Carnegie Mellon University

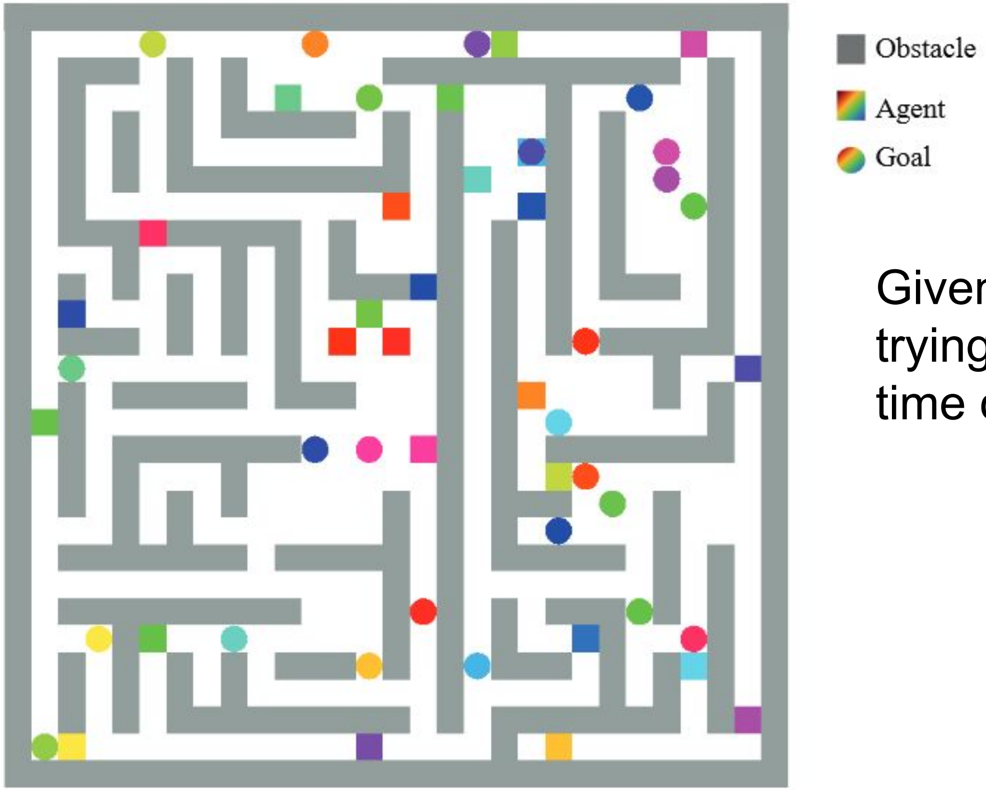
Warebots: Simultaneous Agent Routing

16782- Planning and Decision Making in Robotics Project

Aman Chulawala
Aneesh Sinha
Prakrit Tyagi

Motivation

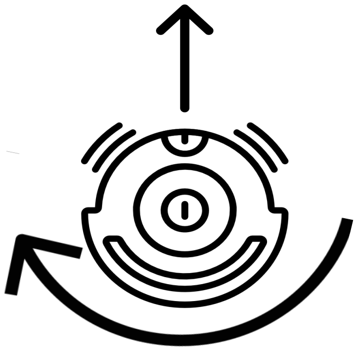
The Problem of Multi Agent Path Planning



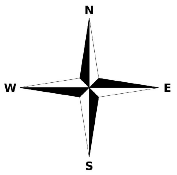
Given a **4 connected Grid**, N-agents are trying to maximize task completion under time constraint while avoiding path conflict.

M. Daman et. al., "PRIMAL_2: Pathfinding Via Reinforcement and Imitation Multi-Agent Learning - Lifelong,"

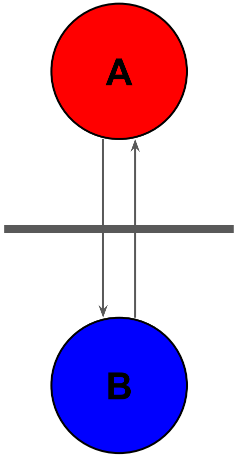
Problem Variables



Robot can move **Forward,** or **Rotating 90 degrees** at one location.

