Multi-Agent Path Finding

Luigi Palopoli, Enrico Saccon

UNIVERSITÀ DI TRENTO

What?

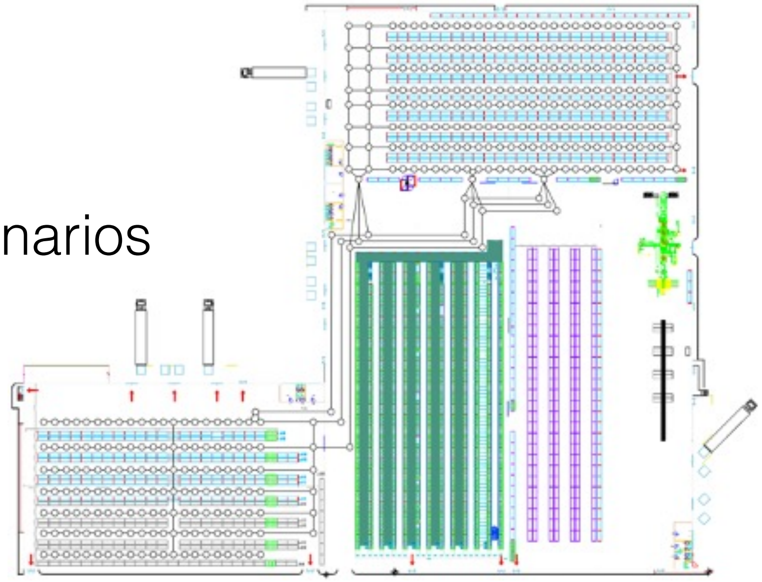
- The standard Multi-Agent Path Finding (MAPF) problem [1] consists in:

given a map and N agents, finding the best *feasible joint plan* Π such that each agent moves from its initial position to its final position minimizing an *objective function*

- It's a combinatorial problem
- In the standard definition, the map is usually a **grid**
- Solution minimizes objective function --> time, space, resource, etc.
- **Centralized approach**

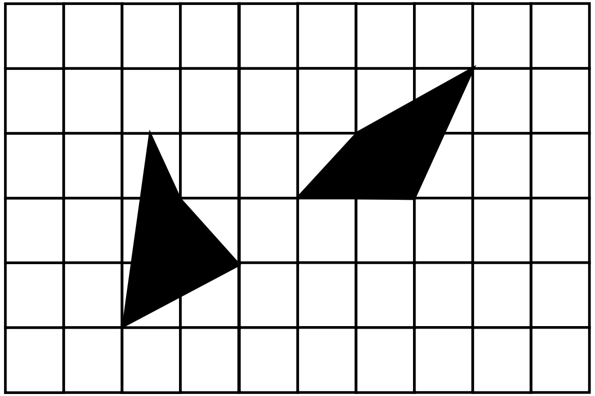
Why?

- Robotics is a main topic in the Industrial Revolution 4.0 and 5.0
- Used in an increasing number of scenarios
- Challanging problem [2]
- Algorithms can be applied to other scenarios



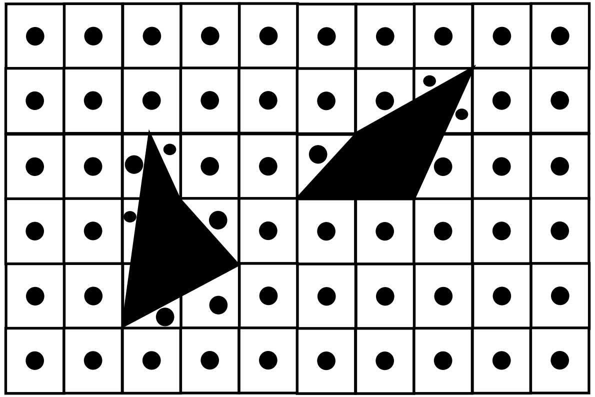
Map Decomposition

- How do I deal with obstacles and map borders?
- Many algorithms to partition the map in cells:
  - Exact cell decomposition
  - Approximate cell decomposition
  - Maximum clearence
  - Morse decomposition
  - Brushfire decomposition
- Each cell is a node --> connectivity graph



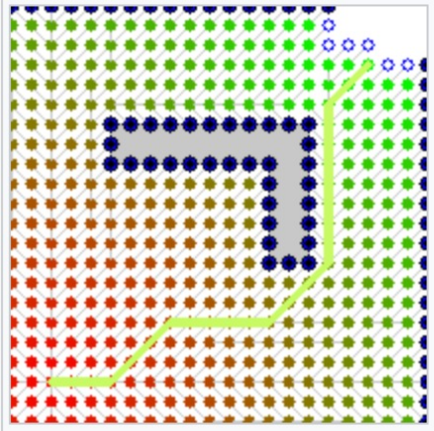
Map Decomposition

- How do I deal with obstacles and map borders?
- Many algorithms to partition the map in cells:
  - Exact cell decomposition
  - Approximate cell decomposition
  - Maximum clearence
  - Morse decomposition
  - Brushfire decomposition
- Each cell is a node --> connectivity graph



Single-Agent Path Finding (SAPF)

- Given a graph $G = (V, E)$, find the *best feasible* plan π_i to go from an initial position to a final position
- The problem usually consists in computing the shortest path between two nodes on a graph
- Deterministic algorithms, e.g., Dijkstra's
- Heuristic algorithms, e.g., A*



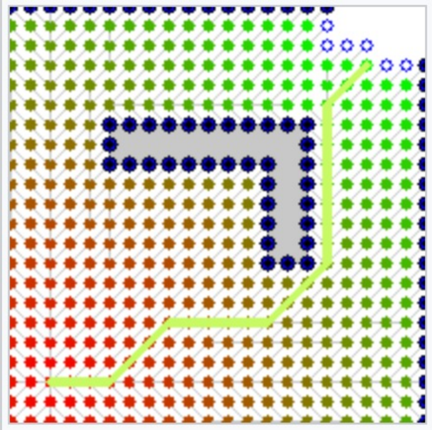
Images taken from the corresponding Wikipedia page

Single-Agent Path Finding (SAPF)

Dijkstra's

- Complete
- Optimal
- Evolution of BFS
- Cost function:

$$f(x) = g(x)$$

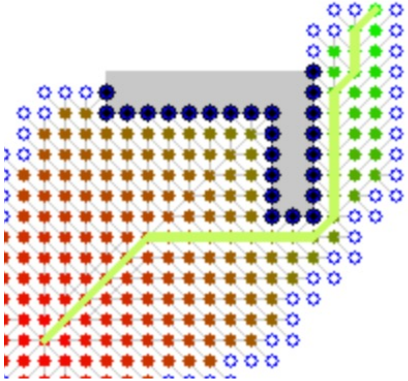


A*

- Complete?
- Optimal?
- Admissible heuristic $h(x)$:

$$f(x) = g(x) + h(x)$$

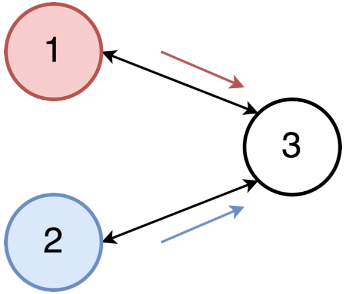
$$h(x) \leq d(x, y) + h\,(y)$$

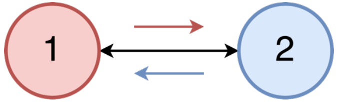


Images taken from the corresponding Wikipedia page

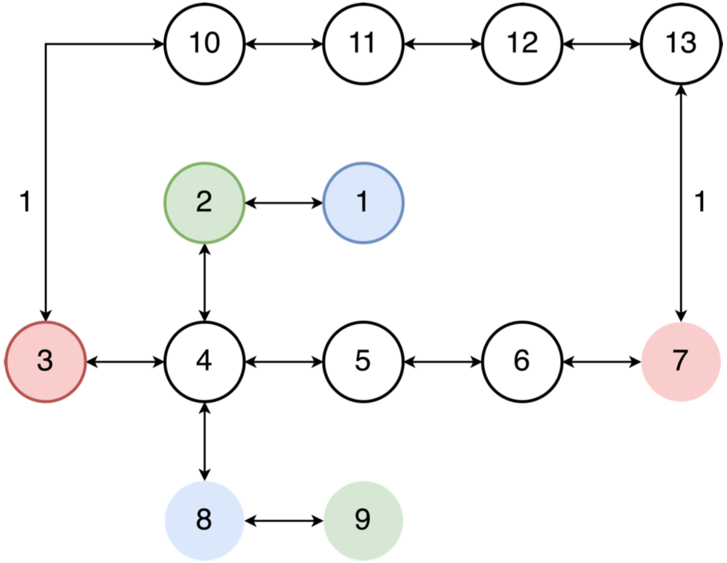
Multi-Agent Path Finding Overview

- Given a graph $G = (V, E)$ and k agents, find the *best feasible joint* plan Π such that each agent moves from its initial position to its final position minimizing an objective function
- A joint plan is a set of single plans: $\Pi = \{\pi_1, \ldots, \pi_k\}$
- A path if feasible if no conflict arises [1]:
  - Vertex conflicts
  - Edge conflicts
  - Swap conflicts
- Objective functions:
  - Makespan (MKS)
  - Sum of individual costs (SIC)





