# Motion Planning of Swarm Robotics

■ Lecture 7

主讲人 Xin Zhou

D. Eng. Candidate in Robotics
Zhejiang University

# Contents



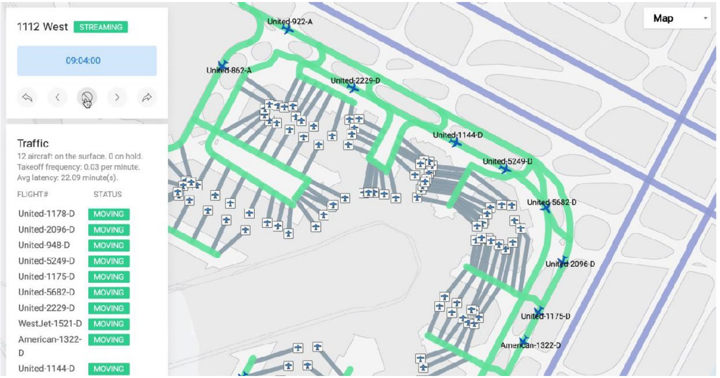
# Contents



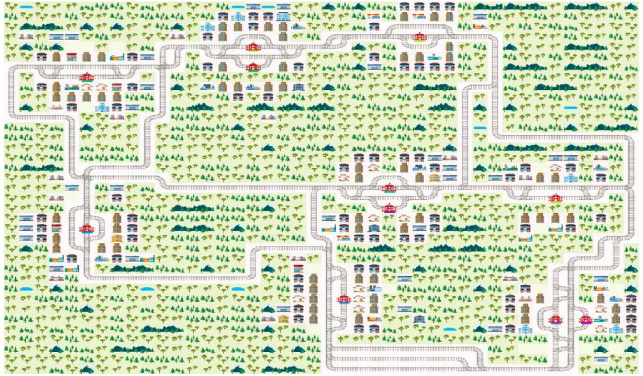
# Background



Warehouse Robot Navigation




Airport Surface Operation



Train Scheduling

Image Credit: Jiaoyang Li in Computer Science at University of Southern California.

# Problem Definition



## Given

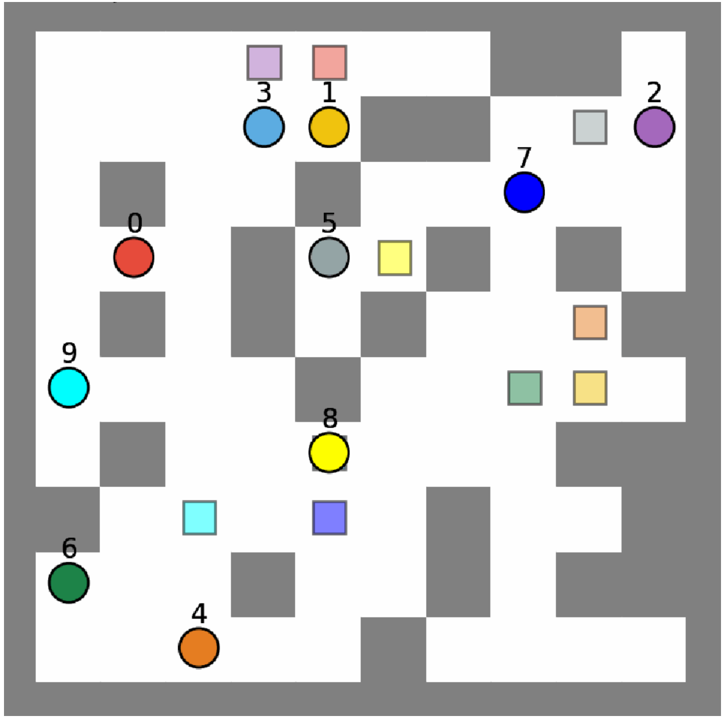
- A graph $G = (V, E)$;
- A set of $k$ agents $a_1, a_2, \cdots, a_k$, each with a start location $s_i \in V$ and a target location $g_i \in V$.

## Goal

- Finds collision-free paths for all agents, and
- Minimizes the sum of their travel times.

## Challenges

- How to solve MAPF **efficiently** and **effectively**?



# Problem Definition



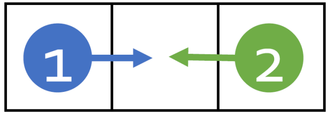
**Actions**

- *Move*: move to a neighboring location.
- *Wait*: wait at its current location.

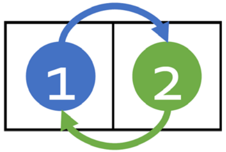
Both actions take one timestep and have one-unit cost.

**Conflicts(= Collisions)**

- *Vertex conflict*: two agents stay at the same location at the same timestep.




- *Edge conflict*: two agents traverse the same edge in opposite directions at the same timestep.

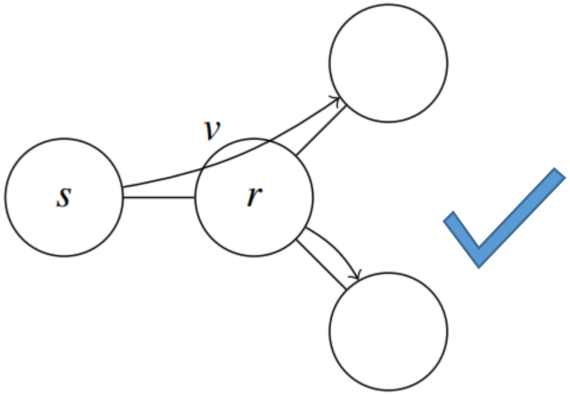


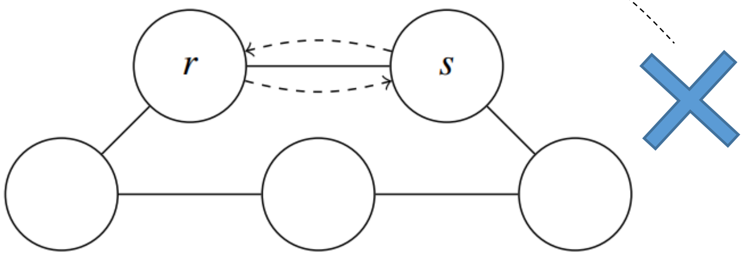
# Rule-based suboptimal solvers



## PUSH AND SWAP[1]

- Agent $s$ with higher priority pushes agent $r$ from a degree-three vertex $v$.
- Non-complete, non-optimal.




[1] Luna, Ryan J., and Kostas E. Bekris. "Push and swap: Fast cooperative path-finding with completeness guarantees." *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

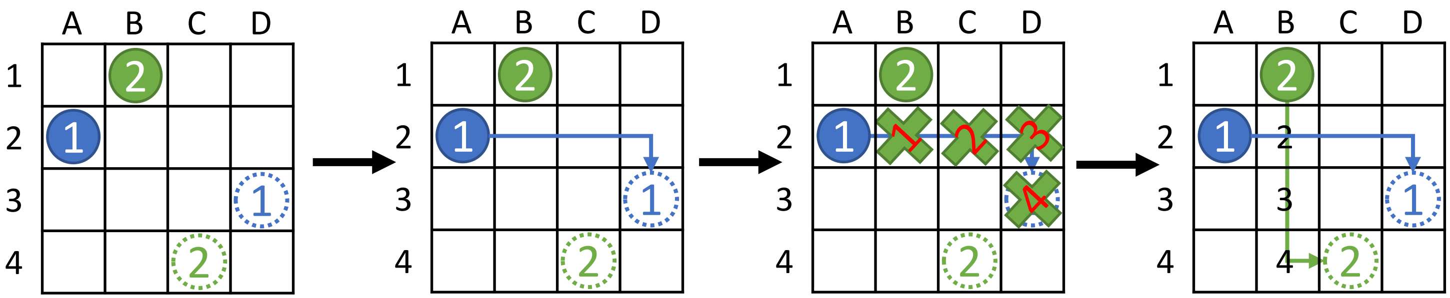
# Search-based suboptimal solvers

## Hierarchical Cooperative A*(HCA*)[1]

**The main idea:** agents plan one by one according to some predefined order.

Former agents mark the vertexes they stayed as occupied at that time point.

