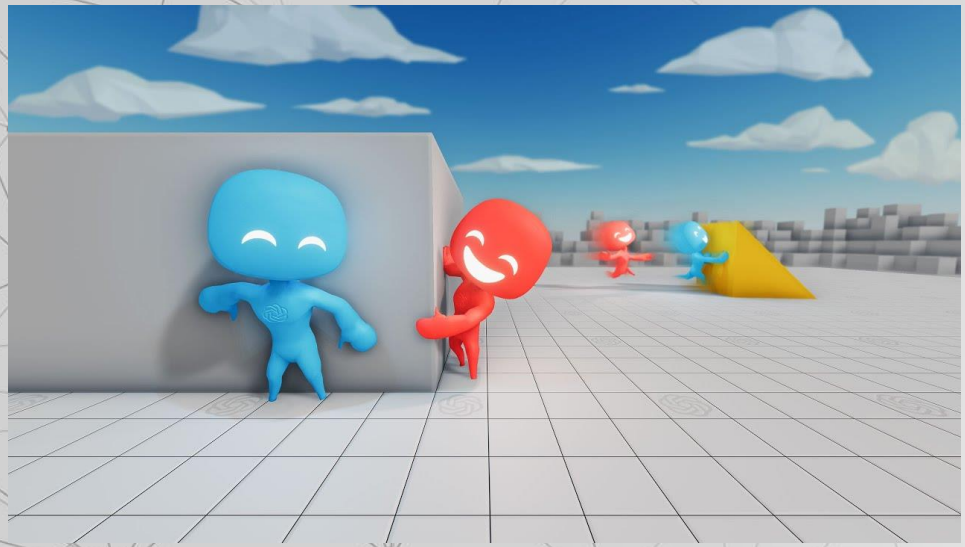
Multiagent Parallel Distributed A*

Tom Eliassy

Supervisor: Professor Jeff Rosenschein



The Hebrew University of Jerusalem

Benin School of Computer Science and Engineering

67537 - Tutorial Work

2019

Overview

01 **Background**
Introduction, Related Work, Platform, Motivation

02 **Multiagent Parallel Distributed A***
Description, Algorithm, Free Flow Specification

03 **Technical Aspects**
Flow Diagram, High-level UML,Modules

04 **Optimizations**
Dead-ends Detection, Stranded Colors and Regions, Chokepoints Detection, Fast-forwarding

05 **Observations**
Experimental Measurements, Findings

06 **Conclusions**

07 **References**

01

Background

Introduction, Related Work, Platform, Motivation

Introduction

- Supporting Re-planning using A* technique
- Interaction between some Agents
- Finding solution based on component division (projection)
- Respecting prioritization and constraints
- Based grid problem
- Offline calculating

Related Work

1. **Multi-Agent A* for Parallel and Distributed Systems (Nisim & Brafman): The ancestor work that our project is derived from (The generalized version)-**
    a. Deals with Parallel Distributed MA-A* in terms of Search problem (Successors, Expansion, Heuristic…).
    b. Generic algorithm which can be adapted to a variety of problems.
    c. Agents' communication (by signaling).
    d. Consideration of accessing to private and shared information.
    e. Distinction between private actions to public ones (existence of pre-conditions for the last actions).
    f. Reducing to a single A* Agent possibility.
    g. Trade-off: Limited computing power Vs. Information access.

Related Work

2. **From One to Many: Planning for Loosely Coupled Multi-Agent Systems (Domshlak & Brafman): Constraint Satisfaction Problem (CSP) for low coordination cases-**

a. Loosely coupled assumption (requires less coordination)

b. Deals with measurements of 2 parameters: Problem Independent (measures the the system's coupling level) and Problem Specific(The min. number and the max. number of actions required from an agent to solve the problem).

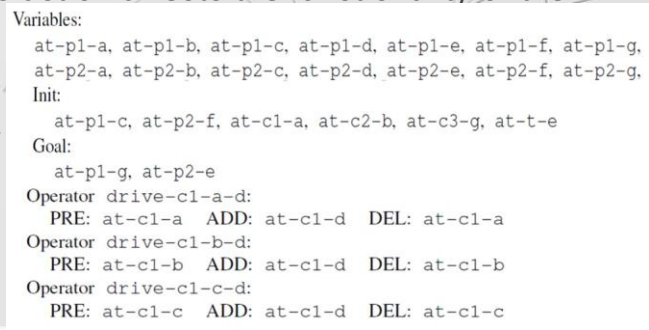
c. Using Constraint Satisfaction Problem (CSP) technique and formulation(STRIPS) with respect to the time dimension.

d. Focus on the worst - case time complexity.

e. Using "agent interaction graph"- Two agents are connected if one agent's action affects the functionality of the other agent (proportionate to the time complexity).

f. Some problems can't be solved due to the required level of coordination.

g. Defines internal actions, public actions as well as coordination points.


Variables:
at-p1-a, at-p1-b, at-p1-c, at-p1-d, at-p1-e, at-p1-f, at-p1-g,
at-p2-a, at-p2-b, at-p2-c, at-p2-d, at-p2-e, at-p2-f, at-p2-g,
Init:
at-p1-c, at-p2-f, at-c1-a, at-c2-b, at-c3-g, at-t-e
Goal:
at-p1-g, at-p2-e
Operator drive-c1-a-d:
PRE: at-c1-a ADD: at-c1-d DEL: at-c1-a
Operator drive-c1-b-d:
PRE: at-c1-b ADD: at-c1-d DEL: at-c1-b
Operator drive-c1-c-d:
PRE: at-c1-c ADD: at-c1-d DEL: at-c1-c

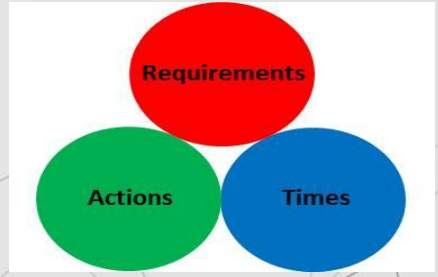
Related Work

3. **A General, Fully Distributed Multi-agent Planning Algorithm (Nissim, Brafman and Domshlak): Constraint Satisfaction Problem (CSP) Heuristic based. Adapted to the Multi-Agent concept -**

a. Loosely coupled, limited agent interaction: Some problems can't be solved due to the required level of coordination.

b. Combines local planning as well as coordination between agents. Defines internal actions, public actions and coordination points. Looking for a global and local consistency.

c. Using Constraint Satisfaction Problem (CSP) technique and formulation adapted for **some** agents(MA-STRIPS) with respect to the time dimension.