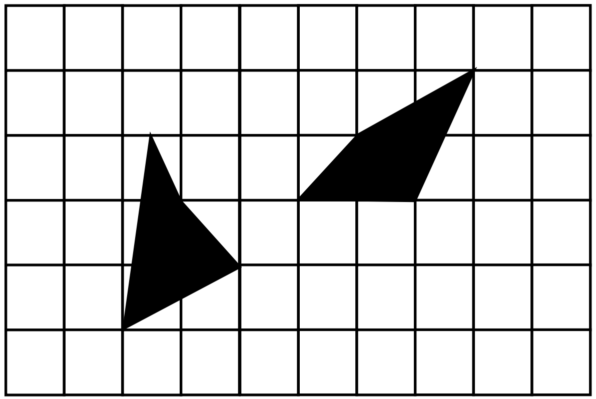
Multi-Agent Path Finding Cost Functions

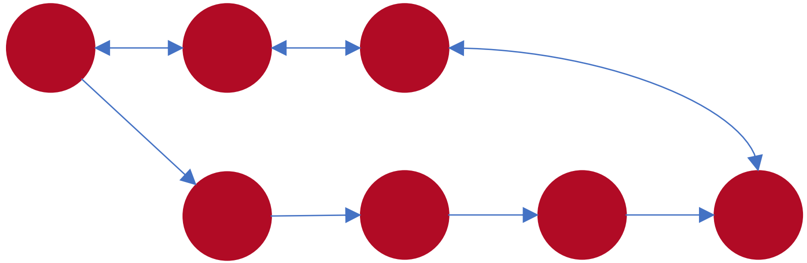


Π_i	$\mathtt{SIC}(\Pi_i)$	$\mathtt{MKS}(\Pi_i)$
$\Pi_1 = \begin{cases} \pi_1 = \{3, 10, 11, 12, 13, 7\} \\ \pi_2 = \{2, 4, 8, 9\} \\ \pi_3 = \{1, 2, 4, 8\} \end{cases}$	14	6
$\Pi_2 = \begin{cases} \pi_1 = \{3, 4, 5, 6, 7\} \\ \pi_2 = \{2, 2, 4, 8, 9\} \\ \pi_3 = \{1, 1, 2, 4, 8\} \end{cases}$	15	5

Multi-Agent Path Finding Overview

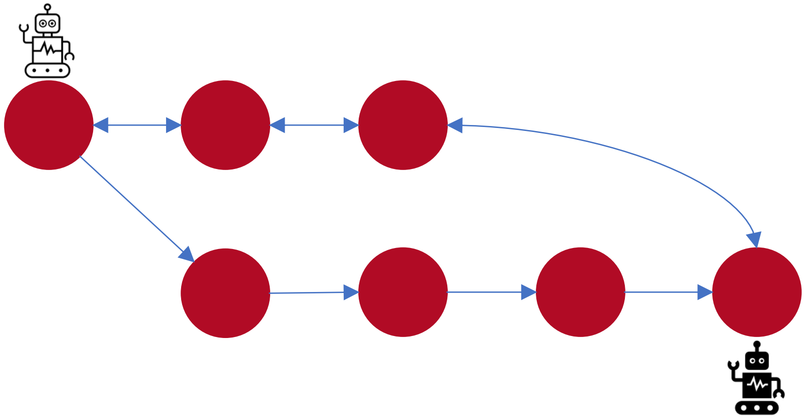
- Given a graph $G = (V, E)$ and k agents, find the *best feasible joint* plan Π such that each agent moves from its initial position to its final position minimizing an objective function
- Time is **discretized**
- Each agent can either
  - move to an adjacent cell; or
  - stay on the same cell --> variation

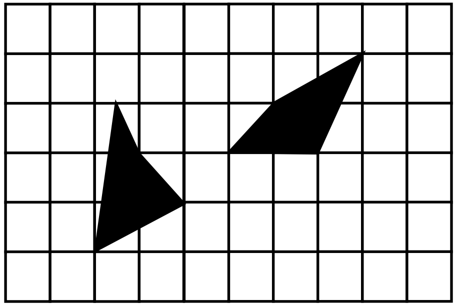




Multi-Agent Path Finding Overview

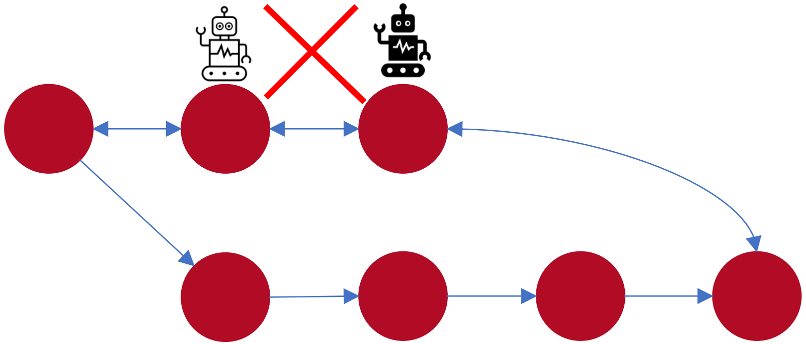
- Given a graph $G = (V, E)$ and k agents, find the *best feasible joint* plan Π such that each agent moves from its initial position to its final position minimizing an objective function
- Time is **discretized**
- Each agent can either
  - move to an adjacent cell; or
  - stay on the same cell --> variation
- Edges have **unitary** costs

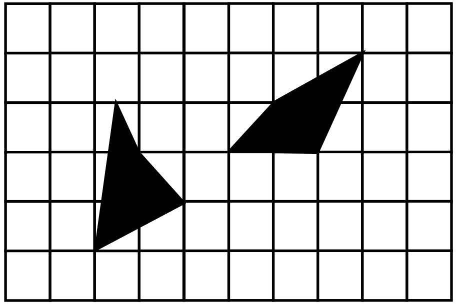




Multi-Agent Path Finding Overview

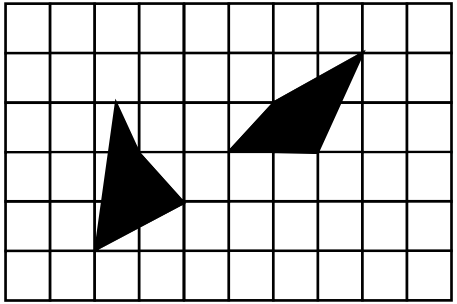
- Given a graph $G = (V, E)$ and k agents, find the *best feasible joint* plan Π such that each agent moves from its initial position to its final position minimizing an objective function
- Time is **discretized**
- Each agent can either
  - move to an adjacent cell; or
  - stay on the same cell --> variation
- Edges have **unitary** costs

