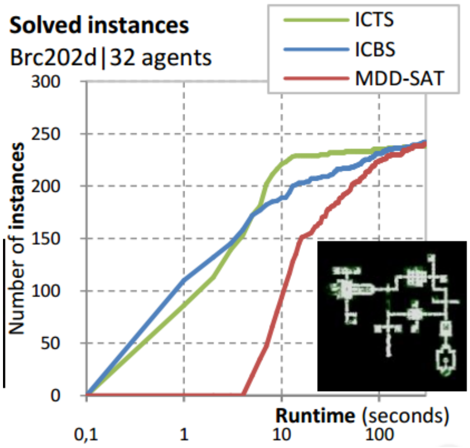
**Where to use what?**

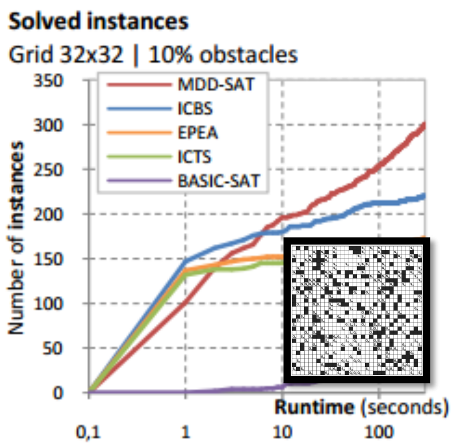







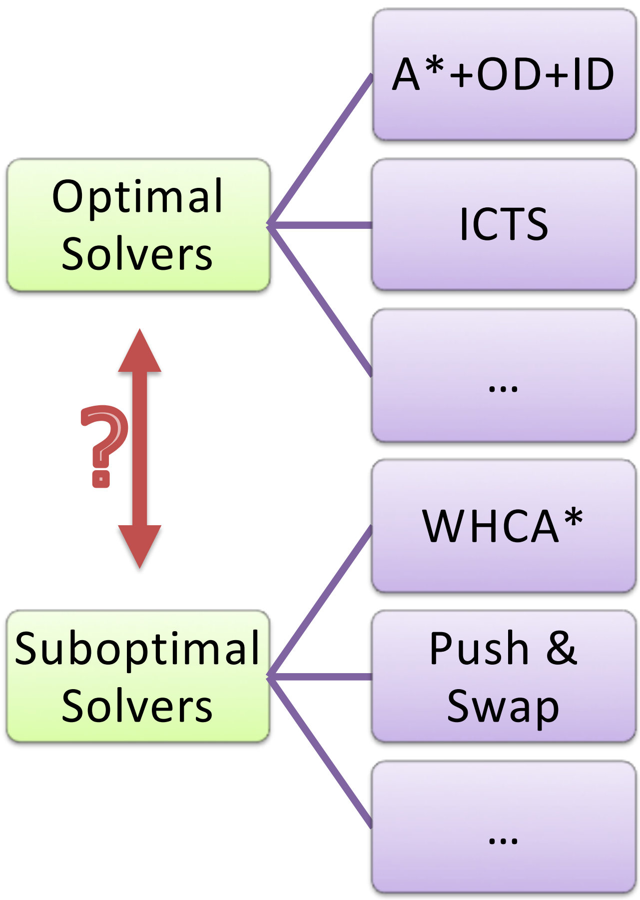



## *Search-based Solvers - Summary*

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

## *Solving MAPF*



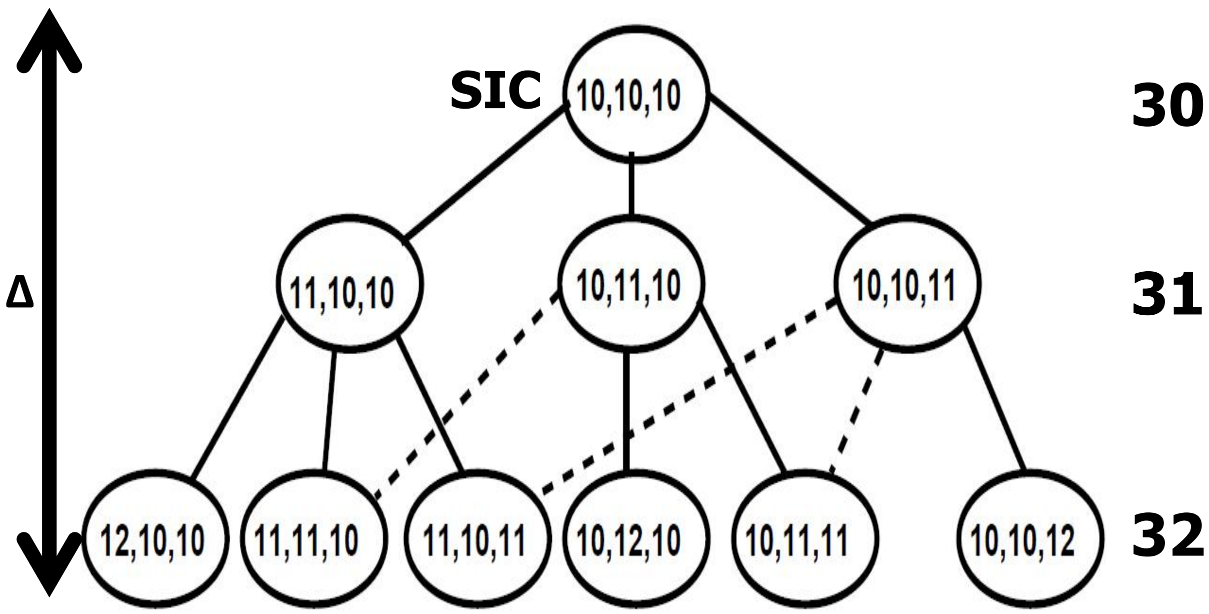
**An algorithm is bounded suboptimal iff**

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

How to create a bounded suboptimal algorithm?

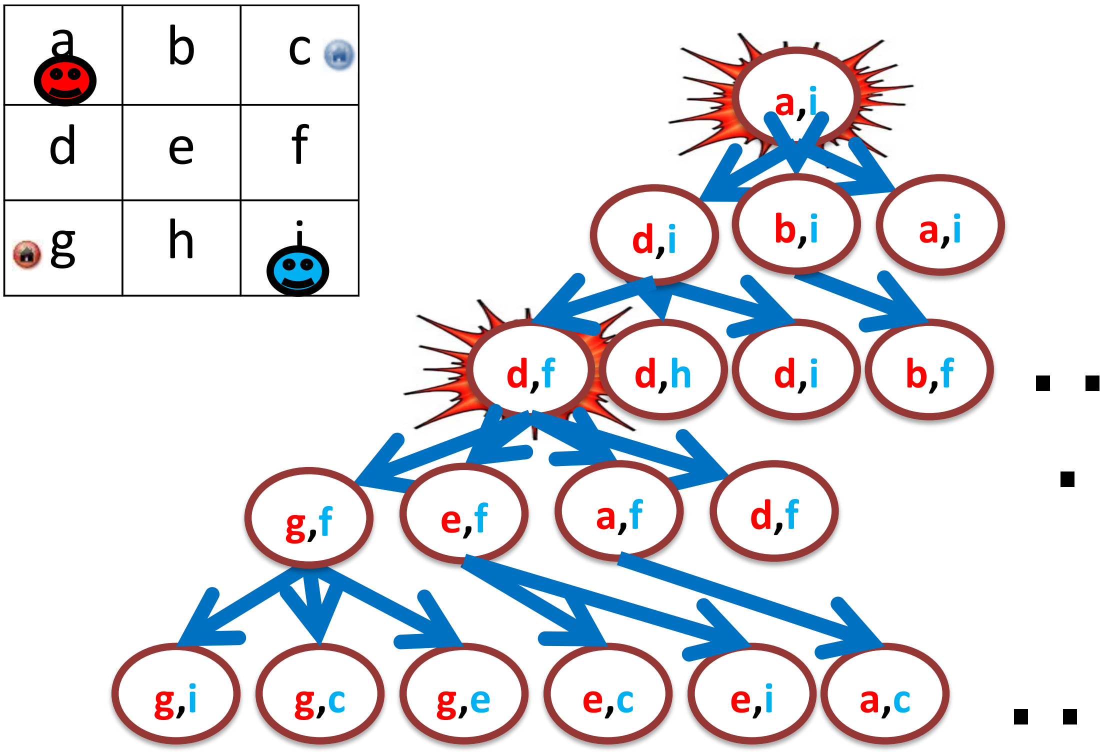
- Different search algorithms
- Inadmissible heuristics