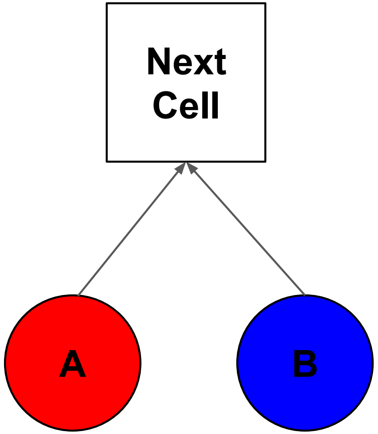
Action	State	Goal
F	X	X
CW 90°	Y	Y
CCW 90°	{N,E,W,S}	
W		

Problem Variables



Edge Conflict

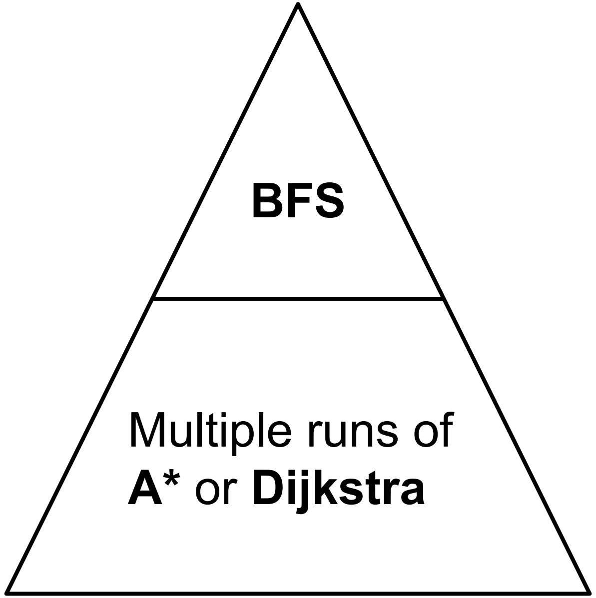
(A, B, V1, V2, t)

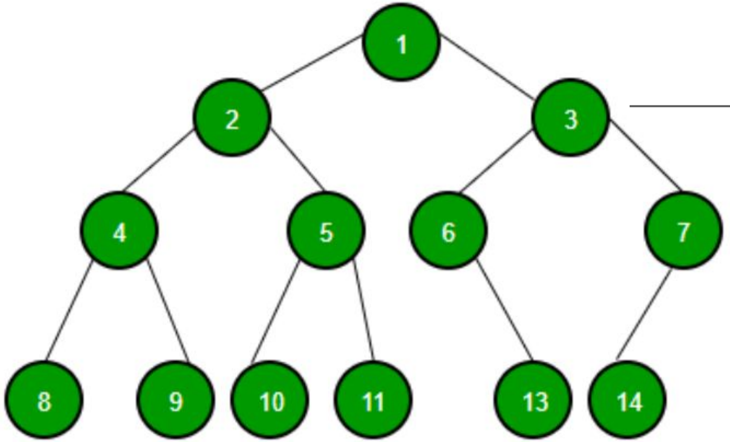


Vertex Conflict

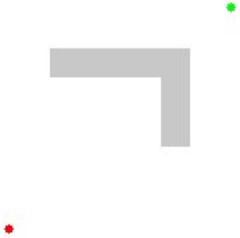
(A, B, V, t)

Planning Algorithm: Conflict Based Search





- Path of robots
- Constraint set
- Sum of cost



Pseudocode

Algorithm 1 Life Long Multi-agent Conflict Based Search

```
Paths = ∅
list<bool> agents_to_plan = {True for all}
bool run ← true
while t ≤ T_sim do
    if run then
        Paths ← CBS(Paths, goal_list, agents_to_plan)
        run ← false
        agents_to_plan= {False for all}
    end if
    action ← Paths[t]
    for each agent a do
        if action leads to goal then
            run ← true
            agents_to_plan[a] = true
        end if
    end for
    t++
end while
```

Initialize variables

Run while loop for time T_sim

CBS give solution & set variables to false

Execute action for time step t

Only if a action of an agent leads to goal, set variables to true. Since one task is complete. We will run CBS again!!