- Easy to deploy;
- Non-complete, Non-optimal;

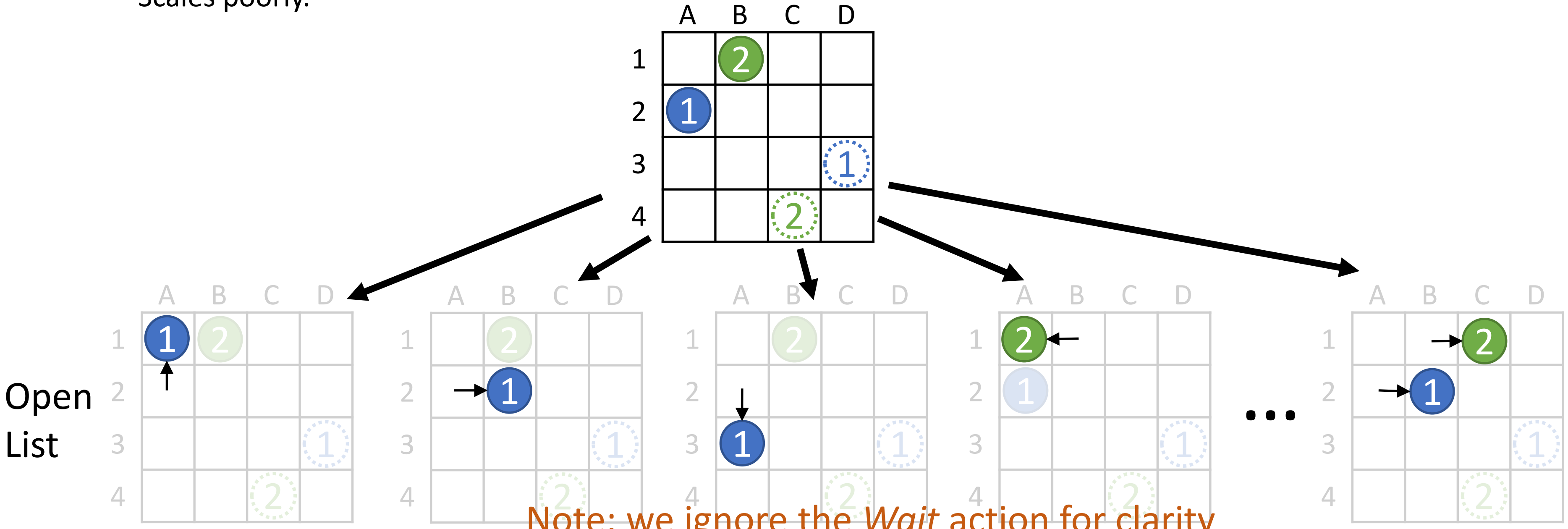



[1] Silver, David. "Cooperative pathfinding." *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*. Vol. 1. No. 1. 2005.

# Single-Agent A*



- Treat the individual agents as a single 'joint agent'.
- The state-space includes all permutations of placing $k$ agents in $V$ locations.
- Node number of the new graph: $\binom{V}{k} = V \times (V-1) \times (V-2) \times \cdots (V-k+1)$, complexity: $O(V^k)$.
- 5 Agents in a 10*10 grid: New node number $\approx 1e10$.
- Scales poorly.

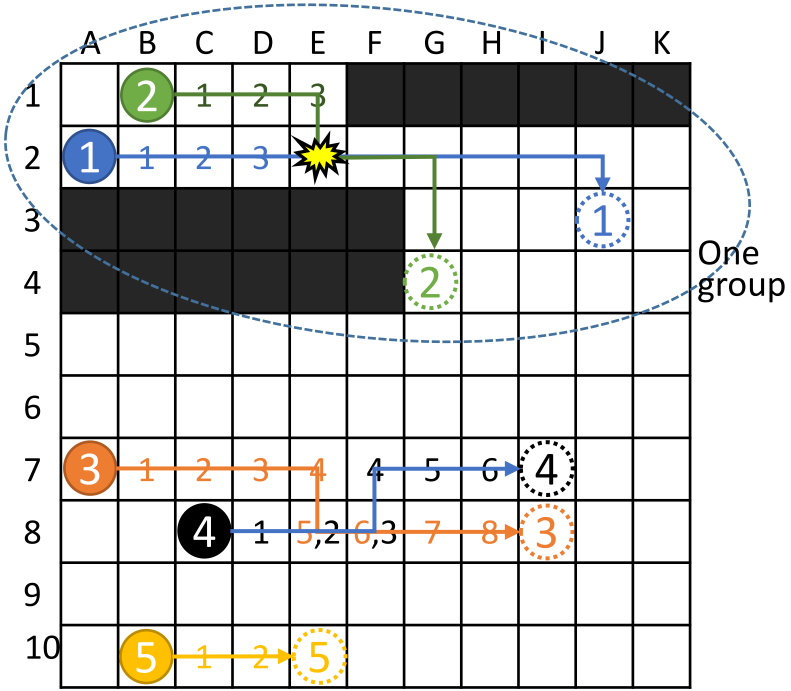



Note: we ignore the *Wait* action for clarity

# Independence Detection (ID)



ID attempts to detect independent groups of agents where there is an optimal solution for each group such that the two solutions do not conflict.




**Algorithm 2** Independence Detection

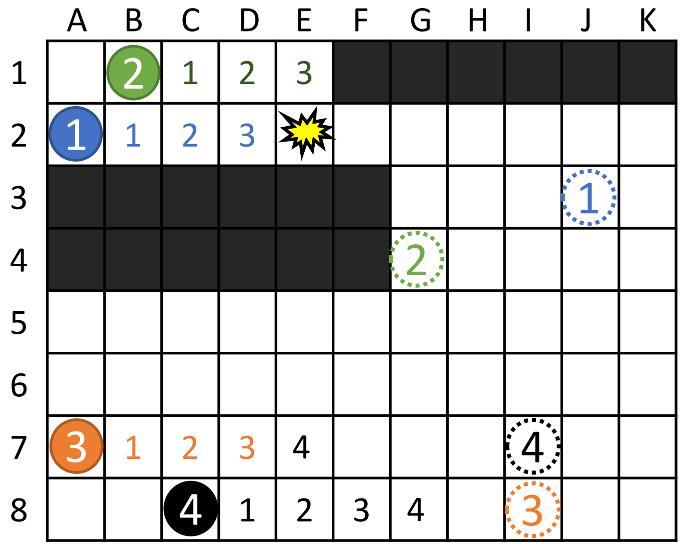
assign each agent to a group
plan a path for each group
fill conflict avoidance table with every path
**repeat**
  simulate execution of all paths until a conflict between two groups $G_1$ and $G_2$ occurs
  **if** these two groups have not conflicted before **then**
    fill illegal move table with the current paths for $G_2$
    find another set of paths with the same cost for $G_1$
    **if** failed to find such a set **then**
      fill illegal move table with the current paths for $G_1$
      find another set of paths with the same cost for $G_2$
    **end if**
  **end if**
  **if** failed to find an alternate set of paths for $G_1$ and $G_2$ **then**
    merge $G_1$ and $G_2$ into a single group
    cooperatively plan new group
  **end if**
  update conflict avoidance table with changes made to paths
**until** no conflicts occur
$solution \leftarrow$ paths of all groups combined
**return** $solution$

[1] Standley, Trevor. "Finding optimal solutions to cooperative pathfinding problems." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 24. No. 1. 2010.

# M*: An enhanced single-agent A* [1]

**The main idea**: expending children in all directions for conflicting agents only, while expending only one child for non-conflicting agents.

- In general, nodes expand only one child where each agent makes its optimal move.
- When conflicts occur between $q$ agents at node $n$, M* traces back from $n$ through all the ancestors until the root node and all these nodes are placed back in *OPEN* set.
- Then $q$ conflicting agents will expend all their branches.
- Similar to ID.




[1] Wagner, Glenn, and Howie Choset. "M*: A complete multirobot path planning algorithm with performance bounds." *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2011.

