Pathfinding and Routing

NAIL137

Jiří Švancara

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

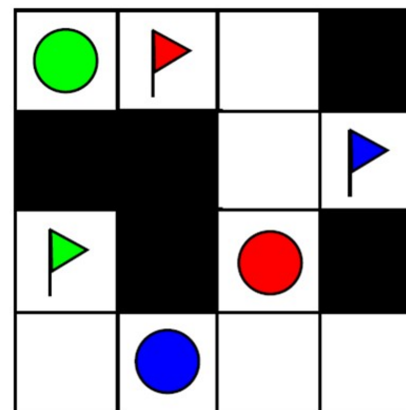
Definitions

- Multi-agent pathfinding (MAPF)

- Alternative names
  - Motion planning for multiple independent objects
  - Motion coordination
  - Multi-robot motion planning
  - Cooperative path finding
  - ...

Definitions - instance

Definition 1. An *instance of MAPF* is a pair (G, A), where G = (V, E) is a graph representing a shared environment and A is a set of agents. Each agent $a_i \in A$ is represented by a pair $a_i = (s_i, g_i)$, where s_i represents the start location (sometimes also called the initial location) of agent a_i, and g_i represents the goal location of agent a_i. In both cases, location corresponds to a vertex in the input graph.



Definitions - plan

Definition 2. A *plan for a single agent* i is a sequence of locations (vertices) where the agent is located at a given timestep. Let π_i denote a plan for agent i, where $\pi_i(j) = v$ represents that agent i is located in vertex v at timestep j. We denote $|\pi_i|$ as the length of the plan.

Definition 3. A plan π_i is said to be a *valid plan* if $\pi_i(0)$ and $\pi_i(|\pi_i|)$ are the initial and goal locations of a_i respectively, and for every $j = (0, \ldots |\pi_i|-1)$ either $\pi_i(j) = \pi_i(j+1)$ or $(\pi_i(j), \pi_i(j+1)) \in E$.

Definitions - collisions

Definition 4. An *edge conflict* occurs when two agents a_i and a_j move on the same edge in the same direction.

$$\pi_i(t) = \pi_j(t) \wedge \pi_i(t+1) = \pi_j(t+1) \text{ and } \pi_i(t) \neq \pi_i(t+1)$$

Definition 5. A *vertex conflict* occurs when two agents a_i and a_j reside in the same vertex at any timestep.

$$\pi_i(t) = \pi_j(t)$$

Definition 6. A *following conflict* occurs when an agent a_i is planned to occupy a vertex that was occupied by another agent a_j in the previous timestep.

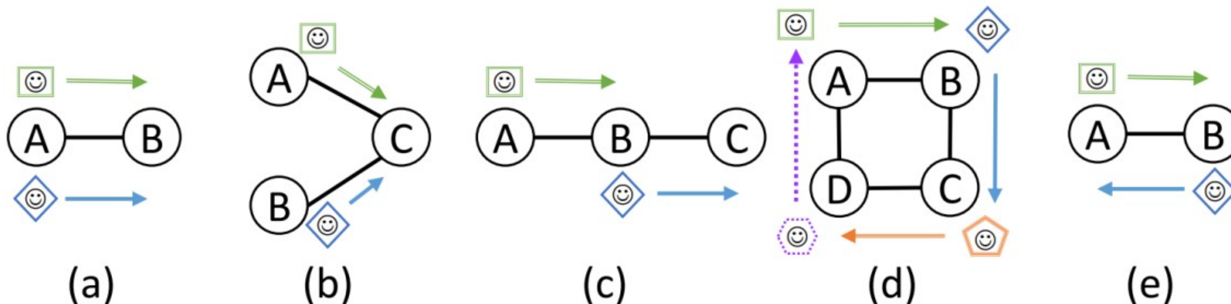
$$\pi_i(t+1) = \pi_j(t)$$

Definition 7. A *cycle conflict* occurs when a set of more than two agents a_i, a_{i+1}, ..., a_{i+k} is planned to move on a fully occupied cycle.

$$\pi_i(t+1) = \pi_{i+1}(t) \wedge \pi_{i+1}(t+1) = \pi_{i+2}(t) \wedge \cdots \wedge \pi_{i+k-1}(t+1) = \pi_{i+k}(t)$$

Definition 8. A *swapping conflict* occurs when two agents a_i and a_j move on the same edge in opposite directions.

$$\pi_i(t+1) = \pi_j(t) \wedge \pi_i(t) = \pi_j(t+1) \text{ and } \pi_i(t) \neq \pi_i(t+1)$$



Definitions - motion primitives

Definition 9. Pebble motion forbids vertex, follow conflicts.

