AAAI-2022 tutorial

Introduction to Multi-agent Pathfinding

Ariel Felner, Ben-Gurion University, Israel

Jiaoyang Li, USC

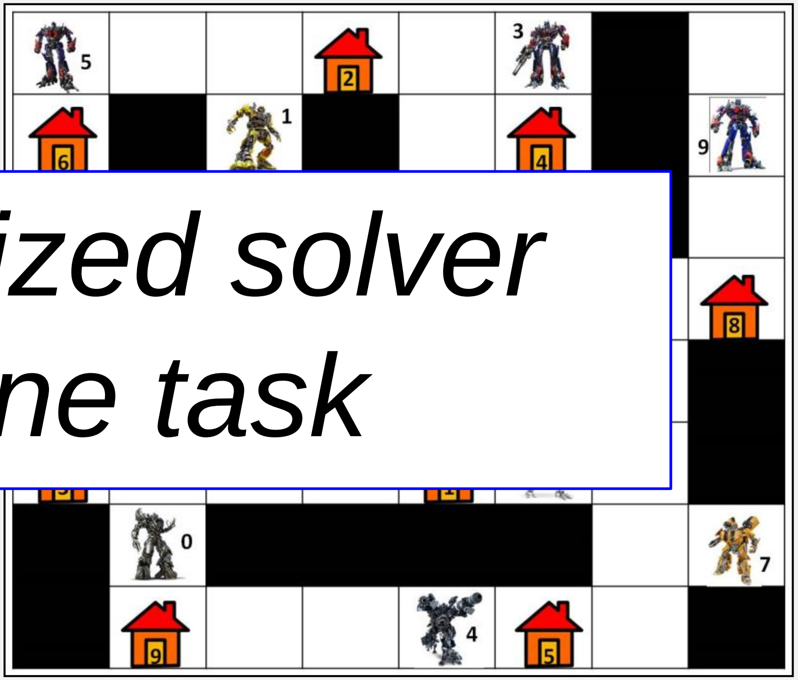
Sven Koenig, USC

Daniel Harabour, Monash University, Australia

Multi-agent path finding (MAPF)

Input

- A graph with **N** states
- A set of **K** agents – each with start and goal state



Actions

An agent can m

Task – a solut

A path for eac

Centralized solver

Offline task

Constraints

Paths shouldn't conflict

- Agents cannot be in the same location at the same time

(Edge constraints, Following policies)

Target

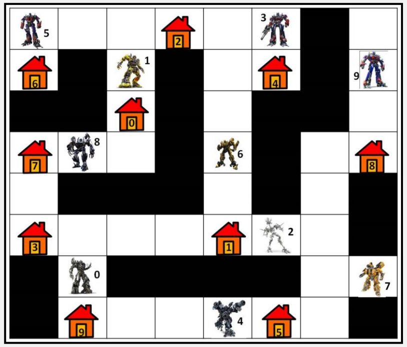
Minimize the cost of the solution

Motivation

- Robotics
- Video games
- Transportation applications
- Warehouse management
- Product assembly