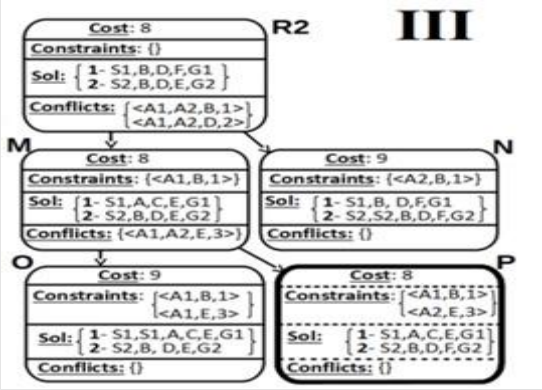
a. Respects the "Forced (essential)-Moves" concept by a global manner.

b. Other agents' planning as a "black-box". Don't match to a centralized planner, improves centralized solutions.

c. Splitting the aspects of an agent planning among 3 variables: (public) Actions, Time(s) and Requirements.

d. Uses Heuristic that dynamically selects an agent: based on *most-constrained* (first) and *goal-achieving*(#achievable sub-goals) concepts. (rather than Min-Domain variable Ordering heuristic that requires fully generated domains).

Related Work



4. **Conflict-Based Search Algorithm for Multi-Agent Pathfinding (Boyarski et al.):**

2 levels centralized search, respects constraints -

a. Pathfinding oriented, based on graph structure with respect to the time dimension.

b. Formalized as a global, single-agent search problem for centralized computing.

c. Allows “wait” action which is an integral part of the described algorithm.

d. Aims to improves the CBS by focusing on cases that it performs poorly, exponentially worse than A*.

e. Composed of 2 levels: 1) High-level constructs a *constraint-tree* (nodes contain time & locations constraints of a single agent) by finding conflicts and constraints addition. 2) Low level search which is performed at each node of the *constraint-tree* in order to find individual paths for all the agents considering the constraints of the corresponded high-level node (can be done by any algorithm, usually A*).

f. Bounds the number of allowed conflicts at the high-level phase between any pair of agents by a predefined parameter B. When exceeding B, the conflicting agents are merged into a single agent that treated by the low-level solver.

g. There are 3 improvements for the algorithm:

   i. Bypass (BP)- Improves the low-level paths in some ways => reducing the CT size.

   ii. Merge & Restart (MR)- When a decision to merge is made, it is suggested to restart the search (avoiding duplicating effort for the low-level solver when calculating the path and the constraints for the merged agent B + 1 times).

   iii. Prioritizing Conflicts (PC)- Conflicts classification by 3 types: *cardinal* (certainly increases the solution’s cost), *semi-cardinal*

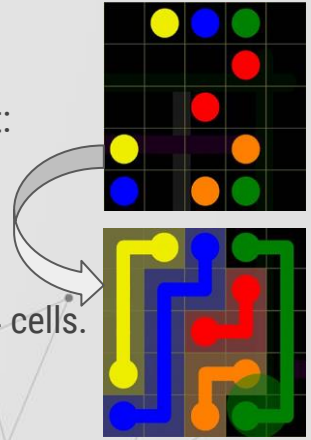
Putting all together

	MA-A* Parallel Distributed	From One to Many	General Fully Distributed MA	(I)CBS
Technique based	A* Search	CSP	CSP	A* Search
Loosely Coupled	X	V	V	preferred
Distributed/Centralized	Distributed	Centralized	Distributed	Centralized
Agents' Communication	Signaling	Constraints	Requirements Vars.	Centralized planner
Distinguish between public & private actions	V	V	V	X
Agent interaction Graph	X	V	V	X
Parameters	X	Problem Independent and Problem Specific	Actions, Times and Requirements	X
Bounding	X	Time complexity	Number of public actions per agent	Number of conflicts
Heuristic	Problem adapted	X	Selecting agent	X
Time Respecting	X	V	V	V

Platform

Flow Free-

- A puzzle game for Android and iOS released by American studio Big Duck Games in June 2012.
- The game takes place on a grid contains some couples of colored dots (the uncolored squares are considered to be empty).
- The objective is to draw flows (paths) between every pair of dots with the same color such that:
    1. All the couples of dots will be connected
    2. All the empty squares will be filled by the flows.
- The difficult level is directly affected by the squared-size of the grid, ranges from 5X5 to 14X14 cells.
- There are some expansions to the game:
    3. “Flow Free: Bridges” (November 2012)- Contains bridges that have to be occupied by a flow that will pass it above other flow (may be the same flow).
    4. “Flow Free: Hexes” (October 2016)- Hexagon- shaped grids.
    5. “Flow Free: Warps” (August 2017)- Different squares- shaped grids such that “entering” an edge “transports” the corresponded flow to the other-side edge by the “back” of the board.



Wikipedia: https://en.wikipedia.org/wiki/Flow_Free , YouTube: https://www.youtube.com/watch?v=nxp8Zg1gmro

Matt Zucker's Programs

Matt Zucker developed 2 programs that solve Flow Free puzzles:

1. **A* Search (C Implementation)-**
   a. Single Agent (Thread), every time there is an activated color that performs a move (Reduces the Branching factor).
   b. For a State s:
      i. Cost to come: g(s) = Number of moves that were made until now (not including Forced-moves).
      ii. Estimated Cost to go: h(s) = Number of empty squares (Inadmissible heuristic).
   c. Optimizations: Dead-ends detection, Stranded Colors & Regions, Chokepoints Detection, Fast-Forwarding.
2. **CSP:Constraint Satisfaction Problem (Python Implementation)-**
   a. Reducing general CSP to SAT using CNF (Conjunctive Normal Form) expression.
   b. Applies the following concepts:
      i. Every cell is assigned a single color.
      ii. The color of every endpoint cell is known and specified.
      iii. Every endpoint cell has exactly one neighbour with the same color.
      iv. The flow in every non-endpoint cell matches exactly one of the six direction types.
      v. The neighbours of a cell specified by its direction type must match its color.
      vi. The neighbours of a cell NOT specified by its direction type must NOT match its color.
   c. For a puzzle of size n with c colors, there are n*c colors vars. And 6*n direction vars.

Since SAT runtime is exponential in the number of vars. in the worst case, the run time is therefore $\mathbf{O}(2^n)$.