Note: By forbidding vertex and follow conflicts explicitly, all 5 are forbidden implicitly.

Definition 10. Parallel motion forbids vertex and swapping conflicts, while following and cycle conflicts are allowed.

Note: Edge conflict is forbidden implicitly, follow and cycle conflicts are allowed.

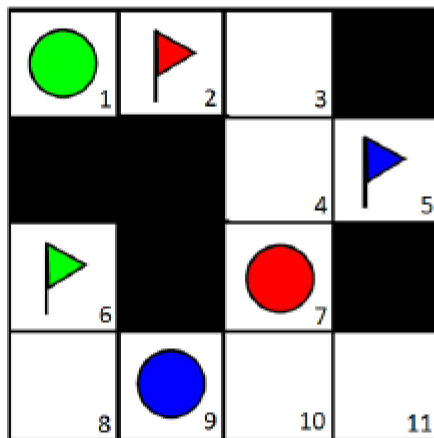
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Definitions - joint plan

Definition 11. A *joint plan* Π is a set of valid plans with the same length, one for each agent. We say that Π is a *valid joint plan*, if there are no collisions between any of the agents. A valid joint plan is a solution to a MAPF instance. We denote |Π| as the length of the joint plan.

Note: Time is discrete and each transition takes 1 time unit.

Note: Individual plans have the same length. After reaching the goal, the robot does not disappear and stays in location.



Timestep	0	1	2	3	4	5	6	7	8	9
Agent Green	1	2	3	4	7	10	9	8	6	6
Agent Red	7	4	5	5	4	3	2	2	2	2
Agent Blue	9	10	11	11	11	11	10	7	4	5

Definitions - cost functions

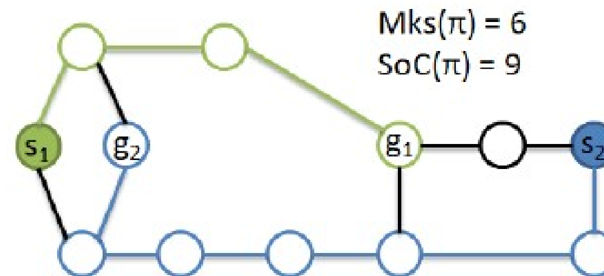
Let T_i be the last timestep agent a_i reached its goal location.

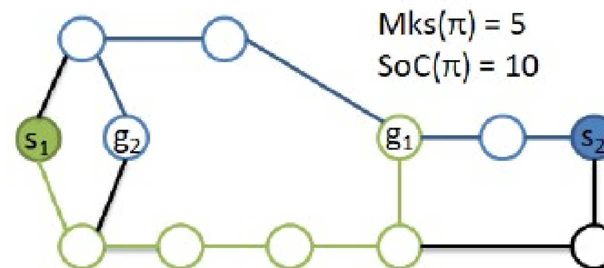
Definition 12. The makespan of plan Π for agents A is

$$Mks(\Pi) = \max_{a_i \in A} T_i$$

Definition 13. The sum of costs of plan Π for agents A is

$$SoC(\Pi) = \sum_{a_i \in A} T_i$$





Resources

Definitions - https://arxiv.org/abs/1906.08291

Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, Eli Boyarski: Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks*. SOCS 2019: 151-158*