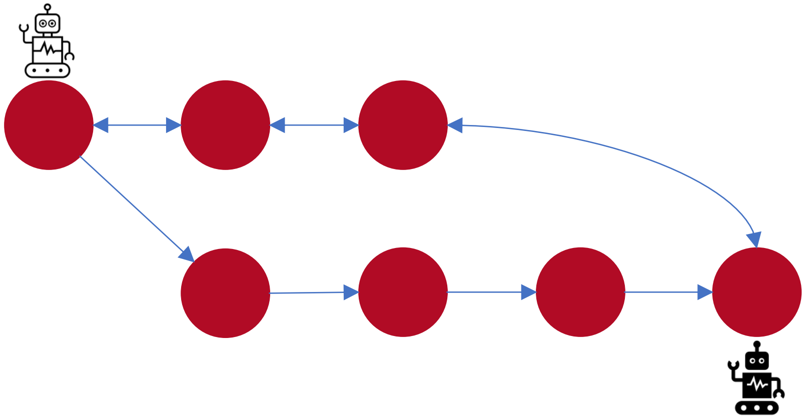




Multi-Agent Path Finding Overview

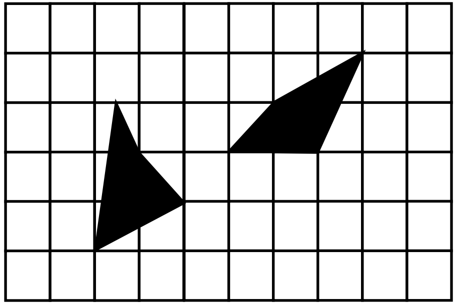
- Given a graph $G = (V, E)$ and k agents, find the *best feasible joint* plan Π such that each agent moves from its initial position to its final position minimizing an objective function
- Time is **discretized**
- Each agent can either
  - move to an adjacent cell; or
  - stay on the same cell --> variation
- Edges have **unitary** costs

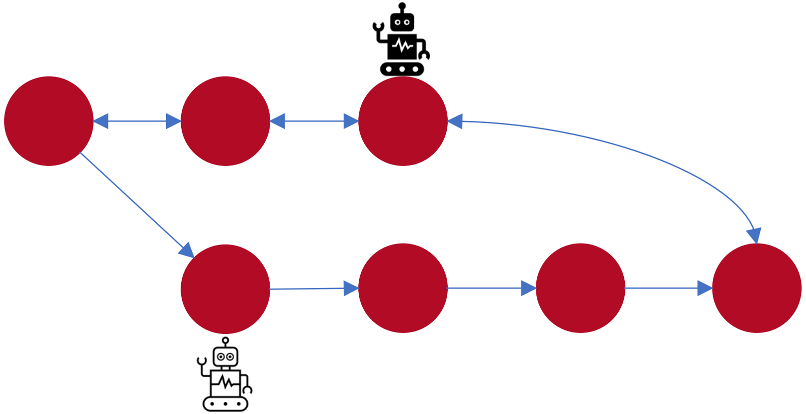




Multi-Agent Path Finding Overview

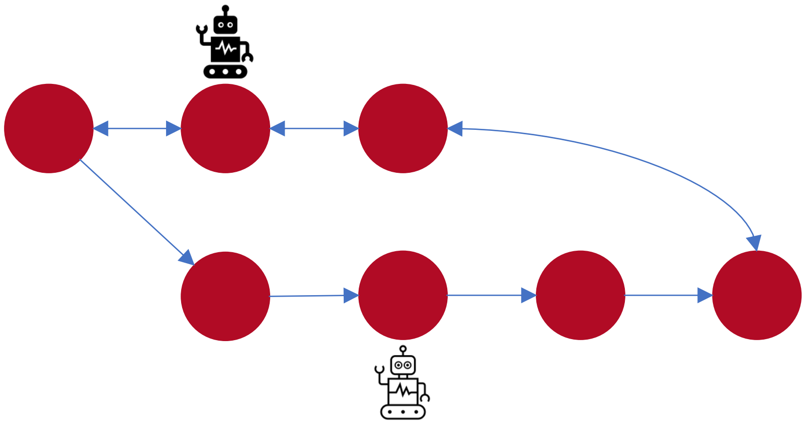
- Given a graph $G = (V, E)$ and k agents, find the *best feasible joint* plan Π such that each agent moves from its initial position to its final position minimizing an objective function
- Time is **discretized**
- Each agent can either
  - move to an adjacent cell; or
  - stay on the same cell --> variation
- Edges have **unitary** costs

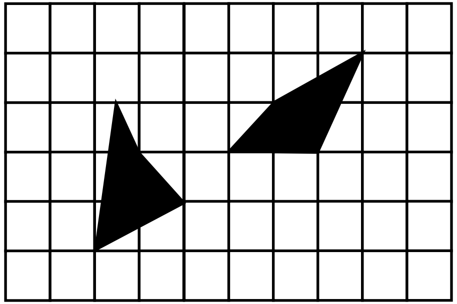




Multi-Agent Path Finding Overview

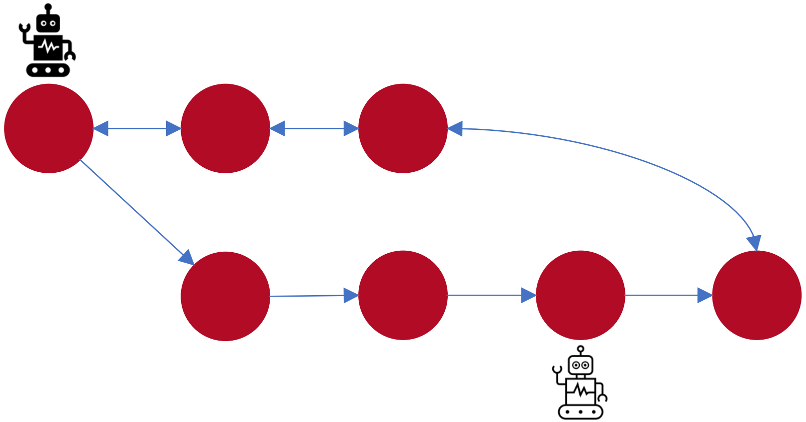
- Given a graph $G = (V, E)$ and k agents, find the *best feasible joint* plan Π such that each agent moves from its initial position to its final position minimizing an objective function
- Time is **discretized**
- Each agent can either
  - move to an adjacent cell; or
  - stay on the same cell --> variation
- Edges have **unitary** costs

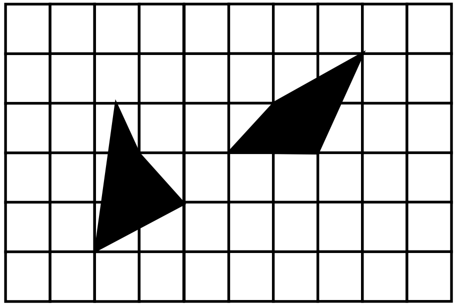




Multi-Agent Path Finding Overview

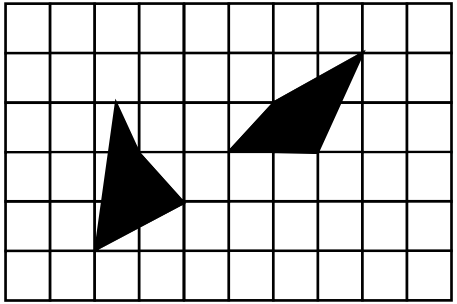
- Given a graph $G = (V, E)$ and k agents, find the *best feasible joint* plan Π such that each agent moves from its initial position to its final position minimizing an objective function
- Time is **discretized**
- Each agent can either
  - move to an adjacent cell; or
  - stay on the same cell --> variation
- Edges have **unitary** costs

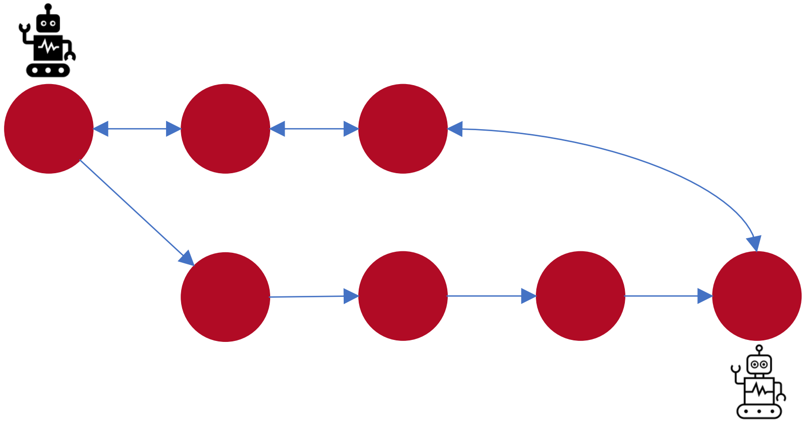




Multi-Agent Path Finding Overview

- Given a graph $G = (V, E)$ and k agents, find the *best feasible joint* plan Π such that each agent moves from its initial position to its final position minimizing an objective function
- Time is **discretized**
- Each agent can either
  - move to an adjacent cell; or
  - stay on the same cell --> variation
- Edges have **unitary** costs