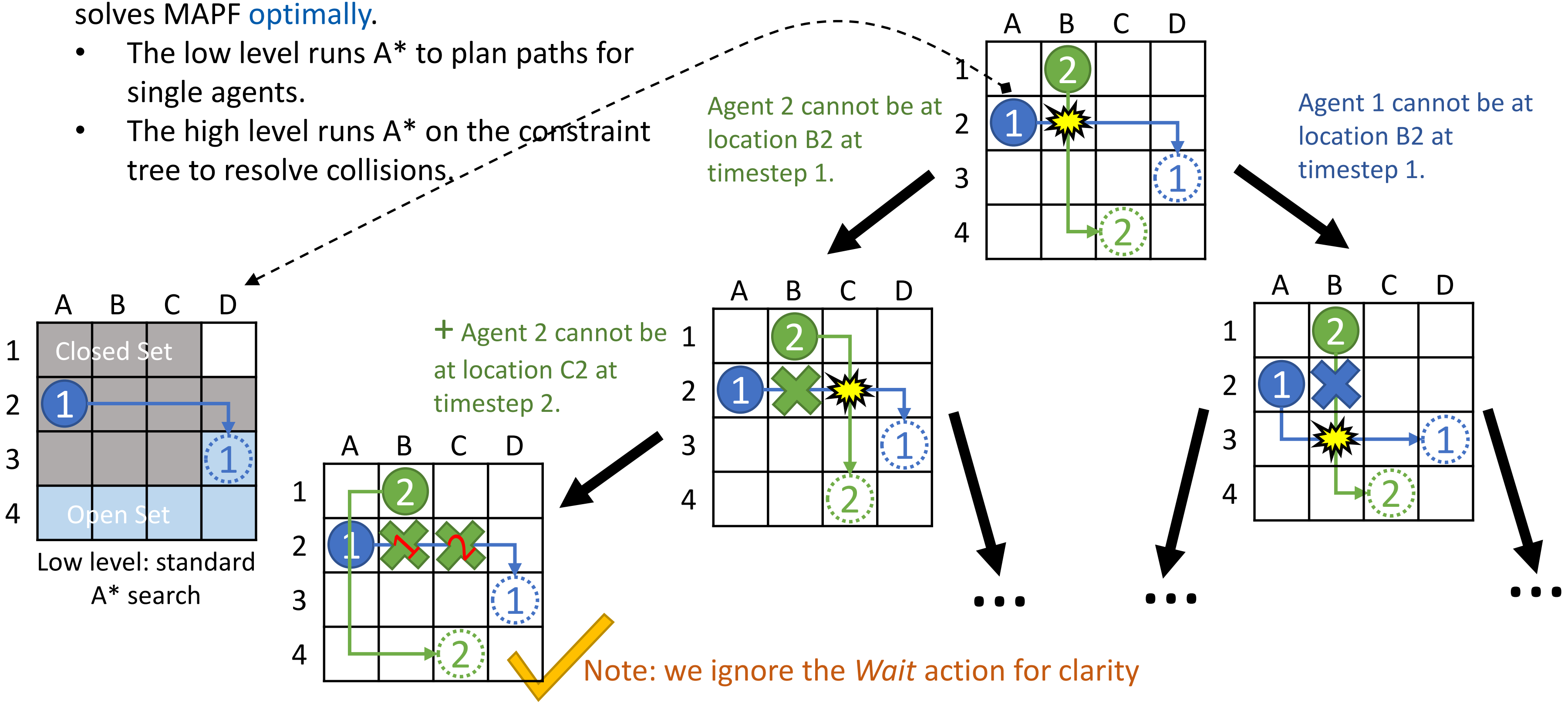
# Conflict-Based Search (CBS)



CBS[1] is a bi-level search algorithm that solves MAPF optimally.

- The low level runs A* to plan paths for single agents.
- The high level runs A* on the constraint tree to resolve collisions.

High-level constraint tree (CT)




Low level: standard A* search

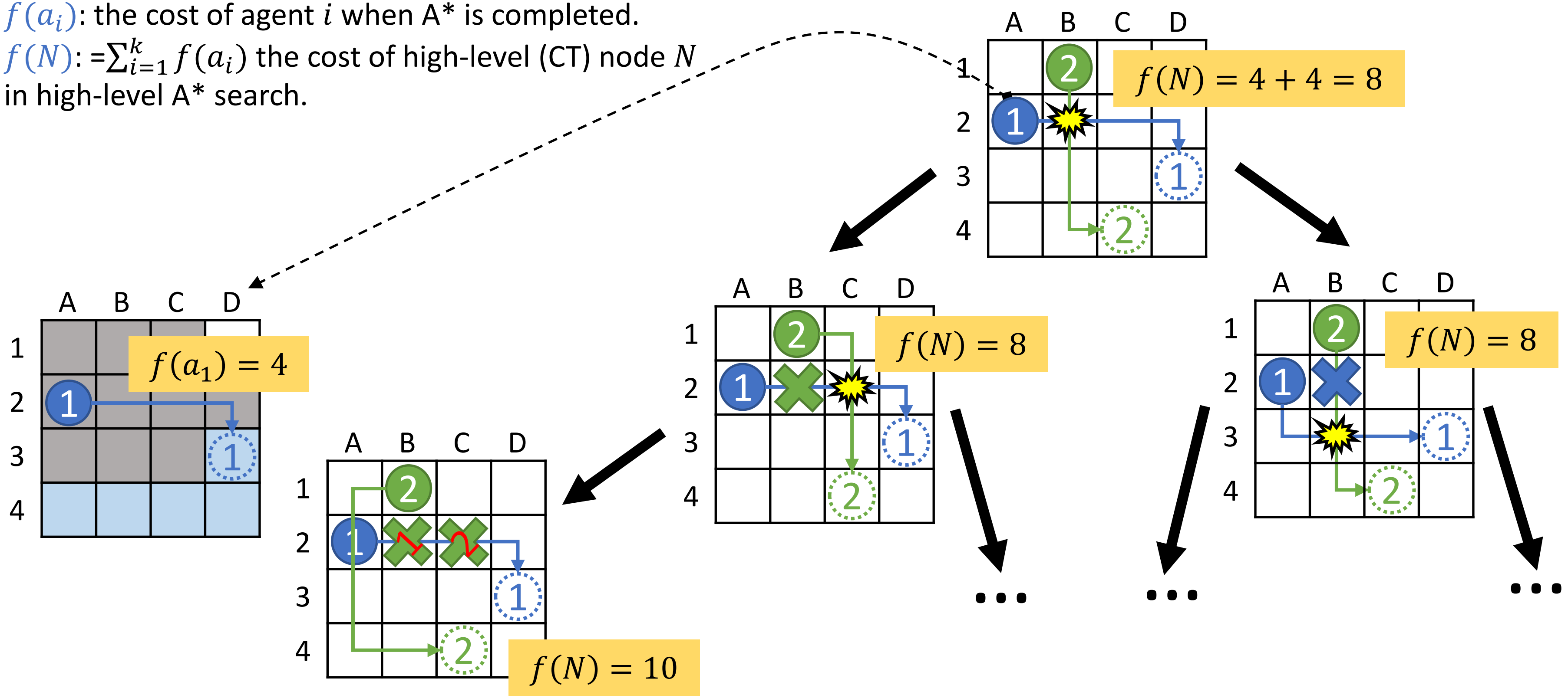
Note: we ignore the *Wait* action for clarity

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

# Conflict-Based Search (CBS)



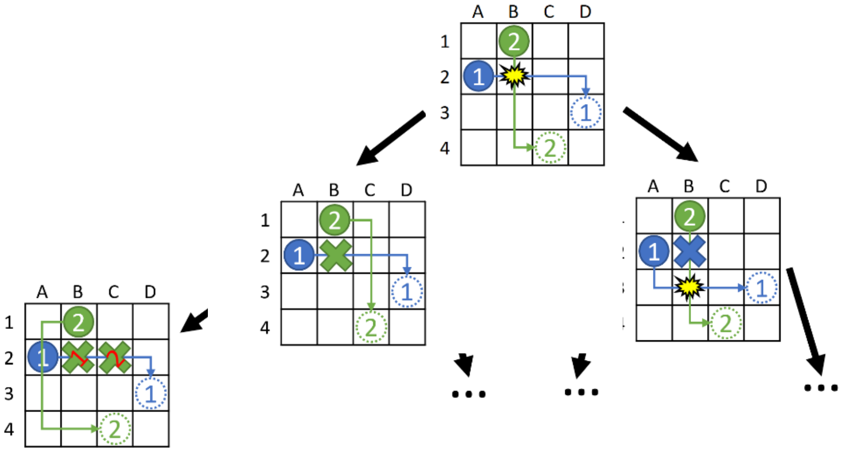
$f(n)$: the cost of a low-level node in standard A* search.
$f(a_i)$: the cost of agent $i$ when A* is completed.
$f(N)$: $=\sum_{i=1}^{k} f(a_i)$ the cost of high-level (CT) node $N$ in high-level A* search.