Different cost functions

$p(a_i)$=Individual path for agent a_i



Cost 1: sum of costs

$$p(a_1)+ p(a_2)+ \ldots + p(a_n)$$

Cost 2: Makepan

$$\max\{p(a_1), p(a_2), \ldots , p(a_n)\}$$

Complexity

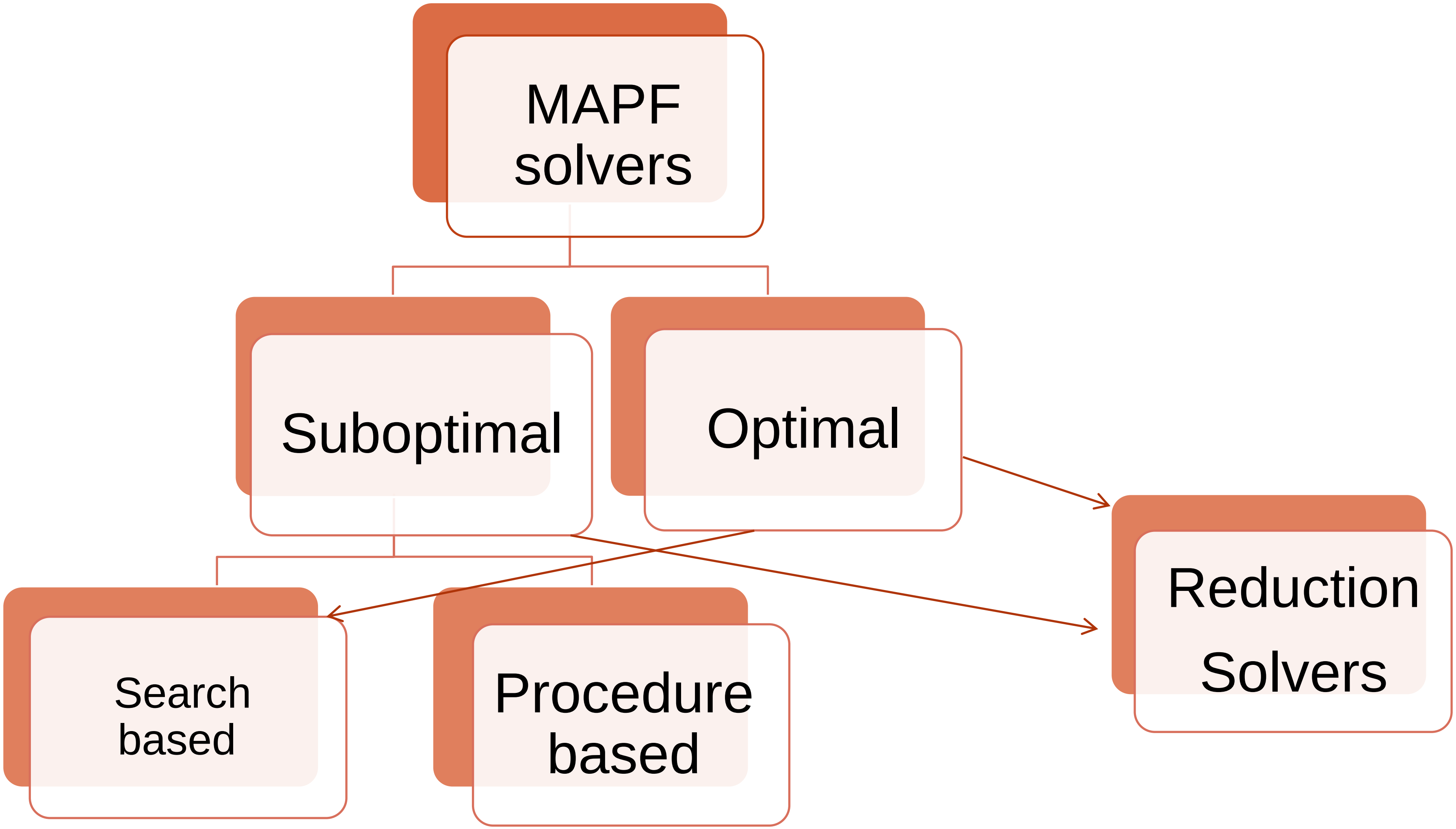
- The problem was proved to be NP-hard

[J. Yu and S. M. LaValle, AAAI-2013]

- The 15-puzzle is a special case of MAPF

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

Main approaches



Suboptimal solvers

Searched-based suboptimal solvers

- Agents are planned individually
  - Then, conflicts and deadlocks are resolved

Attributes:

- Fast
- Easy to understand/implement
- Forfeit optimality/completeness

Cooperative A* [Silver 2005]

Initialize the reservation table T
For each agent do {
    Find a path (do not conflict with T)

Prioritized planning

- Windowed-Hierarchical CA* (WHCA*) [Silver 2005]
- Conflict Oriented WHCA* [Banya and Felner, ICRA 2014]

Procedure-based sub-optimal solvers

- Have specific movement rules (e.g., go on highway)
  - Complete!
  - Very fast!
  - Far from optimal
  - Can solve very large problems

Procedure-based MAPF solvers

- A **complete** polynomial-time algorithm to the pebble motion problem was already introduced by [Kornhauser, FOCS 1984]
- It was recently implemented by Surynek.
- Agents move one at a time.
- Far from optimal.

Procedure-based MAPF solvers

- Slidable Multi-Agent Path Planning, [Wang & Botea, IJCAI, 2009]
  - Complete for slidable grids
- Push and Swap [Luna & Bekris, IJCAI, 2011]
- Parallel push and swap
- Push and Rotate [de Wilde et al. AAMAS 2013]
  - Macro-based
  - Complete for graphs where at least two vertices are always unoccupied
- BIBOX [Surynek 2013]
- Tree-based agent swapping strategy, [Khorshid at el. SOCS, 2011]
  - Complete for tree type graphs

Relaxing Optimal Solvers

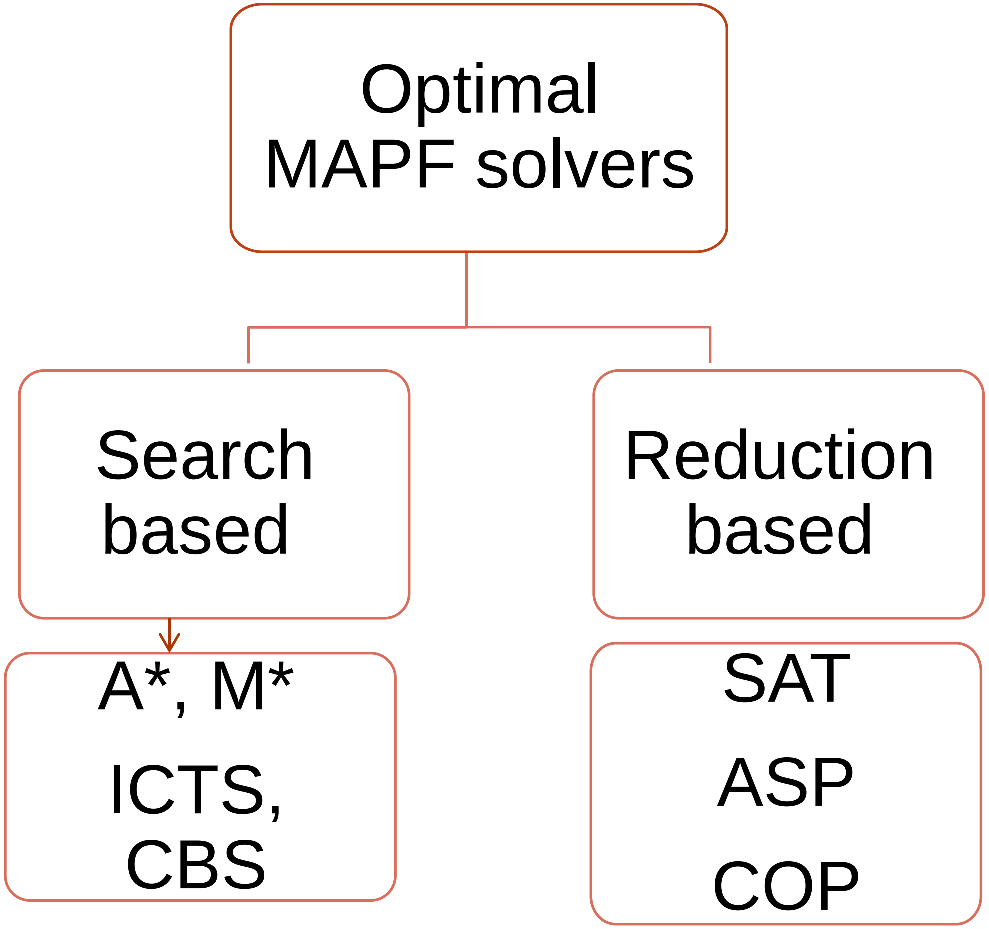
- **Bounded suboptimal solvers**: find a solution which at most W x OPT for $1 \leq W$

- Any optimal solver can be relaxed.
  - WA* of any A*-based algorithm
    - $f(n)=g(n)+Wh(n)$