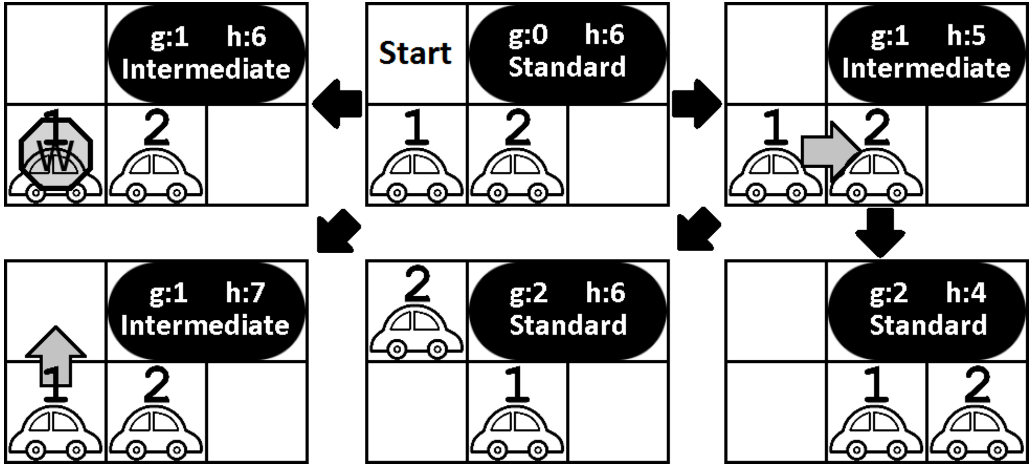


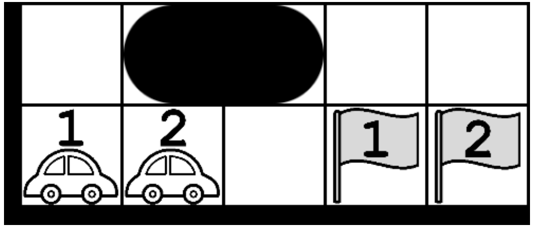
Enhanced Versions of A* [3]

- Instead of considering one position, considers a tuple
- At each timestep:
    - the current state space contains the position of all N agents
    - the next state space has to consider all possible movements of the agents
- This gets bad pretty fast
- **Operator decomposition (OD)**
- **Simple independence detection (SID)**

Enhanced Versions of A* – OD

- Do not consider N agents per each time step
- Considers 1 agent at a time and it requires N operations to advance 1 timestep
- The order in which the agents are chosen is fixed
- It's not complete or optimal without the correct heuristic





Images taken from [3]

Enhanced Versions of A* – SID

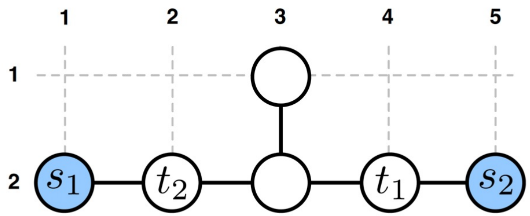
- Also in OD, the search space is still exponential in the number of agents
- Agents whose path *does not collide*, are in **independent** group and should be considered separetely
- The algorithm follow these steps:
  - Start with the optimal paths as if the agents were alone
  - If there are conflicts, divide the agents in conflict groups
  - Solve the conflicts in the group
  - Repeat

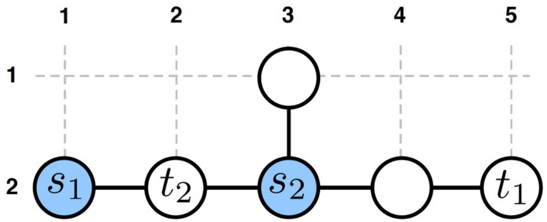
Priority Planning (PP) [4]

- Each agent has a fixed priority
- The higher the priority, the first the agent's path is computed
- Pros:
  - This allows for keeping the complexity small
- Cons:
  - Not complete [5]
  - Not optimal
- Many works, focus on using ML or DL approaches to learning the priorities

Priority Planning (PP)

- There are some instances in which priority planning cannot solve the problem [5]

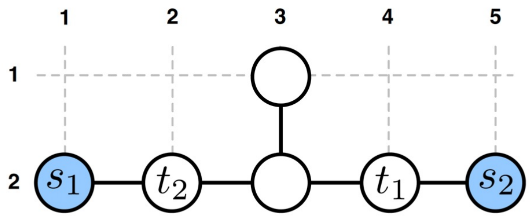




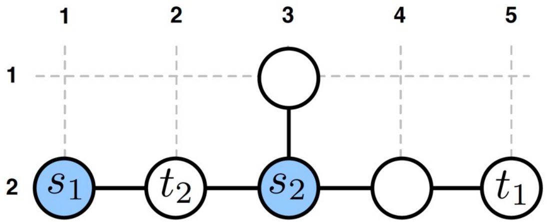
Images taken from [5]

Priority Planning (PP)

- There are some instances in which priority planning cannot solve the problem [5]



Not solvable



Solvable

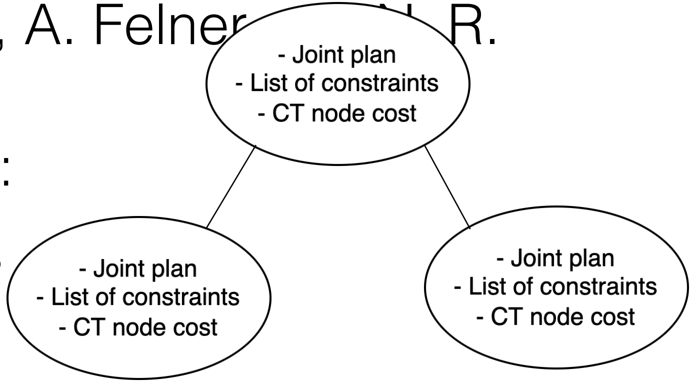
- A MAPF instance must be *well-formed* to be solvable
  - Agents can wait for any amount of time on the initial or final node without blocking other agents

Images taken from [5]

Conflict Based Search (CBS)

- Proposed in 2015 by G. Sharon, R.Stern, A. Felner, ... N. R. Sturtevant [6]
- Optimal algorithm divided in two phases:

1. High-level search → manages conflicts
2. Low-level search → SAPF problem



- Creates a *constraint tree*
- When a conflict is found, it creates two children:
  - Agent a_i cannot be on node n at time t
  - Agent a_j cannot be on node n at time t
- The search continues until a joint plan without conflicts is found
- Nodes to be explored are chosen based on their joint cost

2. Low-level search → SAPF problem
   - The algorithm should be adapted to the problem

CBS Implementation – High-Level

- Start from a root node with:
  - No constraints
  - Joint plan computed as SAPF
- Iterate until a feasible solution is found
  - If one or more vertex conflicts were found, then create two new nodes:
    - Child 1: agent a_i cannot be on node n at time t
    - Child 2: agent a_j cannot be on node n at time t
  - If one or more swap conflicts were found, then create two new nodes:
    - Child 1: agent a_i cannot move from node n_1 to node n_2 at time t
    - Child 2: agent a_j cannot move from node n_2 to node n_1 at time t
- Conflicts are checked by comparing the positions of the solutions
- A joint plan is updated only for the new constraint's agent

CBS Implementation – Low-Level Spanning Tree

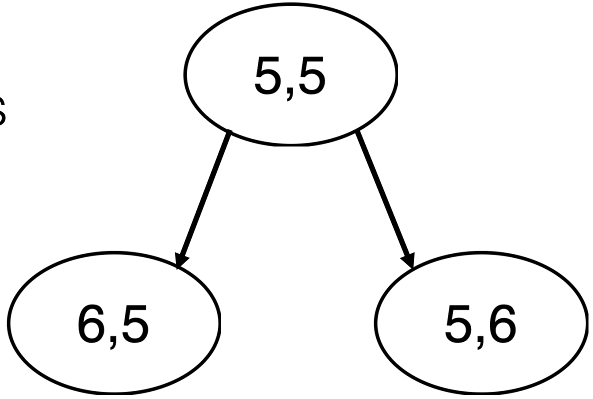
- Observation: difficult to find alternative paths when faced with contraints
- The algorithm computes all the possible paths between two points on a graph
- Then it starts from the shortest path and check if any conflicts arises with the constraints
- If it does, then it inserts waiting actions
- Finally, the shortest path is returned