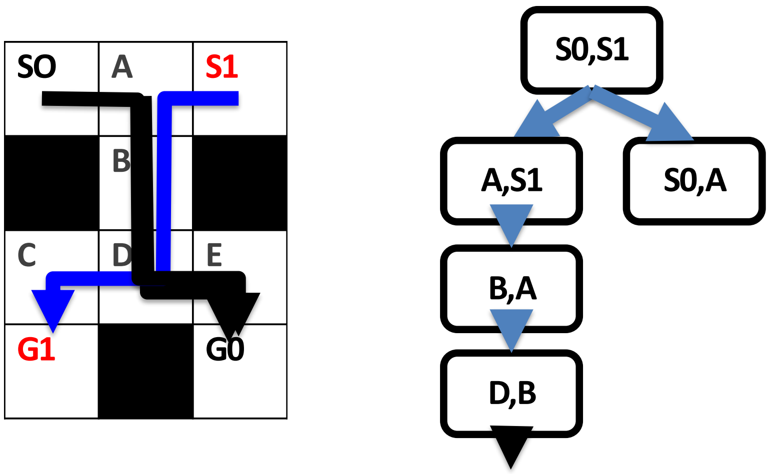


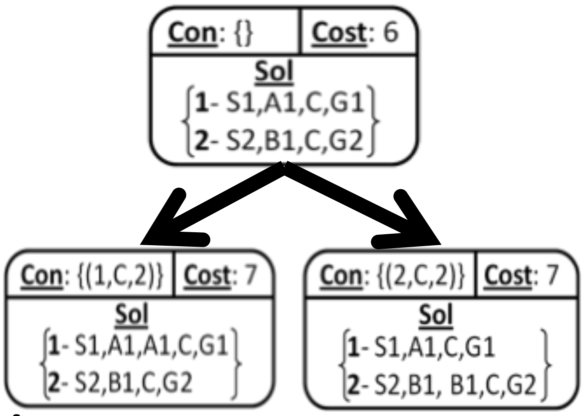

**Open Question!**

## *Suboptimal A**




## *Suboptimal rM**





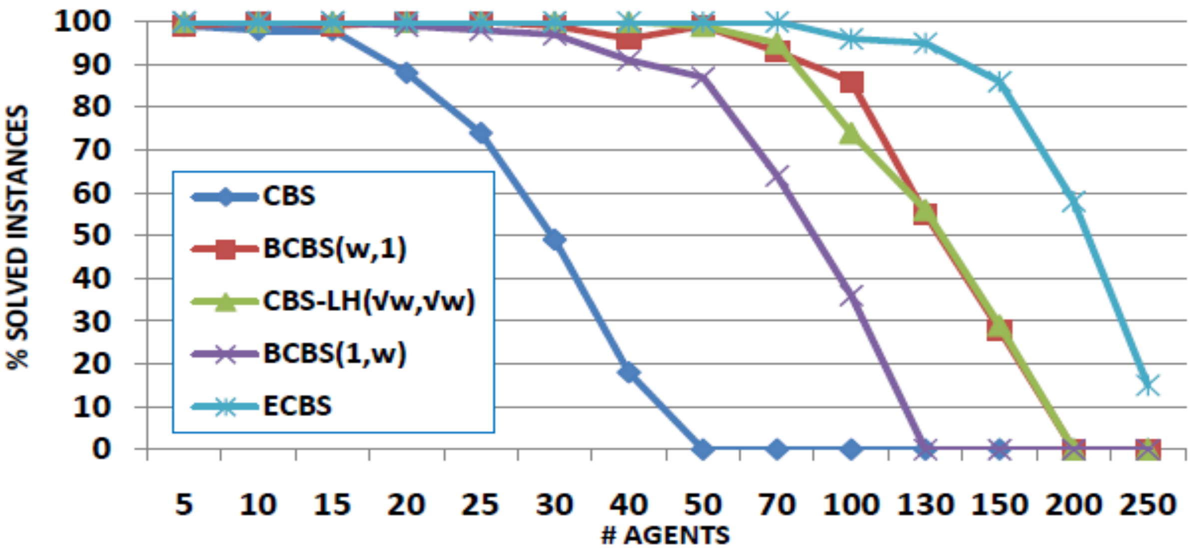

**Observation:**

**Suboptimality can be introduced in both levels**

– ECBS (Barer et al. '14)

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





## *Advanced Issues in Search-based MAPF Algorithms*

- When to use which algorithm? Ensembles?
- Using **knowledge** about past plans [Cohen et al.]
- 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)
- 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<br>(at-least-one(Bs)) | disj(Bs) |
| sum(Bs) = 1 | at-most-one(B) *and* at-least-one(B) |

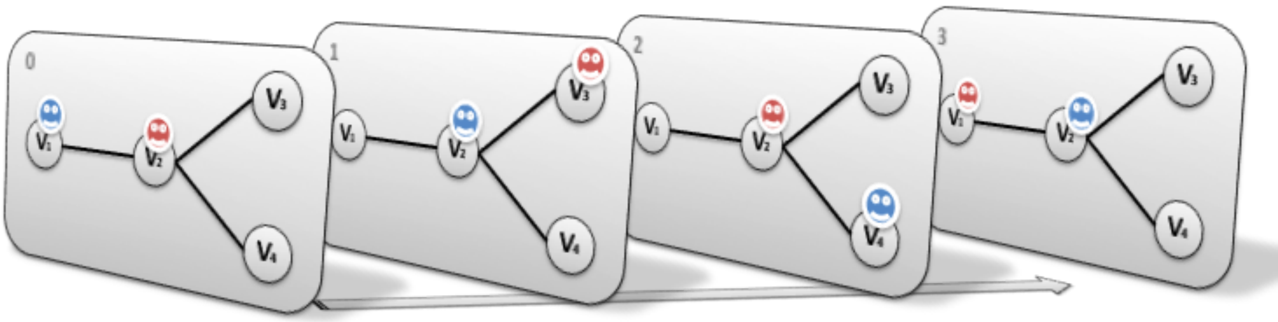
We can even use **numerical variables** (and constraints).

## SAT encoding: core idea

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.




[Surynek, ICTAI 2012]

## SAT encoding with all-different

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


- Agent waits or moves to a neighbor

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

- No-train constraint

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

- Agents are not at the same nodes

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

# 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

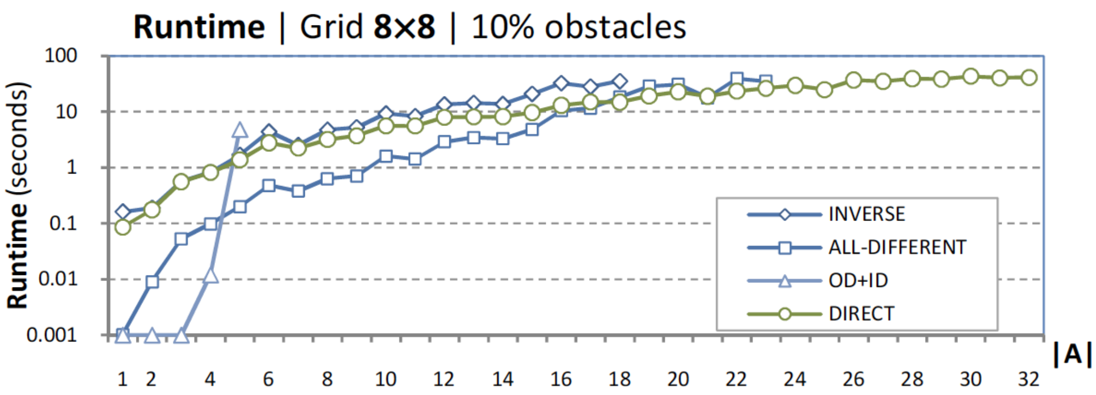
$$\mathcal{X}_{j,k}^{i} \Rightarrow \mathcal{X}_{j,k}^{i+1} \vee \bigvee_{l:\{v_j,v_l\}\in E} \mathcal{X}_{l,k}^{i+1} \qquad \mathcal{X}_{j,k}^{i+1} \Rightarrow \mathcal{X}_{j,k}^{i} \vee \bigvee_{l:\{v_j,v_l\}\in E} \mathcal{X}_{l,k}^{i}$$

- 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



## Mixed model

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

## Picat vs. rest

| 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

## Objectives in SAT

**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]

## *Introduction to CP*

Express the problem as a **constraint satisfaction problem:**

- finite domain variables
- constraints = relations between the variables
- solution = instantiation of variables satisfying all the constraints

Modeling (choice of constraints) is important.

*Example:*

```
E,N,D,O,R,Y in 0..9,
S,M in 1..9,
P1,P2,P3 in 0..1
   D+E = 10*P1+Y
P1+N+R = 10*P2+E
P2+E+O = 10*P3+N
P3+S+M = 10*M +O
all_different(S,E,N,D,M,O,R,Y)
```

```
    S E N D
+   M O R E
= M O N E Y
```

```
    9 5 6 7
+   1 0 8 5
= 1 0 6 5 2
```

## *CP vs. SAT*

Every SAT model is also a CP model.

CP models support **numerical variables and constraints** directly.

CP solvers are based on interleaving local consistency and search

**Consistency techniques** remove inconsistent values

```
all-different({1,2},{1,2},{1,2,3})
-> all-different({1,2},{1,2},{3})
```

**Global constraints** introduce “specialized” solvers into general CP framework

e.g. all-different is based on pairing in bipartite graphs

## CP models: core idea

Separate path planning (which nodes are visited) and time scheduling (when the nodes are visited):

- **find a path for each agent** (planning)
  each agent needs to get from its origin to destination
- **ensure that paths are collision free** (scheduling)
  no two agents meet at the same time at the same node

It is natural to include:

- different **durations** of actions (e.g. different distances between the nodes)
- **capacities** of edges and nodes

## CP models for MAPF

Two versions of the MAPF:

- **no re-visits** allowed (restricted MAPF)
  - flow, path, and scheduling models

  Can be modeled directly as a single CSP (we know the maximum length of plans)
- **re-visits** allowed (classical MAPF)
  - scheduling model with optional activities

  Layered model based on the number of re-visits.