  - Suboptimal A*+OD+ID [Standley and Korf, IJCAI 2011]
  - Suboptimal ICTS [Aljaloud and Sturtevant, SoCS 2013]
  - CBS-e [Barrer et al. SoCS-2014]
  - CBS-Highways [Cohen at el. 2015]
  - EECBS, Li et al. AAAI-2021

Optimal solvers

Main approaches



Main approaches

- Optimal MAPF solvers
  - Search based
    - A*, M*
    - ICTS, CBS
  - Reduction based
    - SAT
    - ASP
    - COP

iii. Reduction solvers

- Reduce MAPF to other known problems in computer science.
  - SAT [Surynek 2012]
  - Integer Linear Programing [Yu et al. ICRA 2013]
  - Answer Set Programming [Erdem et al, AAAI-2013]
  - Branch and cut and price (BCP) [Lamm et al. IJCAI-2019]

- Work extremely fast for small graphs
- May be very slow for large graphs

SAT solver for makespan [Surynek]

- For i=1 to infinity
  - Create a SAT formula that answers:

    "Is there a solution to the problem of cost i"
  - Solve that formula

Integer Programing

[Yu and La-Valle (2013a]

- Used Integer Linear Programming (ILP) to provid a set of equations and an objective function which yield the optimal solution.

Answer Set Programming

[Erden et al, 2013]

- Used the declarative programming paradigm of Answer Set Programming (ASP)

- Represent the path-finding problem for each agent and the inter-agent constraints as a program P in ASP.

- The answer sets of P correspond to solutions of the problem.

Optimal Search-based MAPF solvers

- A*-based algorithms
  - A*
  - EPEA*
  - A*+OD+ID
  - M*
- Other search algorithms
  - ICTS
  - CBS
  - BCP ??

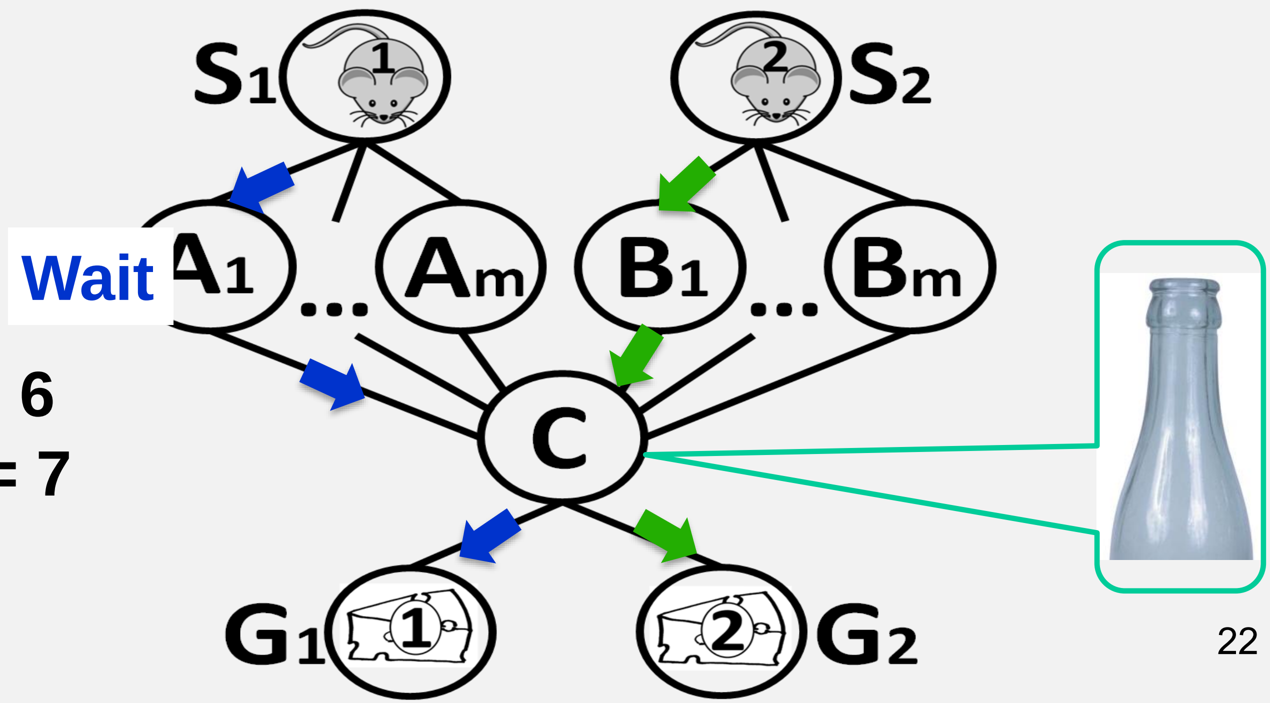
A* approaches

State space: Permutations of **K** agents into **N** locations=$O(N^K)$

Operators: Locations of all agent in the next time step

Heuristic function: Sum of Individual Costs **(SIC)**

SIC = 3+3 = 6
Optimal = 3+4 = 7



Problems with A*

Problem 1: State space is too large

Solution: Let's abstract the underlying graph

- Ryan [2008,2011] abstracted the underlying graph into known shapes such as halls, rings and corridors.
- Have specific expansion schedule for each of these cases.
- Sometimes not optimal.

Problems with A*

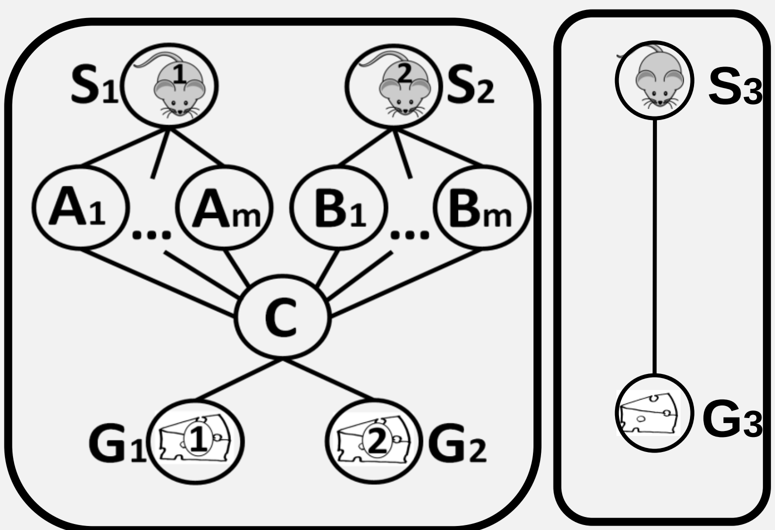
Problem 2: The state space is exponential O($\mathbf{N}^K$)