Direction type	Unicode char	Description
1	─	left-right
2	│	top-bottom
3	┘	top-left
4	└	top-right
5	┐	bottom-left
6	┌	bottom-right

Motivation

Regarding the properties of the Flow Free platform, we would like to have an appropriate planner (which is not based on CSP) that has the following features :

- ❏ Tightly coupled agents are treated
- ❏ It is hard to distinguish between public actions to private/internal ones.
- ❏ Distributed (for supporting projections).
- ❏ Bounded overhead (as a result of access to shared-resources).
- ❏ Contains agents interaction(as a result of the distributing requirement).
- ❏ No time adaption (since there is no meaning to “wait” action in the Flow Free platform).

Note: There is no meaning to the optimality of the solution.

02

Multiagent Parallel Distributed A*

Description, Algorithm, Free Flow Specification

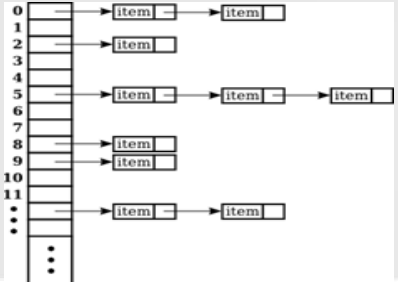
Description

Multiagent A* is a distributed variation of A* that can be done in parallel calculation. There is a shared-resource which is actually a data-structure contains information that is relevant for all the participant agents so they have to communicate each other using this data structure (may no one of them deals the whole knowledge). Because of the shared- resource use, there is a need to use a mutual- exclusion object that will limit the simultaneous access to this shared-resource only for a single agent every time. In turn, every agent may store a state that is necessary for the other agents (with specifying to which ones of them) and will also pick a relevant state that an other agent has posted for him .

Description

In addition, every agent has a **minimum** priority- queue which stores the states that are candidates for expansion (so called openList). In every iteration of the A* algorithm (next slide) we'll expand the state s with the minimum f(s) = g(s) + h(s) value (explanation in the next few slides). Alongside the openList, there is a closedList of already expanded states. Note: While giving the same state to two different agents they will generate **different** successors states (since each of them has his own considerations).

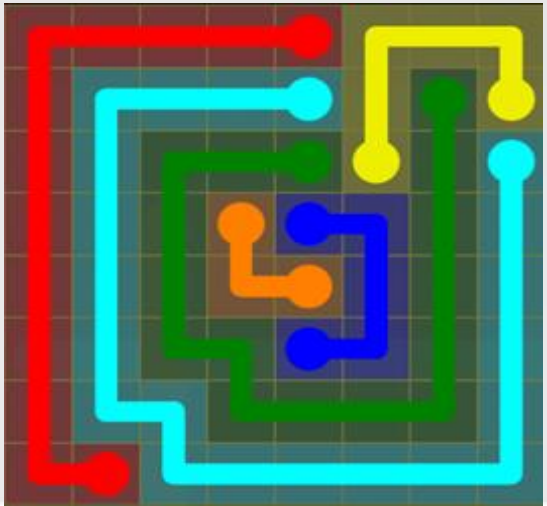
Expansion prioritization: First, an agent will try to expand a state taken from the shared- resource. If there is no such state, the agent will try to expand a state from his openList and in case that it is empty - the agent will go to sleep.



Description

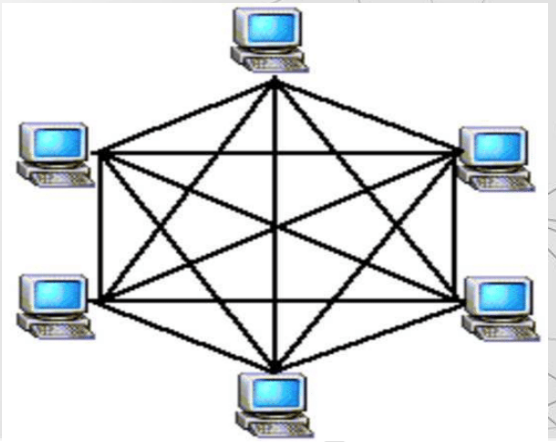
Termination Detection: Occurred when an agent completes his flow while all the other agents have already executed their flows on the same state before. In such a case, the agent will notify all the other agents about the solution finding and will update the global variable that stores the solution state (which is also a shared-resource).

Notifies



Algorithm

1. **Multi-Agent A*:**
    a. First expansion (The shared-resource and the openList are empty, no states to get)
    b. While (Not (global goal state)) do
        i. Acquire the mutual-exclusion object
            1. If (there is a relevant state in the shared-resource which hasn't been expanded by this agent yet)
                a. Get the state from the shared resource and expand it
        ii. Release the mutual-exclusion object
        iii. If (didn't get a state from the shared-resource)
            1. If (the openList is Not empty)
                a. Get a state from the openList and expand it
            Else
                a. Go to sleep



Algorithm

2.Expand (state):

a. Insert state to the closedList // Mark as visited

b. If (state is a global goal state) -> return

c. successors <- Find successors (state)

d. Each successor that is NOT in the closedList will be inserted to the openList

e. If (one of the successors is a local goal state) -> Post the state to the relevant agents using the shared resource

3.Find successors(state):

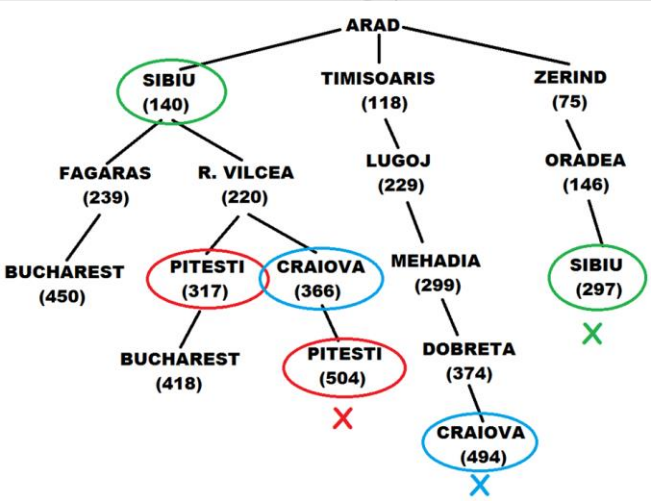
a. Optional moves <- Calculate the optional moves for the current agent

b. While (There is only a single optional move) do Fast-forwarding

c. For every move in Optional moves

   i. If (The move has to be eliminated) -> add move to the closedList

d. Return Optional moves



Flow Free - Problem Specification

- **Shared-Resource (Data Structure)** → Dictionary maps <Agent, min. Priority queue of pending states for agent>. The priority queue ordered regarding the f = g+h value of the pending states.

