

## MAPF algorithms

### Non optimal

**Algorithm: Prioritized Planning** -Explanation: Prioritized Planning assigns priority levels to agents based on their importance and solves subproblems in a prioritized order. It is complete, but the solutions may not be optimal due to the disruption caused by higher-priority agents to lower-priority agents' plans.

Efficient for solving multi-agent pathfinding problems by prioritizing agents based on their importance. Inefficient for problems with conflicting priorities or complex dependencies.

### Optimal

#### Algorithm: ICTS

**Algorithm: CBS** -Explanation: Two-level algorithm, first finds a conflict-free path assignment using a high-level search tree and then resolves conflicts at the low-level. It guarantees optimality when using optimal low-level solvers. Efficient for solving multi-agent pathfinding problems with conflicting paths and bottlenecks. Inefficient for problems with a large number of agents or complex environments.

**Algorithm: M-star** -Explanation: A single-agent pathfinding algorithm that uses lazy search to adapt to dynamic obstacles in grid-based environments. Efficient for solving single-agent pathfinding problems in grid-based environments with an unbounded number of obstacles. Inefficient for problems with a large number of agents or complex terrain.

**Algorithm: A-star OD ID** -Explanation: A-star OD is a variant of A-star where 1 agent moves every time. Deeper search but lower branching factors, hopefully can explore less of the tree this way.

## MAPF algorithms (cont)

**Algorithm: EPE A-star (Enhanced Partial Expansion A-star)** -Explanation: EPE A-star selectively expands nodes, focusing on promising areas using a heuristic on actions, to reduce computational overhead.

## Independence Detection

### Simple Independence Detection

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

### Independence Detection

Try to avoid conflict with **same cost** before merge

### Online Independence Detection

Solve for every group of agents separately, merge groups if necessary.  
Advantages: Replan only required agents, Minimize disturbance, maintain snapshot optimality.

## Planning Domain Description Language (PDDL)

Fact = a predicate, State = a conjunction of facts, The initial state: all that I know about the world

The goal: a conjunction of facts that I wish true,

Action = a method for moving between states

- Can have preconditions and effects

**STRIPS** - Problem is <Predicates, Actions, Initial, Goal>

**Approach 1** - State Space Search. Can search Forward to the goal or backward from the goal. Possible heuristics: Delete relaxation/Abstraction(Remove a predicate)

**Approach 2** - Partial order planning- try to achieve goals in parallel, Only make choices when conflicts occur.

**Approach 3** - SAT compilation.

## MA-STRIPS

**MA-STRIPS** - Replace Actions in strips with a set of actions per agent. It has different levels:

Centralized | Centralized but allow parallel execution| Decentralized observations| Decentralized execution| Decentralized planning

**Approach 1** - STRIPS compilation: If agents can act in parallel, A in STRIPS will be the cartesian product of all A's.

Else, The possible actions are a union of all actions.

### In the parallel setting:

- Full knowledge of other agent' abilities
- Global heuristic estimates, as in centralized planning

### In the distributed setting:

- Partial knowledge of other agents' abilities
- Local, less informed heuristic estimates

### In the centralized setting:

- Agents represent a natural factoring of the problem

Can we plan in a mostly decoupled way?

Pros: - Can use regular STRIPS planner

Cons: - Does not fit a distributed setting

- Branching factor exponential w. # of agents

**Private actions** affect and are affected by agent's actions only.

**Public actions** affect or are affected by other agents.

**Approach 2:** Planning + CSP

## Suboptimal MAPF

**Algorithm: Weighted A-Star** - A-Star that has a weight factor to prioritize nodes close to the goal by increasing the weight of h in the formula.

Formula:  $f(n) = g(n) + w * h(n)$

## Suboptimal MAPF (cont)

**Algorithm: Suboptimal CBS** -First expand the routes with the least conflicts OR use single agent A-Star for low level search OR use BFS for high level search.

