Leader Election for Optimal Resource Allocation in Greedy Multi-Agent Systems

Aman Agarwal, Anshuman Das

***Abstract*—In decentralized multi-agent systems, self-motivated decisions made by individual agents can conflict with the performance of the group as a whole. The election of a leader by the system allows for the possibility of directing individual agents to avoid greedy choices and instead make decisions that optimize for the whole group's benefit. In this project, we analyze and compare various decentralized leader election criteria and their impact on the efficiency of a dynamic group of agents, that is neighbouring vehicles in a simulated multi-lane highway, where standard right-of-way traffic rules are suboptimal and can cause clogging. The leader is responsible for optimising lane allocation to vehicles in order to avoid clogging by slower agents and ensure the fastest time of completion for the whole group. So far, leader election and consensus algorithms have been seen as an isolated problem, separate from the the system performance. We explore whether and how by varying the election criteria of the leader, a dynamic system of selfish agents can be made to show different behaviours.**

***Index Terms*—Leaders, election, decentralized, consensus.**

I. Introduction

Decentralized multi-agent systems have a variety of advantages over their centrally-controlled counterparts. Decentralized control is attractive because it allows for higher scalability, and lacks a single point of failure which allows for greater fault tolerance. Through solely local communication and other interactions, multi-agent systems such as swarm robots have displayed the ability to create collective intelligence and demonstrate highly complex behaviours[1, 12]. Such phenomena have been exploited in swarm robotics to achieve impressive results [3, 13].

However, when the agents may have selfish or conflicting interests, the system tends to move towards anarchy and leads to sub-optimal solutions for the group, losing all advantages of such collective intelligence. The election of a *leader* enables us to execute centralized algorithms in a decentralized environment, therefore mitigating some of the above risks and improving the quality of the solutions that the group converges to, while also being resilient to cascading errors that are prevalent in anarchist systems[**error1**, **error2**] or disturbances to the system topology since the system lacks any single point of failure [9, 6, 15].

However, most leader election algorithms in the literature do not take into account dynamic systems in which the agents may be constantly moving at different relative speeds, and entering/exiting the system constantly. In such a situation, the asynchronous nature of communication can further reduce the stability of the established consensus due to constant delays in receiving up-to-date information by the whole group.

In this project, we explore leader election in such a system of selfish agents. We choose a simple system of a multi-lane highway and a large number of individual cars moving at different speeds, which all start from one end of the highway and move in the same direction. Since the cars have different speeds, and they arrive at the highway sequentially, it is natural for agents to greedily choose the emptiest lane possible. This self-motivated approach quickly clogs some available lanes with slow vehicles that should have ideally queued in the same lane, ordered such that faster cars behind them can enter the faster lanes, and therefore reduce the time taken for all cars to get through the full highway.

Therefore, it is reasonable to elect a leader which can make the optimal lane allocation decisions for incoming cars, to overrule the greedy decision-making algorithm of individual cars and reduce chances of lane clogging. In this paper we present: An election algorithm that enforces a leader selection within fixed number of election time steps, overruling consensus in favor of speed; and also present some of the results obtained.

A. Problem Formulation

The setup for the simulated environment begins with initial lists of N lanes and P cars, which have different randomly generated speeds, such that $1 \leq N \leq P$.

The minimum time of travel completion for the full system of cars, t_{min}, corresponds to the case when $N = P$: with greedy choice, each car occupies its own lane, so the t_{min} is dominated by the slowest car. At the same time, the maximum time of travel completion for all P cars, t_{max}, corresponds to $N = 1$, when the slowest car is the first car to enter the highway from the queue.

For any pair (P, N) in an anarchist arrangement, the cars do a *greedy* lane selection and the total time for all cars to cross is t_{greedy}. With a leader-led lane allocation,

the optimized time is t_{optim}. Therefore, the first question that arises is whether a leader is necessary; or in other words, if it is possible that $t_{optim} \leq t_{greedy}$.

The next question is whether t_{optim} changes with different leader selection criteria.

In the simulations, we look into both of these questions.

II. Related Work

For traffic management, especially at intersections and lane merges, several alternatives to leader election have been explored in previous work

Leader selection is an important primitive for distributed computing, useful as a subroutine for any application that requires the selection of a unique processor among multiple candidate processors.