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

# Conflict-Based Search (CBS)

$N$: A CT (high-level) node;
$N.constraints$: A set of constraints imposed on each agent;
$N.solution$: A single consistent solution, solution, i.e., one path for each agent that is consistent with $N.constraints$.
$N.cost$: The cost of the current solution. Here, it is the sum of all agent costs.
$R$: The root node;
$SIC(\cdot)$: the sum of all single-agent cost.
$OPEN$: The open list of CT.
$(a_i, a_j, v, t)$: A tuple describing a conflict that agent $a_i$ and $a_j$ stay in vertex $v$ together at timestep $t$.




**Algorithm 1:** high-level of CBS

```
Input: MAPF instance
R.constraints = ∅
R.solution = find individual paths using the low-level()
R.cost = SIC(R.solution)
insert R to OPEN
while OPEN not empty do
    P ← best node from OPEN // lowest solution cost
    Validate the paths in P until a conflict occurs.
    if P has no conflict then
        return P.solution // P is goal
    C ← first conflict (a_i, a_j, v, t) in P
    foreach agent a_i in C do
        A ← new node
        A.constraints ← P.constraints + (a_i, v, t)
        A.solution ← P.solution.
        Update A.solution by invoking low-level(a_i)
        A.cost = SIC(A.solution)
        Insert A to OPEN
```

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

# Conflict-Based Search (CBS)



## Optimal Solvers

Single-Agent A*
M*
CBS

## Bounded Suboptimal Solvers

BCBS
ECBS

## Suboptimal Solvers

PUSH AND SWAP
HCA*

Bounded suboptimal solvers return a solution $f$ that guarantees $f \leq \omega \cdot C^*$, where $C^*$ is the cost of the unknown optimal solution. $\omega$ is user-defined.

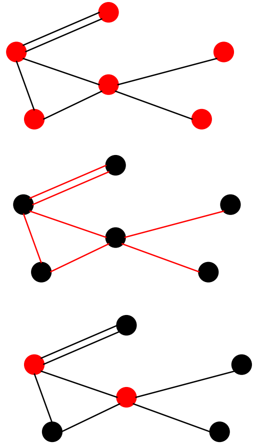
[1] Sharon, Guni, et al. "Conflict-based search for optimal multi-agent pathfinding." Artificial Intelligence 219 (2015): 40-66.

# Enhanced CBS (ECBS)



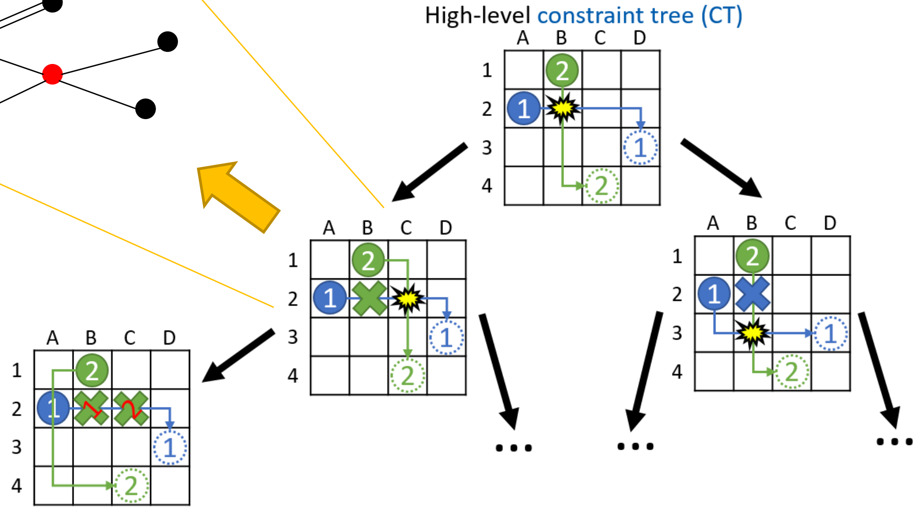
**Heuristics $h_c$**

- Number of conflicting agents
- Number of conflicting pairs
- Vertex cover
- Alternating heuristic [1]



Vertex: a conflicting agent

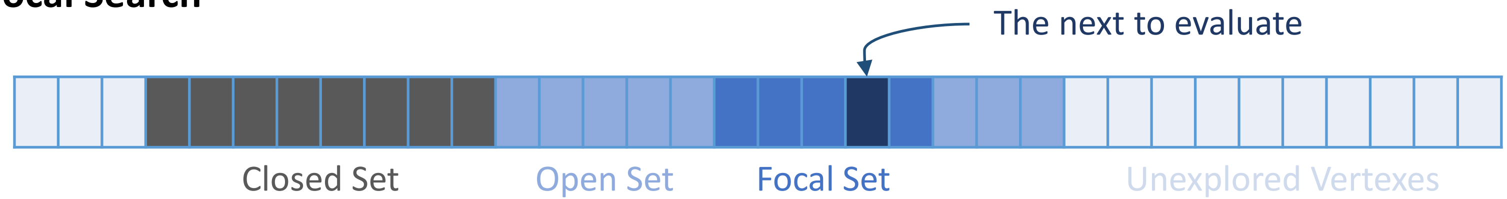
Edge: a conflict




[1] Röger, Gabriele, and Malte Helmert. "The more, the merrier: Combining heuristic estimators for satisficing planning." *Twentieth International Conference on Automated Planning and Scheduling*. 2010.

# Enhanced CBS (ECBS)

## Focal Search




- We select a promising subset in *OPEN* as *FOCAL* according to a policy (function) $f_1$.
- We select the best node in *FOCAL* to evaluate according to another policy (function) $f_2$.

### $f_1$ of the Low Level

$n$: Nodes; $OPEN$: The open set of the low level;

$f(n) = g(n) + h(n)$: The general A* cost.

$f_{\min} = f(n) | n = \min_{n \in OPEN} f(n)$

$FOCAL = \{n | n \in OPEN, f(n) \leq \omega \cdot f_{\min}\}$,

### $f_2$ of the Low Level

$f_2$: $n_{\text{next}} = \min_{n \in \text{FOCAL}} h_c(n)$;

$h_c(n)$: Number of conflicting pairs (see previous slide)

[1] Röger, Gabriele, and Malte Helmert. "The more, the merrier: Combining heuristic estimators for satisficing planning." *Twentieth International Conference on Automated Planning and Scheduling*. 2010.

# Enhanced CBS (ECBS)



## Focal Search for the High Level

### $f_1$ of the High Level

$OPEN$: the open set of the high level; $OPEN_i$: the open set of the agent $a_i$ in the low level;

$f_{\min}(i)=f(n)|n = \min_{n \in OPEN_i} f(n)$; $n$: low-level node; $N$: high-level node;

$LB(N) = \sum_{i=1}^{k} f_{\min}(i)$; $\boldsymbol{LB} = \min(LB(N)|N \in OPEN)$;

$N.cost$: sum of all agents' solution costs in CT node $N$ (same to CBS);

$FOCAL = \{N|N \in OPEN, N.cost \leq \omega \cdot \boldsymbol{LB}\}$.

### $f_2$ of the High Level

$f_2$: $N_{\text{next}} = \min_{N \in FOCAL} h_c(N)$;

$h_c(N)$: number of conflicting pairs.

[1] Röger, Gabriele, and Malte Helmert. "The more, the merrier: Combining heuristic estimators for satisficing planning." *Twentieth International Conference on Automated Planning and Scheduling*. 2010. **The proof is in page 24.**

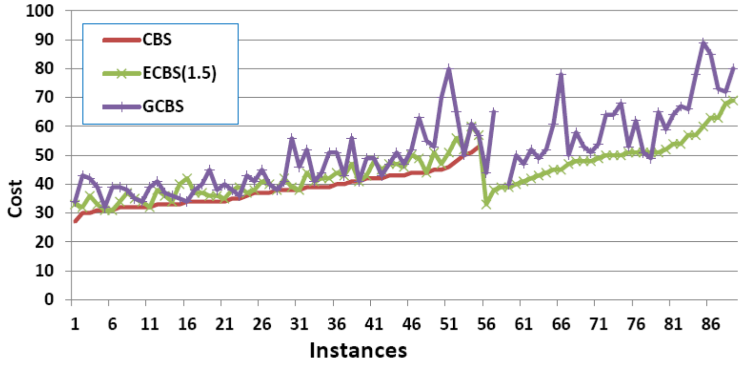
# Enhanced CBS (ECBS)



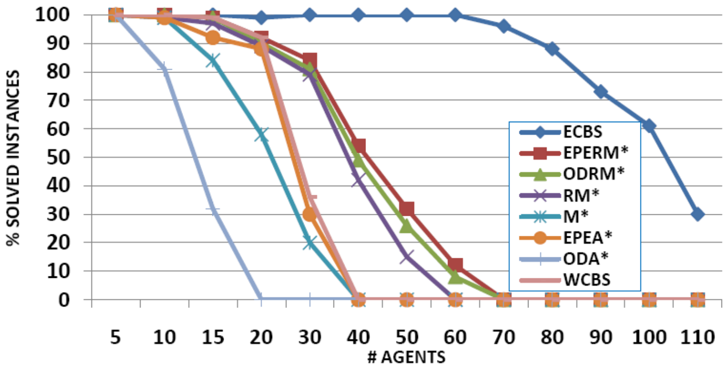
## The main idea

**Bounded optimality:** If we keep selecting the lowest-cost node in the *OPEN* set to evaluate, the solution is optimal (original A*). Now we relief this by selecting from the *FOCAL* set which includes nodes with costs no larger than ($\omega \cdot$ lowest-cost).