Increment time

Pseudocode

1. Create a root node
2. Reuse solution from previous CBS call

```
function CBS(Paths, goal_list, agents_to_plan)
    Root.solution = Paths[t,End]
    Update Root.solution using low_level(goal_list[t],agents_to_plan)
    Root.cost = Sum_of_cost(Root.solution)
    Root.constraints = ∅
    push Root to OPEN
    while OPEN is not empty do
        P ← best node from OPEN
        Check the paths in P for conflicts
        if P has no conflict then
            return P.Solution
        end if
        C ← first conflict(a_i, a_j, v, t) in P
        for each agent a in C do
            A ← new node
            A.constraints ← P.constraints + (a, v, t)
            A.solution ← P.solution
            Update A.solution
            A.cost = Sum_of_cost(A.solution)
            push A to OPEN
        end for
    end while
end function
```

Pseudocode

3. Call low level planner only on agents for which agents_to_plan is true.

```
1: function CBS(Paths, goal_list, agents_to_plan)
2:     Root.solution = Paths[t,End]
3:     Update Root.solution using low_level(goal_list[t],agents_to_plan)
4:     Root.cost = Sum_of_cost(Root.solution)
5:     Root.constraints = ∅
6:     push Root to OPEN
7:     while OPEN is not empty do
8:         P ← best node from OPEN
9:         Check the paths in P for conflicts
10:        if P has no conflict then
11:            return P.Solution
12:        end if
13:        C ← first conflict(a_i, a_j, v, t) in P
14:        for each agent a in C do
15:            A ← new node
16:            A.constraints ← P.constraints + (a, v, t)
17:            A.solution ← P.solution
18:            Update A.solution
19:            A.cost = Sum_of_cost(A.solution)
20:            push A to OPEN
21:        end for
22:    end while
23: end function
```