- **Mutual-Exclusion Object** → A single Semaphore, Free of deadlock and starvation.

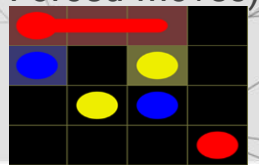
- **Notifying about global goal state** → Using Python's Event, function as a sleeping flag. In Python it is possible to send signals only to the main thread (https://docs.python.org/3/library/threading.html)

- **Prioritizing states contains complete flows of other agents** → By setting g_value = 0 for the state that is sent to them.

- **g value and h value** → **g:** #moves executed only by the current agent (not include Forced-moves).

  h: #empty squares in the state

  In the example taken from https://mzucker.github.io/2016/08/28/flow-solver.html: g(s) = 1, h(s) = 8



03

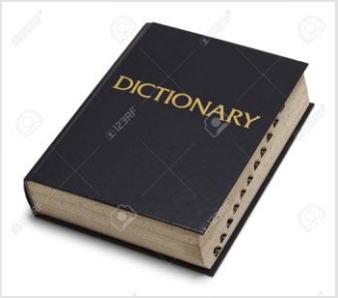
Technical Aspects

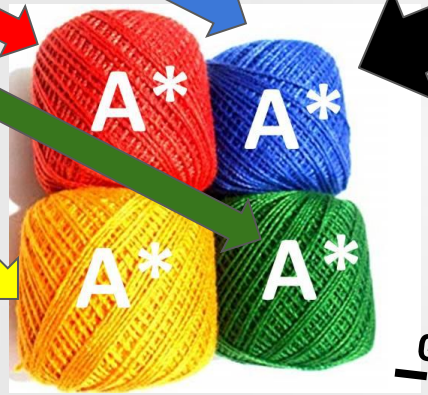
Flow Diagram, High-level UML,Modules

Flow - Diagram

Main Thread

1. Converting the problem to Search format
2. Agents Creation
3. Threads Creation





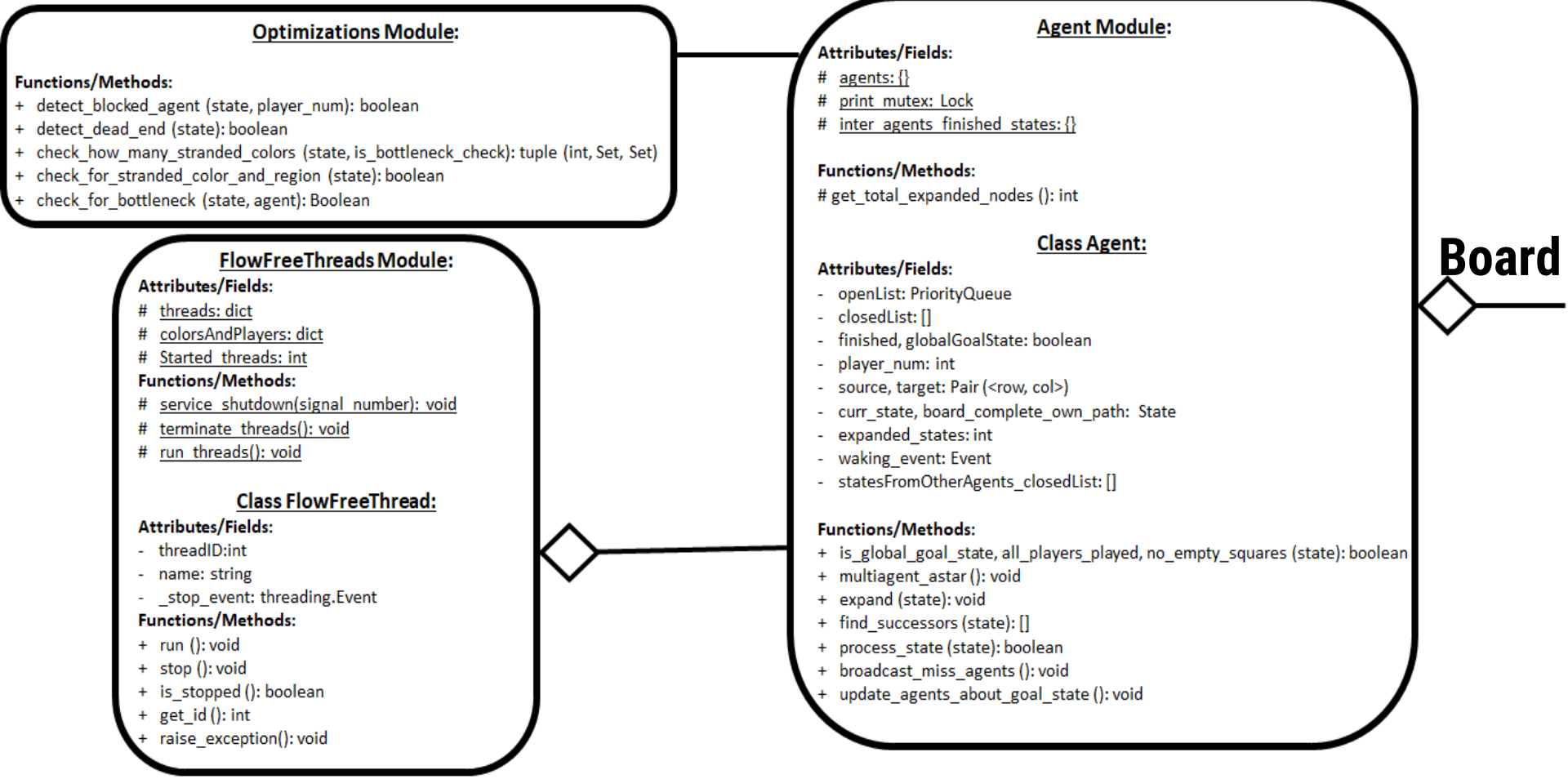
Global Goal State has been found

Main Thread

4.Threads Termination

5. Producing Summary

High-Level UML - Part 1



High-Level UML - Part 2

Agent

Board Module:

Attributes/Fields:
goal_state: State
update_global_goal_mutex: Lock

Class State:

Attributes/Fields:
- size: int
- players: dict
- board: []
- g_value, h_value: int
- sources, targets: dict
- finished: dict
- num_of_finished_agents: int
- head: Pair (<row, col>)
- player: int

Functions/Methods:
+ convertToNpFormat (boardRepresentationString): void
+ set_head (row, col): void
+ determining_targets_and_sources(): void
+ update_finished_agents (): void

+ is_agent_goal_state (agent_num): boolean
+ check_move_valid (row, col): boolean
+ get_possible_moves_for_player (): []
+ is_forced_moved (row, col, player): boolean
+ perform_move (row, col, agent): void

+ check_for_player_flow_neighbour (row, col): Boolean
+ is_head_a_neighbour (row, col): boolean
+ num_of_free_neighbours (row, col): int
+ edgepoints_neighbour_didnt_finish (row, col): boolean

+ is_same_board (other): Boolean
- __eq__, __ne__, __lt__, __le__, __gt__, __ge__ (other): boolean

+ print_board (): void

RegionsMap Class and Module:

Attributes/Fields:
- size: int
- Orig_state: State
- board: []
- dependencies: dict

Functions/Methods:
+ regions_map_init_and_first_row_calculation (regions_map, current_label, need_to_decrease_label) : Pair (<boolean, int>)
+ produce_regions_map_pass1(): {}
+ dependencies_updating (row, col): void
+ find_representative (given_depends, item): int
+ produce_regions_map_pass2 (dependencies): Set
+ find_regions (row, col): Set
+ regions_lists_contains_mutual_area (region_list1, regions_list2, agent_num): boolean

FlowFreeThreads Module

Global:

Attributes/Fields-

- Threads container
- Map <Color, Player Number>
- Started threads counter

Functions/Methods-

- Starting threads
- Terminating threads
- Signaling the Main thread

FlowFreeThread Class:

Attributes/Fields-

- Info: Name and ID
- Stopping event

Functions/Methods-

- Run
- Stop
- Get Info
- Raise Exception

Agent Module

Global:

Attributes/Fields-

- Agents container
- Print Mutex
- **Shared-Resource:** bulletin board- where every Agent can post relevant States for the other Agents

Functions/Methods-

- Summarizing

Agent Class:

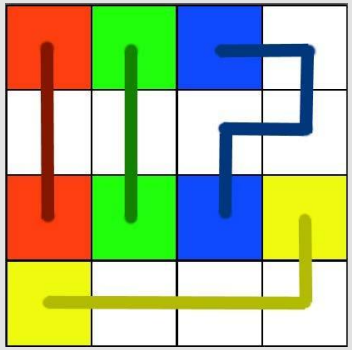
Attributes/Fields-

- Lists: OpenList & ClosedList, statesFromOtherAgents
- Edgepoints: Source & Target
- Relevant Boards: current, last completed
- Info: ID number, #expanded nodes
- Finish Flags: Local & Global
- Waking Event

Functions/Methods-

- Recognizing global Goal-State
- Multiagent A* interface (Multiagent A*, Expand, Find Successors)
- State Processing
- Communication with the other Agents

Board Module



Global:

Attributes/Fields-

- Global Goal-State and it's Mutex

State Class:

Attributes/Fields-

- General Info: size, mapping <Color, Player Number>, finished Agents tracking, Sources & Targets
- Values: g & h
- Player: Number, Head, current Board
- **Regions Map**

Functions/Methods-

- Formalizing the puzzle as a Search Problem
- Initialization: Setting the Head, Determining the Sources and the Targets
- Recognizing local Goal-State
- Move: Validation, Performing, Options, Forced checking
- Neighbours: #Free neighbours, Recognizing a Flow's Head as a neighbour, Neighbours that are edge points of incompleted Agents, Make sure that there is a neighbour of the current Flow
- Comparable Interface
- Help Methods: Print, Indices Converting, Test, Manhatten Dist. Heur., #Empty tiles
- Keep track of the number of finished Agents

RegionsMap Module

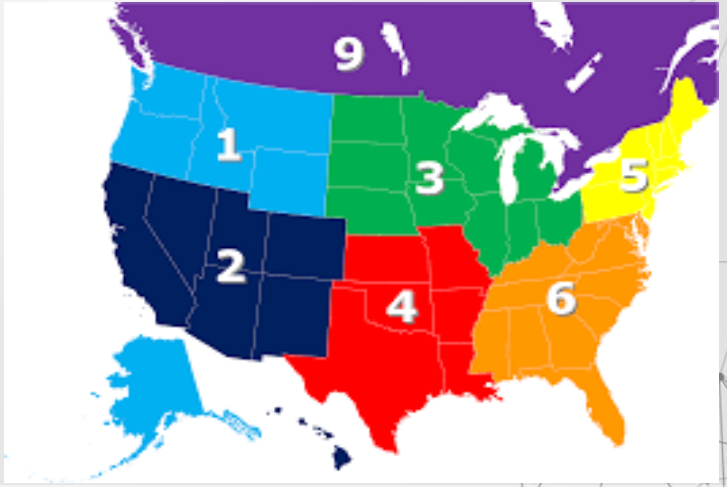
RegionsMap Class:

Attributes/Fields-

- Ancestor Info: size, board and instance
- Dependencies List

Functions/Methods-

- First Row Calculation (looking only Left)
- Pass 1 & Pass 2 (Connected Component Labeling algorithm)
- Dependencies Manipulations
- Regions Manipulations



04

Optimizations

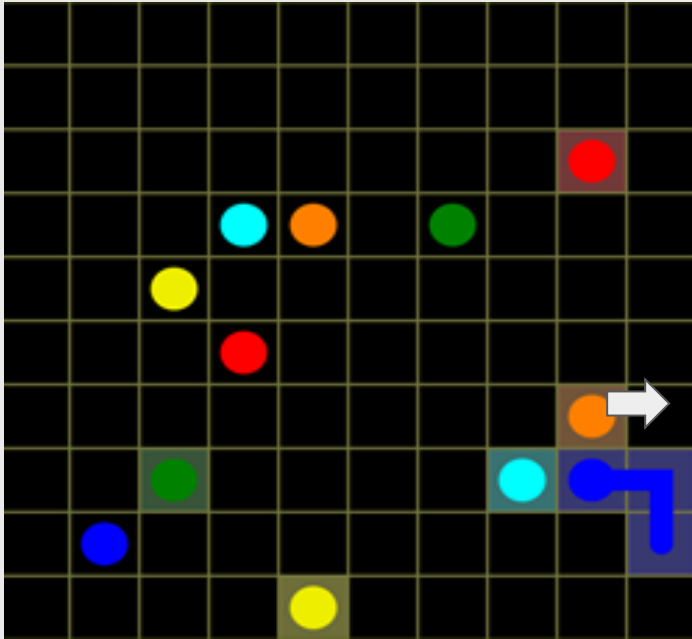
Dead-ends Detection, Stranded Colors and Regions, Chokepoints Detection, Fast-forwarding

Forced-Moves consideration

A move is considered to be forced if a flow has only one single possible square to move into.

