Cooperative solutions to exploration tasks under speed and budget constraints

Karishma & Shrisha Rao

JOURNAL OF SIMULATION
2023, VOL. 17, NO. 6, 676–687
https://doi.org/10.1080/17477778.2022.2043792

RESEARCH ARTICLE

Cooperative solutions to exploration tasks under speed and budget constraints

Karishma and Shrisha Rao

International Institute of Information Technology - Bangalore, Bangalore, India

ABSTRACT
We present a multi-agent system where agents can cooperate to solve a system of dependent tasks, with agents having the capability to explore a solution space and make inferences, as well as query for information under a limited budget. Re-exploration of the solution space only happens when an older solution expires. We investigate the effects of task dependencies, increasing the speed of the agents, the complexity of the problem space, and the query budgets available to agents. Specifically, we evaluate trade-offs between the agent's speed and query budget. We observe that increasing the speed of a single agent improves the system performance to a certain point only and increasing the number of faster agents may not improve the system performance due to task dependencies. Favouring faster agents during budget allocation enhances the system performance, in line with the "Matthew effect".

ARTICLE HISTORY
Received 14 April 2021
Accepted 10 February 2022

KEYWORDS
Task exploration; cooperative agents; resource constraints; multi-agent system

1. Introduction

Many applications like military concept development (Cares, 2002), battlefield intelligence (Hongwei et al., 2010; Ismail et al., 2018), health care and medical diagnosis systems (Gupta & Pujari, 2009), etc. distribute tasks to achieve the goal(s). Tasks are distributed based on the agent's capabilities, and not all the agents need to get tasks of the same complexity. In the case of a complex task, an agent may seek external help as well.

There are also fundamental trade-offs involved between computation and communication (Li et al., 2018), as also seen in high-performance computing (HPC; Xiao & Peng, 2019), where in some contexts, it is better to compute a solution locally, and in others, to fetch a solution stored elsewhere. The same sort of trade-off can also be seen in cloud robotics (Salmerón-Garcı et al., 2015) and in 5 G mobile networks (Eramo et al., 2016).

The Matthew effect is also well known to exist in various forms in various settings (Rigney, 2010). However, until now, there has not been any satisfactory simulation of the same in a broader context that transcends the specific features of particular domains although attempts have been made to simulate it in specific settings, such as scientific peer review (Squazzoni & Gandelli, 2012) and computational social systems (J. J. Zhang et al., 2021).

More generally, simulation is well known to be a useful technique to understand trade-offs and other aspects involved in resource utilisation strategies (Dear & Sherif, 2000; Wilsdorf et al., 2019) and to better understand how ranking and selection may be made (Waeber et al., 2012). However, there has not, until now, been a study of the issues involved in how trade-offs between speed and budgets may affect the choices made.

Exploration by multiple agents has been known to particularly be important in the context of multi-robot exploration (Burgard, Moors, Stachniss & Schneider, 2005), which continues to offer interesting problems for research (Viseras et al., 2020). However, the sorts of problems that are addressed in this work have not hitherto been addressed at all. Simulations of multi-robot systems have likewise not dealt with them (Choi et al., 2021; Dawson et al., 2010).

Network traffic flow evolution (Wang et al., 2019), waste collection management (Gruler et al., 2017), discrete event systems of wireless networking (Tavanpour et al., 2020), efficient disaster management (S. Lee et al., 2022), etc. are domains where the impact of cooperation and collaboration is studied using simulation. In such applications, cooperation may be required among communities or individuals in the society, but collaboration may increase the complexity.

In this work, we identify the fundamental problem of solving a set of tasks cooperatively by a set of agents, which can directly explore a solution space (to represent local computation) or can query an oracle (to represent bandwidth usage or offloaded computation), subject to a query budget. The agents can also infer some new solutions in line with previously known solutions and can share their solutions with other agents. Tasks have dependencies and need to be worked in an order

CONTACT Karishma karishma@iiitb.ac.in International Institute of Information Technology, 26/C Electronics City, Hosur Road, Bangalore 560003, India

specified by a program graph. In this setting, we formulate and answer the following types of questions:

(1) If there is a choice between agents with greater speed or more query budget, which should be preferred and why?

