```
   ____     ____       _          _        _
U /"___|U | __")uU  /"\  u U  /"\  u  |"|
\| | u   \|  _ \/ \/ _ \/   \/ _ \/ U | | u
 | |/__   | |_) | / ___ \   / ___ \  \| |/__
  \____|  |____/ /_/   \_\ /_/   \_\  |_____|
 _// \\  _|| \\_  \\    >>  \\    >>  //  \\
(__)(__)(__) (__)(__)  (__)(__)  (__)(_")("_)
  CONSENSUS-BASED AUCTION ALGORITHM LIBRARY
            by Pierre LAGUE
```

> Simulation of the CBBA algorithm on a ring topology network. Multi-Task Assignement

> Simulation of the CBAA algorithm on a fully connected topology network. Single-Task Assignement

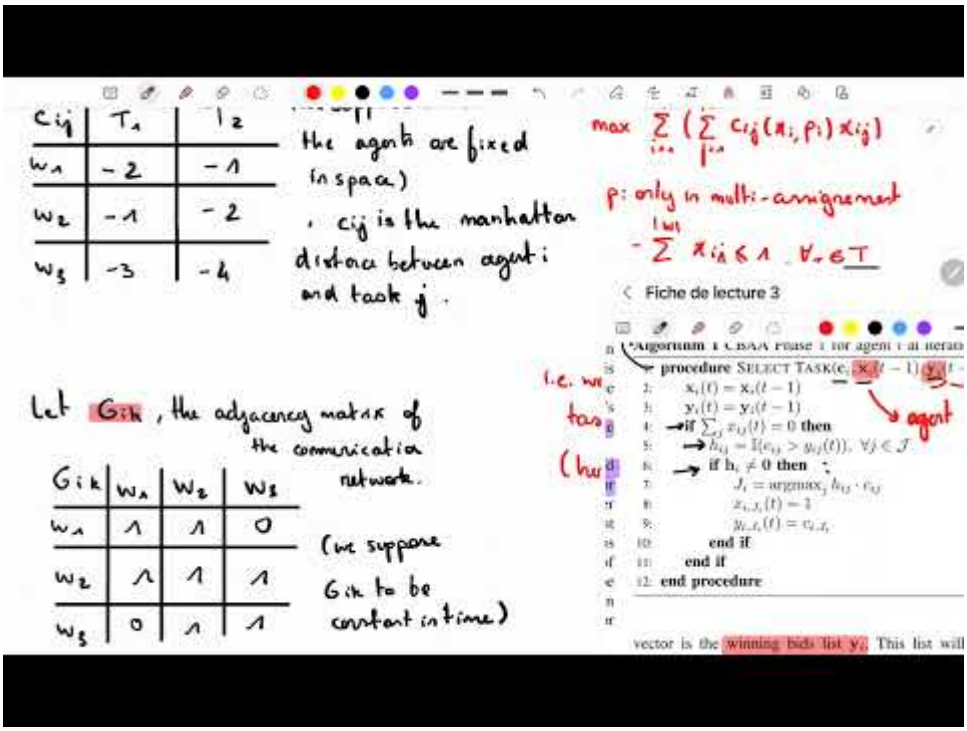
CBAAL - Consensus Based Auction Algorithm Library

Overview

CBAAL is a library developed as part of my end-of-year assignment for my first-year master's degree. It focuses on implementing consensus-based auction algorithms (CBAA) and consensus-based bundle algorithm (CBBA) primarily for dynamic load balancing problems. The algorithms implemented in this library are inspired by the publication of Choi et al., which introduced these innovative approaches to solving load balancing challenges in dynamix environments. 📑💼

CBAA - Consensus Based Auction Algorithm - Single Task Assignement

Video : Breaking down the CBAA



According to Choi et Al., the consensus-based auction algorithm (CBAA) is a single assignment strategy that makes use of both auction and consensus. The algorithm consists of iterations between two phases. The first phase of the algorithm is the auction process, while the second is a consensus algorithm that is used to converge on a winning bids list. By iterating between the two, CBAA can exploit convergence properties of decentralized consensus algorithms as well as the robustness and computational efficiency of the auction algorithms.

Phase 1 : The Auction process

In this phase, each agents places a bid on a task asynchronously with the rest of the fleet. Let c_{ij} be the bid that is placed on a task. In CBAAL, we consider this bid to be the euclidean distance between an agent and a task. Initially multiple agents may bid on the same task. In this phase, two vectors of lenght N_t are also defined (N_t being the number of tasks in the simulation). They are both initialized as zero vectors.