[Barták et al, MISTA 2017]

## Restricted MAPF: flow model

Based on network flows

### Path planning

if agent enters the node, it must also leave it (flow preservation constraint)

$$\forall x \in V \setminus \{orig(p)\} : \sum_{a \in InArcs(x)} Used[a,p] = Flow[x,p]$$

$$\forall x \in V \setminus \{dest(p)\} : \sum_{a \in OutArcs(x)} Used[a,p] = Flow[x,p]$$

### Scheduling

time intervals spent in a node do not overlap

$$(Flow[x,p_1] \land Flow[x,p_2]) \Rightarrow (OutT[x,p_1] < InT[x,p_2] \lor OutT[x,p_2] < InT[x,p_1])$$

### Temporal constraints

$$Used[a,p] \Rightarrow OutT[x,p] + w(a) = InT[y,p].$$

$$InT[x,p] \leq OutT[x,p].$$

[Barták et al, MISTA 2017]

## Restricted MAPF: path model

Based on covering by cycles

### Path planning

each node has predecessor and successor

$$Prev[x,p] = y \Leftrightarrow Next[y,p] = x.$$

### Scheduling

time spent in a node modeled as activity N

$$NoOverlap(\bigcup_{p \in P} N[x,p]).$$

### Temporal constraints

$$EndOf(N[x,p]) + w(x, Next[x,p]) = StartOf(N[Next[x,p],p]),$$

[Barták et al, MISTA 2017]

## Restricted MAPF: scheduling (opt) model

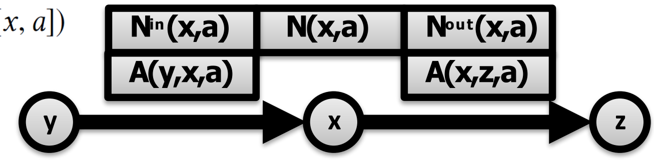
**Based on optional activities**

### Path planning

Activities for traversing arcs and visiting nodes

$$PresenceOf(N[x,a]) \Leftrightarrow PresenceOf(N^{in}[x,a])$$

$$Alternative\left(N^{in}[x,a], \bigcup_{(y,x)\in E} A[y,x,a]\right)$$




### Scheduling

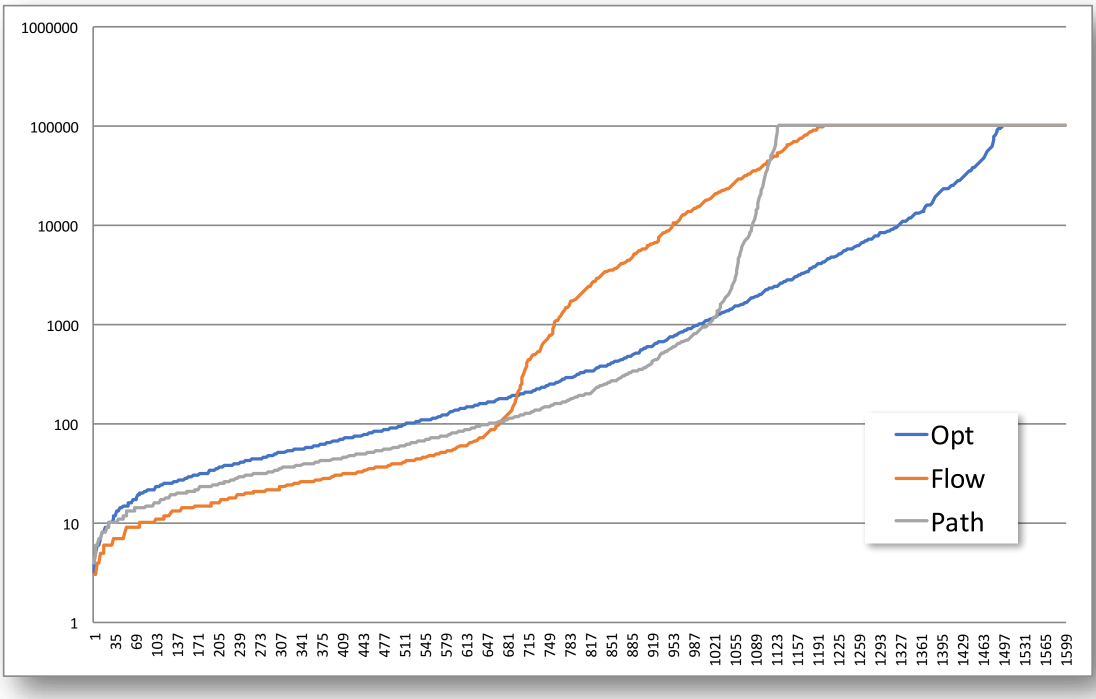
$$NoOverlap\left(\bigcup_{a\in A} N[x,a]\right)$$

### Temporal constraints

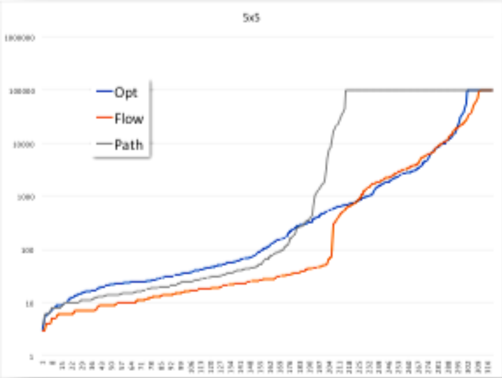
$$StartOf(N[x,a]) = EndOf(N^{in}[x,a])$$

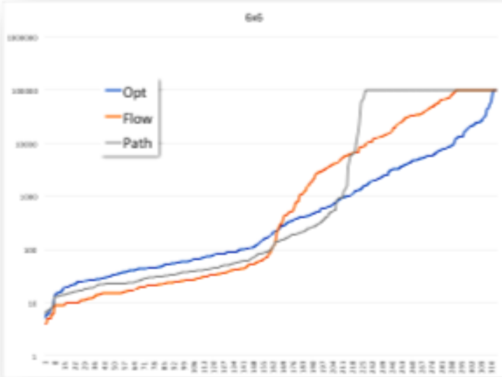
$$EndOf(N[x,a]) = StartOf(N^{out}[x,a])$$

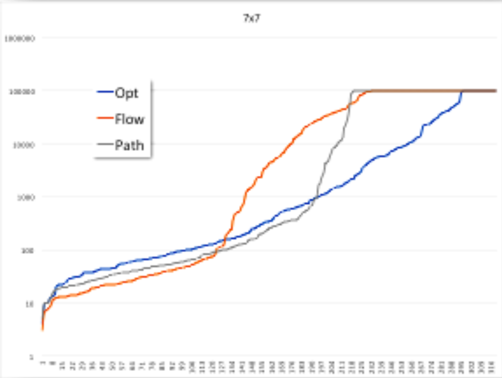
## Comparison of CP models

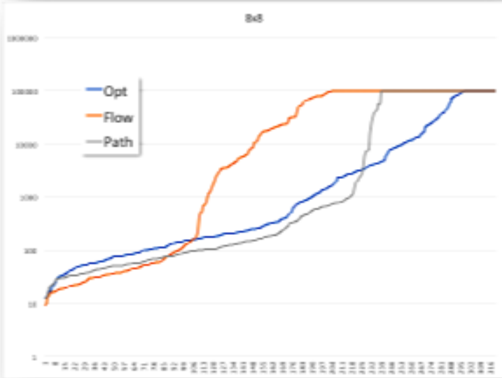


## Comparison of CP models (map size)

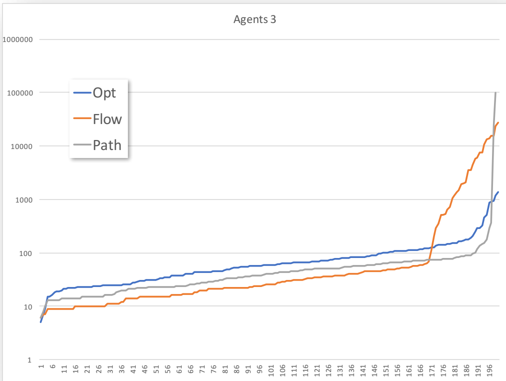


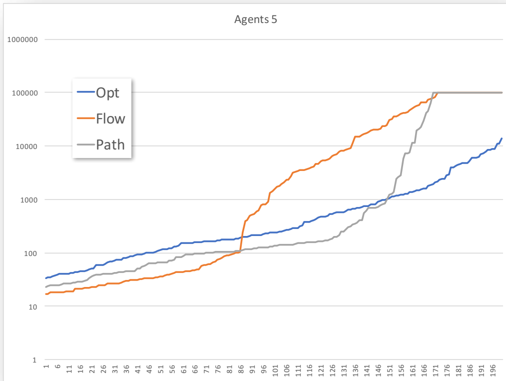



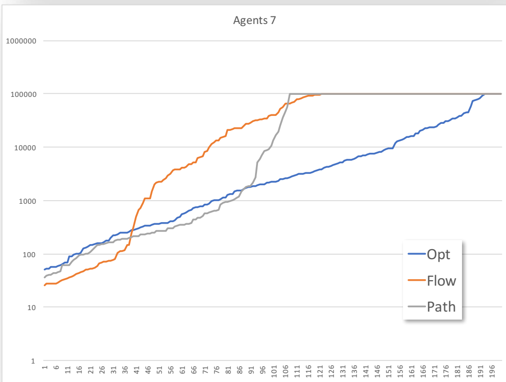



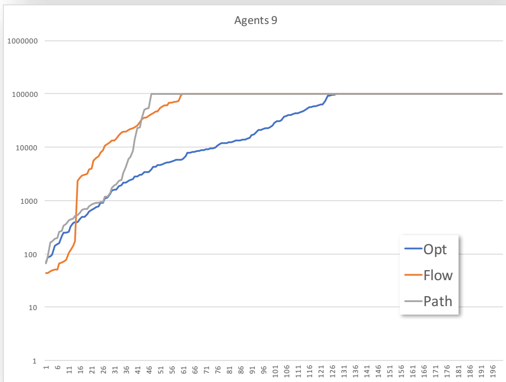




## Comparison of CP models (#agents)



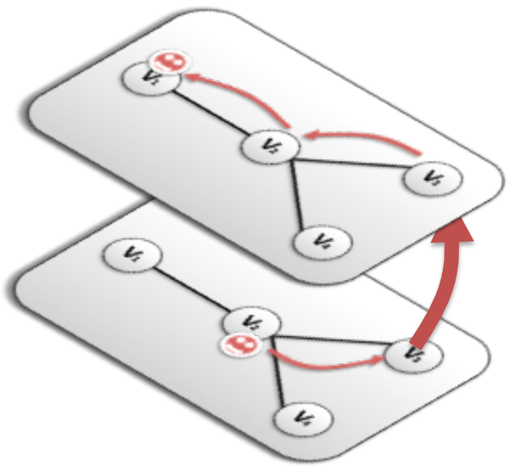



[Barták et al, AAMAS 2018]