OR

An empty square is adjacent to only one single flow's head and it has only one free neighbor (like the Orange agent in the picture)

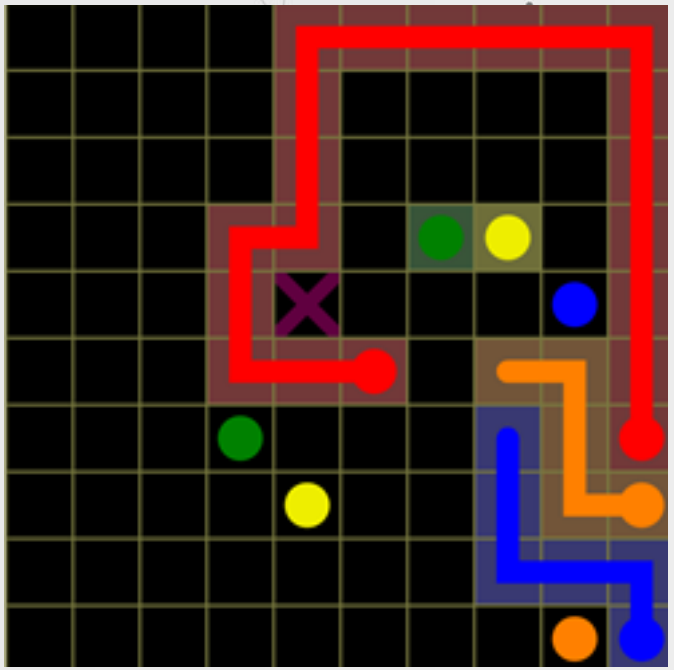


Taken from: https://mzucker.github.io/2016/08/28/flow-solver.html

Dead-ends Detection

A square that has a way into it but there is no way out of it (like the X square in the picture).

A state contains a dead-end like this is unsolvable and should be eliminated in order to reduce the branching factor.

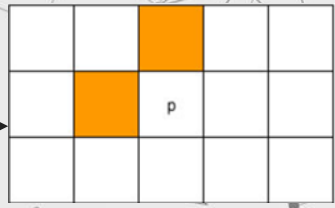


Taken from: https://mzucker.github.io/2016/08/28/flow-solver.html

Connected Component Labeling Algorithm

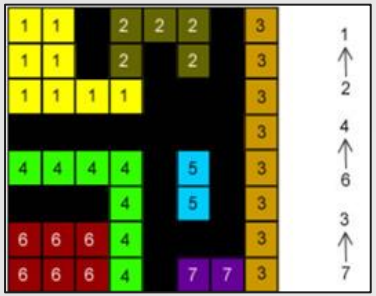
First Pass:

- We'll look at the **up** neighbor and the **left** neighbor of each square

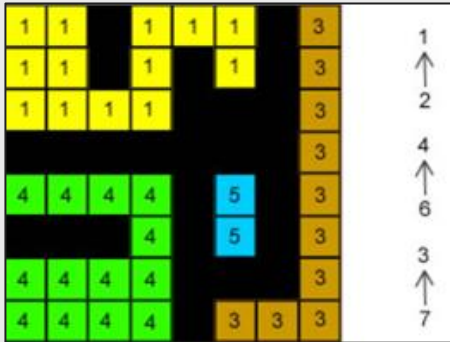


- In case of conflict, we'll generate a dependency (For example 2->1)

Finally →



Second Pass: Unification of regions considering dependencies-

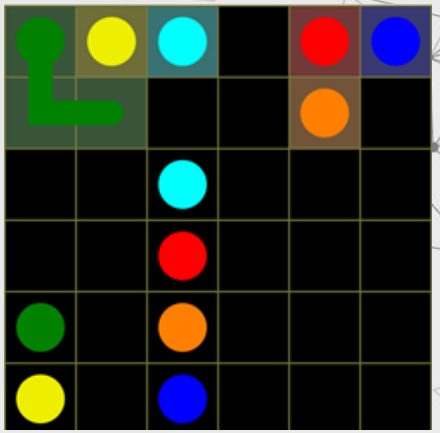


Taken from: http://aishack.in/tutorials/connected-component-labelling/

Stranded-Colors

1) Blocking a source/target of other agent.

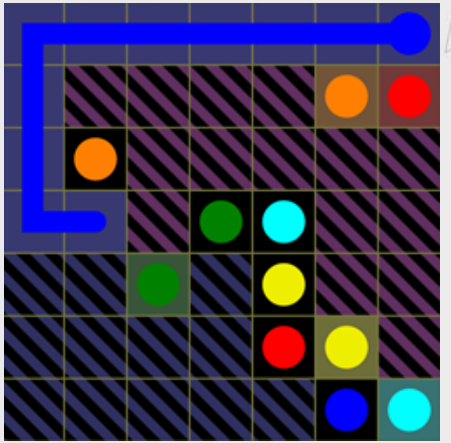
   (Like Green blocks Yellow)



1) For each non-completed color, there must be

   a region contains both the current position

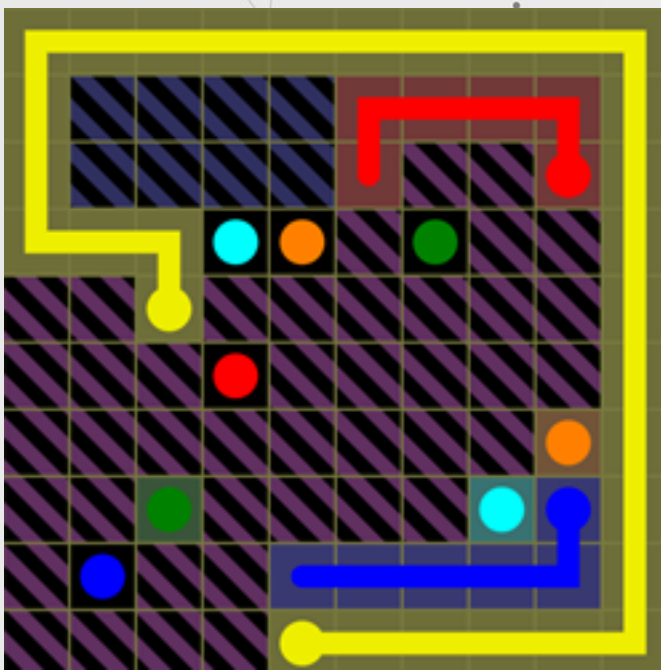
   (head/source) as well as it's target.