- the first vector is x_i, that is the agents task list. If $x_{ij}=1$, agent i has been assigned to task j, and 0 otherwise.
- the second vector is y_i, that is the winning bid list. It is an up-to-date estimate of the highest bid made for each task in the simulation.

After the definition of these two vectors, the valid tasks list h_i is defined as $$h_{ij} = \mathbb{1} (c_{ij} > y_{ij}), \forall j \in J$$
An unnasigned agent i will compute the valid task list h_i. If there are valid tasks, it then select the task $j*$ giving it the maximum score based on the current winning bid list, and updates its task list x_i and winning bid list y_i accordingly. In the case where an agent has already been assigned a task, this auction phase is skipped and the agent jumps to phase 2 : Consensus Process.

Phase 2 : Consensus Process

The second phase of the CBAA is the consensus section of the algorithm. Here, agents make use of a consensus strategy to converge on the list of winning bids, and use that list to determine the winner. This allows conflict resolution over all tasks while not limiting the network to a specific structure.

The simulation uses a network represented as a graph and its adjacency matrix. The network $\mathbb{G}(\tau)$, is the undirected communication network at time τ with a symetric adjacency matrix $g(\tau)$. This adjacency matrix is defined such that $g_{ik}=1$ if a link exists between agents i and k, and 0 otherwise. Agents i and k are said to be *neighbors* if such a link exists. For the implementation and as convention, each agent is a neighbor to himself (self connection nodes, $g(\tau)_{ii}=1$).

At each iteration of phase 2, each agents receives, and transmits their winning bid list y_i and the consensus is reached in a way that each agent i replaces its y_{ij} values with the largest value between itself and its neighbors. Also, if an agent is outbid, it looses it's assignement and jumps back to phase 1.

CBBA - Consensus-Based Bundle Algorithm - Multi-Task Assignement

Choi et al. define CBBA as an extension to the single-assignment problem. In CBBA, each agent has a list of tasks potentially assigned to itself, but the auction process is done at the task level rather than at the bundle level. Similar to CBAA, CBBA consists of iterations between two phases – bundle construction and conflict resolution.

Phase 1 : Bundle Construction

During phase 1 of the algorithm, each agent continuously adds tasks to its bundle until it is incapable of adding any others. The tasks are added into the bundle in the following way, until it has reached a final condition.

- each agent carries two types of lists of tasks both of cardinalty inferior to L_t, the maximum assignement number:
    - the bundle b_i, tasks are ordered based on which ones were first added.
    - the path p_i, tasks are ordered based on their location in the assignement.

The authors define $S_i^{p_i} the total reward value for the agent i performing the tasks along the path p_i. They also mention the notion of marginal score improvement when a task is added to the bundle, which we will not discuss here. In the end, each agent carries four vectors:

- a winning bid list $y_i \in \mathbb{R}_+^{N_t}$
- a winning agent list $z_i \in \mathbb{I}^{N_t}$
- a bundle $b_i \in (\mathbb{J} \bigcup {\emptyset })^{L_t}$
- a path $p_i \in (\mathbb{J} \bigcup {\emptyset })^{L_t}$

If a task j is added to the bundle, it incurs the marginal score improvement of

- $c_{ij}[b_i]$ = 0 if the task j is already in the bundle b_i.
- $c_{ij}[b_i] = max_{n \leq |p_i|} S_{i}^{p_i \oplus *n{j}}* - *S*{i}^{p_i} \text { otherwise }$

In other words, the CBBA scoring scheme inserts a new task to the location tha incurs the largest score improvement **after** the task was inserted in the bundle vs before it was inserted, and this value becomes

the marginal score associated to that task given the current path. Naturally, if the task is already in the path, then it brings no improvement in score.

The score function is initialized as $S_i^{{\emptyset}}$ = 0, while the path p_i and bundle b_i are recursively updated as

$$b_i = b_i \oplus _{end} {J_i}, p_i = p_i \oplus _{n_i, J_i} {J_i}$$
where :

- $J_i = argmax (c_{ij[b_i]} \times h_{ij})$ -> the index of the task that maximizes the score improvement.
- $n_{i,J_i} = argmax_n S_i^{p_i \oplus _n {J_i}}$ -> the index of at which the task to be added maximises the marginal score.
- $h_{ij} = \mathbb{I}(c_{ij} > y_{ij})$ the valid tasks list, as defined earlier.