Depending on the network topology, several algorithms have been presented till today, chiefly the Ring Election Algorithm [5] and its variations [7, 2]. However in a dynamic network where communication links and network topology change continuously, these algorithms are not applicable.

The most promising alternatives adapt network routing algorithms such as Temporally Ordered Routing for dynamic leader selection [11]. However, even TORA fails in the case when multiple topology changes occur. One of the well known asynchronous variations is [8]. They describe a more general algorithm does a good job in situations when the network topology continuously changes. The main drawback of this approach is that it leads to some circumstances when it an agent may not have a leader at all (not even itself), opting to wait for new information instead. Our approach focuses more on ensuring that every agent has a leader and follows the leader's instructions, given that the agent's actions may have a detrimental effect on other agents' performance if it does not follow the leader's decision.

In swarms of aerial robots, Sato et al[14] and some other works propose leader selection scheme based on leader influence, and combine that with the concept of boids or programmable matter to make the agents automatically follow the leader[4]. This scales well to many agents, but only aims to keep the boid cohesed together. There is little intentional direction from the leader telling agents what to do. In our project, such an approach would not work because instead of physically following a leader, each agent needs explicit instructions on which lane to enter.

Luo et al[10] frame the leader selection problem as finding a core set which can be used to calculate the Minimum-Volume-Enclosing-Ellipsoid (MVEE) representing the swarm boundary. They even demonstrate its effectiveness with 50 robots in simulation and 10 turtlebots, with varying topologies. However, the swarm is not as dynamic as the system in our project.

III. Methods

A. Simulation

The simulation environment was built in MATLAB, and an object-oriented design was used to create individual, independent agents each with different kinematic capabilities.

Every agent had a unique *ID*, a collision sphere of influence (if another agent is within this range, the agent shall slow down to avoid collisions) *collSOI*, and a communication sphere of influence (the range within which it can establish a direct communication channel with another agent) *commSOI*. The agent had a limitation on maximum velocity *vLim*, a maximum acceleration (or deceleration, if negative) *aLim*, a *state* variable containing the current position, velocity and acceleration, a *path* that tells it which lane to enter or stay in, a *leader* attribute that contains the unique ID of the agent that the ego agent believes is its leader, a list *know* containing the information of all the other agents currently in direct communication, and finally a binary flag called *election* that tracks if an election is currently being held or not.

Each agent also has a built-in method that enables it to choose a lane based on greedy selection, and some helper methods for collision avoidance.

At initialization, all cars begin at rest. The first car in the list enters the first lane, and then at each timestep, subsequent cars wait for a random flag to turn True until they can start themselves and begin entering a lane. This is done to introduce random starting conditions into the system.

More about the simulation can be seen in section IV (Experimental Results).

B. Algorithm

The election algorithm in this project focuses on high flexibility of the criteria used to select a leader from a pool of candidate agents. We describe the election algorithm used, from the perspective of an individual agent. This can be shown in Algorithm 1.

IV. Experimental Results

At the current stage of implementation, the simulation supports three phases of transit: (1) a pre-leader selection, (2) lane selection, and (3) re-iteration. Phase (1) works to move stationary cars at the start of the queue into the allotted lanes. Once the separation between a car that has just entered the queue and the next-to-enter car is greater than the sum of their *collSOI*, the waiting

Algorithm 1: Leader selection algorithm

```
if No agent in communication then
    Set self as leader;
else
    if Election is occurring then
        Find the agent that scores the highest based
          on criteria;
        Set that agent as the believed leader;
        for Every other agent do
            if Another agent's leader has a higher
              score than own leader then
                Set that as the leader;
            end
            if You yourself become the leader then
                End election for all other agents;
            end
        end
    else
        Accept whoever the current leader is;
    end
    Ping the leader;
    if No response from leader then
        Restart election
    end
end
```

car is randomly assigned a *go_car* flag (either 0 or 1) which determines whether the car will go or not. The probability of obtaining 0 or 1 is 50%. Phase (2) is the leader's action on its own subset of the agents. In practice, this process is akin to some form of optimal path planning. For the sake of argument, we assume that the leader has completed this process the instant all cars reach the midpoint of the path, and treat the resulting leader decision as a reordering of the queue in each lane. Phase (3) is a reiteration of Phase (1) without any leader selection. This provides a way of estimating the performance of queue reordering.