- On a 10x10 grid with 10 agents: =$\mathbf{100^{10}}$ =$\mathbf{10^{20}}$

Solution: let's reduce the number of agents!

i) Independence detection (ID) [Standely 2010]

- Divide the agents into independent groups
- Solve each group separately



Problems with A*

Problem 3: The branching factor is exponential:

$$b_{global}=b^K$$

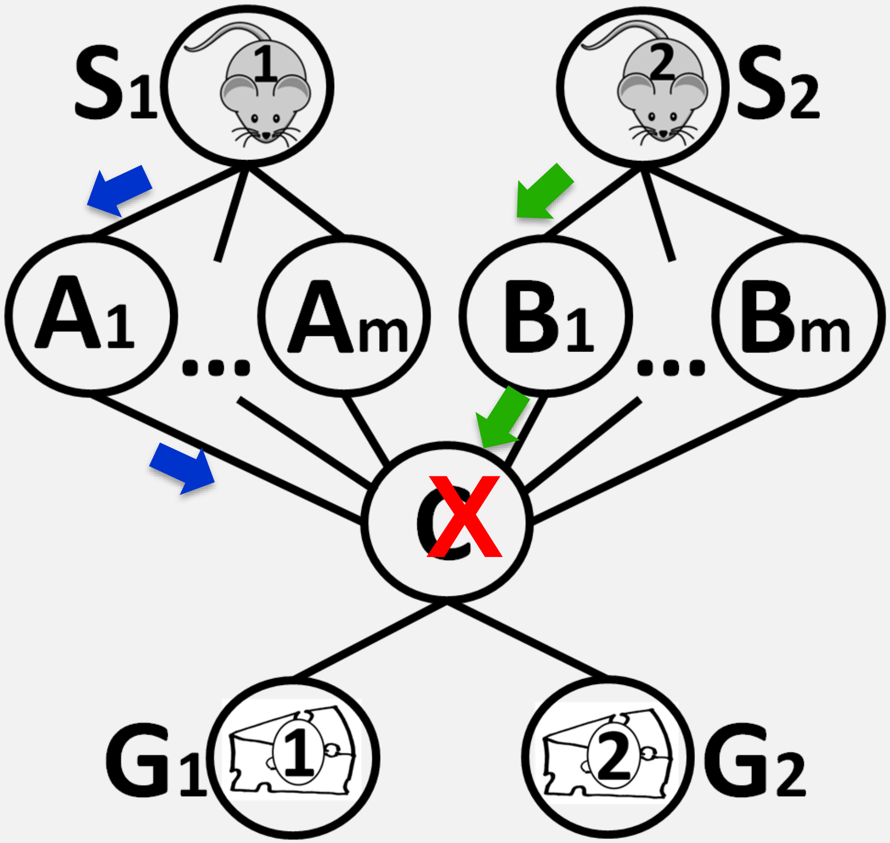
- On a grid with 20 agents: $b^k = 5^{20} =$ **95,367,431,640,625**

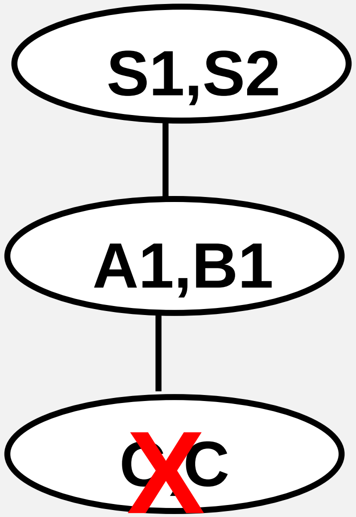
Solution: let's reduce the branching factor!

M* [Wagner 2011]

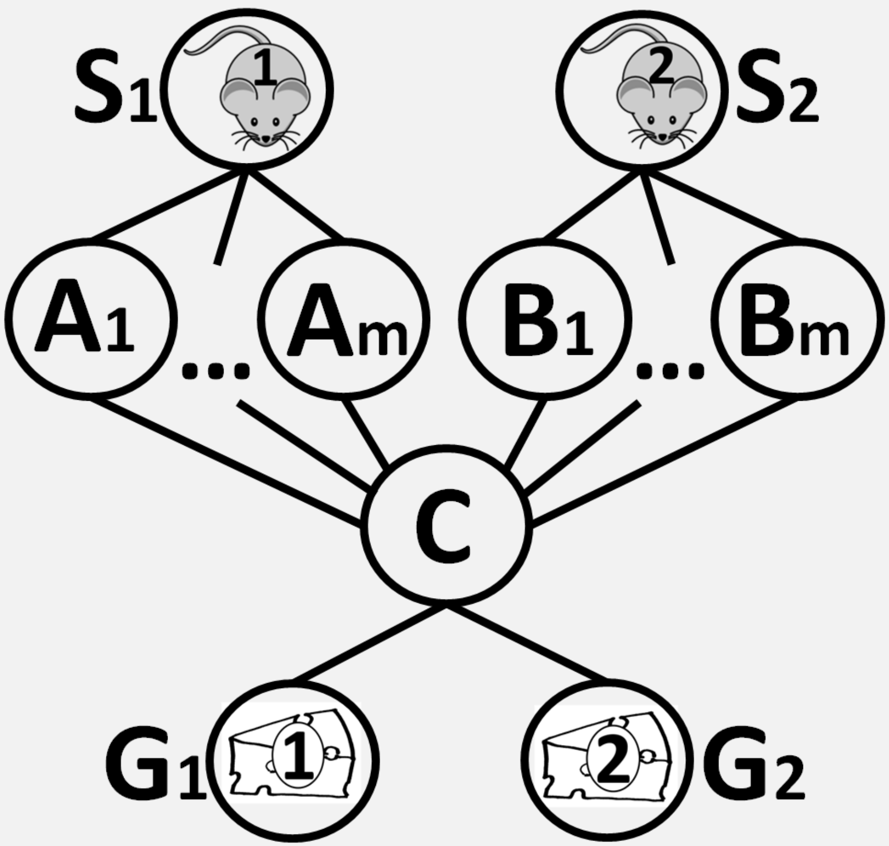
- Dynamically change the branching factor based on conflicts.
- Works on the global search space but starts with single moves of agents
- When a conflict occurs between two agents M* moves back to all ancestors and generates ALL possible children.