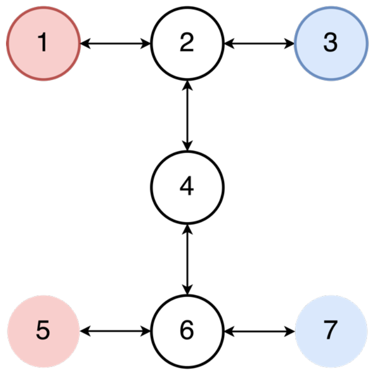
CBS Implementation – Low-Level TDSP

- This algorithm strongly takes from Dijkstra's
- The connectivity matrix is changed with Connection types
- Connection stores:
    - A vector of time steps
    - Type of connections: ONE, ZERO, LIMIT_ONCE, LIMIT_ALWAYS
- Adds placeholders to avoid vertex and swap conflicts by miming the wait action.
- Usually A* is used

Increasing Cost Tree Search (ICTS)

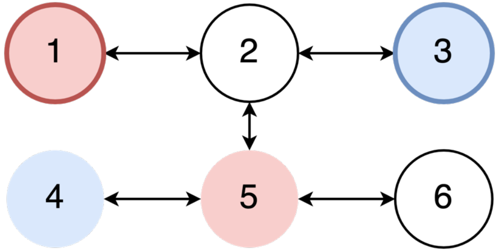
- It was proposed in 2013 by G. Sharon, R. Stern, M. Goldenberg and A. Felner and it is optimal [7]
- Similarly to CBS, ICTS is divided in two searches: high-level and low-level
- It uses an Increasing Cost Tree
- For each new level, k new nodes are created
- The search continues until a feasible solution is found

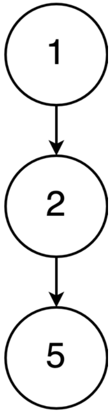




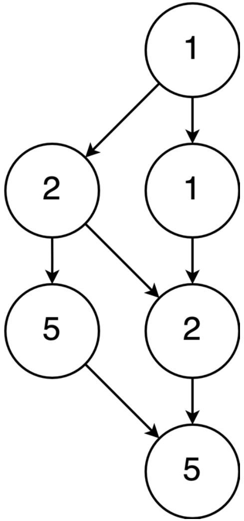
ICTS – Low-Level Search

- The low-level search is implemented using Multi-value Decision Diagrams (MDDs) [8]
- An MDD contains all the paths for an agent going from its initial position to their goal with a certain cost

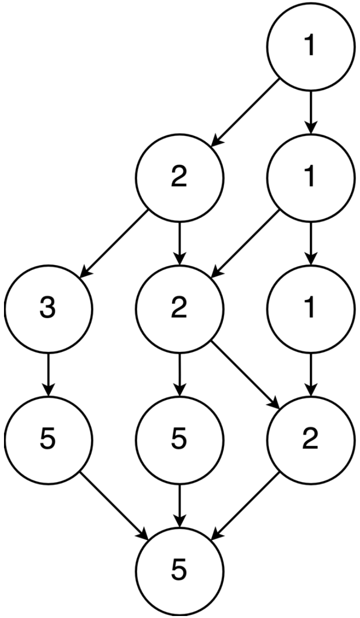




Cost 3



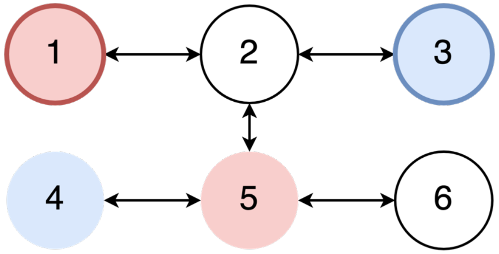
Cost 4

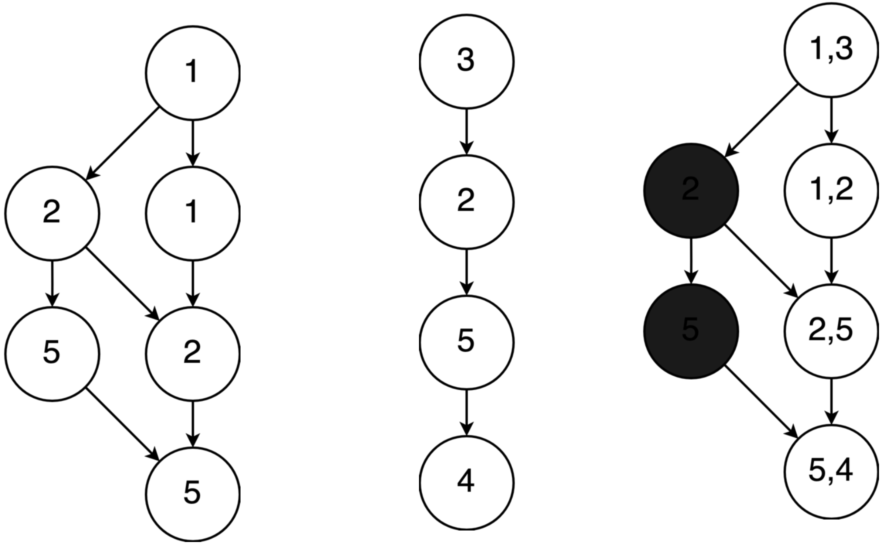


Cost 5

ICTS – Low-Level Search

- We can merge the MDDs of different agents to check for possible solutions
- The branches that have conflicts are removed





Red agent   Blue agent   Merged MDD

Constraint Programming (CP) [9]

- MAPF has been proven to be NP-Hard [10] → it can be reduced to SAT and MILP
- Constraint programming is a mathematical modeling paradigm in which some constraints are placed over some variables.
- Some constraints are:
  - Agents must be only on one vertex at each time step;
  - A node can be occupied by at most one agent at a time;
  - Agents start from their initial position and must be on their arrival position at the end;
  - Agents must move along edges.

Constraint Programming – Implementation

- Started from the work of Bartak et al using Picat
- Moved to IBM's CPLEX for performance
- Decision variables:

  X[n_steps][n_nodes][n_agents]   movement[n_agents][n_steps]

  goal_points[n_steps][n_nodes][n_agents]   edges[n_agents][n_steps]

  - An agent a can be only on one node n at a time s:

$$\forall s \in S, \forall a \in A, \sum_{n \in N} X[s][n][a] = 1$$

  - CPLEX: Python:

```python
for s in steps:
    for a in agents:
        m.add_constraint((m.sum(x[(s, n, a)] for n in nodes) == 1))
```

  - C++:

```cpp
FOREACH(s, steps) {
  FOREACH(a, agents) {
    IloExpr expr(env);
    FOREACH(n, nodes) {
      expr += x[s][n][a];
    }
    model.add( x: expr <= 1);
  }
}
```