Benchmarks - https://movingai.com/benchmarks/mapf/index.html

Community web - https://mapf.info/

My dissertation - https://dspace.cuni.cz/handle/20.500.11956/123554

Computational hardness

NP-Complete decision problems

- Is there a solution in given Mks
- Is there a solution in given SoC

    → Optimization is NP-Hard

Deciding if a problem is solvable is in P

-

Computational hardness

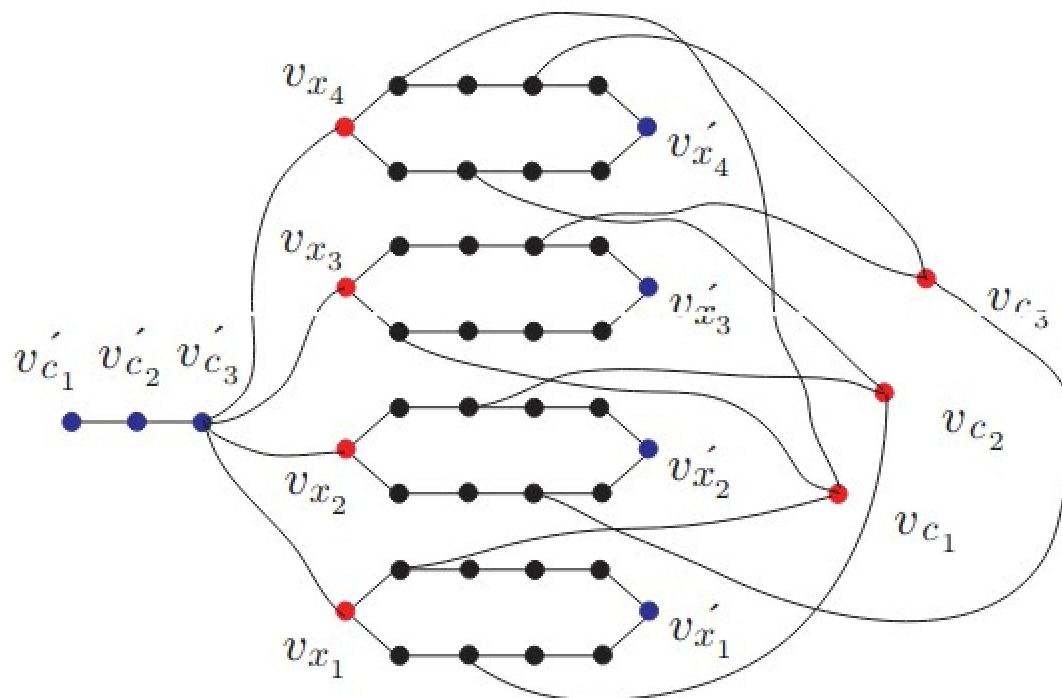
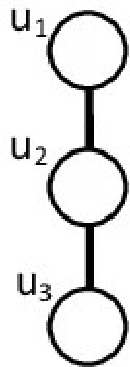
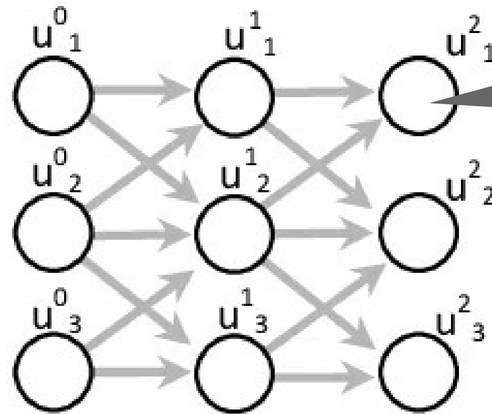


Figure 3: An $\mathrm{MPP}_{\mathrm{pr}}$ instance constructed from the **3SAT** instance $(\{x_1, x_2, x_3, x_4\}, \{x_1 \vee \neg x_3 \vee x_4, \neg x_1 \vee x_2 \vee \neg x_4, \neg x_2 \vee x_3 \vee x_4\})$. The red vertices are the start vertices and the blues one the goals.

Single agent algorithm - A*

- A* from Intro to AI or AI1 class





Some locations are blocked in given timestep. Keep track of „conflict avoidance table" (CAT)

May be added on demand

P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Systems Science and Cybernetics, 1968.

Single agent algorithm - A*

Can we do better than A*?