Taken from: https://mzucker.github.io/2016/08/28/flow-solver.html

Stranded-Regions

For each region, there must be at least one contained color whose current position (head/source) and target are located at the same region.



Taken from: https://mzucker.github.io/2016/08/28/flow-solver.html

Chokepoint Detection

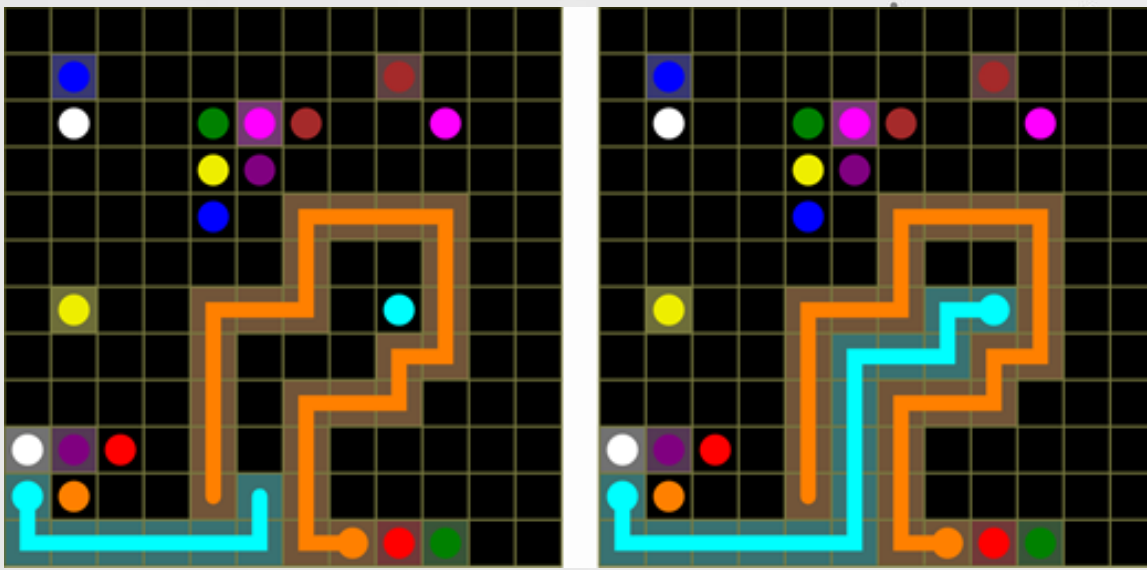
A bottleneck of width n can’t enable flow of more than n colors. This is an unsolvable situation.

Taken from: https://mzucker.github.io/2016/08/28/flow-solver.html

Fast-Forwarding

Speeding up the expansion of Forced-Moves every time

that there is such a move.



Taken from: https://mzucker.github.io/2016/08/28/flow-solver.html

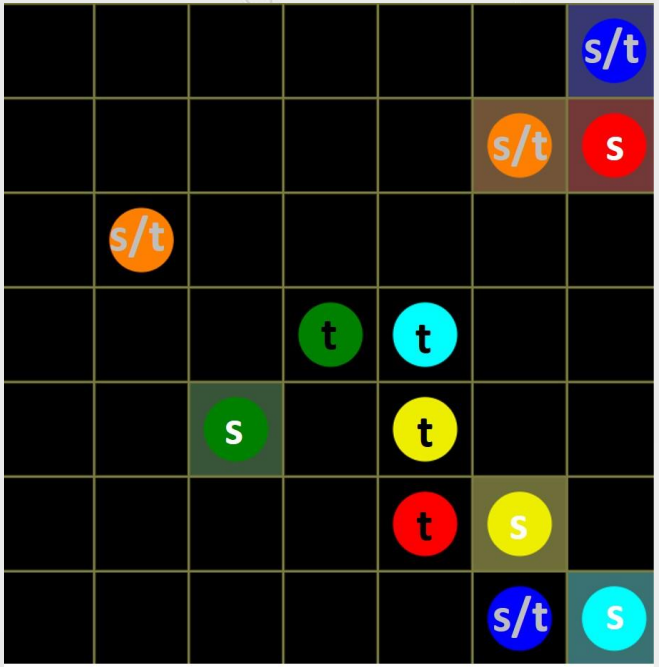
Source/Target Determining

Rule of Thumb: We would prefer the source to be closer to the outer boundary than the target.

s = Necessarily source

t = Necessarily target

s/t = can be source or target



Taken from: https://mzucker.github.io/2016/08/28/flow-solver.html

05

Observations

Experimental Measurements, Findings

Experimental Measurements

	MA-A*		CSP			A* (C Implement.)	
	avg. Time	avg. Nodes	Time	SAT vars.	clauses	Time	Nodes
regular 5X5	0.23	102	0.003	184	2,031	0.001	16
regular 6X6	0.46	119	0.004	322	4,353	0.001	25
regular 7X7	3.01	1,060	0.007	452	6,418	0.001	40
regular 8X8	1.56	282	0.010	616	9,452	0.001	55
regular 9X9	9.82	1,607	0.021	1009	17,595	0.001	166
extreme 8X8	31.36	5,769	0.011	571	8,472	0.001	76
extreme 9X9	3 min.<	-	0.017	661	9,112	0.001	111
extreme 10X10	3 min.<	-	0.019	938	14,818	0.034	4,625

Notes: 1)The avg. values of MA-A* were taken regarding the mean of **5** measurements 2)The times are in sec.
3) Processor: Intel(R) Core(TM) i5-5200 CPU @ 2.20 GHz 2.19GHz

Findings

- **Problem classification:** Tightly coupled agents, Low coordination.