We study various systems to better understand the selected leader. For the leader selection criteria, we decide to implement a popularity/knowledge based selection: the agent with the most knowledge of other agents will be the leader. Since the implementation currently only provides the chosen leader, we give an analysis on the selected leader in each case.

The inputted queue of cars can be organized based on various characteristics of the system. To intentionally cluster the system for leader selection, we organize the queue with *limiting velocity*.

A. Increasing velocity

If a queue of cars is organized such that the cars enter in order of increasing velocity, a single lane limits the velocity of every car behind the first; for this case, a small communication radius ensures that the median-positioned car will be the leader in the full system.

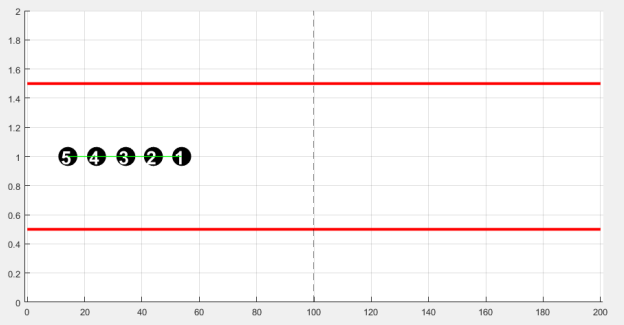

Fig. 1. One lane, increasing velocity queue with 5 agents.

This choked flow in the single lane, while detrimental to the time of travel completion, unexpectedly also allows a leader to be robustly chosen due to the length of time each agent spends with every other agent. With this information, we can make the prediction that for the multi-lane counterpart, the first agent will always be the elected leader, since it spends the most time in the system (thus establishing the most communication links) in the system.

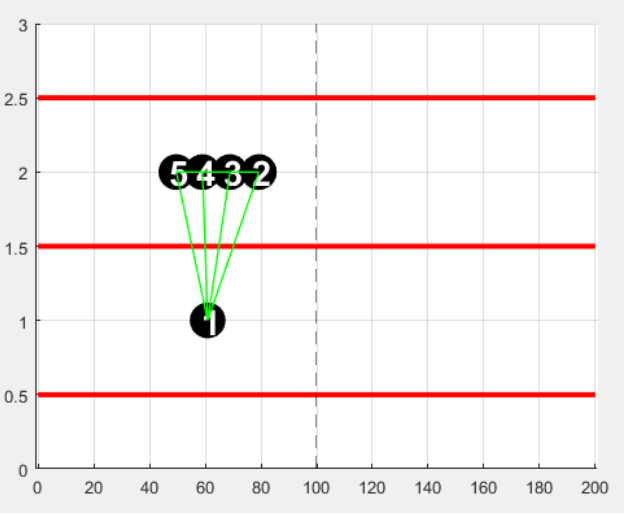

Fig. 2. Two lane, increasing velocity queue with 5 agents.

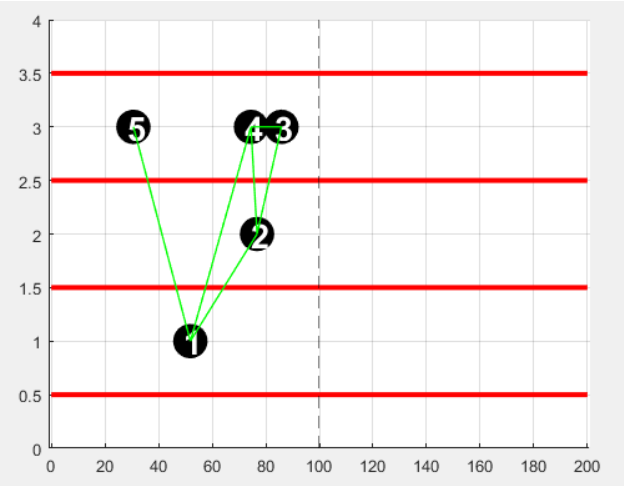

Fig. 3. Three lane, increasing velocity queue with 5 agents.