- Theoretically, no
- Practically, YES
  - The graphs are usually grids

    Grid-Based Path Planning Competition

    https://gppc.search-conference.org/
  - The pathfinding is done with time expansion

Single agent algorithm - SIPP

- State is location and time
- Keep track of moving obstacles' times (unsafe intervals)
- Same as A*, but generate successors based on safe intervals

```
1  g(s_start) = 0; OPEN = ∅;
2  insert s_start into OPEN with f(s_start) = h(s_start);
3  while(s_goal is not expanded)
4    remove s with the smallest f-value from OPEN;
5    successors = getSuccessors(s);
6    for each s' in successors
7      if s' was not visited before then
8        f(s') = g(s') = ∞;
9      if g(s') > g(s) + c(s, s')
10       g(s') = g(s) + c(s, s');
11       updateTime(s');
12       f(s') = g(s') + h(s');
13       insert s' into OPEN with f(s');
```

```
1  getSuccessors(s)
2    successors = ∅;
3    for each m in M(s)
4      cfg = configuration of m applied to s
5      m_time = time to execute m
6      start_t = time(s) + m_time
7      end_t = endTime(interval(s)) + m_time
8      for each safe interval i in cfg
9        if startTime(i) > end_t or endTime(i) < start_t
10         continue
11       t = earliest arrival time at cfg during interval i with no collisions
12       if t does not exist
13         continue
14       s' = state of configuration cfg with interval i and time t
15       insert s' into successors
16   return successors;
```

Taken from SIPP paper, there might be a mistake...

M. Phillips, M. Likhachev. SIPP: Safe interval path planning for dynamic environments. ICRA, 2011.

Single agent algorithm - SIPP

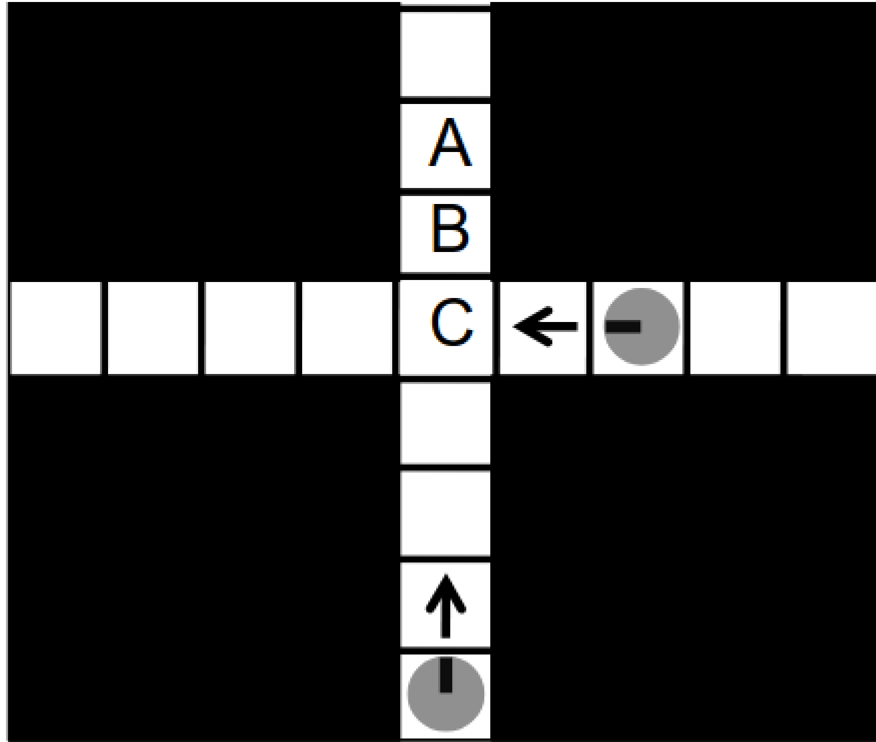


Fig. 6. An example environment with two dynamic obstacles moving at a speed of one cell per timestep. There are three highlighted configurations (A,B,C) and the robot is located at configuration B.