## *Multi-layer model*

SAT uses layers to encode time slices (number of layers = makespan)

CP uses **layers to encode re-visits** of nodes (number of layers = number of re-visits)



using activities for nodes and arcs

N(x,a) → N(x,a,k)

A(y,x,a) → A(y,x,a,k)

transitions to next layers via A(x,x,a,k)

[Barták et al, AAMAS 2018]

## *Number of layers*

Upper bound for the number of layers:

Upper bound on makespan

Length of the shortest path, over all agents, from the origin node to the destination node

$$\frac{MB - p_{min}}{2} + 1$$

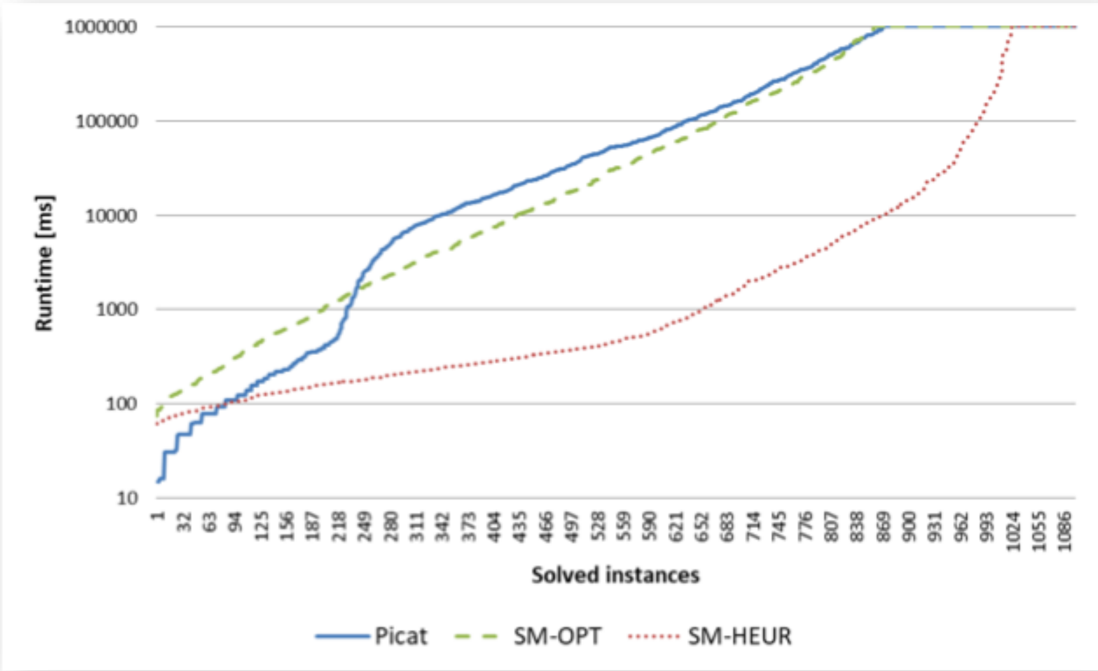
Could be a huge number (leading to a big model).

Layers can be incrementally added until a solution is found.

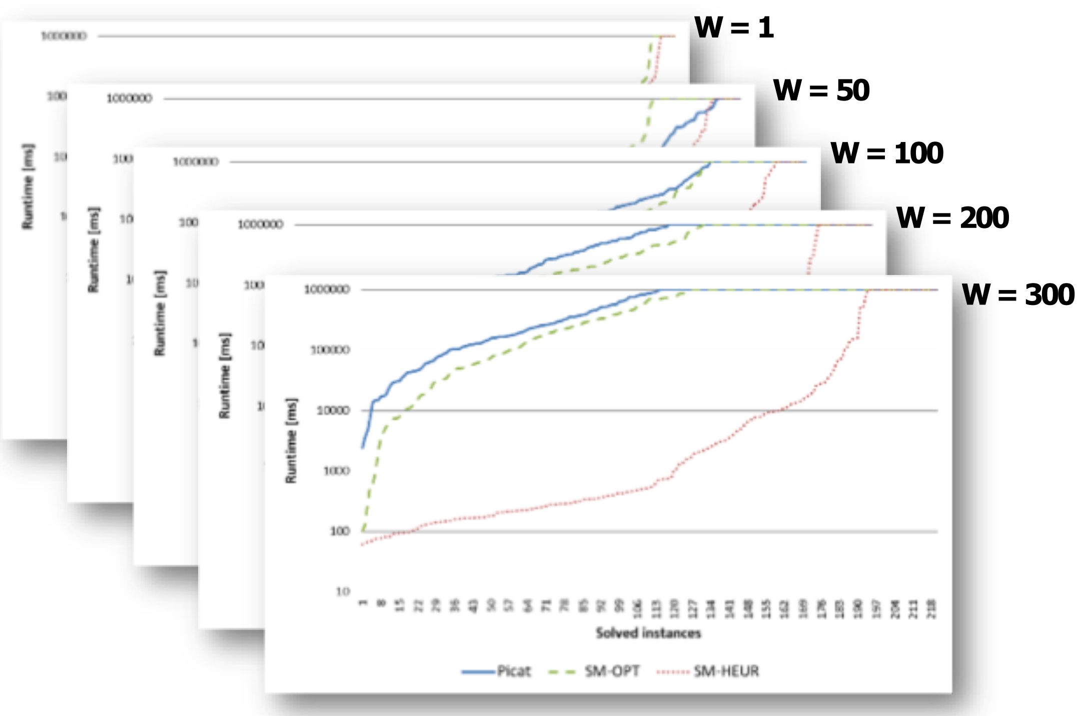
Makespan of the solution can used to estimate the number of layers (if we optimize makespan).

[Barták et al, AAMAS 2018]

## Model comparison




## Model comparison (length of arcs)

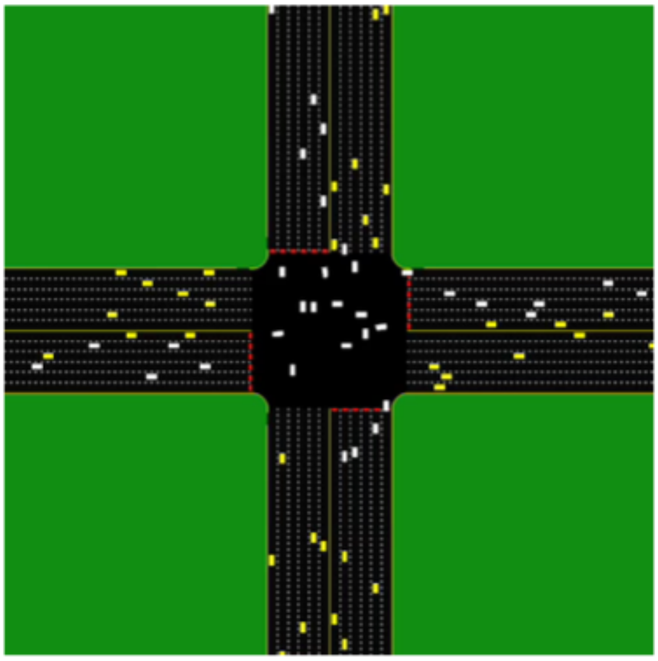


Part IV:

# FROM PLANNING TO EXECUTION

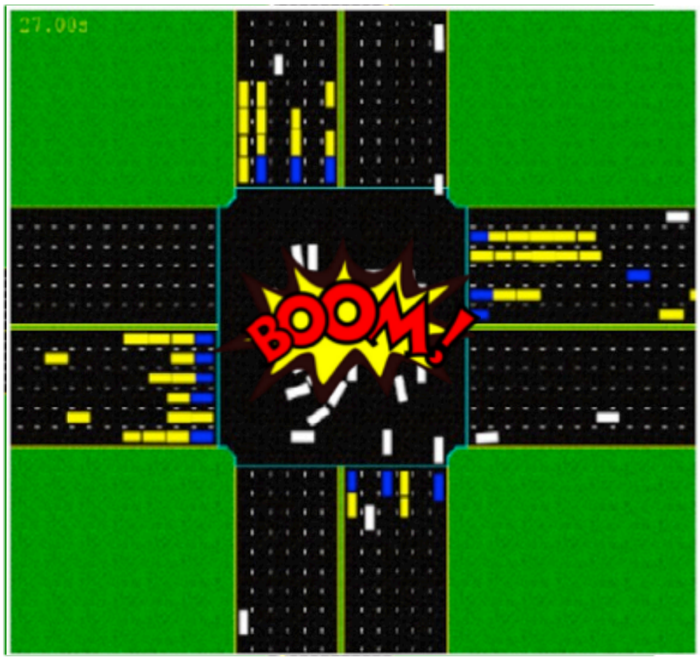
*Man (or AI) Make Plans and God Laughs*

## *Automatic Intersection Manager*



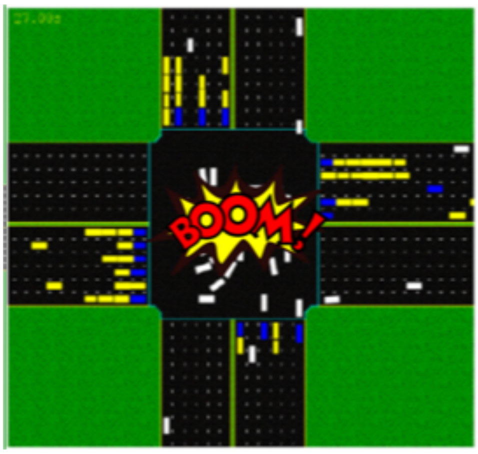
(Stone et al., UT Austin)

## *Automatic Intersection Manager*



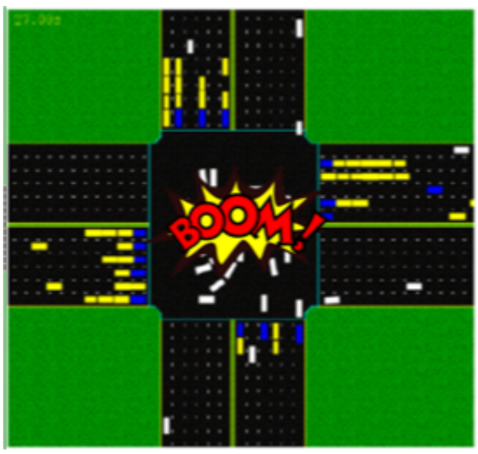

**Who is to blame?**
[Elimelech et al. ‘17]

- How to **react** when an unplanned event occur?
- How to **plan a-priori** if we know such events may occur?





