Decentralized Reinforcement Learning

Dhanush Giriyan, Computer Science
Mentor: Andrea W. Richa, President's Professor
School of Computing and Augmented Intelligence

Introduction & Motivation

Distributed computing is an approach to solving problems using multiple processors that work together to achieve a collective goal while restricting inter-system communication based on their locality. Often, these distributed computing problems are also optimization problems that can only be solved using approximate methods. **Multiagent Reinforcement Learning (MARL)** is a powerful class of algorithms that provides (sub-)optimal solutions to such multiagent problems. Existing MARL algorithms rely on a centralized server to perform all major computations, making them impractical to use in cases where no such centralized server is feasible.

Where could such an instance occur?

Mine fields: during war; the exact location of each mine is unknown, and no centralized system exists that can find them all with a single look…

This work proposes an algorithm called Decentralized Multiagent Rollout that answers the following question:

Can Multiagent Reinforcement Learning algorithms be implemented in a distributed manner?

Problem Description

This problem is a specific instance of the Vehicle Routing Problem (VRP). Our problem takes place on a grid world where each grid cell has at most four neighboring cells, while in the general VRP each cell can have an arbitrary number of neighboring cells.

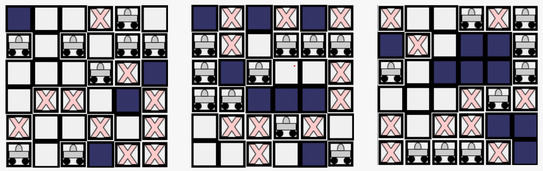
Environment: A grid world where each cell can either be free or a wall (obstacle).

Tasks: These are scattered uniformly at random across the free cells of the environment. They are stationary. A task is said to be solved when an agent arrives at its grid cell.

Agents: The entities that live on the grid world. During each time step, the agents move from one free grid cell to an adjacent one (no diagonal moves) or wait in their current cell. An agent can only (directly) see tasks and other agents in its **local view**.

View: (of an agent) covers all the free grid cells within *k*-hops of the agent's current position; *k* is an input parameter required at initialization.

Objective: Minimize the number of steps required to collectively solve all tasks.



Decentralized Multiagent Rollout

The Multiagent Rollout algorithm as described in [1] involves sequentially computing controls or actions for each agent in the environment while minimizing an estimate of the number of steps required to solve the problem instance from the current arrangement of tasks and agents. This estimation is done by using a heuristic; in our case we use the shortest path heuristic. The required computations to determine an action for each agent are completed on a central server to which each agent is connected. Our work shows that one can capitalize on the strengths of Multiagent Rollout while simultaneously abiding by the locality constraints. We call this algorithm Decentralized Multiagent Rollout. This implementation not only permits the use of Multiagent Rollout in cases where such locality constraints must be maintained but also provides a massive speed-up in the runtime of the algorithm!

Decentralized Multiagent Rollout has the following three phases:

Self Organizing Agent Clusters (SOAC)

In SOAC, the agents segregate themselves into clusters centered around the task vertices while using only local information. Agents that are close to task vertices become representatives of these clusters and agents that see these representatives in their view greedily add themselves to the representative's cluster. In case multiple agents see the same task, we perform a round-robin-based leader election to ensure that the representative agent of each cluster is a unique agent. Further, we limit the number of agents in a cluster using a parameter ψ. In our simulator, clusters are identified with colors as shown in the figures below.

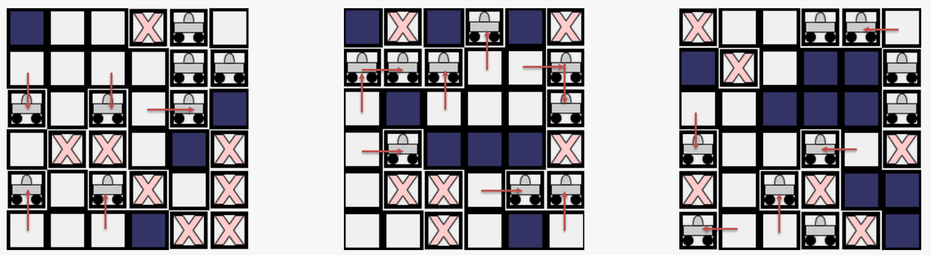
Local Map Aggregation (LMA)

The aim of this phase of the algorithm is to provide each agent in a cluster with the view of the whole cluster, i.e., the union of the views of each agent in the cluster. As a result of the previous phase, we obtain an agent tree for each cluster rooted at the cluster's representative. We traverse this tree using a depth-first search and, while doing so, pass the local view of each agent, starting with the representative agent all the way down to the leaf agents and then back up to the representative agent. This results in the representative agent obtaining an internal representation of the view of the whole cluster.

Cluster-Aided Multiagent Heuristic Rollout (CAMAHR)

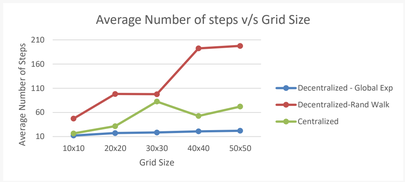
This phase applies the Multiagent Rollout as described in [1] to each cluster. Starting from representative agent we compute an action for each agent and pass the computed action to the next agent in the agent tree (again, given by the depth-first search order) for the cluster. The agents in a cluster coordinate their moves with each other due to this passing of moves. Once every agent in a cluster has computed an action to take, we assign an exploratory action to any agent that was never assigned any cluster. This is possible if an agent does not see any tasks or other agents that are already part of a cluster in its local view. After this is completed, all the agents simultaneously take the action that was assigned to them. Agents then remove themselves from any cluster they may have been a part of, and the process resumes from the SOAC phase until no tasks remain.

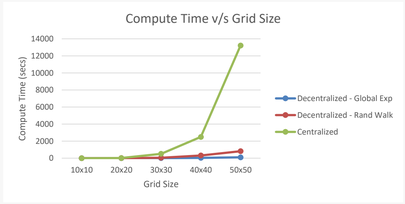
The following image shows the result a single round of the above the three phases for the three instances shown earlier (red arrows point the moves taken by agents):

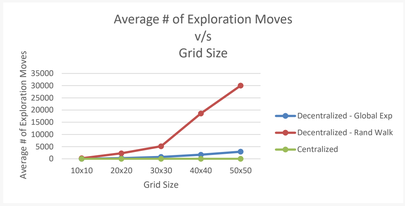


Results

The following graphs compare the performance of Decentralized Multiagent Rollout with Centralized Multiagent Rollout based on the total number of steps taken to solve the problem instance and compute time averaged over ten randomly generated instances of different grid sizes.







Future Work & References

- **Physical Experiments:** We intend to use the robots to further test our algorithm in a more realistic setting.
- **Dynamic Tasks**: Consider the possibility where new tasks are added to the problem instance as old tasks are still being solved.

Reference:

[1] Bertsekas, Dimitri P. Multiagent Rollout Algorithms and Reinforcement Learning. arXiv preprint arXiv:1910.00120, April 2020.

FURI

Ira A. Fulton Schools of Engineering
Arizona State University