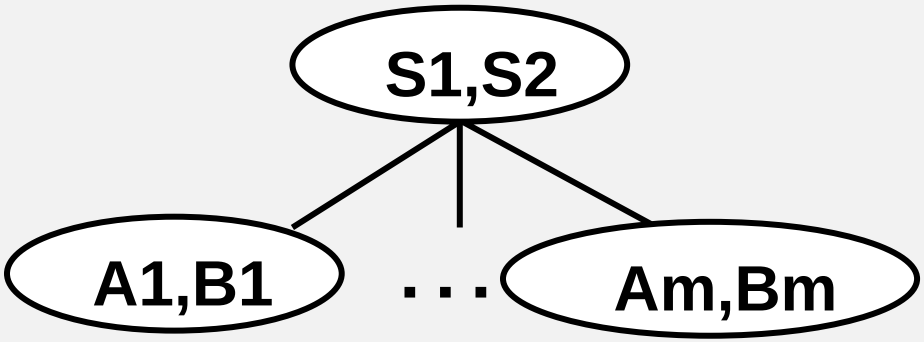
M*





M*





Problems with A*

Problem 4: Surplus nodes (those with f>C*)

Solution: let's avoid them

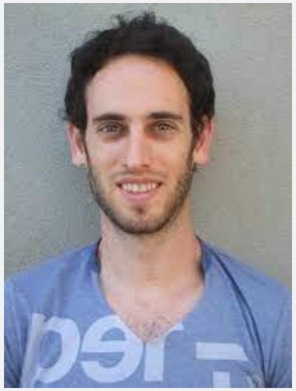
i) Operator Decomposition (OD) [Standley AAAI-2010]
  Intermediate states
  Each level in the tree moves a single agent
  Every K levels we have a full state (as A*)

ii) Enhanced partial expansion A* (EPEA*) [AAAI-2012]
  EPEA* never generates surplus nodes

→ [Goldenberg et al. 2012] studied combinations of these approaches

New non-A* algorithms

Guni Sharon



1) The Increasing Cost Tree Search (ICTS)

[Sharon et al. IJCAI-2011, AIJ-2012]

2) Conflict-Based Search (CBS)

[Sharon et al. AAAI-2012]

3) Meta-agent Conflict-Based Search (MA-CBS)

[Sharon et al. SoCS-2012]

4) Branch and cut and price,

[Lam et al. IJCAI-2019]

These algorithms are exponential in different parameters

Algorithm 1: ICTS [AIJ-2013]

two level algorithm

High-level

Is there a solution with costs

10 10 10

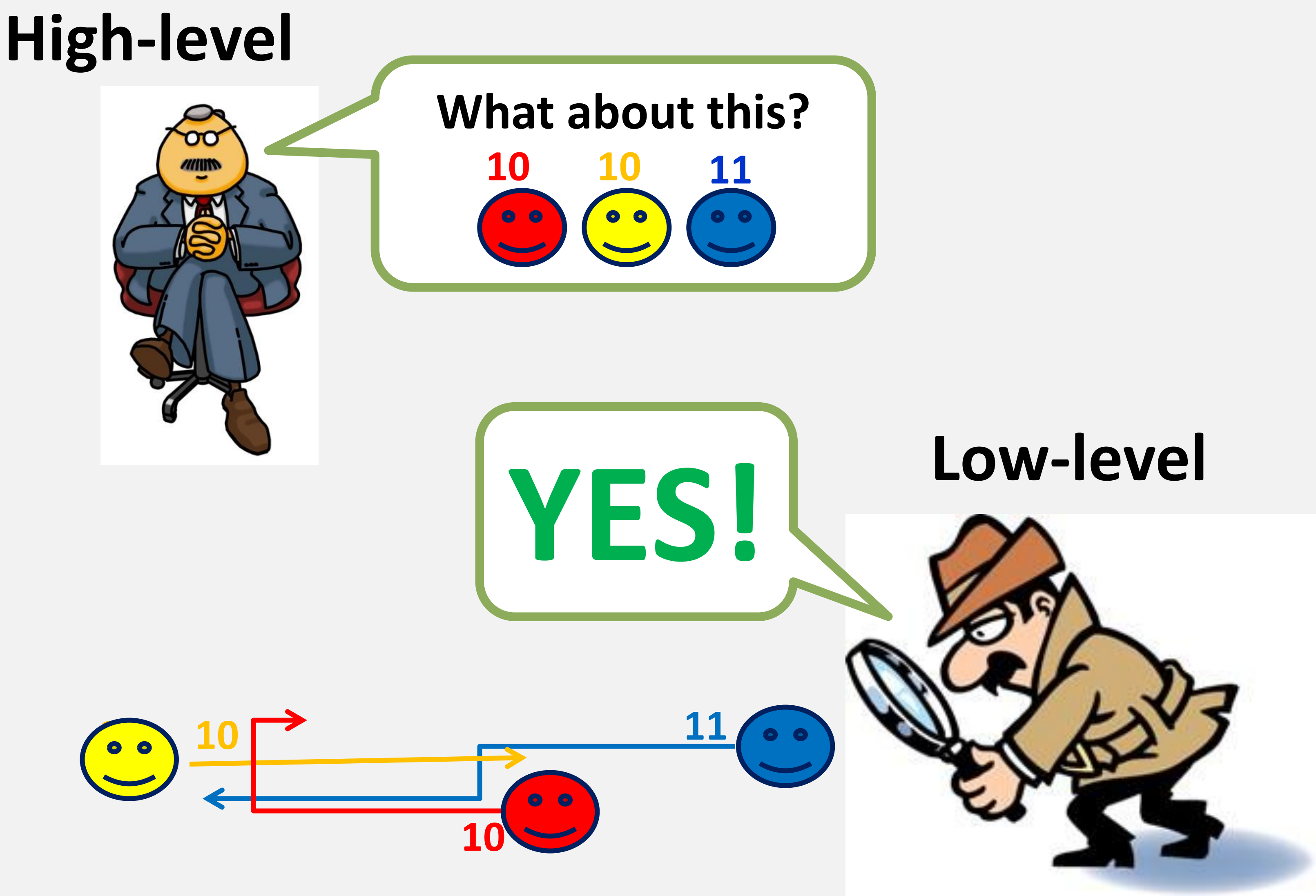
?