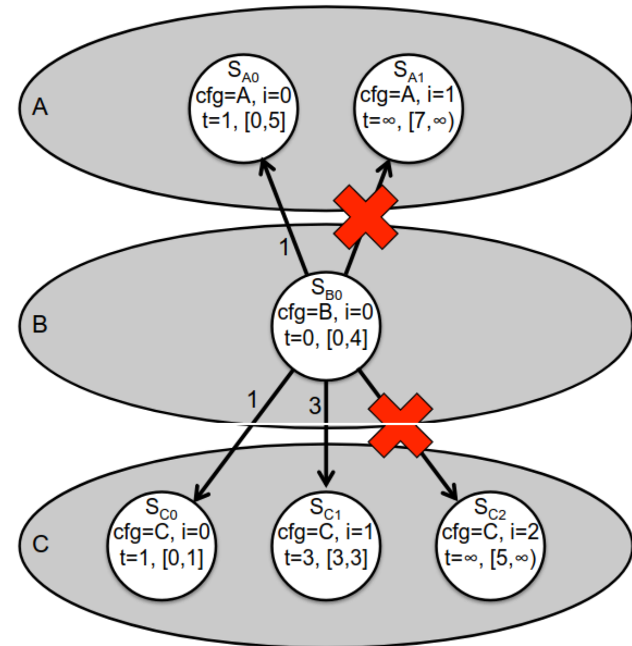


Fig. 7. An illustration of an expansion of state S_{B0}. Each white circle in this figure represents a state. The first line in each state is its name. The second line defines the state by indicating its configuration (cfg) from Figure 6 and its safe interval (i). The third line shows the earliest known time we can reach this state (t) and the safe interval [a,b]. The gray ovals group states by their configuration. The arrows indicate transitions from one state to another, labeled with their cost. The red X indicate invalid transitions due to collision with a dynamic obstacle.

Single agent algorithm - SIPP

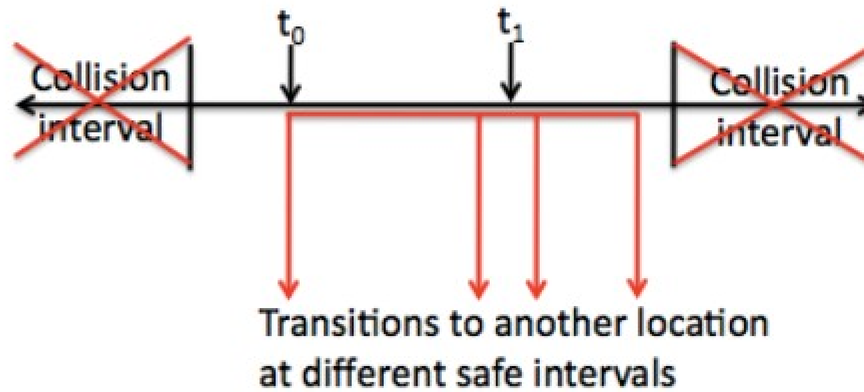


Fig. 8. The set of successors generated from t_0 is a superset of the set of successors generated from t_1

Conflict-based search (CBS)

1) Navigate agents individually
2) Check for conflicts
   I. No conflicts – return
   II. Conflicts – add constraints
3) Resolve conflicts in best-first manner

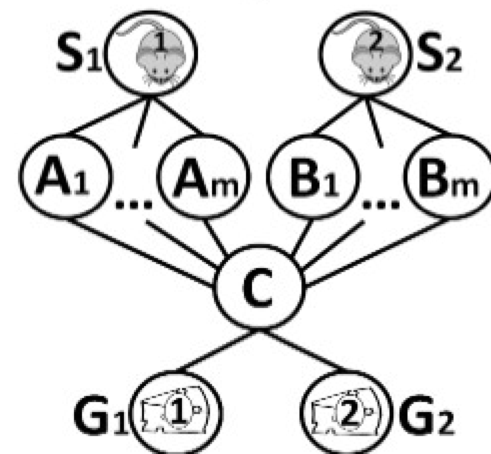
G. Sharon, R. Stern, A. Felner, and N. Sturtevant. Conflict-based search for optimal multi-agent path finding. AAAI, 2012.