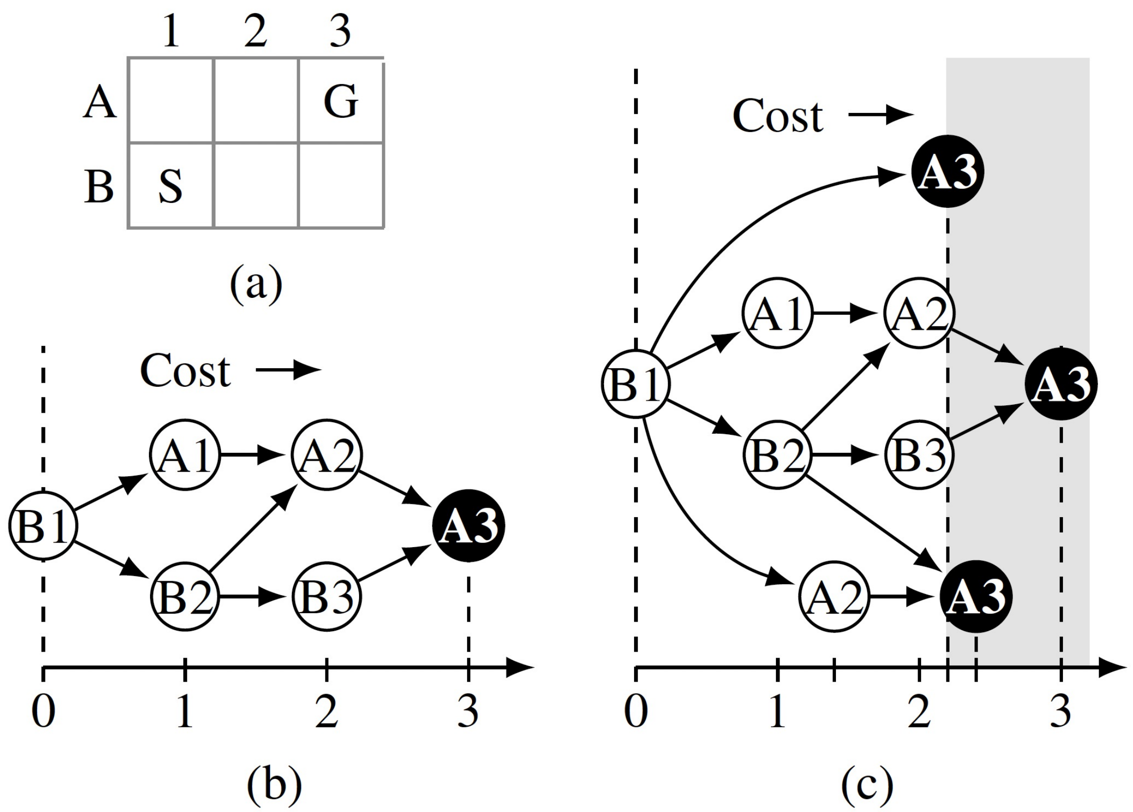
Constraint Programming – Constraints

- Examples of constraints are:
  - Agents cannot be on more than node each time step
  - A node cannot be occupied by more than one agent at time
  - An agent must occupy a neighbor of the node it is on at time $t + 1$ or stay on the same node
  - Agents start on their initial positions and end on their final positions
  - At a certain time, an edge cannot be used in more than one direction
  - The movement cost of an agent at a given time is the cost of the edge it is traversing
  - The agent must go through all the goals and only once before reaching the final position

Extended ICTS [11]

- Standard MAPF considers edges with unit costs
- Assumptions on agents' movements:
  - Wait on the center of the node
  - Moves in a straight line
  - Collision is the overlapping in an instant of time
- Two problem:
  - Partial time overlap conflict detection
    - Detect conflicts
  - Partial time overlap successor generation
    - Generate successive states

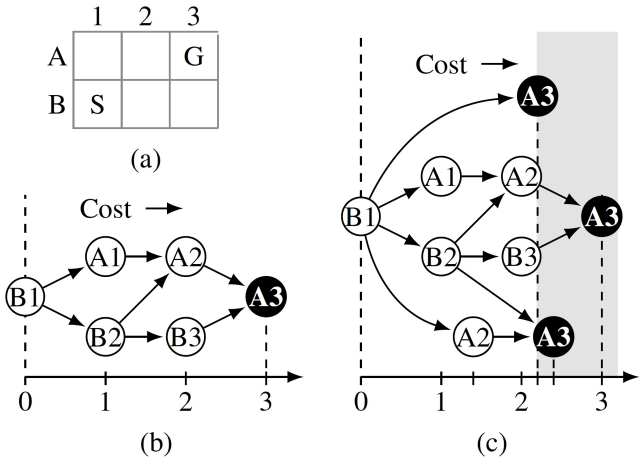
Extended ICTS



Images taken from [11]

Extended ICTS – Optimal

- In ICTS, the MDD had one root and one sink
- The next child is not obtained with an increment of 1, but we need to set an increment value δ
  - If δ is small --> the depth of the ICT may become too big
  - If δ is large --> search is reduced, but the solution may not be optimal
- The ICT nodes now contain intervals:
  - Lower bound is the solution minimum
  - Higher bound is the solution maximum
- The low-level is changed from a satisfactory problem to an optimal one: find the solution with the minimum cost in the interval



Images taken from [11]

Extended ICTS – Heuristics

- ϵ-ICTS: considers the low-level a satisfactory problem
  - This allows the algorithm to find a solution that is bounded sub-optimal
- w-ICTS: we can bound the sub-optimality for the generation of the next step to values of δ by adding a weight value w

CBS Heuristics

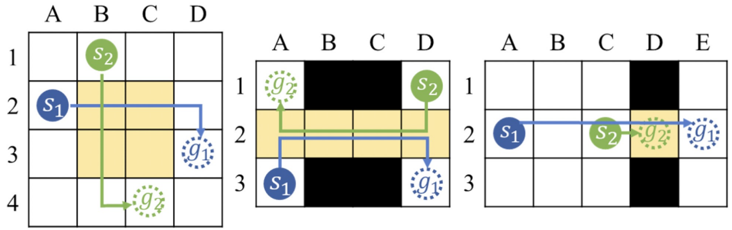
- CBS has been the start of the show with many improvements:
  - Bypassing conflicts [12]
  - Prioritizing conflicts [13]
  - Symmetry reasoning [14]
- And also a number of different heuristics:
  - ECBS [15]
  - EECBS [16]
  - EEEEECBS (Saccon et al., 2025)
  - EEEEEEEEEEEEEEECBS (Saccon et al., 2030).

CBS Heuristics

- CBS has been the start of the show with many improvements:
  - Bypassing conflicts [12]
  - Prioritizing conflicts [13]
  - Symmetry reasoning [14]
- And also a number of different heuristics:
  - ECBS [15]
  - EECBS [16]

CBS Heuristics

- Bypassing conflicts:
  - Do not split the node every time a conflict is found
  - When analyzing a conflict, modify the agents' path
  - If the cost if the new solution is the same as before and the number of conflicts is reduced --> do not split, but substitute
- Prioritizing conflicts:
  - A conflict is cardinal iff by solving the cost of both child CT nodes increases
  - A conflict is semi-cardinal iff the cost of only one child increases
  - A conflict is non-cardinal iff neither children's cost increased
- Symmetry reasoning



(a) Rectangle (b) Corridor (c) Target

Images taken from [16]

Enhanced CBS (ECBS)

- Instead of using A* uses Focal search [17]
  - Bounded suboptimal algorithm
  - Uses OPEN and FOCAL sets
  - FOCAL contains all those nodes that are have a weights suboptimal cost
  - The values in FOCAL are sorted using a function to estimate the cost-to-go
- ECBS implements focal search both for the low-level search and the high-level search:
  - The low-level is not sped up --> given an agent and a CT node, it returns
    - the path that minimizes the number of conflicts with other agents
    - the cost of the shortest path
  - The high-level search gets the costs of the shortest paths and can use focal search to speed up the analyses of the tree

Explicit Estimation Search

- Two main problems with ECBS high-level search:
  - It considers only the cost-to-go --> solution cost may be greater than the sub-optimality bound
  - At each time, the number of CT nodes with a similar cost is large --> FOCAL is rarely emptied
- The world is full of heuristic searches!
- Explicit Estimation Search (EES) [18] is a bounded-suboptimal search algorithm --> uses one more heuristic to overcome said focal behavior

Explicit Estimation Search (EES)

- EES is a bounded-suboptimal search algorithm --> uses one more heuristic to overcome said focal behavior
- It uses $\hat{h}$ and $\hat{d}$ to estimate the cost-to-go and the distance-to-go
- It keeps track of:
  - $best_f$, the node minimizing $f(n) = g(n) + h(n)$ from the FOCAL list
  - $best_{\hat{f}}$, the lowest predicted solution cost
  - $best_{\hat{d}}$, the node between the w admissible ones that appears closer to the target
- The node to explore is chosen based on
  1. $\hat{f}(best_{\hat{d}}) \leq w \cdot f\left(best_f\right) \rightarrow best_{\hat{d}}$ --> chose the node nearest to the goal
  2. $\hat{f}\left(best_{\hat{f}}\right) \leq w \cdot f\left(best_f\right) \rightarrow best_{\hat{f}}$ --> chose the node with the best path
  3. $best_f$ --> trust A*

Explicit Estimation CBS (EECBS)

- EECBS improves on ECBS by using EES on the *high-level* search
- It maintains 3 lists of CT nodes:
  - CLEANUP: regular list of A* sorted by the lower bound
  - OPEN: regular list of A* sorted by a *potentially inadmissible* function
  - FOCAL: nodes with cost bounded by w sorted by the distance-to-go
- The choice of the node to expand is similar to EES:
  - $cost\left(best_{h_c}\right) \leq w \cdot lb(best_{lb}) \;\rightarrow best_{h_c}$
  - $cost\left(best_{\hat{f}}\right) \leq w \cdot lb(best_{lb}) \;\rightarrow best_{\hat{f}}$
  - $best_{lb}$