NO!

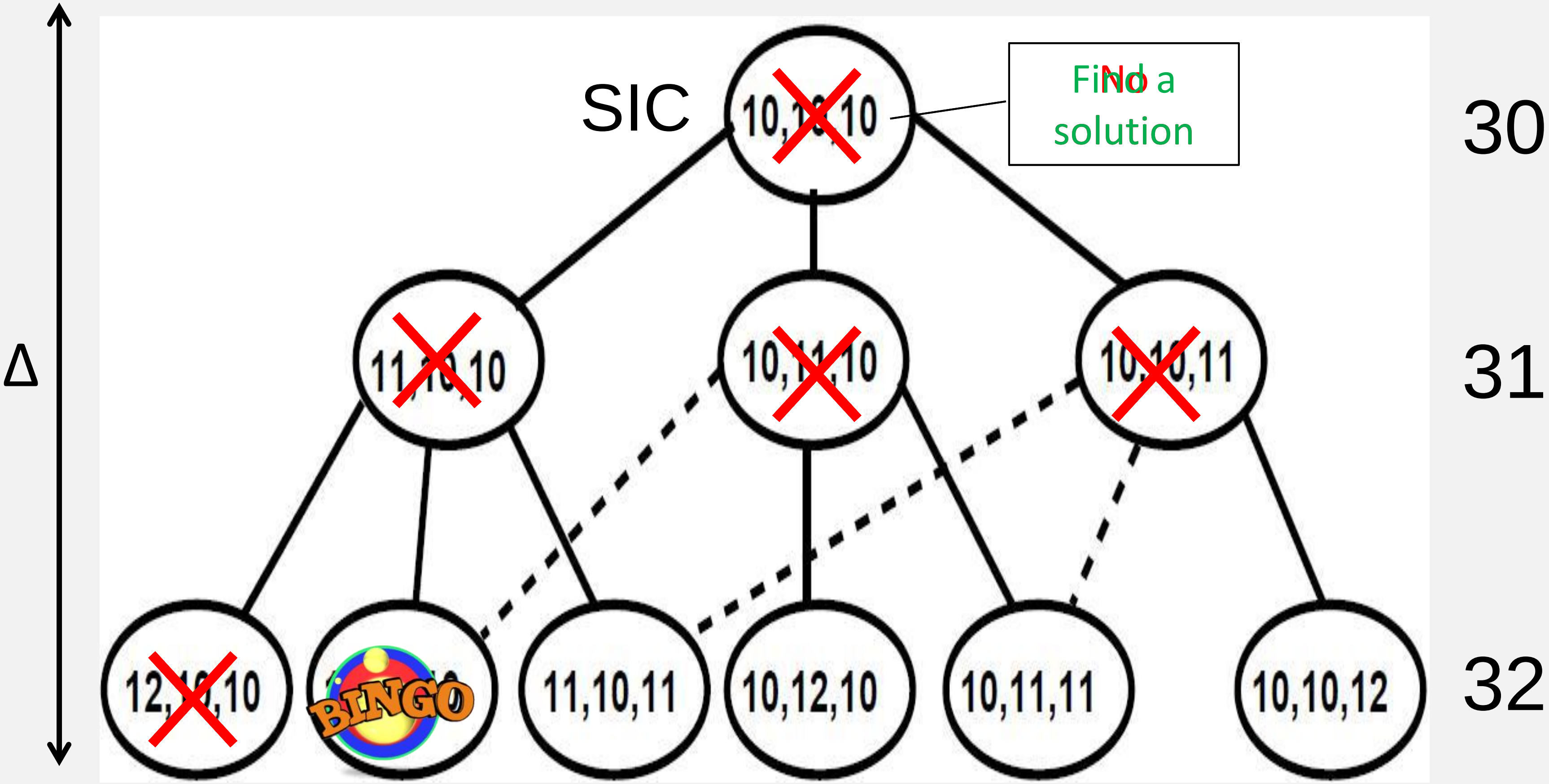
Low-level

Algorithm 1: ICTS [IJCAI-2011]