**Quicker search:** we choose nodes that are unlikely to have conflict with others ($h_c$) to reduce the number of CT nodes generated.




(a) Cost comparison, 8 agents, $w = 1.5$




(b) Bounded algorithms, $w = 1.1$

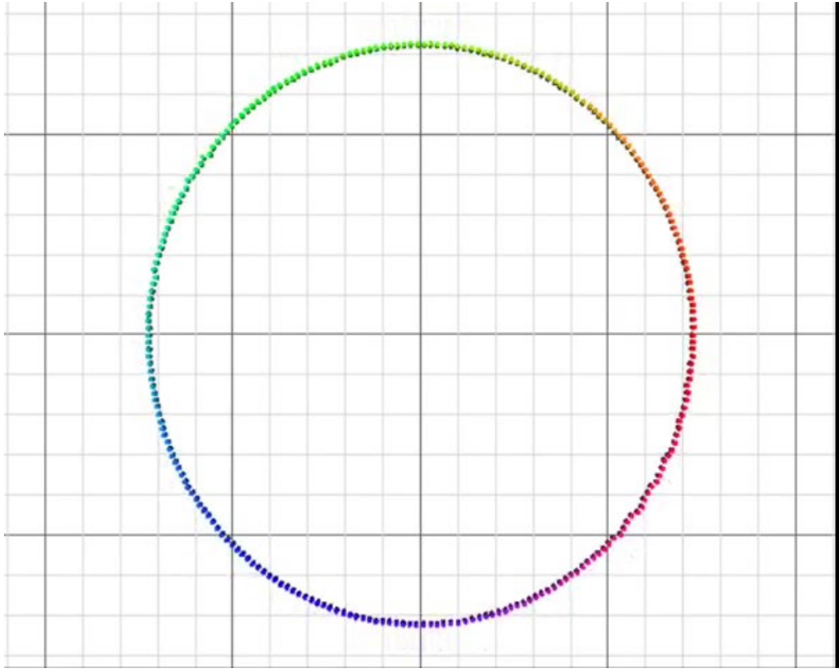
[1] Röger, Gabriele, and Malte Helmert. "The more, the merrier: Combining heuristic estimators for satisficing planning." *Twentieth International Conference on Automated Planning and Scheduling*. 2010. **The proof is in page 24.**

# Contents



# Background






Berg, Jur van den, et al. "Reciprocal n-body collision avoidance." Robotics research. Springer, Berlin, Heidelberg, 2011. 3-19.

# Evolution



VO[1] (Velocity Obstacle)

↓

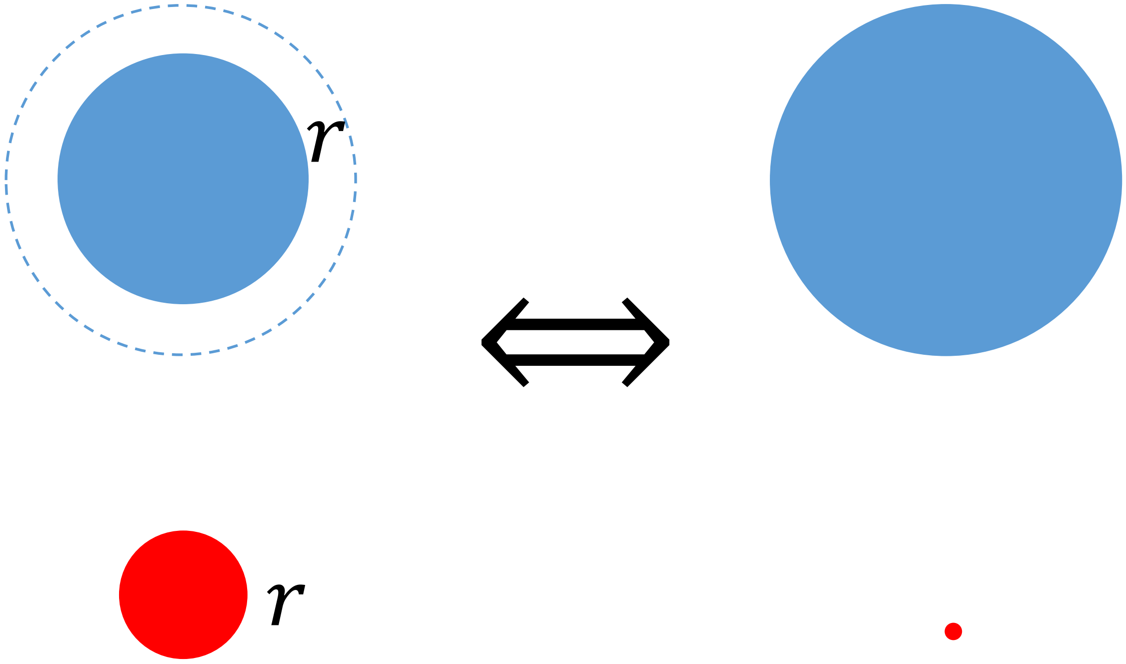
RVO[2] (Reciprocal Velocity Obstacle)

↓

ORCA[3] (Optimal Reciprocal Collision Avoidance)

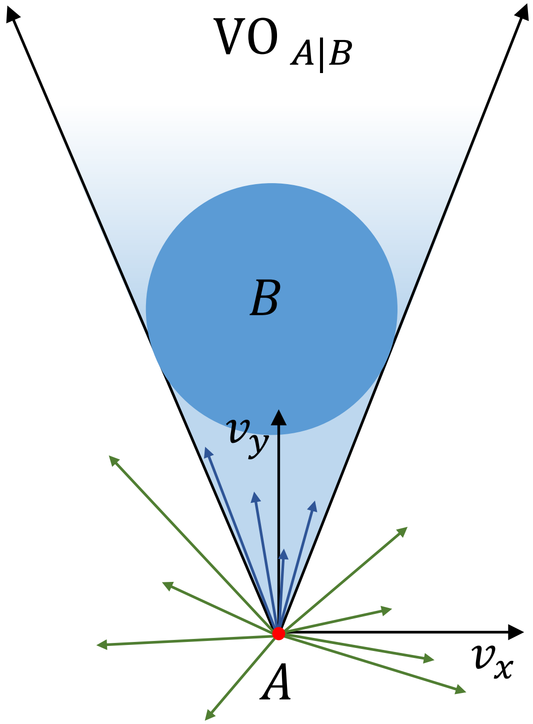
[1] Fiorini P, Shiller Z. Motion planning in dynamic environments using velocity obstacles[J]. The International Journal of Robotics Research, 1998, 17(7): 760-772.
[2] Van den Berg J, Lin M, Manocha D. Reciprocal velocity obstacles for real-time multi-agent navigation[C]. 2008 IEEE International Conference on Robotics and Automation. IEEE, 2008: 1928-1935.
[3] Van Den Berg J, Guy S J, Lin M, et al. Reciprocal n-body collision avoidance[M]. Robotics research. Springer, Berlin, Heidelberg, 2011: 3-19.