## MAPF to SAT Encoding

MAPF to SAT Encoding

$V_{a_i, t, s} = 1$  (1)  $(k)$   
 $V_{a_i, t, s} = 0$  (2)  $(k)$   
 $V_{a_i, t, s} \rightarrow \bigvee_{s \in S} V_{a_i, t, s} = 1$  (3)  $(k \times |S|)$   
 $V_{a_i, t, s} \wedge V_{a_j, t, s} = 0$  (4)  $(k^2 \times |S|)$   
 $V_{a_i, t, s} \wedge V_{a_j, t, s} = 0$  (5)  $(k \times |S|^2)$   
 $V_{a_i, t, s} \wedge V_{a_j, t, s} = 0$  (6)  $(k \times |S|^2)$

$k \times |S| \times |V| = \text{Var}$  of the problem  
 -  $k$ : מספר אגנטים  
 -  $|S|$ : מספר תאים  
 -  $|V|$ : מספר זמנים

## SIPP- Low Level Planner for Continuous

### Low-Level Planner - SIPP

- Using SIPP as the low-level solver for CCBS
- The CCBS constraints creates safe intervals

Constraint: {Blue, wait\_at(A), [4, 7]}



Two states in A: {A, [0, 4]} and {A, [7, ∞]}

## Continuous Time

### Classical CBS vs CCBS Constraints

MAPF vs MAPF<sub>c</sub>

Conflict: {Blue, Green, (2,4) × (3,4), 5}  
 Constraint: {e, t}

Conflict: {Green, (2,4) × (3,4), 5}  
 Constraint: {Blue, (2,4) × (3,4), 5}

How to find replan a single agent efficiently under such a constraint?

To eliminate a conflict, CBS needs to constrain the colliding agents.

And again in case of classical CBS constraints are imposed on locations and contain only one timestep, while in CCBS the one needs to constrain actions. And these constraints need to have time-intervals as the agent can wait for arbitrary amount of time and start to perform the action at any moment.

## Large Agents

MAPF for Large Agents (Li et al., '19)

**Large agents:**  
Agents occupying multiple nodes

What constraints to impose by CBS to avoid a conflict  $(a_1, a_2, t_1, t_2)$ ?

**Definition:** Constraints  $C_1$  and  $C_2$  are **mutually disjunctive** if any pair of conflict free paths satisfies at least one of them

**Theorem:** Resolving conflicts with any pair of mutually disjunctive constraints preserves optimality and completeness

**Asymmetric:**

- Constraint 1:  $a_1$  avoids node  $l_1$
- Constraint 2:  $a_2$  avoids every node that conflicts with  $a_1$  occupying  $l_1$

**Symmetric:**

- Choose point  $p$  in the overlap of agents  $a_1$  and  $a_2$  when occupying  $l_1$  and  $l_2$
- Constraint 1:  $a_1$  avoids every node  $l_1$  that if  $a_1$  occupies  $l_1$ , then it overlaps with  $p$
- Constraint 2:  $a_2$  avoids every node  $l_2$  that if  $a_2$  occupies  $l_2$ , then it overlaps with  $p$

Every grid cell is a node. And agent at a node is defined here as having the top-left of the agent at that node.

## MA STRIPS Plan

Agent 1: A1, A2, A3  
Agent 2: A4, A5  
Agent 3: A6, A7, A8  
Agent 1: A9

A coordination scheme

Agent	1	2	3	1
Action	A3	A5	A8	A9

Private plans

Agent a1	Agent a2	Agent a3
A1 A2 A3	A4 A5	A6 A7 A8
A9		

## Planning + CSP

### Planning + CSP (very very high-level)

- While a full plan for all agents do not exist:
  - Decide the **public actions** for every agent
- For each agent:
  - Search for a plan to perform its public actions (using only **private actions**)
  - If no such plan was found
    - Goto 1.1, search for better plan
- Return the full plan

**Pros** -Exploits loosely coupled agents  
-Adding agents may only add complexity polynomially

**Cons** -CSP variables have huge domain  
-Regular CSP heuristic are not relevant and regular solvers can't run a planner

## Approach 2+: Planning First

High level idea: **plan first, let others coordinate**

- Replace the "blind" selection of coordination points
- First plan for one agent, then let others try to follow

A "message passing" scheme:

- First agent plans, assumes all others will help
- Result: a plan for this agent, but also
  - The public actions this agent did
  - The public actions it needs other to do
- Next agent gets the required public actions as landmarks