B. Decreasing velocity

If a queue of cars is organized such that the cars enter in order of decreasing velocity, a single lane does not limit velocity. This means that each car moves at its maximum speed, and communication can, at minimum, span one unit of *collSOI*.

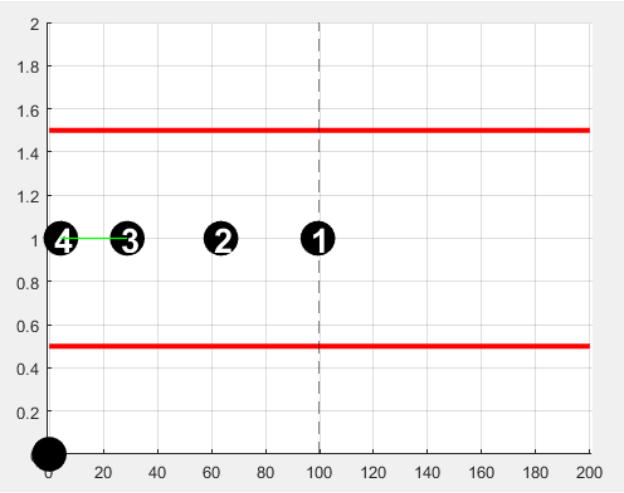

Fig. 4. One lane, decreasing velocity queue with 5 agents.

C. Random car queues

For the random car queue, a random seed is initialized for a repeatable random experiment with differently-sized lanes. The initialized cars have randomized velocity limits and acceleration limits, but all fall within a desired range. For single-lane behavior, we can expect that the system will behave as some combination of the single-lane increasing velocity and single-lane decreasing velocity systems. The problem becomes much more interesting, however with multi-lane systems:

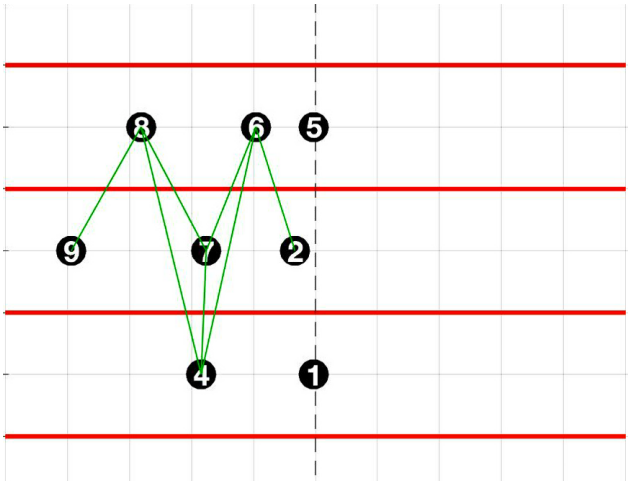

Fig. 5. Three lane, random queue with 10 agents.

With the example above of the three-lane, 10-agent system, we essentially see disconnects between agents at the extremes, but strong interconnectedness with agents 4 and 7. Systems like these are don't have easily predictable leaders because of the disorder of the queue with respect to velocity limits. In this system, the group leader was agent 7.

V. Discussion

The concept of leader selection was explored through simulation of a multi-agent system. A MATLAB simulation of the interaction between agent objects was created. The program was designed for a three-phase simulation for before, during, and after leader decision process; the work in this paper was focused around the first phase. Given "system knowledge" as the leader election criteria, we expect the agent with the most established communication links to be the elected leader. This agent, in turn, is most often either the slowest vehicle in the system or the agent that remains at the system's median-position at the longest period of time.

In order to evaluate the performance of the system, further testing is necessary through leader *lane allocation*. Once a lane allocation algorithm (sorting algorithm) is ran, a time-based performance index can be measured by comparing the ratio of before to after leader decision process times, with a large ratio representing improvement.

There were many simplifications made to the system given. One major simplification was during the leader decision process, by treating the optimal trajectory planning as a queue sorting algorithm. Since the leader lane selection will sort the vehicles in order of decreasing speed, the time-based performance index is guaranteed to be 1 at minimum, which may not always be true. To better understand the downsides of leader selection,

we thus need a more realistic way for the leader to be able to optimally plan trajectories, which may include more sophisticated algorithms involving lane-switching and model predictive control.