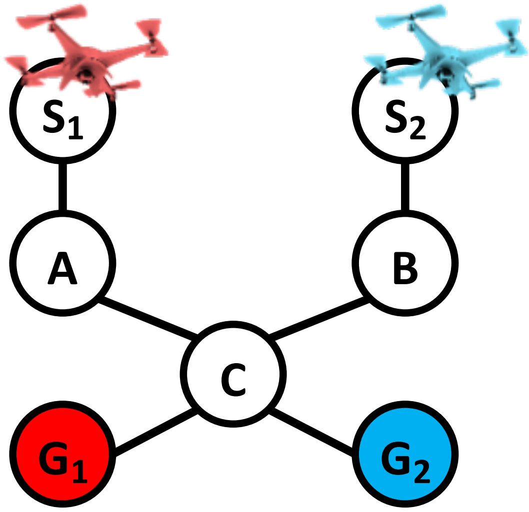

- How to **react** when an unplanned event occur?
- How to **plan a-priori** if we know such events may occur?



## Running Example – the Plan

Plan

| | | | | | |
|---|---|---|---|---|---|
| **Red** Agent | S1 | A | A | C | G1 |
| **Blue** Agent | S2 | B | C | G2 | G2 |

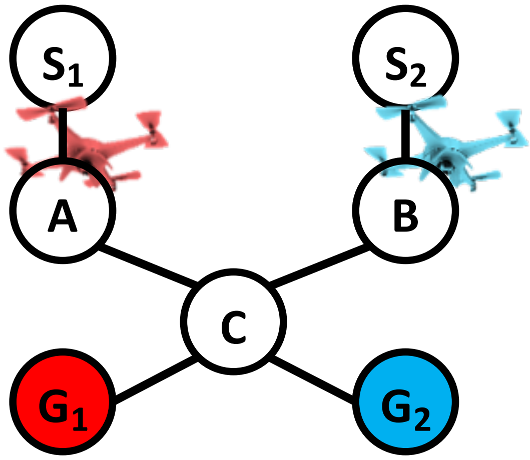




## Running Example – the Plan

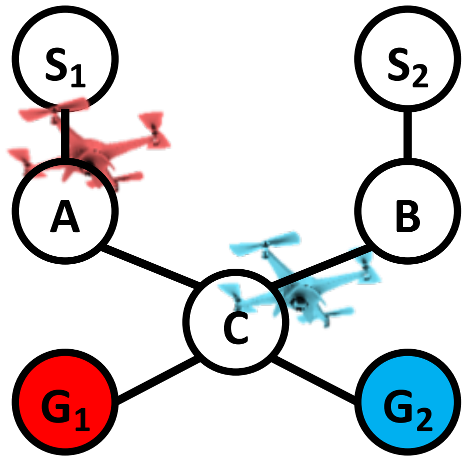

Plan

| | | | | | |
|---|---|---|---|---|---|
| **Red** Agent | S1 | A | A | C | G1 |
| **Blue** Agent | S2 | B | C | G2 | G2 |



## Plan

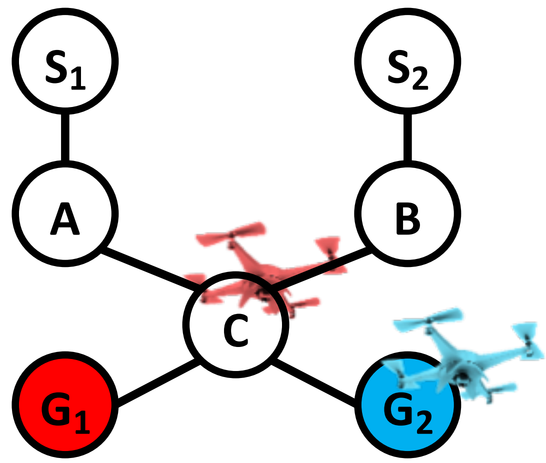
| | | | | | |
|---|---|---|---|---|---|
| **Red** Agent | S1 | A | A | C | G1 |
| **Blue** Agent | S2 | B | C | G2 | G2 |






## Plan

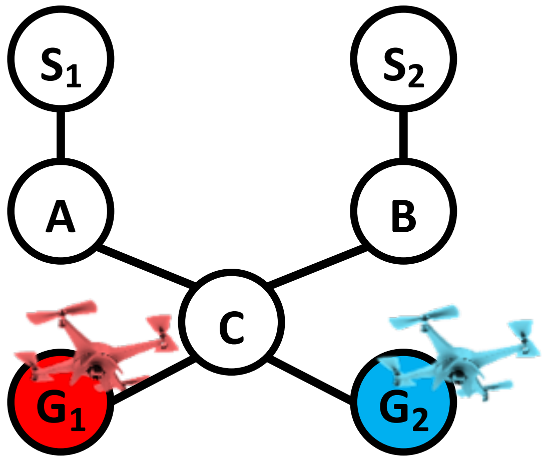
| | | | | | |
|---|---|---|---|---|---|
| **Red** Agent | S1 | A | A | C | G1 |
| **Blue** Agent | S2 | B | C | G2 | G2 |



## *Running Example – the Plan*

Plan

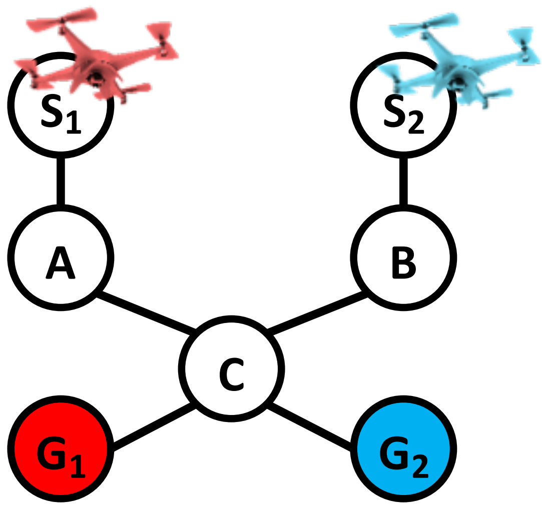
| | | | | | |
|---|---|---|---|---|---|
| **Red** Agent | S1 | A | A | C | G1 |
| **Blue** Agent | S2 | B | C | G2 | G2 |




## *Running Example - Execution*

Exec.

| | | | | | |
|---|---|---|---|---|---|
| **Red** Agent | S1 | A | A | C | G1 |
| **Blue** Agent | S2 | B | C | G2 | G2 |

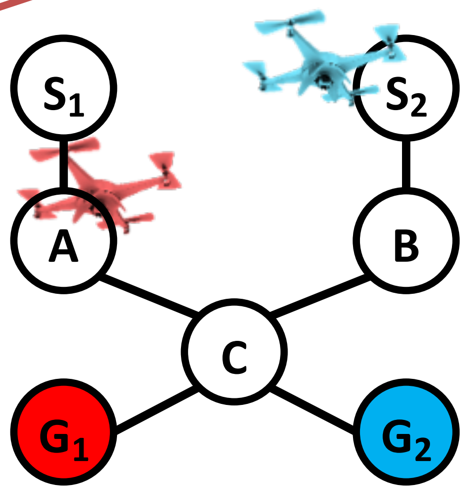


## *Running Example*

**Exec.**

| | | | | | | |
|---|---|---|---|---|---|---|
| **Red** Agent | S1 | A | A | C | G1 | |
| **Blue** Agent | S2 | **S2** | B | C | G2 | G2 |

**Unexpected delay ☹**



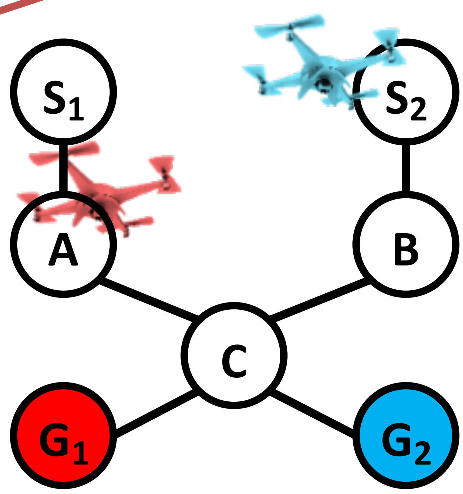

## *Running Example*

**Exec.**

| | | | | | | |
|---|---|---|---|---|---|---|
| **Red** Agent | S1 | A | A | **C** | G1 | |
| **Blue** Agent | S2 | **S2** | B | **C** | G2 | G2 |

**Unexpected delay ☹**

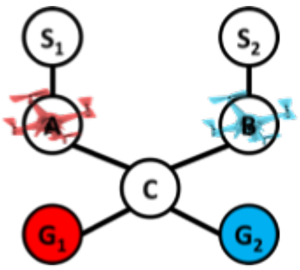
**Potential future conflict**

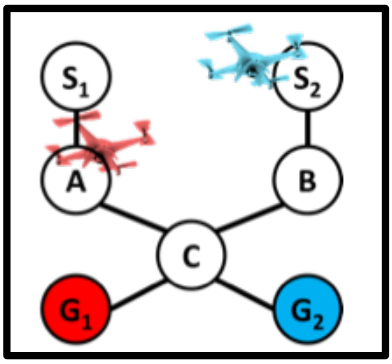


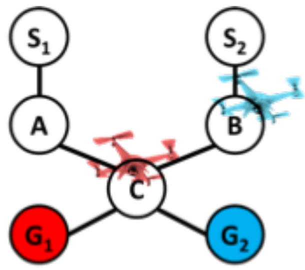
## Repair or Replan?