**Pros**- The choice of public actions is more informed

**Cons**- Which agent goes first?  
How to know what the other agents can do?

## Greedy Privacy Preserving Planner (GPPP):

- High level planning
  - Outcome: coordination scheme
  - Approach: Solve a relaxed planning problem
- Grounding
  - Input: a coordination scheme
  - Output: a set of private plans to follow the scheme
- Or false, if not possible
- Notes High level planning
  - Done using a heuristic forward search (GBFS)
  - Each agent generates runs search locally using only private actions to identify relevant states to publish
  - Uses heuristics to guide the search
- Notes Grounding
  - Grounding is done in parallel by all agents
  - Agents can use different heuristics/p-planners

## Online MAPF

### Replan Single

Plan optimally one by one for new agents  
New agent avoids current agents' plans

### Replan Single Grouped

Plan optimally for new agents together (MAPF)

New agents avoids current agents' plans  
**Replan All**

Replans optimally all agents when new agents appear

+Is snapshot optimal- optimal solution assuming no new agent appear in the future

-May unnecessarily disturb agents  
Possible objective functions: Sum of steps, Number of agents in graph at any time, Sum of steps over shortest path- all equal.  
Throughput at time x, Maximal service ration (ratio between shortest path and actual path).

## Online MAPF variants

### Online MAPF Variants

Definition:

**Incomplete** = problem may be unsolvable

**Unsound** = some collisions may be unavoidable

	Stay at goal	Disappear
Appears in grid	Incomplete & Unsound	Unsound
Appears in garage	Incomplete	Complete & Sound

## Suboptimal ID

### Online MAPF Algorithms – Suboptimal ID

- Online MAPF algorithms can not be optimal
- SubID: allow small suboptimality while re-planning
  - ✓ Improve computation time
  - ✓ Decrease number of disturbances to other agents

## ITF

### The Iterative Taxation Framework (ITF)

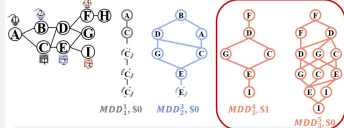
- **Path planning (PLAN)**.
    - Generates a set of paths with **lowest** penalized cost for every agent.
  - **Conflict detection (DETECT)**.
    - Detects a set of conflicts between the plans of the agents.
  - **Conflict resolution (RESOLVE)**.
    - Attempts to resolve these conflicts by adding penalties.
  - **Winner penalization** - Winner agrees to pay penalty that the loser cannot afford
  - **Loser penalization** - Loser is motivated to alter its preference.
- While Condition:**  
 PLAN  
 DETECT  
 RESOLVE
- Exhaustive ITA
    - Try to find a taxation to resolves conflicts for **all shortest paths**
  - Monte Carlo ITA
    - Simulate actions, resolve conflicts of this simulations

## iBundle

### iBundle for MAPF: Price Update

- At the end of each round, prices of all bundles that **unhappy** agents bid on are raised by  $\epsilon$  (assume  $\epsilon = 1$ )
- This will lead agent 3 to bid on higher cost (longer) paths

#### Round 2 bids:



## Action Graph

- Graph to model the interaction between actions
- Nodes correspond to actions
- An aedge  $(a_1, a_2)$  exists iff at least 1 of the following hold
  - $a_1$  achieves a precondition for  $a_2$  (or the other way around)
  - $a_1$  destroyed a precondition for  $a_2$  (or the other way around)
  - $a_1$  and  $a_2$  have conflicting effects
- Key: a partition of the AG induces MA-STRIPS

## Approach 3: Multi Agent Forward Search

### Algorithm for agents

1. While (not done)
  - 1.1. Handle received messages
  - 1.2. Extract best state  $s$  from OPEN
  - 1.3. For every action  $a$  of this agent
    - 1.3.1.  $s' = \text{generate}(a, s)$
    - 1.3.2 Add  $s'$  to OPEN
    - 1.3.3. If  $a$  is public, broadcast it
      - Can be optimal! But needs halting mechanism.
      - Distributed with distributed heuristics
      - Sometimes, MAFS achieves super linear speedup since it exploits the structure of the problem