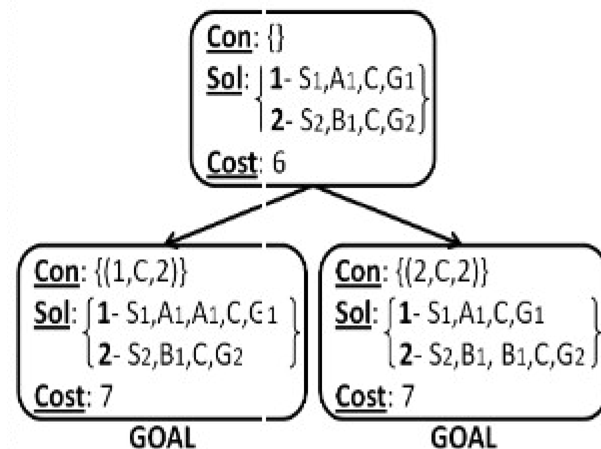
CBS

Algorithm 1: high-level of CBS

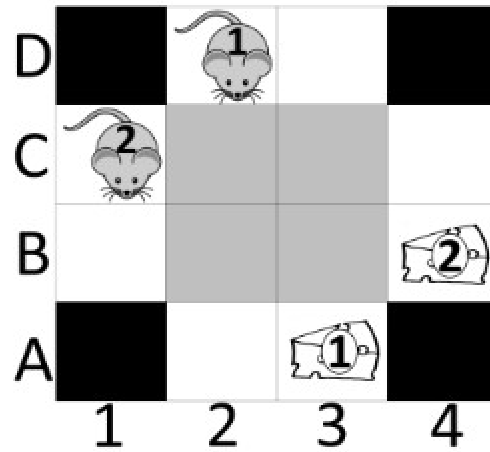
Input: MAPF instance

1. $R.constraints = \emptyset$
2. $R.solution =$ find individual paths using the low-level()
3. $R.cost = SIC(R.solution)$
4. insert R to OPEN
5. **while** OPEN *not empty* **do**
6. P $\leftarrow$ best node from OPEN // *lowest solution cost*
7. Validate the paths in P until a conflict occurs.
8. **if** *P has no conflict* **then**
9. **return** P.solution // *P is goal*
10. C $\leftarrow$ first conflict (a_i, a_j, v, t) in P
11. **foreach** *agent* a_i *in C* **do**
12. A $\leftarrow$ new node
13. A.constriants $\leftarrow$ P.constriants + (a_i, s, t)
14. A.solution $\leftarrow$ P.solution.
15. Update A.solution by invoking low-level(a_i)
16. $A.cost = SIC(A.solution)$
17. Insert A to OPEN





CBS - pathological instance



CBS - properties

1) Optimal

- Proof idea: CT is searched in best-first manner

2) Not complete!

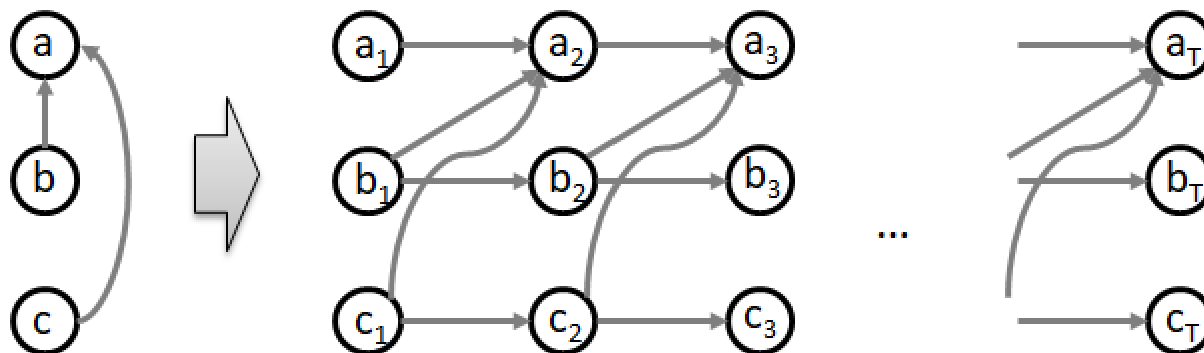
- Can not detect unsolvable instances
- Will expand CT forever

CBS - improvements

1) Use CAT
2) Prioritize conflicts
3) Positive constraints (disjoint splitting)
4) Symmetry / Conflict reasoning
5) Meta agents
6) Add heuristic functions
7) ...

https://github.com/Jiaoyang-Li/CBSH2-RTC

Reduction to SAT - makespan



- Variable At(v,i,t) – agent a_i is at vertex v at timestep t
- Variable Pass(u,v,i,t) – agent a_i moves over (u,v) at timestep t
- Iteratively add layers
- First feasible solution is (makespan) optimal