**Repair** the existing plan

**Replan** a new plan




Wait!

| | | | | | | |
|---|---|---|---|---|---|---|
| Red Agent | S1 | A | A | A | C | G1 |
| Blue Agent | S2 | S2 | B | C | G2 | G2 |

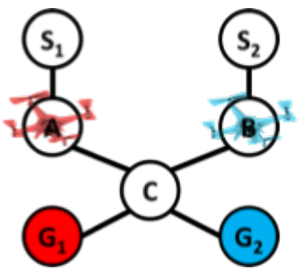
New plan

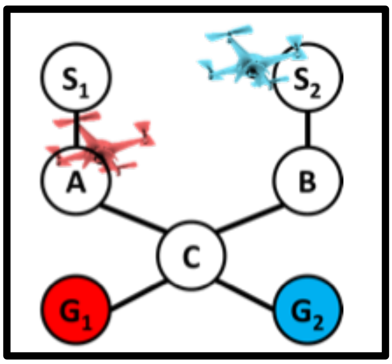
| | | | | | | |
|---|---|---|---|---|---|---|
| Red Agent | S1 | A | C | G1 | | |
| Blue Agent | S2 | S2 | B | C | G2 | G2 |

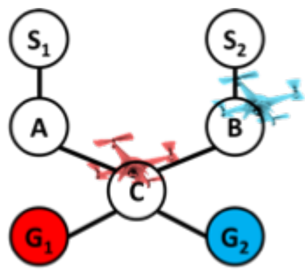

## Repair or Replan?

**Repair** the existing plan

**Replan** a new plan


+ Fast to compute (O(1))
+ Fewer messages
- Solution quality may vary

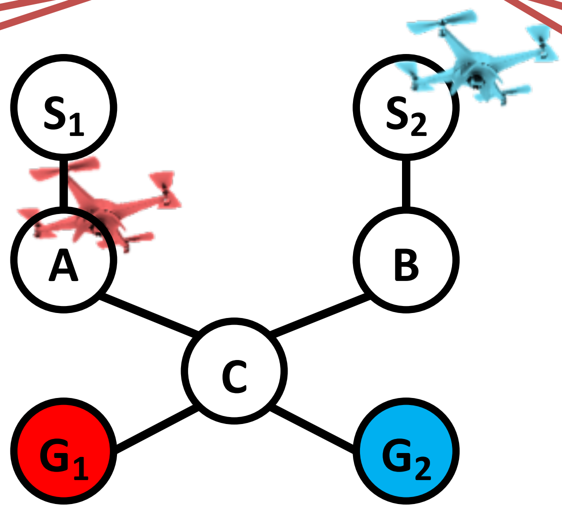
- Hard to compute
- Need full sync.
+ High solution quality

## *When to Repair/Replan?*

| Exec. | | | | | | |
|---|---|---|---|---|---|---|
| **Red** Agent | S1 | A | A | C | G1 | |
| **Blue** Agent | S2 | S2 | B | C | G2 | G2 |

**Unexpected delay ☹**

**Potential future conflict**

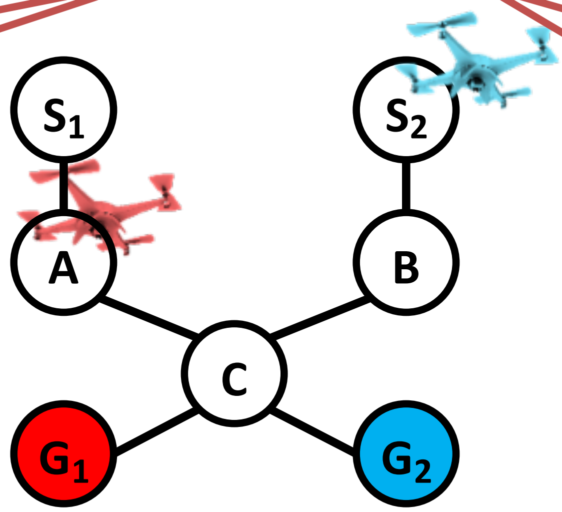



## *When to Repair/Replan?*

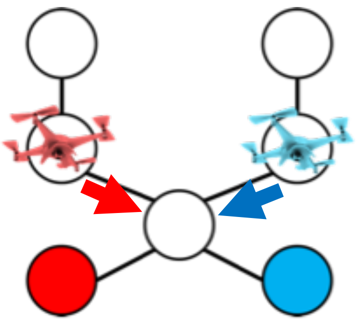
| Exec. | | | | | | |
|---|---|---|---|---|---|---|
| **Red** Agent | S1 | A | A ? | C ? | G1 ? | |
| **Blue** Agent | S2 | S2 | B ? | C ? | G2 ? | G2 ? |

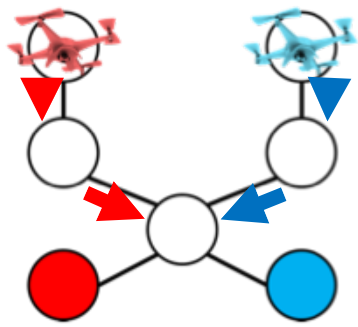
**Unexpected delay ☹**

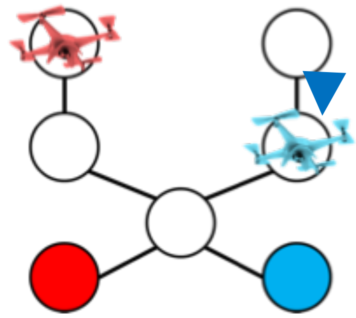
**Potential future conflict**



| Lazy | Reasonable | Eager |
|---|---|---|







When collision is about to occur

When collision will occur

When an agent is delayed

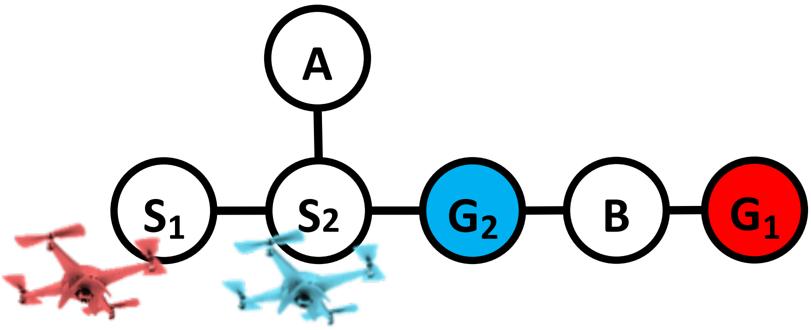


| | Lazy | Reasonable | Eager |
|---|---|---|---|
| Repair | N/A | | |
| Replan | | | |

**When agents need to communicate?**

Minimal Communication Protocol (MCP)
[Ma et al. '16]

## *Minimal Communication Protocol (MCP)*

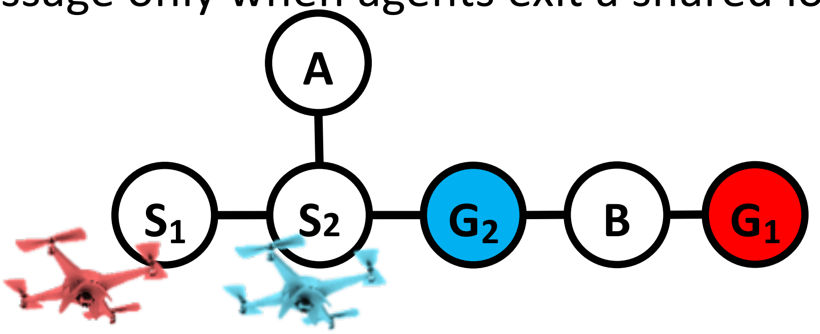



| | | | | | |
|---|---|---|---|---|---|
| **Red** Agent | $S_1$ | $S_2$ | $G_2$ | B | $G_1$ |
| **Blue** Agent | $S_2$ | A | $S_2$ | $G_2$ | $G_2$ |

## *Minimal Communication Protocol (MCP)*

MCP

- Preserve **ordering of visits to locations**
- Repair only to avoid breaking this order
- Send a message only when agents exit a shared location



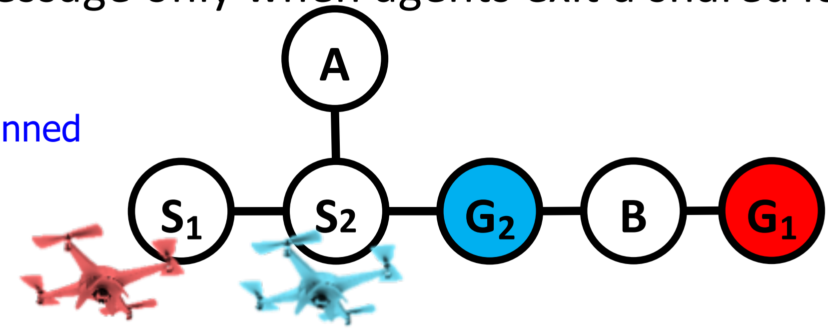

| | | | | | |
|---|---|---|---|---|---|
| **Red** Agent | $S_1$ | $S_2$ | $G_2$ | B | $G_1$ |
| **Blue** Agent | $S_2$ | A | $S_2$ | $G_2$ | $G_2$ |