Anytime Solvers

- We have seen that:
  - optimal algorithms do not scale well for the problem, but
  - sub-optimal algorithms may return a solution that is too inadequate
- Here come anytime solvers!
- The idea is:
  - return a sub-optimal solution in the shortest amount of time possible;
  - refine said solution in the remaining time available
- Many algorithms focus on intersections --> nodes with two or more neighbors
- We will see:
  - MAPF-LNS [19]
  - X* [21]

MAPF-LNS

- The algorithm uses Large-Neighborhood Search (LNS) [20]
  - The idea is to take a solution, remove an area, consider what remains as good and replan only on the sub-area which is a sub-problem of the initial
- It starts by finding an initial sub-optimal solution with either
  - EECBS, PP, or heuristics on PP
- The important aspect is how to extract a neighborhood
  - Agent-based
  - Map-based
  - Random-based

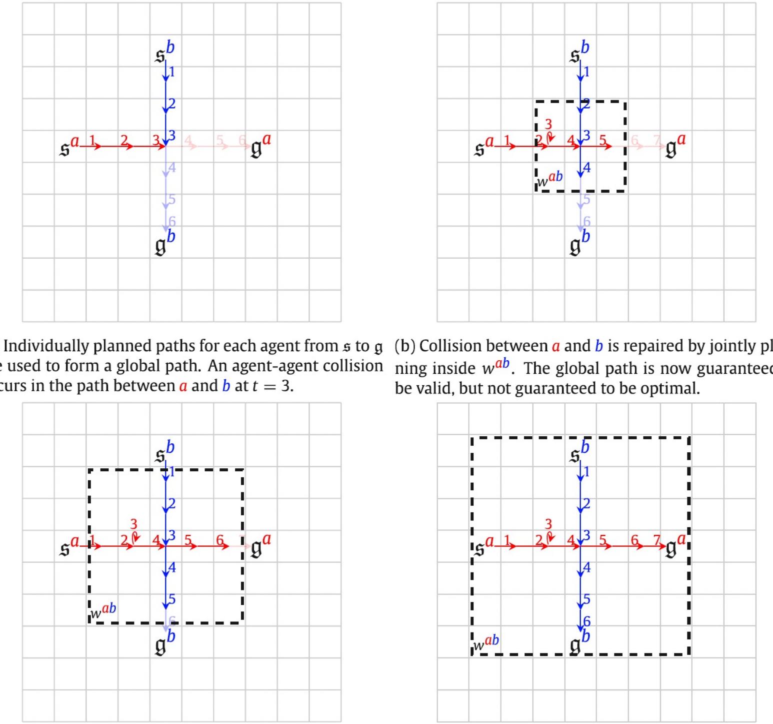
MAPF-LNS

- Agent-based neighborhood:
  1. Extract those agents that are not following the shortest path they could
  2. For each, compute a shorter random path
  3. Find agents that are colliding
  4. Replan groups of agents
- Map-based neighborhood:
  1. Identify the intersections --> higher probability of collision
  2. Identify agents moving through intersection, or in the area
  3. Change order in which agents pass through intersection
- Random neighborhood:
  1. Randomly choose N agents to replan for
  2. Replan

X*

- They introduce a concept called *window*
  - Identify agents and states around a conflict and repair the conflict
  - Each window has a successor, which basically is a larger window --> windows can grow
  - Two windows can also be merged together
- How does it work?
  1. Plan each agent individually
  2. Then starting by time t_0 it looks for conflicts
  3. For each conflict it creates a window and tries to solve it locally to the window
  4. The fixes are optimal within the window
- By enlarging windows and merging them, it can produce an optimal solution
- By preventing changes to following windows, it can produce a sub-optimal solution

X* – Example



(a) Individually planned paths for each agent from $\mathfrak{s}$ to $\mathfrak{g}$ are used to form a global path. An agent-agent collision occurs in the path between a and b at $t = 3$.

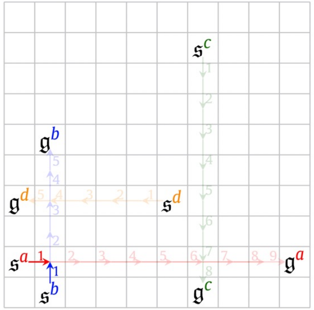
(b) Collision between a and b is repaired by jointly planning inside w^{ab}. The global path is now guaranteed to be valid, but not guaranteed to be optimal.

(c) w^{ab} is grown and a new repair is generated for a and b. The window does not yet encapsulate the search from $\mathfrak{s}^{ab}$ and $\mathfrak{g}^{ab}$, so the repaired global path is not yet guaranteed to be optimal.

(d) w^{ab} is grown and a new repair is generated. The repair search is from $\mathfrak{s}^{ab}$ to $\mathfrak{g}^{ab}$ and unimpeded by w^{ab}, thus allowing w^{ab} to be removed and the global path returned as optimal.

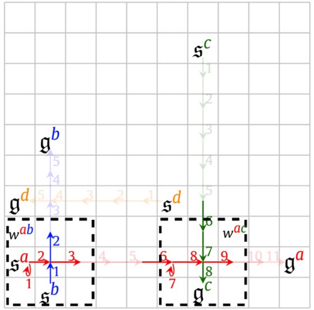
Images taken from (Veder and Biswas, 2021)

X*– Merging

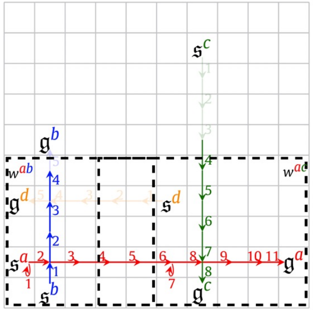


(a) Individually planned paths for each agent from $\mathfrak{s}$ to $\mathfrak{g}$ are used to form a global path. An agent-agent collision occurs between a and b at $t = 1$.

(b) Collision between a and b is repaired by jointly planning inside w^{ab}. The repair creates a collision between a and c at $t = 7$.

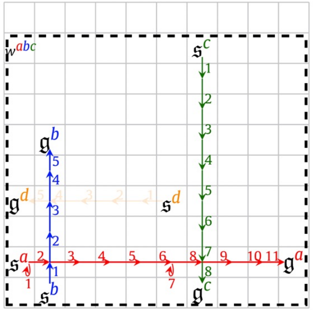


(c) Collision between a and c and is repaired by jointly planning inside w^{ac}. No collisions exist, thus producing a valid global path.



(d) All windows are grown in order to improve repair quality.

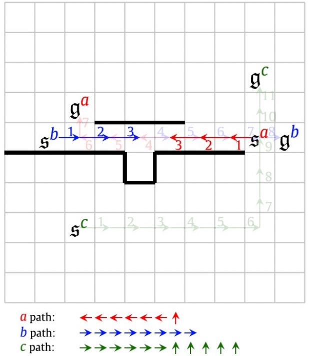
(e) As they overlap in agent set and states, w^{ab} and w^{ac} are merged to form w^{abc}, and a new repair is generated and inserted into the global path.



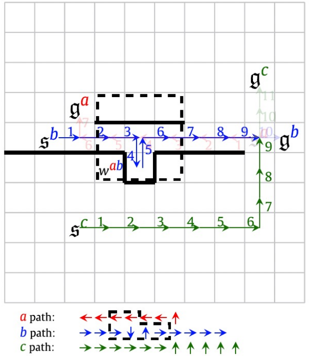
(f) w^{abc} is repeatedly grown and searched until the search of w^{abc} takes place from $\mathfrak{s}^{abc}$ to $\mathfrak{g}^{abc}$ unimpeded, thus allowing w^{abc} to be removed and the global path returned as optimal.

Images taken from (Veder and Biswas, 2021)

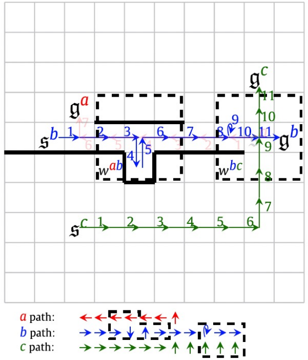
X* – Example

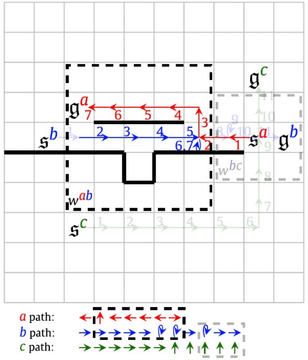


(a) Individually planned paths for each agent from $\mathfrak{s}$ to $\mathfrak{g}$ are used to form a global path. An agent-agent collision occurs in the path between a and b between $t = 3$ and $t = 4$. Walls are depicted by thick black lines.