The recursion continues until one of the following conditions is reached :

- either $|b_i| = L_t$ -> reached maximum assignement number
- or $h_i = 0$ -> the number of valid taks is 0

In CBBA, each agent needs information about not only whether or not it is outbid on the task it selects but also who is assigned to each task; this enables better assignments based on more sophisticated conflict resolution rules.

Phase 2 : Conflict Resolution

In CBAA, agents bid on a single task and release it upon receiving a higher value in the winning bids list. On the contrary, in CBBA, agents add tasks to their bundle based on their currently assigned task set. Suppose that an agent is outbid for a task and thus releases it; then, the marginal score values for the tasks added to the bundle after this task are no longer valid. Therefore, the agent also needs to release all the tasks added after the outbid task. Otherwise, the agent will make further decisions based on wrong score values, which may lead to poor performance.

In the multi-assignment consensus stage, three vectors are communicated for consensus. Two were described in the bundle construction phase: the winning bids list y_i Nt and the winning agent list z_i. The third vector $s_i \in \mathbb{R}^{N_u}, with N_u the number of agents, represents the time stamp of the last information update from each of the other agents. Each time a message is passed, the time vector is populated with

- $s_{i k}= \tau_r \text { if } g_{i k}=1$
- $s_{ik}= max_{m: g_{i m}=1} s_{m k}\text { otherwise }$

where $\tau _r$ id the message reception time. When agent i receives a message from another agent k, z_i and s_i are used to determine which agent's information is the most up-to-date for each task. There are three possible actions agent i can take on task j :

- update: $y_{ij} = y_{kj}$, $z_{ij}=z_{kj}$ -> the new task added is the one kept
- reset: $y_{ij} = 0$, $z_{ij} = \emptyset $
- leave: $y_{ij} = y_{ij}$, $z_{ij} = z_{ij}$ -> the agent doesn't take the new task

This notion of belief is important but adds more complexity to the problem. The authors define 17 decision rules based on the situation an agent might find itself in. If a bid is changed by the decision rules in Table I, each agent checks if any of the updated or reset tasks were in their bundle, and if so, those tasks, along with all of the tasks that were added to the bundle after them, are released. From here, the algorithm returns to the first phase and new tasks are added

Features of the project

- Implementation of CBAA and CBBA algorithms (Centralized/Sequential implementation using object oriented model).
- Designed for dynamic load balancing scenarios.
- Provides a framework for simulating load balancing strategies over various network topologies.
- Well-documented codebase for easy understanding and modification.
- **Work in progress** : implementation of the algorithms using the actor programming model for decentralized/asynchronous execution 🛠️📝

Installation

1. Clone the repository:

```
git clone https://github.com/yourusername/CBAAL.git
```

2. Navigate to the cloned directory:

```
cd CBAAL
```

3. Install dependencies:

```
pip install -r requirements.txt
```

Execution

The program `CBAAL.py` can be executed in command line with fully customizable parameters like so :

```
 python .\CBAAL.py cbaa D --nb_agents 20 --nb_tasks 20 --max_t 100 --topology 2
--v
```

The parameters are defined as follows:

Parameter	Description	Type
algorithm	Algorithm to execute (CBAA/CBBA)	string

Parameter	Description	Type
method	Method (C/D - Centralized/Decentralized)	string
--nb_agents	Number of agents in the simulation	integer
--nb_tasks	Number of tasks in the simulation	integer
--max_t	Maximum iteration	integer
--topology	Network topology [(1)star, (2)fc, (3)ring, (4)mesh, (5)random]	integer
--viz	If set to 'gui', displays the GUI viz, if not, the plot viz	string
--v	If set to true, adds verbose output, else it doesn't	boolean

Documentation

Detailed documentation can be found in scripts of the repository. This includes explanations of the algorithms, and relevant remarks about the code and theory behind the algorithms. 📖🔍

Contribution

Contributions to the project are welcome! If you find any bugs or have suggestions for improvements, please open an issue or submit a pull request on the GitHub repository. 🚀👥

License

This project is licensed under the MIT License. See the `LICENSE` file for more details. 📜

Acknowledgements

We would like to express our gratitude to Choi et al [https://dspace.mit.edu/bitstream/handle/1721.1/52330/Choi_Consensus-Based-Decentralized.pdf?sequence=2] . for their pioneering work in consensus-based auction algorithms, which served as the inspiration for this project. 🙏

Contact

For any inquiries or questions regarding the library, please contact [pierre.lague @ protonmail.com]. 📧