- **Time and Space Complexities:** Regarding a board (grid) contains n squares and branching factor of **3** -
  Space Complexity- O(n*3^n), the complexity of A*/UCS Search s.t every node contains a few grids.
  Time Complexity - O(n*3^n), since we perform checks all over the board in each turn (Expansion).

- **C is much faster than Python** (pycosat is also implemented in pure C).

- **MA-A* generates much more nodes than A*** => Leads to running time that is much longer.

- **There is no meaning to the optimality of the solution** apart from finding such one. The usage of the Cost function and the Heuristic function is to guide our search toward finding a solution by efficient manner regarding the aspects of Nodes expansion and Time complexity.

06

Conclusions

Conclusions

- **Regarding the platform of Flow Free and our algorithm, an addition of agent/s (color/s) to a given puzzle increases the run time complexity only by a polynomial rate** as opposed to the grid size increasing which results in exponential growth of the time complexity.

- **CSP is more effective than A* Search** in case that the branching factor is greater than 2 and the number of CSP variables asymptotically equals to the max depth of the search tree.

- **There are duplicated local calculations in the MA-A*** (Although avoiding duplicated global calculations like 1->2->3 which is similar to 1->3->2/2->1->3/2->3->1 / 3->1->2/ 3->2->1).

- **Splitting the closedList to local part and global part may reduce the running time for some cases.**

Conclusions (More trivial ones)

- Using parallelism doesn't necessarily reduces the running time (since there is an issue of access to shared resources and maybe duplicated calculations).
- There are problems which have to be solved by centralized manner in their nature.
- The Hardware (Processor, Cores) has a significant influence on the performance.
- **Future work:** To check whether based graphplan heuristics improve the performance (since they are more accurate) although it will take more time to calculate them.

07

References

References

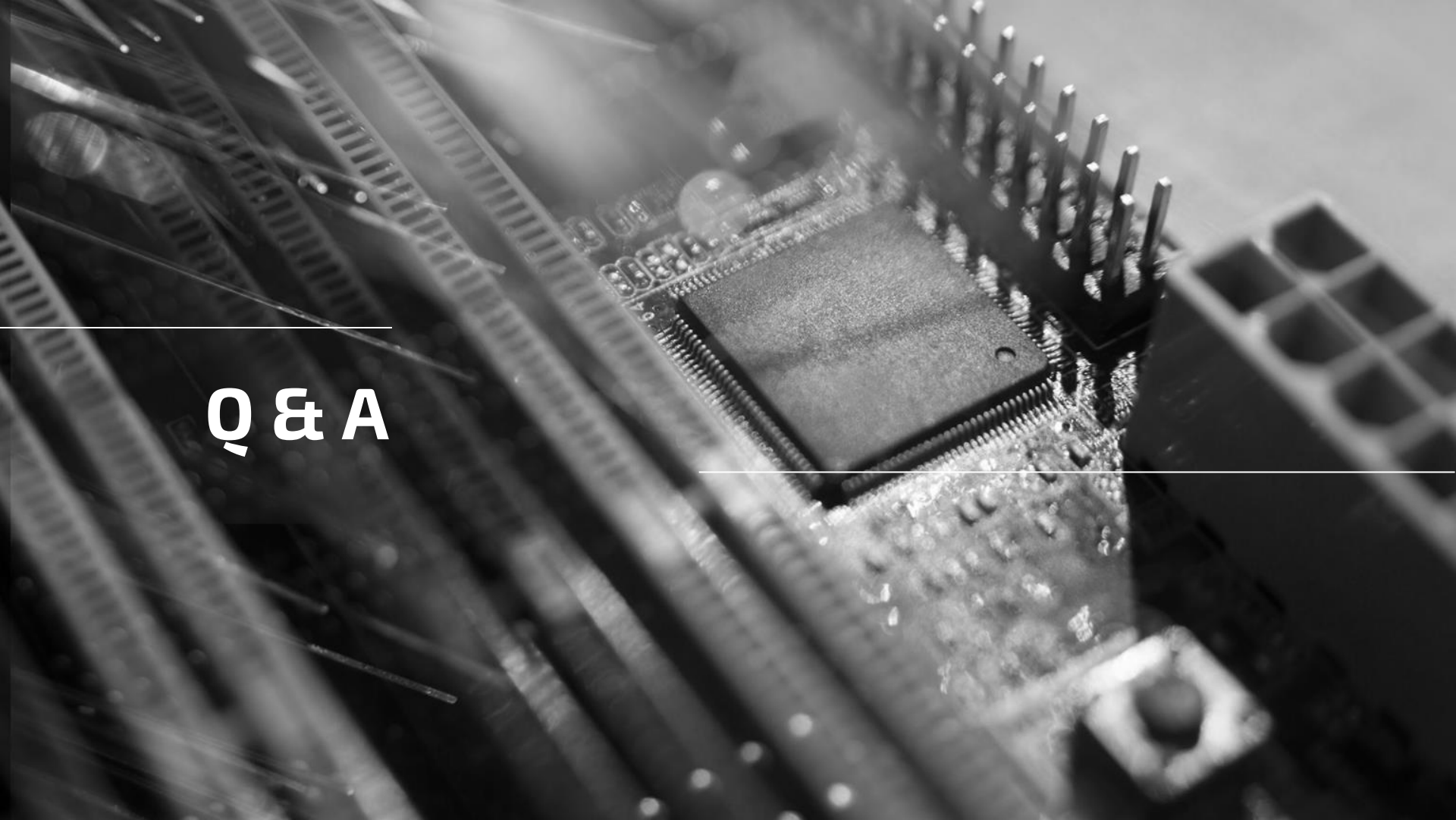
[1] **Multi-Agent A* for Parallel and Distributed Systems** - Raz Nissim and Ronen Brafman

[2] **From One to Many: Planning for Loosely Coupled Multi-Agent Systems** - Ronen I. Brafman and Carmel Domshlak

[3] **A General, Fully Distributed Multi-agent Planning Algorithm** - Raz Nissim, Ronen I. Brafman and Carmel Domshlak

[4] **Meta-Agent Conflict-Based Search For Optimal Multi-Agent Path Finding** - Guni Sharon, Roni Stern, Ariel Felner and Nathan Sturtevant

[5] **ICBS: Improved Conflict-Based Search Algorithm For Multi-Agent Pathfinding** - Eli Boyarski, Guni Sharon, Roni Stern, Ariel Felner, David Tolpin, Oded Betzalel and Eyal Shimony


Q & A

Thanks You!