Reduction to SAT - constraints

$$At(a_i, s_i, 1) \quad (1)$$

$$At(a_i, g_i, T_i) \quad (2)$$

$$\forall t \in \{1, \ldots, T_i\}, \forall u, v \in V, u \neq v :$$

$$\neg At(a_i, u, t) \vee \neg At(a_i, v, t) \quad (3)$$

$$\forall t \in \{1, \ldots, T_i - 1\} :$$

$$At(a_i, u, t) \implies \bigvee_{(u,v)\in E} Pass(a_i, u, v, t) \quad (4)$$

$$\forall t \in \{1, \ldots, T_i - 1\} :$$

$$Pass(a_i, u, v, t) \implies At(a_i, v, t + 1) \quad (5)$$

$$\forall a_i, a_j \in A, a_i \neq a_j, \forall t \in \{1, \ldots, \min(T_i, T_j)\} :$$

$$\neg At(a_i, v, t) \vee \neg At(a_j, v, t) \quad (6)$$

$$\forall a_i, a_j \in A, \forall t \in \{1, \ldots, \min(T_i, T_j) - 1\}, \forall (u, v) \in E :$$

$$\neg Pass(a_i, u, v, t) \vee \neg Pass(a_j, v, u, t) \quad (7)$$

Start and goal

Agent is in single location

Agent moves through an edge

Agent on edge ends in a vertex

No vertex conflicts

No swap conflicts

Reduction to SAT - sum of costs

- Increasing Mks may decrease SoC. How many layers to add?
- How to decide if agent is in g_i for the last time?
- How to limit number in SAT?

- Late(i,t) - a_i is late in t $\neg At(a_i, g_i, t) \implies Late(a_i, t)$
- Propagate in time $\neg Late(a_i, t) \implies \neg Late(a_i, t+1)$
- "At most k" encoding for all Late variables
- Increase both Mks and SoC limit at the same time

P. Surynek, A. Felner, R. Stern, and E. Boyarski. Efficient SAT approach to multi-agent path finding under the sum of costs objective. ECAI, 2016.

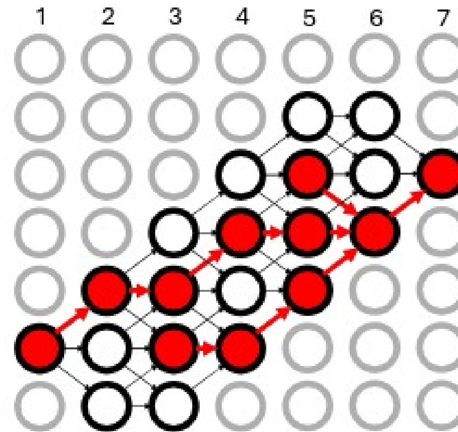
Reduction to SAT - improvements

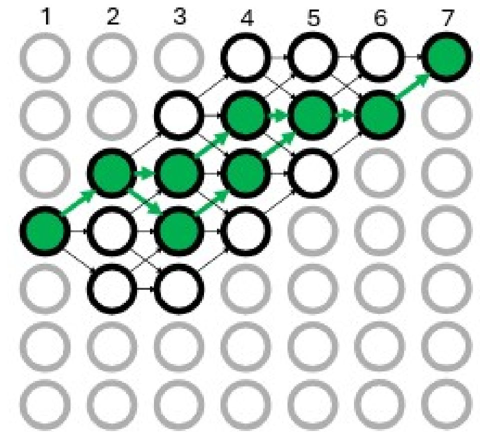
1) Do not include unreachable states (MDD-SAT)
2) Do not include collisions at first (Lazy-SMT)
3) Do not include all positions (Subgraph method)
4) Change encoding
5) Use better solvers
6) Use more tools (Monosat)

https://github.com/svancaj/MAPF-encodings https://github.com/potassco/mapf-subgraph-system

Reduction to SAT - Monosat

- Monosat (SMT solver with graph theories)
- Actually build the TEG





$$At(a_i, s_i, 1)$$

$$At(a_i, g_i, T_i)$$

$$\forall t \in \{1, \ldots, T_i\}, \forall u, v \in V, u \neq v :$$

$$\neg At(a_i, u, t) \vee \neg At(a_i, v, t)$$

$$\forall t \in \{1, \ldots, T_i - 1\} :$$

$$At(a_i, u, t) \implies \bigvee_{(u,v) \in E} Pass(a_i, u, v, t) \quad (4)$$

$$\forall t \in \{1, \ldots, T_i - 1\} :$$

$$Pass(a_i, u, v, t) \implies At(a_i, v, t+1) \quad (5)$$

$$\forall a_i, a_j \in A, a_i \neq a_j, \forall t \in \{1, \ldots, \min(T_i, T_j)\} :$$