(2) In a system of dissimilar agents operating at different speeds, how should a fixed small budget be shared among them so that the overall system performance is the best possible?

2. Related work

A multi-agent system (MAS) contains multiple agents to solve complex problems by subdividing them into smaller tasks. Agents act autonomously to make wise decisions based on their intelligence and experience (Dorri et al., 2018). In a MAS, where each agent is assigned a local task with requirements, an agent may require multiple agents' collaboration with a coordination strategy, if needed (Guo & Dimarogonas, 2017). Interactions between tightly coupled MASs are one of the effective means to gather the partially observable information, while coordination policies among loosely coupled agents are still a big challenge (M. Liu et al., 2020). Multi-agent cooperative behaviour can occur in a dynamic environment as well (Xu & Yang, 2009), where a multi-agent cooperative processing model performs cooperative work to process tasks quickly and efficiently. MAS can control several aspects of smart grids like management of energy, scheduling energy, reliability, the security of the network, fault handling capability and communication between agents (Mahela et al., 2020).

Many real-time complex systems contain task execution dependencies (Lu, Nolte, Kraft et al., 2010), data dependencies (Ndoye & Sorel, 2013) and shared resources dependencies (Shi et al., 2019). Dependencies among tasks also need to be noted in scheduling tasks on a system of machines where the total energy consumed by the system is to be reduced. Different heuristic approaches exist for this although the task of energy minimisation is known to be NP hard (Agrawal & Rao, 2014). Thus, scheduling the task sets needs to be aware of the dependencies (David et al., 2001). A dependency graph is one of the optimal approaches to represent task dependencies (Shi et al., 2019). Execution dependencies arise among an embedded program's tasks due to task priority, task precedence and inter-task communication (Lu, Nolte, Bate et al., 2010; Yang et al., 2019).

Task dependency exists during the multi-task allocation in various applications like complex mobile crowds (Yang et al., 2019) and distributed computing (Y. C. Lee & Zomaya, 2011). Unhandled dependency can cause high latency and allocation errors and even bring the system into a wrong state. Thus, a dependency-aware task scheduling approach is required to obtain accuracy and efficiency.

The table scheduling algorithm and scheduling algorithm based on task replication can be used to schedule the dependent task in a distributed system (Qin et al., 2018). The table scheduling algorithm is simple in design and low in complexity, whereas scheduling algorithm based on task replication uses the backtracking methods for task scheduling. Due to that, the time complexity is high and the solution space is quite large.

The clustering scheduling algorithm usually divides the tasks into smaller clusters and merges the cluster after completion. Existing examples of clustering algorithms are EZ, DSC, LC and MD (Topcuoglu et al., 2002). We have used the priority-based scheduling algorithm with dependency constraints to eliminate the high time complexity.

A task scheduling algorithm with resource attribute selection utilises the resource efficiently by selecting the optimal node to execute a task (Y. Y. Zhao et al., 2014). The same has not considered the choice between the resources.

Task allocation is a crucial problem for agents' cooperation in multi-agent systems. A distributed and self-adaptable scheduling algorithm can adapt to the task arrival process on itself, considering the influence from task flows on other agents (Ghassemi et al., 2019).

Dominant Resource with Bottlenecked Fairness (DRBF) is a multi-resource fair allocation mechanism to improve resource utilisation under well-studied fairness constraints (L. L. Zhao et al., 2018). We have evaluated the agent's performance with and without fair allocation of the resources, which is useful when efficient system performance is required instead of fairness.

Multi-agent MDP is a popular method for solving sequential optimisation, decision-making, and learning problems in an uncertain environment where the outcome depends on the previous actions (Mukhopadhyay & Jain, 2001). The presence of uncertainty regarding agent states and actions can lead to performance issues. Policy iteration (PI) and value iteration (VI) are the standard techniques to solve an MDP within large action spaces (Ashutosh et al., 2020), where the iteration complexities increase with the number of controls (Fiscko et al., 2021; Littman et al., 2013).