## Plan Repair via Adjusting Agent Velocity

MCP

- Preserve ordering of visits to locations
- **Repair** only to avoid breaking this order
- Send a message only when agents exit a shared location

Can also move
**faster** than planned




| | | | | | |
|---|---|---|---|---|---|
| **Red** Agent | $S_1$ | $S_2$ | $G_2$ | B | $G_1$ |
| **Blue** Agent | $S_2$ | A | $S_2$ | $G_2$ | $G_2$ |


## Plan Repair via Adjusting Agent Velocity

MCP

- Preserve ordering of visits to locations
- **Repair** only to avoid breaking this order
- Send a message only when agents exit a shared location

Can also move
**faster** than planned

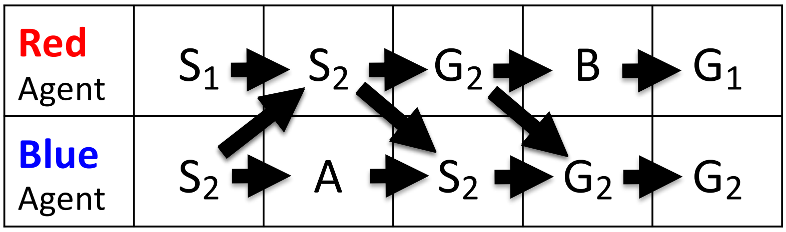
| | | | | | |
|---|---|---|---|---|---|
| **Red** Agent | $S_1$ | $S_2$ | $G_2$ | B | $G_1$ |
| **Blue** Agent | $S_2$ | A | $S_2$ | $G_2$ | $G_2$ |

## Plan Repair via Adjusting Agent Velocity

MCP

- Preserve ordering of visits to locations
- **Repair** only to avoid breaking this order
- Send a message only when agents exit a shared location

Can also move **faster** than planned



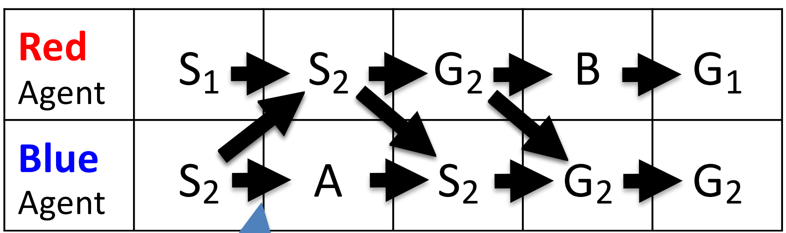

## Plan Repair via Adjusting Agent Velocity

MCP

**Ma et al. '16, '18**

- Preserve ordering of visits to locations
- **Repair** only to avoid breaking this order
- Send a message only when agents exit a shared location

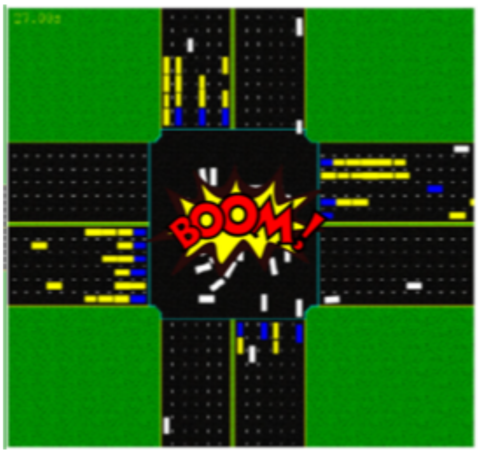
Can also move **faster** than planned




Label each edge with the robot's velocity constraints

→ A Simple Temporal Network
→ Solvable in poly-time

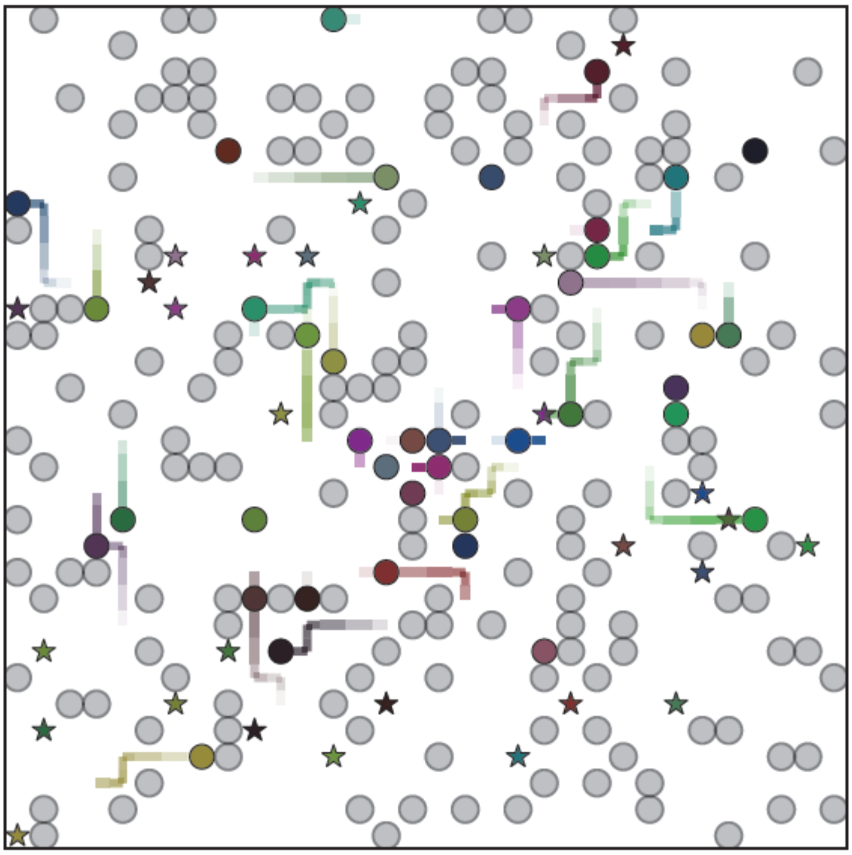
- How to **react** when an unplanned event occur?
- How to plan a-priori if we know such events may occur?






## How to consider unpredictable changes a-prior?

- Find a plan whose expected (*) cost is minimal
  - AME (**Ma et al. '17**)
- Find a plan that is executable with high probability
  - UM* (**Wagner & Choset '17**)
- Find a plan that is robust to a fixed number of changes
  - K-robust MAPF solvers (**Atzmon et al.**, see SoCS and AAMAS '18)

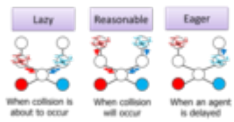


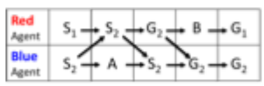


## Planning and execution in MAPF

- Under-studies aspect of MAPF
- Dilemma #1: replan vs. repair

- Dilemma #2: when to repair/replan?
  - Eager, reasonable, lazy, or MCP

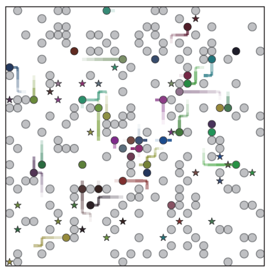



- Dilemma #3: a-prior planning: robust or expectation

## Many open challenges

- How to consider solution quality?
- Relation to conformant and contingent planning
- Life-long MAPF planning



Part V:

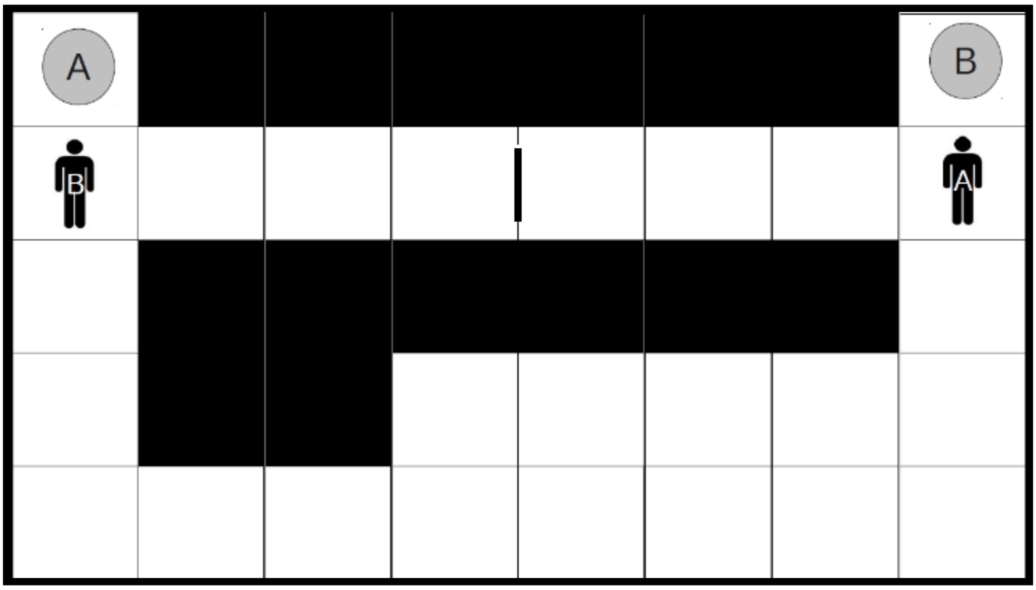
# CHALLENGES AND CONCLUSIONS



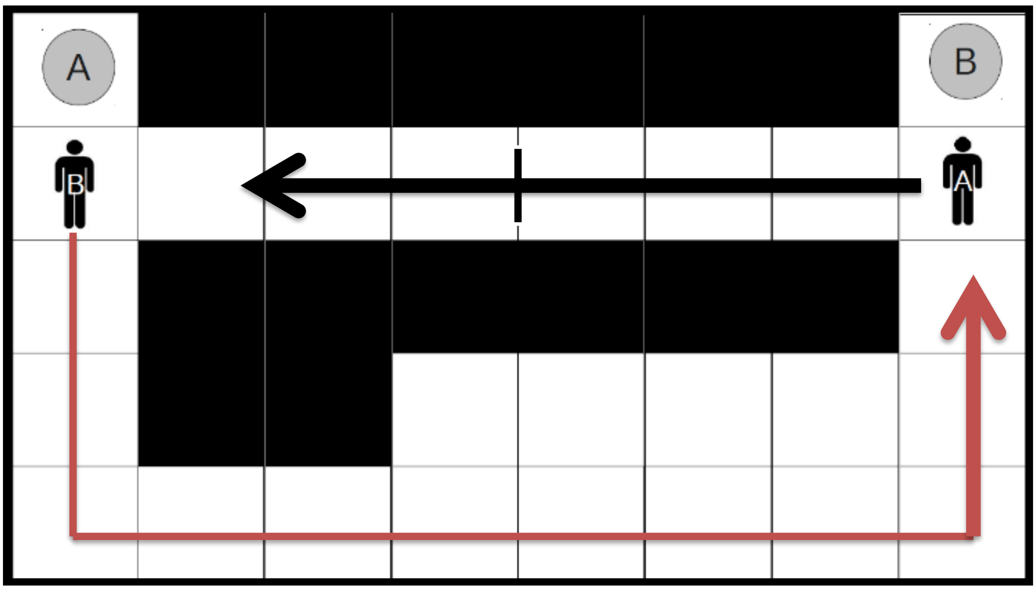
## Why I like to work on Multi-Agent Pathfinding

- 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

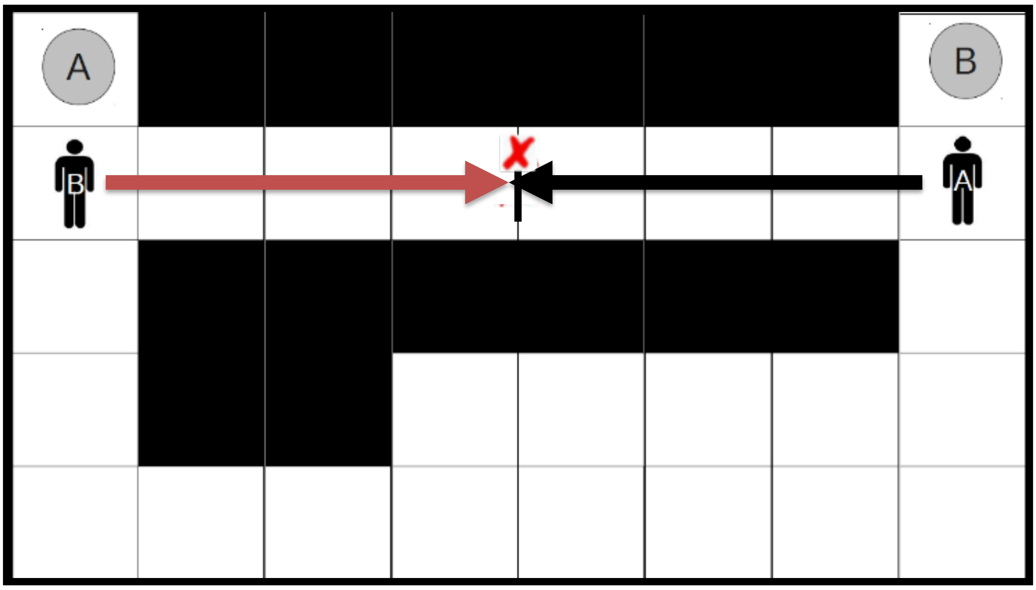



## Challenge: MAPF with Self-Interested Agents



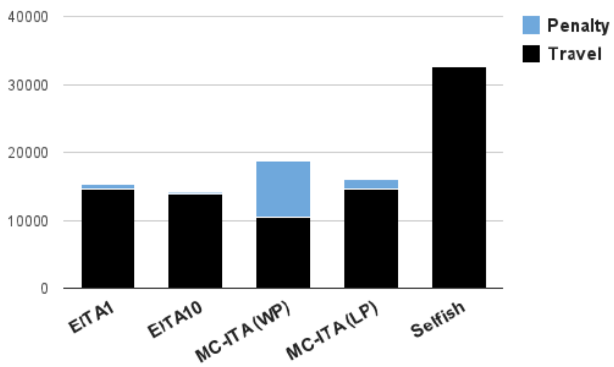
## Challenge: MAPF with Self-Interested Agents

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

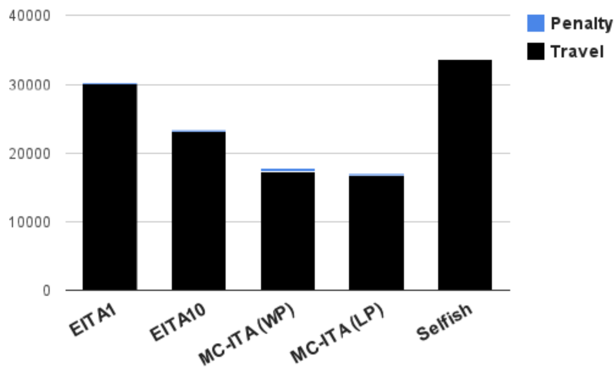



What if the other agent is **adversarial**?
or even worse, a **human**?

## Preliminary Results: MAPF with a Taxation Scheme



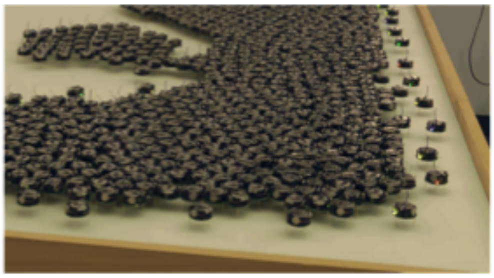

(a) 50x50 grid with 20% for 20 agents




(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 and others
- Online settings

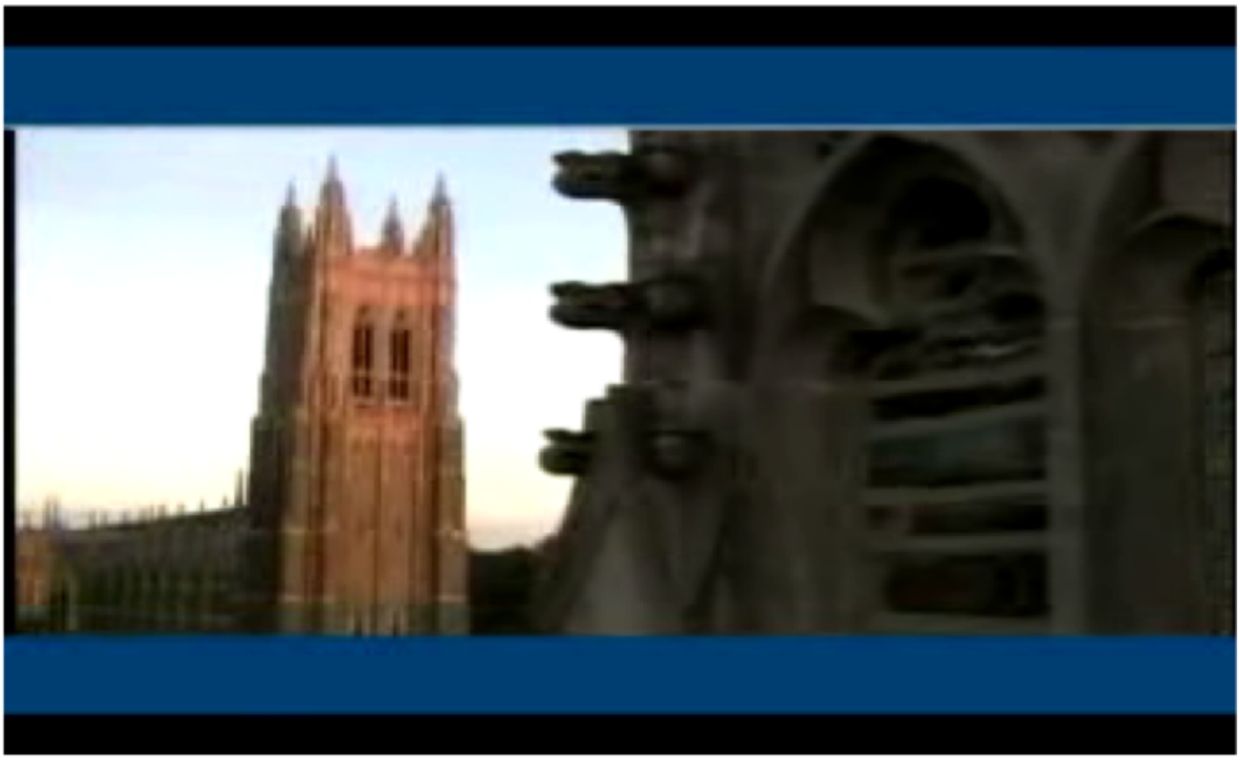
*Challenge: Relation to General Multi-Agent Planning*

## 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

*Agents can be Anything*



# Thanks!

Roman Barták, Roni Stern