# Particle Model



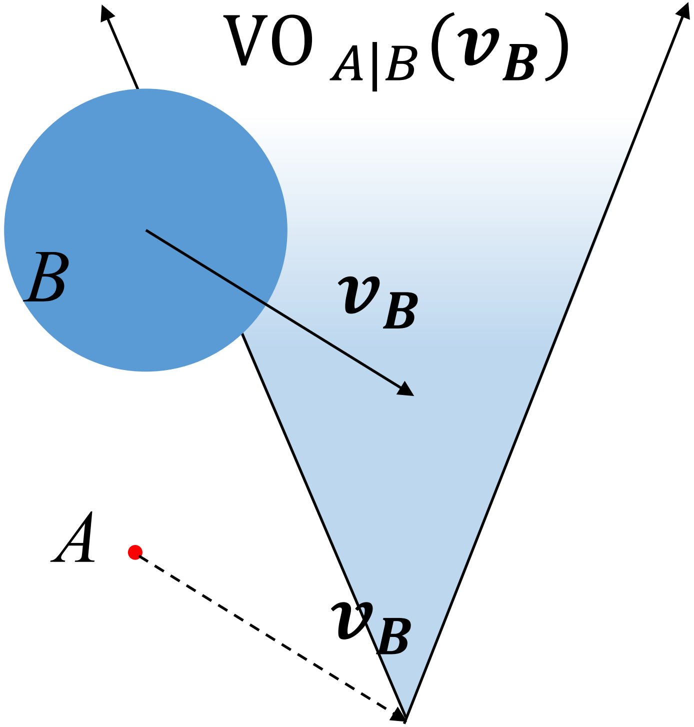
# Velocity Obstacle (VO)

## Static Case



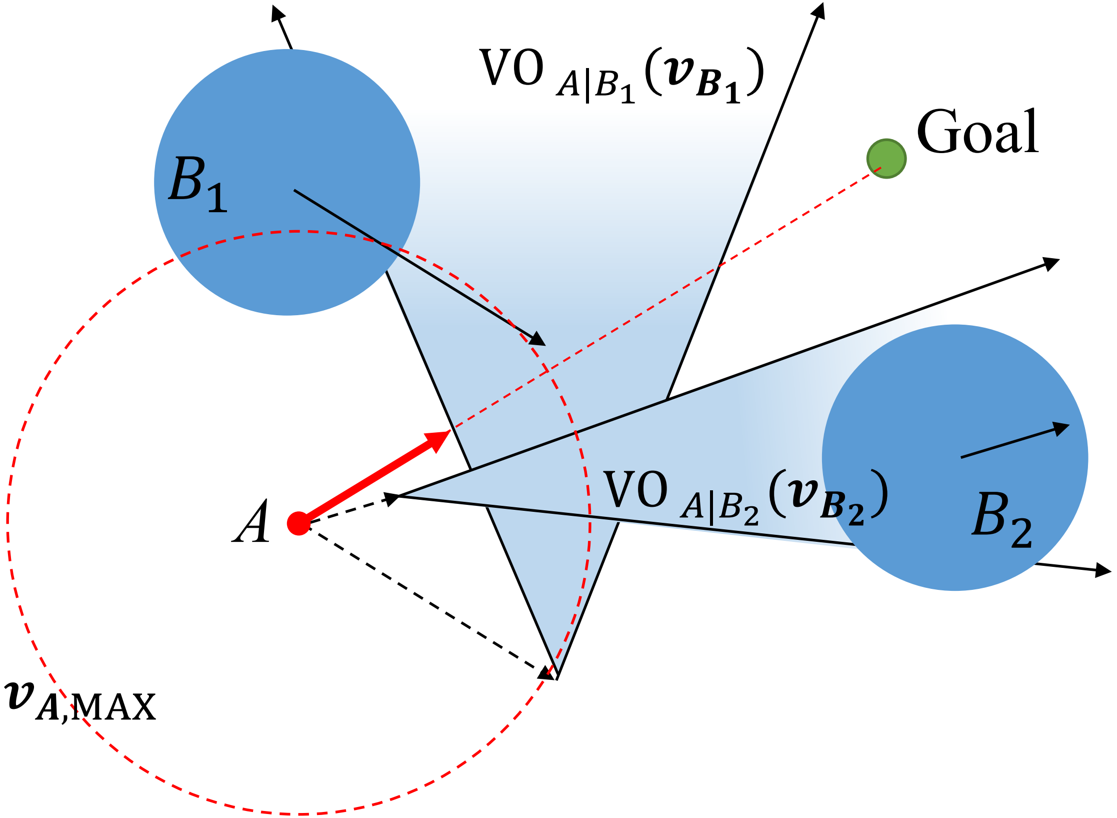
# Velocity Obstacle (VO)

**Dynamic Case**



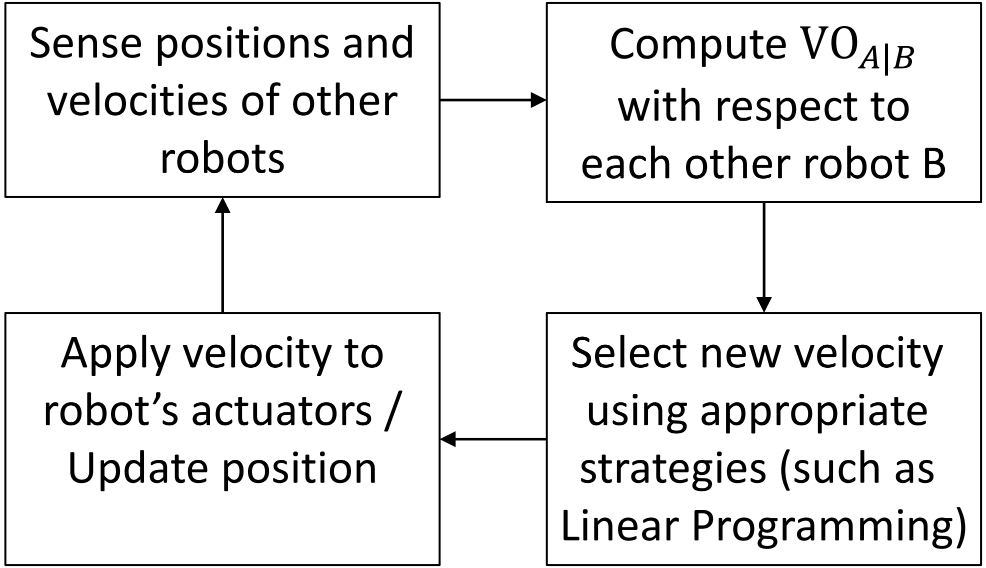
# Velocity Obstacle (VO)

## Multiple Obstacle Case



# Velocity Obstacle (VO)

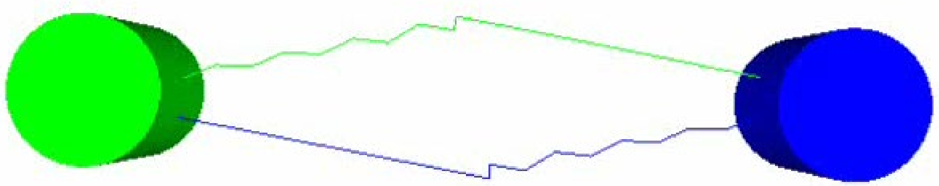
## Control Loop



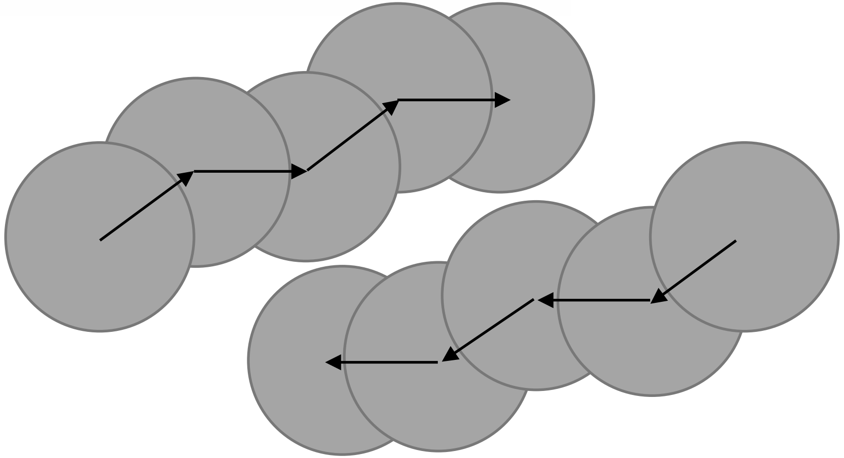
# Reciprocal Velocity Obstacle (RVO)



## Oscillation of VO



The key problem: agents don’t consider others’ velocities in the next circle.