(b) Collision between a and b is repaired by jointly planning inside w^{ab}. b now side steps into the slot to allow a to pass, but this repair causes a collision with c at $t = 9$. The region of the paths repaired by w^{ab} is surrounded by dashed lines.





Images taken from (Veder and Biswas, 2021)

MAPF – Variants

- Different types of conflicts
- Different behaviors of agents when reaching goal
- MAPF with agents of different sizes
- Lifelong MAPF
- MAPF with non discrete time
- Multi-Objective MAPF
- MAPF in combination with task planning

Recap

- Optimal:
  - Enhanced A* [2] (complete with correct heuristic)
- Complete and optimal:
  - CBS [6]
  - ICTS [7]
  - Constraint/logic programming [9]
- Fast:
  - PP [4]
- Suboptimal:
  - ϵ-ICTS and w-ICTS
  - CBS with improvements
  - ECBS
  - EECBS
- Anytime solvers:
  - MAPF-LNS
  - X*

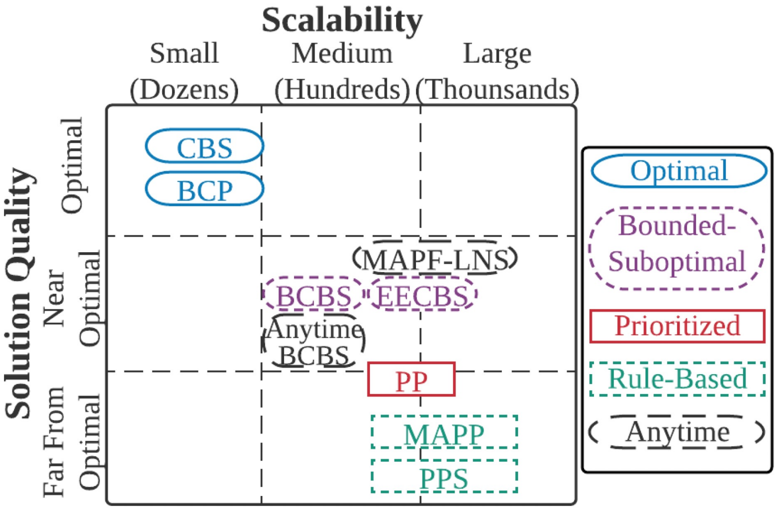


Image taken from [19]

Recap

- Optimal:
  - Enhanced A* [2] (complete with correct heuristic)
- Complete and optimal:
  - CBS [6]
  - ICTS [7]
  - Constraint/logic programming [9]
- Fast:
  - PP [4]
- Suboptimal:
  - ϵ-ICTS and w-ICTS
  - CBS with improvements
  - ECBS
  - EECBS
- Anytime solvers:
  - MAPF-LNS
  - X*

[MAPF Bible]

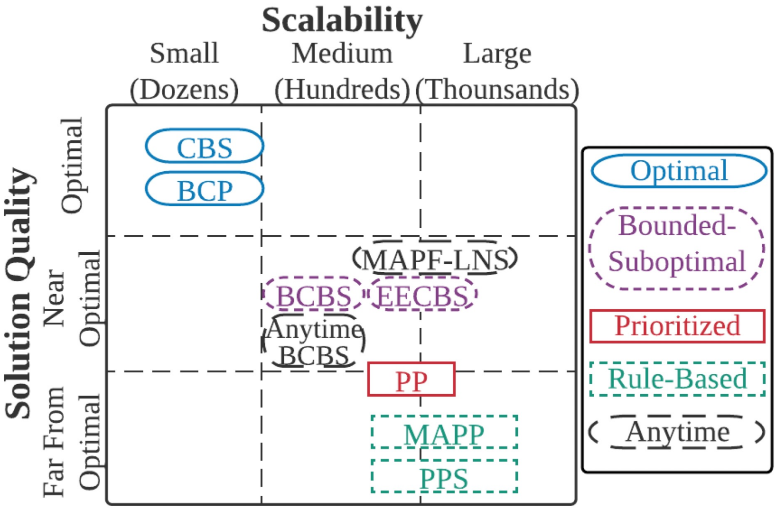


Image taken from [19]

References

[1] Stern, Roni, et al. "Multi-agent pathfinding: Definitions, variants, and benchmarks." *Proceedings of the International Symposium on Combinatorial Search*. Vol. 10. No. 1. 2019.

[2] https://www.leagueofrobotrunners.org/

[3] Standley, T. (2010). Finding Optimal Solutions to Cooperative Pathfinding Problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, *24*(1), 173-178. https://doi.org/10.1609/aaai.v24i1.7564

[4] Silver, D. (2021). Cooperative Pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, *1*(1), 117-122. https://doi.org/10.1609/aiide.v1i1.18726

[5] Ma, H., Harabor, D., Stuckey, P. J., Li, J., & Koenig, S. (2019). Searching with Consistent Prioritization for Multi-Agent Path Finding. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 7643-7650. https://doi.org/10.1609/aaai.v33i01.33017643

[6] Sharon, Guni, et al. "Conflict-based search for optimal multi-agent pathfinding." *Artificial Intelligence* 219 (2015): 40-66.

[7] Sharon, Guni, et al. "The increasing cost tree search for optimal multi-agent pathfinding." *Artificial intelligence* 195 (2013): 470-495.

[8] A. Srinivasan, T. Ham, S. Malik and R. K. Brayton, "Algorithms for discrete function manipulation," *1990 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*, Santa Clara, CA, USA, 1990, pp. 92-95, doi: 10.1109/ICCAD.1990.129849.

[9] R. Barták, N. -F. Zhou, R. Stern, E. Boyarski and P. Surynek, "Modeling and Solving the Multi-agent Pathfinding Problem in Picat," *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, Boston, MA, USA, 2017, pp. 959-966, doi: 10.1109/ICTAI.2017.00147.

[10] Yu, J., & LaValle, S. (2013). Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, *27*(1), 1443-1449. https://doi.org/10.1609/aaai.v27i1.8541

[11] Walker, Thayne T., Nathan R. Sturtevant, and Ariel Felner. "Extended Increasing Cost Tree Search for Non-Unit Cost Domains." *IJCAI*. 2018.

References

[12] Boyrasky, Eli, et al. "Don't split, try to work it out: Bypassing conflicts in multi-agent pathfinding." *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 25. 2015.

[13] Boyarski, Eli, et al. "Icbs: The improved conflict-based search algorithm for multi-agent pathfinding." *Proceedings of the International Symposium on Combinatorial Search*. Vol. 6. No. 1. 2015.

[14] Li, Jiaoyang, et al. "New techniques for pairwise symmetry breaking in multi-agent path finding." *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 30. 2020.

[15] Barer, Max, et al. "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem." *Proceedings of the International Symposium on Combinatorial Search*. Vol. 5. No. 1. 2014.

[16] Li, J., Ruml, W., & Koenig, S. (2021). EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(14), 12353-12362. https://doi.org/10.1609/aaai.v35i14.17466

[17] Pearl, Judea, and Jin H. Kim. "Studies in semi-admissible heuristics." *IEEE transactions on pattern analysis and machine intelligence* 4 (1982): 392-399.

[18] Thayer, Jordan Tyler, and Wheeler Ruml. "Bounded suboptimal search: A direct approach using inadmissible estimates." *IJCAI*. Vol. 2011. 2011.

[19] Li, Jiaoyang, et al. "Anytime multi-agent path finding via large neighborhood search." *International Joint Conference on Artificial Intelligence 2021*. Association for the Advancement of Artificial Intelligence (AAAI), 2021.

[20] Shaw, Paul. "Using constraint programming and local search methods to solve vehicle routing problems." *International conference on principles and practice of constraint programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998

[21] Vedder, Kyle, and Joydeep Biswas. "X*: Anytime multi-agent path finding for sparse domains using window-based iterative repairs." *Artificial Intelligence* 291 (2021): 103417.