$$\neg At(a_i, v, t) \vee \neg At(a_j, v, t) \quad (6)$$

$$\forall a_i, a_j \in A, \forall t \in \{1, \ldots, \min(T_i, T_j) - 1\}, \forall (u, v) \in E :$$

$$\neg Pass(a_i, u, v, t) \vee \neg Pass(a_j, v, u, t) \quad (7)$$

https://github.com/svancaj/MAPF-encodings

https://github.com/potassco/mapf-subgraph-system

BCP for MAPF

- Follows branch-and-cut-and-price algorithm (from combinatorial optimization)

1) Master problem
2) Pricer
3) Separators
4) Branching

E. Lam, P. Bodic, D. Harabor, P. Stuckey. Branch-and-cut-and-price for multi-agent path finding. Comput. Oper. Res., 2022.

BCP - master problem

- Given a set of paths for each agent
- LP to select path for each agent
  - Might be fractions!!!

$$\min \sum_{a \in \mathcal{A}} \sum_{p \in \mathcal{P}_a} c_p \lambda_p \tag{1a}$$

subject to

$$\sum_{p \in \mathcal{P}_a} \lambda_p = 1 \qquad \forall a \in \mathcal{A}, \tag{1b}$$

$$\lambda_p \geq 0 \qquad \forall a \in \mathcal{A}, p \in \mathcal{P}_a. \tag{1c}$$

BCP - pricer

- A* to find better paths for agents
- These paths are added into the set of paths the master problem chooses from
- The edges are weighted by use of other agents (i.e. constraints on conflicts)

BCP - separators

- Constraints to avoid conflicts
- Added to the master problem upon detection

4.3.2. Vertex conflicts

A vertex conflict occurs at $v \in \mathcal{V}$ whenever v is visited by more than one agent. That is, whenever

$$\sum_{a \in \mathcal{A}} X_v^a > 1. \tag{4}$$

Given a solution to the master problem, the separator for vertex conflicts first computes X_v^a for all $a \in \mathcal{A}$ and $v \in \mathcal{V}$ using Eqs. (2) and (3). Next, it builds the constraint

$$\sum_{a \in \mathcal{A}} X_v^a \leq 1, \tag{5}$$

for every $v \in \mathcal{V}$ that satisfy Condition (4). It then substitutes for X_v^a in Constraint (5) using Eqs. (2) and (3), forming a constraint over the λ_p variables in the master problem. Finally, the separator adds this constraint to the master problem.

4.3.3. Edge conflicts

Consider a move edge $e = ((l_1, t), (l_2, t + 1)) \in \mathcal{E}$ where $l_1 \neq l_2$. An edge conflict occurs at e whenever e or its reverse e' are traversed by more than one agent. That is,

$$\sum_{a \in \mathcal{A}} (X_e^a + X_{e'}^a) > 1.$$

The separator for edge conflicts is similar to the separator for vertex conflicts. The edge conflict at e is removed by adding the constraint

$$\sum_{a \in \mathcal{A}} (X_e^a + X_{e'}^a) \leq 1 \tag{6}$$

to the master problem after substitution using Eq. (2).

BCP - branching

- The solution may be fractional! (LP)
- If it is, split in two branches, each forcing whole number solution
- Repeat the master problem

Independence detection

Algorithm 1: Independence Detection.

1: assign each agent to a group
2: compute plan for each group
3: **while** there is a conflict in plans **do**
4: $G_1, G_2 \leftarrow$ conflicting groups
5: **if** G_1 and G_2 conflicted before **then**
6: merge G_1 and G_2 into new group G
7: find plan for G
8: continue
9: **else if** can replan G_1 and avoid G_2 **then**
10: replan G_1 and avoid G_2
11: continue
12: **else if** can replan G_2 and avoid G_1 **then**
13: replan G_2 and avoid G_1
14: continue
15: **else**
16: merge G_1 and G_2 into new group G
17: find plan for G
18: **end if**
19: **end while**
20: solution $\leftarrow$ path of all groups combined

Honorable mentions

- Cooperative A*
- Operator decomposition
- Increasing cost tree search