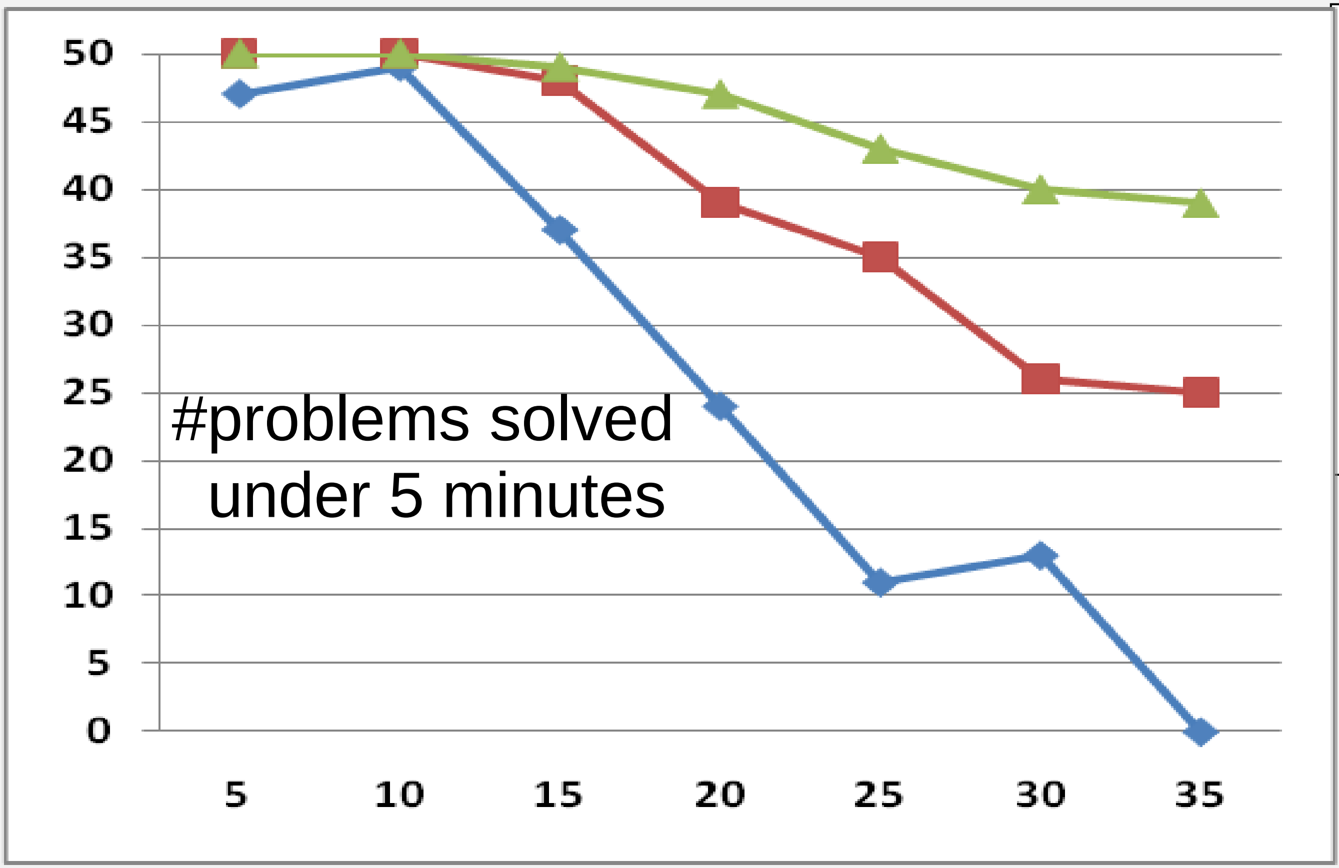
two level algorithm

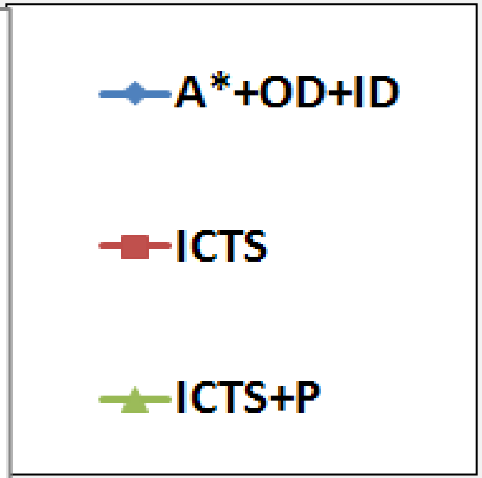


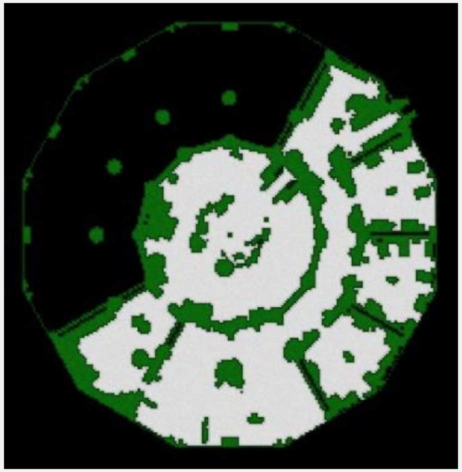
ICTS: High level



Experiments: Dragon-Age Origin [Sturtevant]

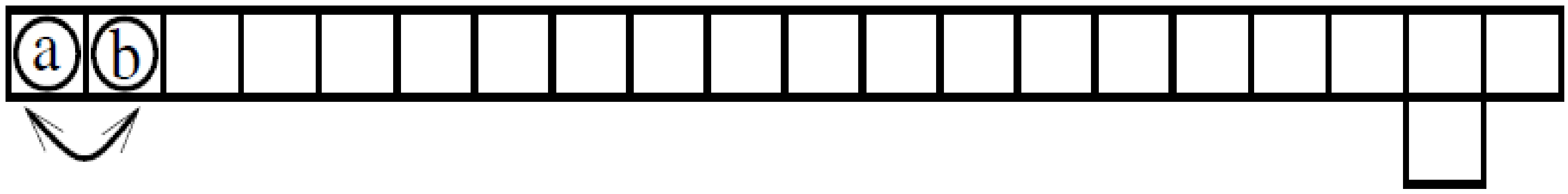






number of agents

ICTS: pathological case



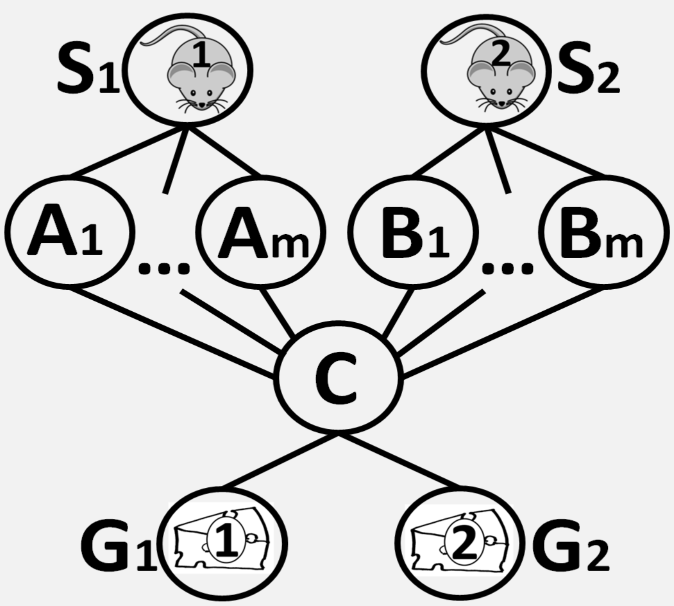
- SIC =2
- Optimal solution =74
- Δ=72
- A*: solved in **51**ms
- ICTS: solved in **36,688**ms