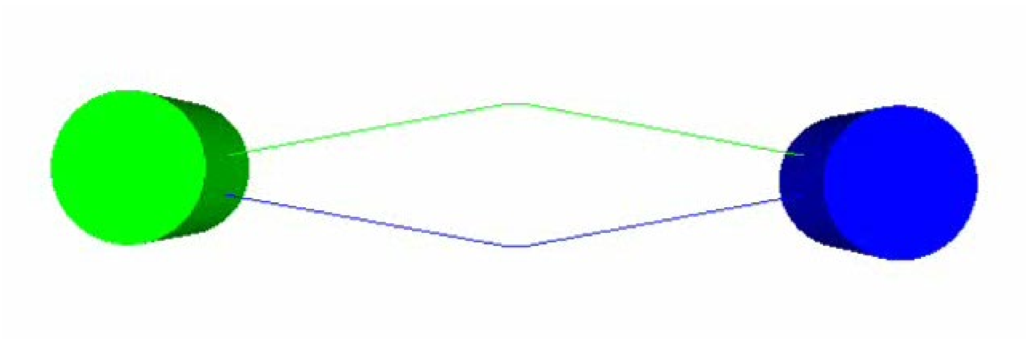


Van den Berg J, Lin M, Manocha D. Reciprocal velocity obstacles for real-time multi-agent navigation[C]. 2008 IEEE International Conference on Robotics and Automation. IEEE, 2008: 1928-1935.

# Reciprocal Velocity Obstacle (RVO)



## Oscillation Avoidance

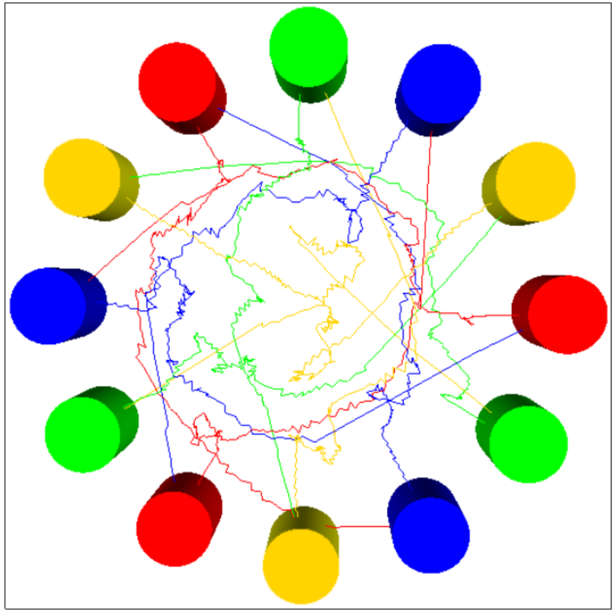


Van den Berg J, Lin M, Manocha D. Reciprocal velocity obstacles for real-time multi-agent navigation[C]. 2008 IEEE International Conference on Robotics and Automation. IEEE, 2008: 1928-1935.

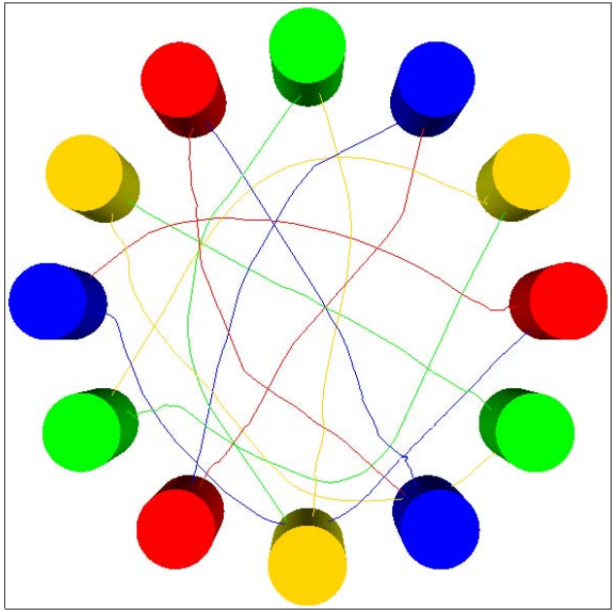
# Reciprocal Velocity Obstacle (RVO)



## Comparison



VO



RVO

Van den Berg J, Lin M, Manocha D. Reciprocal velocity obstacles for real-time multi-agent navigation[C]. 2008 IEEE International Conference on Robotics and Automation. IEEE, 2008: 1928-1935.

# Reciprocal Velocity Obstacle (RVO)

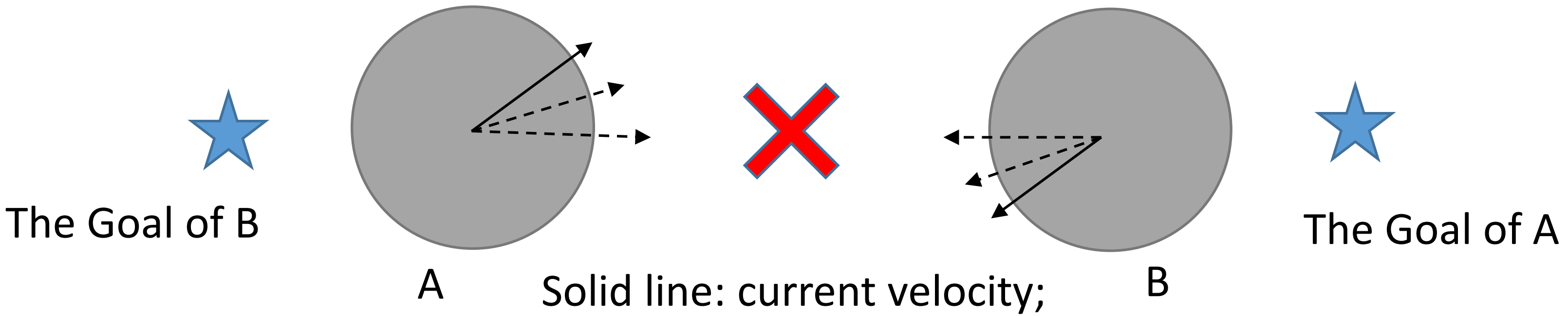


Main idea:

<u>Each agent takes limited responsibility of velocity change.</u>

We choose a new velocity $v'_A$ that is the average of its current velocity $v_A$ and a velocity $v$ that lies outside the velocity obstacle.

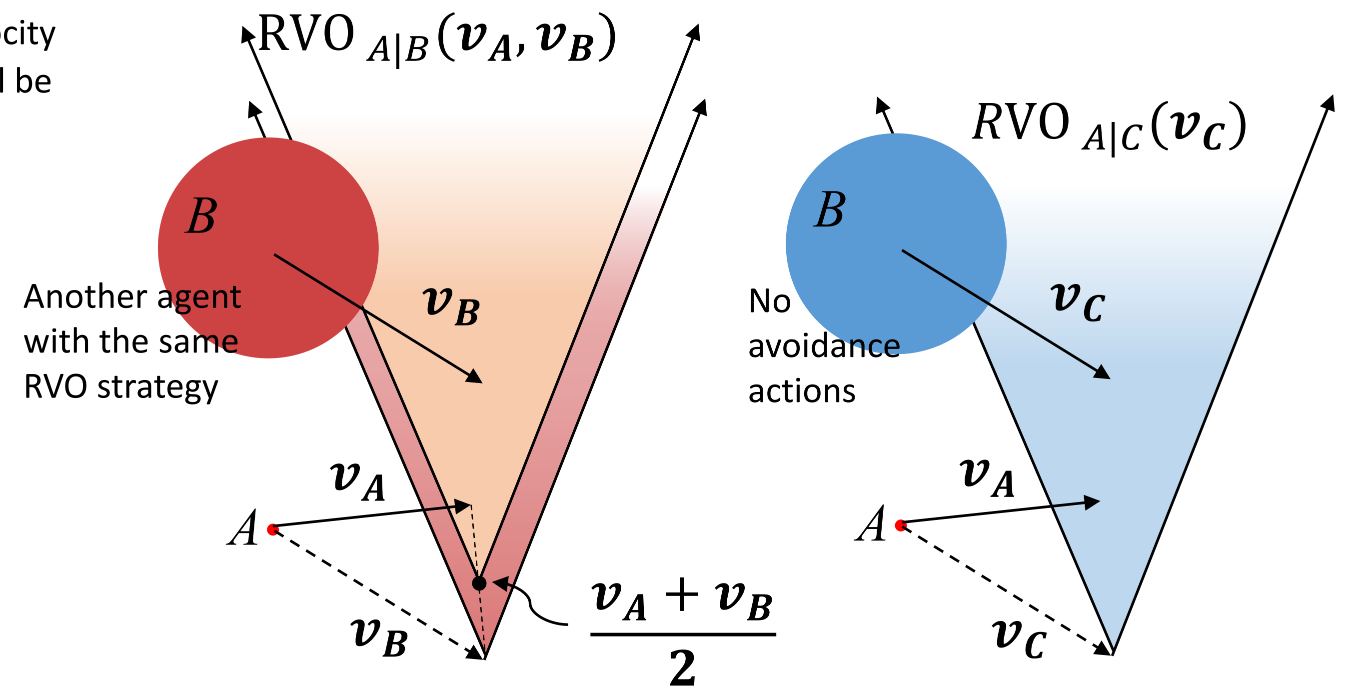
RVO is a rule determining how much responsibility each agent should take.