Pseudocode

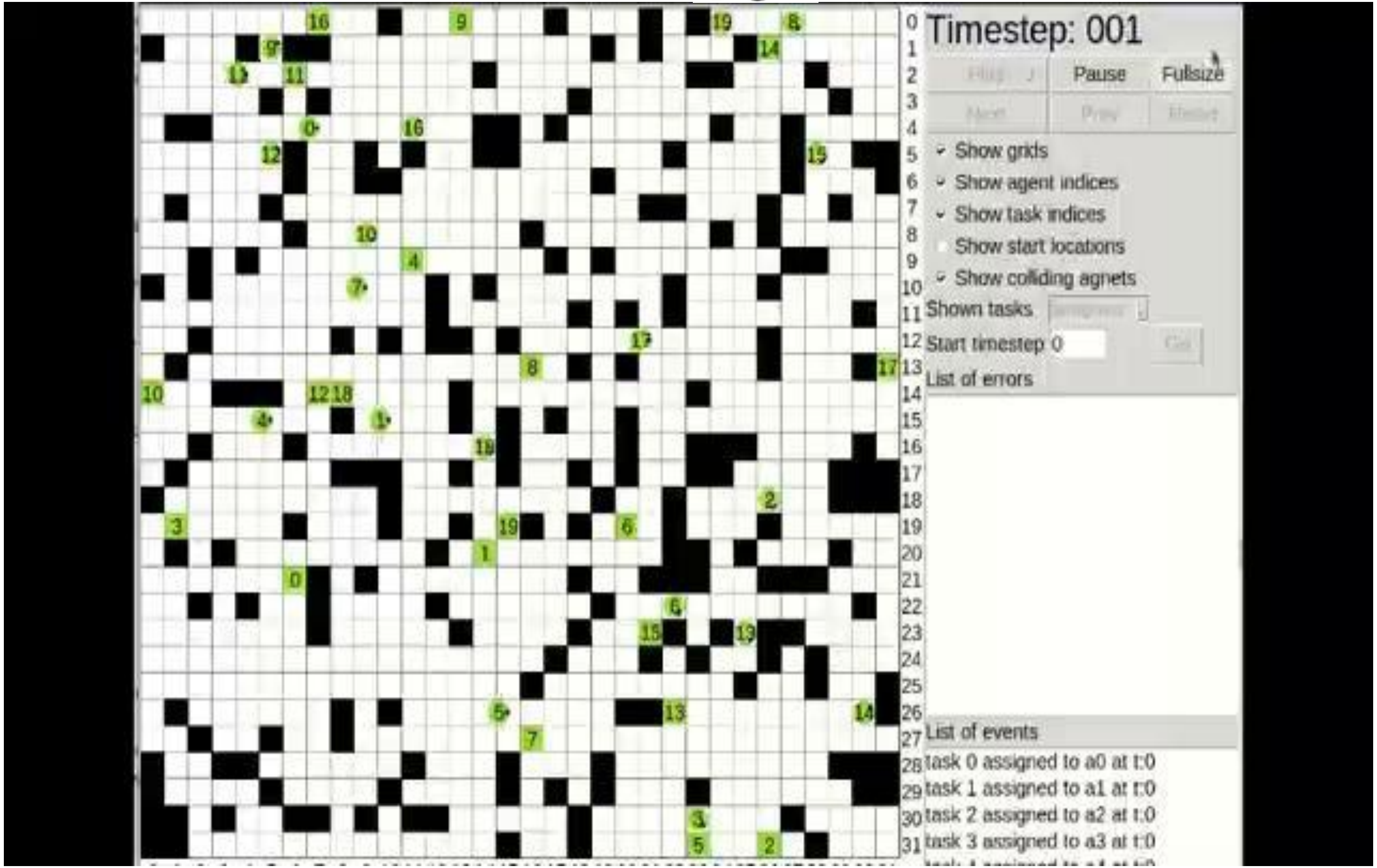
```
function CBS(Paths, goal_list, agents_to_plan)
    Root.solution = Paths[t,End]
    Update Root.solution using low_level(goal_list[t],agents_to_plan)
    Root.cost = Sum_of_cost(Root.solution)
    Root.constraints = ∅
    push Root to OPEN
    while OPEN is not empty do
        P ← best node from OPEN
        Check the paths in P for conflicts
        if P has no conflict then
            return P.Solution
        end if
        C ← first conflict(a_i, a_j, v, t) in P
        for each agent a in C do
            A ← new node
            A.constraints ← P.constraints + (a, v, t)
            A.solution ← P.solution
            Update A.solution
            A.cost = Sum_of_cost(A.solution)
            push A to OPEN
        end for
    end while
end function
```

Take the first conflict that you find and split the parent node.

Parent nodes constraints are propagated into child

Copy parent solution into child node and only update agent **a** path by calling low lever planner

Demonstration Video: PlanViz



2x Speed

Results

Metric	Warehouse map	Random map
Dimension	33x57 cells	32x32 cells
# of Agents	25	20
Task done	3666	3416
Solve time	72.722 sec	70.821 sec
Std in solve time	3 sec	3 sec
Avg Tree expansion	8.3 per CBS calls	5.2 per CBS calls
Total CBS calls	2625	2492

Discussion and Future Work

- Augmented CBS Planning is capable of managing more agents than Naive CBS
- Lifelong Augmented CBS is capable of planning for new tasks while keeping track of best results from the last run

Currently working on implementing features of

- Improved Conflict Based Search
- Enhanced Conflict Based Search
- Increasing Scalability of existing pipeline

Thank You

http://idm-lab.org/bib/abstracts/papers/aaai21b.pdf