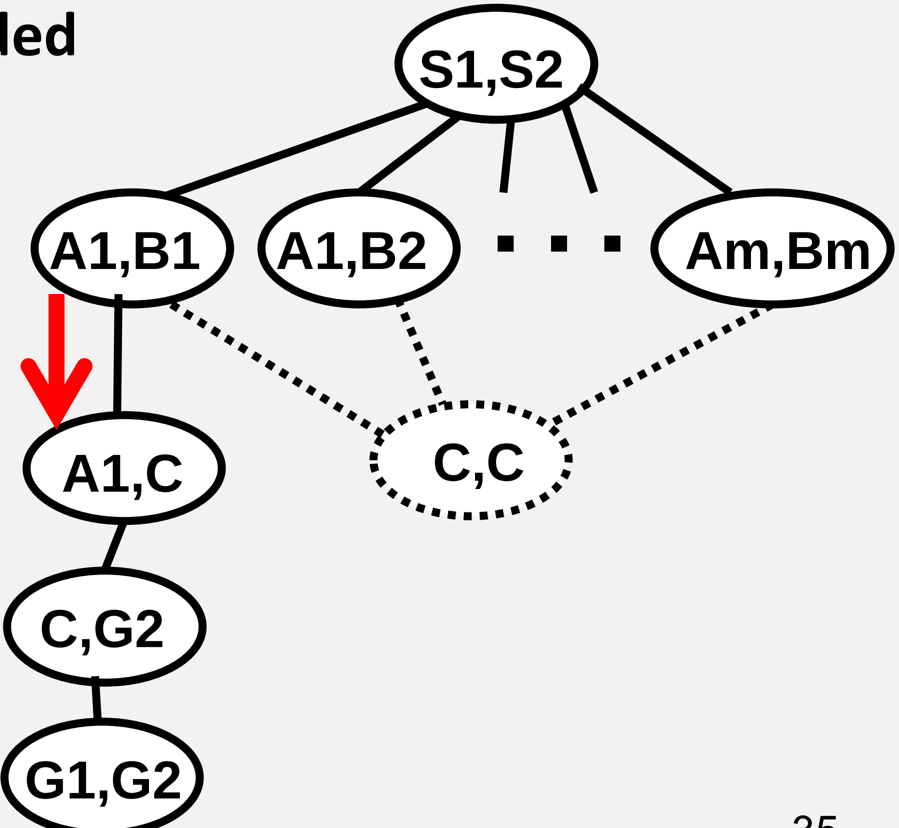
Algorithm 2: Conflict-based Search (CBS) [AAAI-2012, AIJ-2015]

Motivation: cases with bottlenecks:

A*

- **f=6:** ***All m^2 combinations*** of **(A_i,B_j)** will be generated and expanded - all will generate **(C,C)** which is illegal
- **f=7: 3 states are expanded**





CBS – underlying idea

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

CBS plans for single agents
but under constraints

Conflicts and constraints

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

Conflict is resolved by adding either [**A,X,T**] or [**B,X,T**]

CBS: general idea

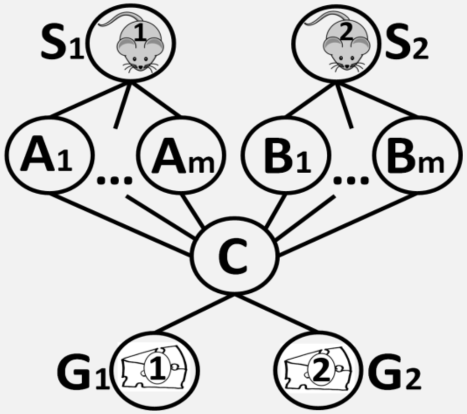
1. Plan for each agent individually
2. Validate plans
3. If the plans of agents A and B conflict

   Constrain A to avoid the conflict

   or

   Constrain B to avoid the conflict

The constraint tree

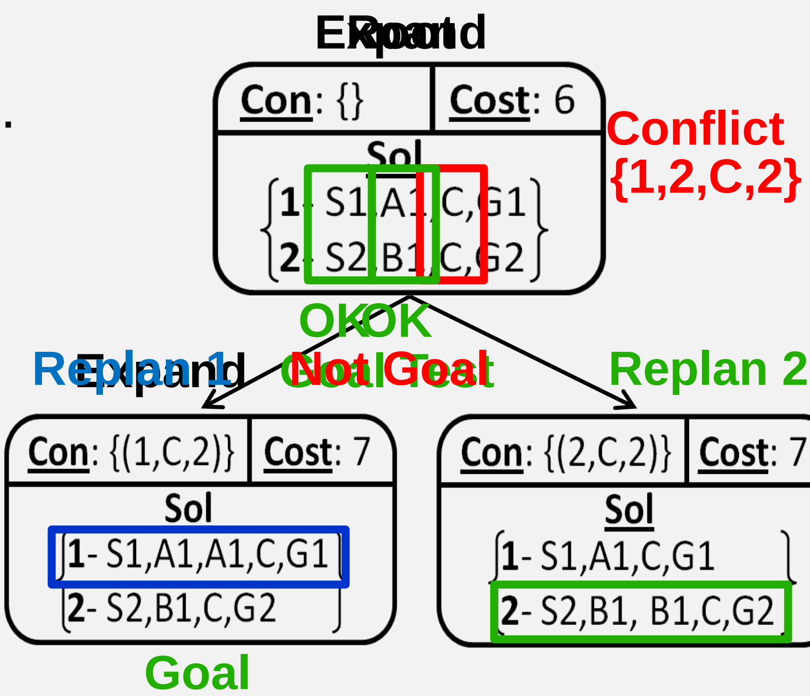


Nodes:

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

Goal test:

- Are the paths conflict free.



CBS is a very popular framework!

Thank you