Solid line: current velocity;
Dotted line: planned velocity

# Reciprocal Velocity Obstacle (RVO)

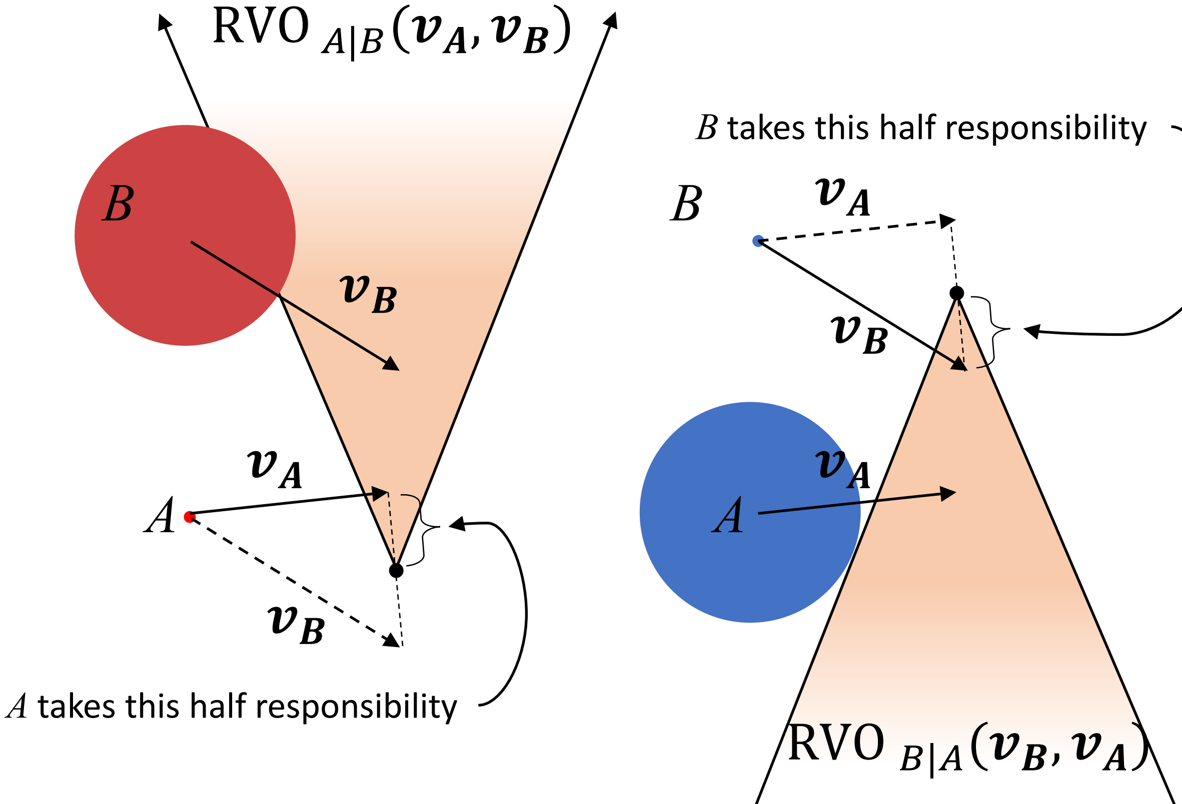
$\mathrm{RVO}_{A|B}(\boldsymbol{v_A}, \boldsymbol{v_B})$ : the velocity space of A that A of $\boldsymbol{v_A}$ will be in collision with B of $\boldsymbol{v_B}$.



# Reciprocal Velocity Obstacle (RVO)



The criteria of collision avoidance is $(\boldsymbol{v_A}-\boldsymbol{v_B})^T(\boldsymbol{p_A}-\boldsymbol{p_B}) \geq 0$, i.e., their relative velocity keeps them away.

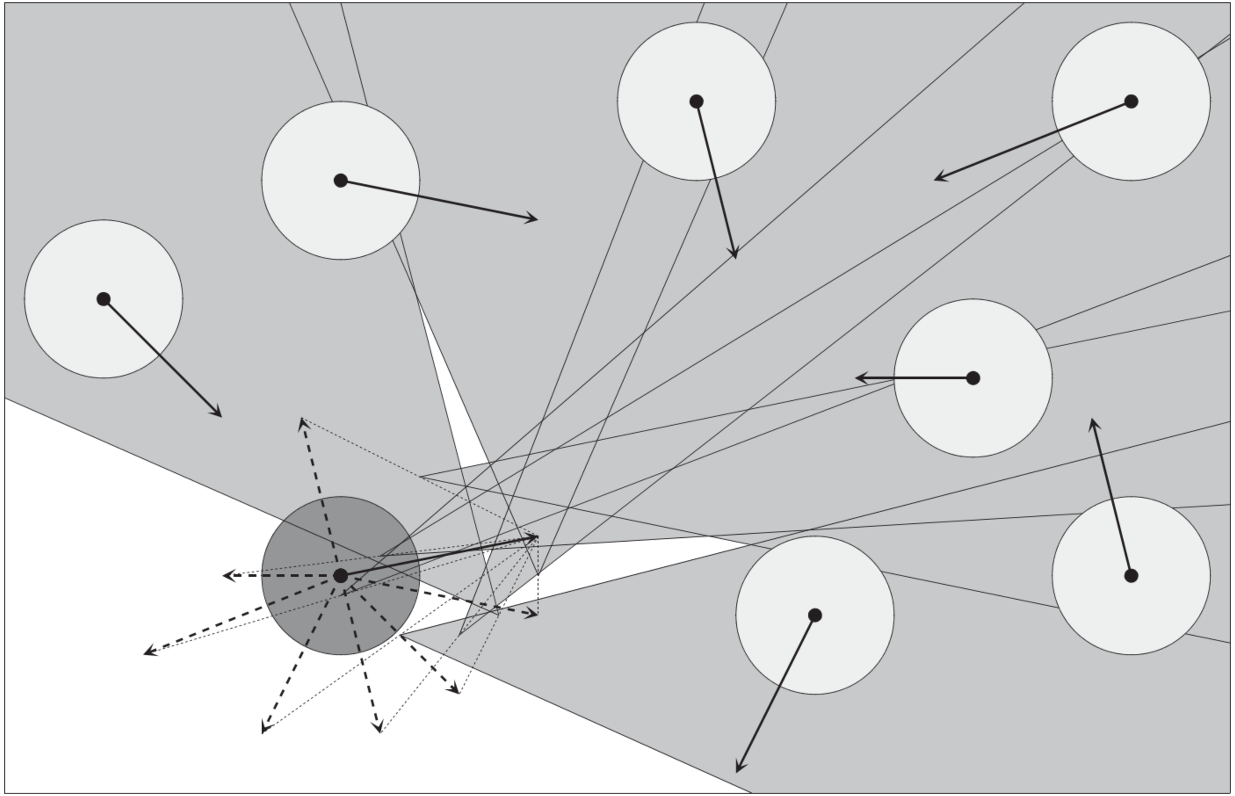



Strict proof can be found in page 1931 of the paper:
Van den Berg J, Lin M, Manocha D. Reciprocal velocity obstacles for real-time multi-agent navigation[C]. 2008 IEEE International Conference on Robotics and Automation. IEEE, 2008: 1928-1935.

# Reciprocal Velocity Obstacle (RVO)



## Multiple Obstacle Case



Van den Berg J, Lin M, Manocha D. Reciprocal velocity obstacles for real-time multi-agent navigation[C]. 2008 IEEE International Conference on Robotics and Automation. IEEE, 2008: 1928-1935.