Robots are increasingly becoming autonomous and swarms continue to find new applications. As centralized control and traditional leader selection schemes do not scale well with highly dynamic swarms, our work paves the way for faster coordination and semi-centralized mission control in groups of robots.

ACKNOWLEDGMENT

The authors would like to thank Dr Kirstin Petersen, Assistant Professor, Cornell University. Anshuman Das did all of the coding for all simulations and experiments, and prepared the final presentation slides. Aman Agarwal did literature review, designed the algorithms and wrote the high-level pseudo code, and planned the project steps and milestones.

REFERENCES

[1] Carl Anderson, Guy Theraulaz, and J-L Deneubourg. "Self-assemblages in insect societies". In: *Insectes sociaux* 49.2 (2002), pp. 99–110.

[2] Rena Bakhshi et al. "Leader election in anonymous rings: Franklin goes probabilistic". In: *Fifth Ifip International Conference On Theoretical Computer Science–Tcs 2008*. Springer. 2008, pp. 57–72.

[3] Jan Carlo Barca and Y. Ahmet Sekercioglu. "Swarm robotics reviewed". In: *Robotica* 31.3 (2013), 345359. DOI: 10 . 1017 / S026357471200032X.

[4] Davide Canepa and Maria Gradinariu Potop-Butucaru. "Stabilizing flocking via leader election in robot networks". In: *Symposium on Self-Stabilizing Systems*. Springer. 2007, pp. 52–66.

[5] Ernest Chang and Rosemary Roberts. "An improved algorithm for decentralized extrema-finding in circular configurations of processes". In: *Communications of the ACM* 22.5 (1979), pp. 281–283.

[6] Iain Couzin et al. "Effective leadership and decision-making in animal groups on the move". In: *Nature* 433 (Mar. 2005), pp. 513–6. DOI: 10.1038/nature03236.

[7] Randolph Franklin. "On an improved algorithm for decentralized extrema finding in circular configurations of processors". In: *Communications of the ACM* 25.5 (1982), pp. 336–337.

[8] Rebecca Ingram et al. "An asynchronous leader election algorithm for dynamic networks without perfect clocks". In: *The Proceedings of the International Symposium on Parallel and Distributed Processing*. Citeseer. 2009.

[9] Andrew J. King, Dominic D.P. Johnson, and Mark Van Vugt. "The Origins and Evolution of Leadership". In: *Current Biology* 19.19 (2009), R911 –R916. ISSN: 0960-9822. DOI: https://doi.org/10.1016/j.cub.2009.07.027. URL: http://www.sciencedirect.com/science/article/pii/S0960982209014122.

[10] Wenhao Luo et al. "Asynchronous distributed information leader selection in robotic swarms". In: *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2015, pp. 606–611.

[11] Navneet Malpani, Jennifer L Welch, and Nitin Vaidya. "Leader election algorithms for mobile ad hoc networks". In: *Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*. ACM. 2000, pp. 96–103.

[12] Nathan J Mlot, Craig A Tovey, and David L Hu. "Fire ants self-assemble into waterproof rafts to survive floods". In: *Proceedings of the National Academy of Sciences* 108.19 (2011), pp. 7669–7673.

[13] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. "Programmable self-assembly in a thousand-robot swarm". In: *Science* 345.6198 (2014), pp. 795–799. ISSN: 0036-8075. DOI: 10.1126/science.1254295. eprint: https://science.sciencemag.org/content/345/6198/795.full.pdf. URL: https://science.sciencemag.org/content/345/6198/795.

[14] Hiroshi Sato, Masao Kubo, and Akira Namatame. "Adaptive Leader Selection based on Influential Individuals". In: *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*. ICST (Institute for Computer Sciences, Social-Informatics and. 2016, pp. 367–370.

[15] Radka rov et al. "Graded leadership by dominant animals in a herd of female beef cattle on pasture". In: *Animal Behaviour* 79.5 (2010), pp. 1037 –1045. ISSN: 0003-3472. DOI: https://doi.org/10.1016/j.anbehav.2010.01.019. URL: http://www.sciencedirect.com/science/article/pii/S0003347210000412.

APPENDIX

The open source code can be found at: https://github.com/ad765/leaderSelection