A partially observable Markov decision process (POMDP) is an agent decision process for uncertainties in the planning problem (Hubmann et al., 2018). A POMDP's policy is a mapping from the observations (or belief states) to actions. However, the application of POMDPs has been minimal for a long time because of the enormous dimensionality and history

(F. Liu & Liu, 2018). Point-based methods (Kurniawati et al., 2008; Shani et al., 2013) use heuristic methods to find the search space and improve computational efficiency (Z. Z. Zhang et al., 2014). Many point-based approximate value iteration algorithms evaluate a value function to update the estimated set of belief points (Vlassis & Spaan et al., 2004). Subsequently, the exploration proficiency remains to be improved, particularly when managing large-scale POMDP applications.

The method presented here does not have issues related to uncertainty regarding agent states similar to MDP because an agent's current task exploration is independent of its previous exploration and does not bring about an increase in dimension and history. Likewise, it is advantageous in comparison with the POMDP, as it does not require any value function evaluation on account of being independent of any set of belief points. Thus, it is feasible for larger applications with longer run times.

3. Methodology

We present a model for a multi-agent system having cooperative agents where tasks have some dependency structure among them and are assigned to the system. We evaluate the solution exploration under speed and budget constraints.

3.1. System specification

Each task has a variable reward and a dependency list of other tasks on which it is dependent. A task cannot be scheduled for exploration until all the tasks in its dependency list get explored first. Thus, we form a subset of tasks whose dependency lists are empty (all tasks on which these tasks are dependent have already been completed). Second, we prioritise tasks from this subset based on their associated rewards where a task with a higher value of reward gets greater priority.

Later, the tasks are distributed among the available agents in the system. An agent can explore the solution space for an assigned task and also collect inference data for future reference, which are stored in its knowledge base. The advantage of the inference data is that if an agent gets a task that can be performed using prior inference data, then the solution space exploration is not required. A solution provided by an agent is validated, and a reward is given to an agent based on the validation outcome.

Our model has also considered complex tasks that an agent is not able to explore by itself, and in this scenario, it can ask for help from an oracle by making a query. Query utilisation is limited as per the allocated budget, which may be either shared or individual. A budget available to an agent being greater than the number of unaccomplished tasks can eliminate the need for exploration.

When an agent gets a task assigned that belongs to its knowledge set, it can accomplish the same quickly; however, over a while, the same solution may no longer be valid, then an agent explores again and updates its knowledge set. Thus, the system is capable of adapting to dynamic changes in the environment.

Agents are cooperative by sharing their knowledge with others and can vary in terms of speed. A cooperative faster agent is capable of exploring the solution space and collecting inference data faster as well. Shared knowledge from a faster agent in the early phase can improve the performance of others as well.

For the experiments, we generated random mazes with random target locations. An agent traverses the maze for the assigned task, which corresponds to the exploration of the solution space. If an agent fails to reach the solution in the generated maze, then it may query an oracle if it has a budget available. The oracle provides a hint to explore the task in a maze instead of providing the exact solution. After receiving a hint from an oracle for the task, an agent explores the maze again and finds the solution.

Once an agent explores the solution, it also checks if the same target location may contain solutions for other possible tasks as well. If so, it stores this information as inference data. Thus, with each current task solution found by exploration, an agent also collects inference data.

Solution exploration is performed on a maximum 400×400 maze size by multiple agents in parallel. The obtained results show that, as may be expected, over a while, agent's knowledge increases and improves performance by reducing the exploration time for a task.

We consider the dependencies between tasks by way of program graphs G_{40} and G_{18} (Zomaya & Lee, 2012). For each task in the program graph, there is a target location defined in a maze. It is possible that multiple task solutions are available at the same maze location. If a task has a dependency on others as per the program graph, an agent can only attempt the task, by exploration, inference, or query, if the prior tasks are already completed.

We also evaluated the system exploration, by increasing the speed of a single agent where the faster agent explores the solution and collects inference data in less time. The knowledge shared by the faster agent can help other agents with their assigned tasks. However, the experimental results show that a faster agent improves the system performance to a certain point only due to task dependencies (see Figure 6(a)).

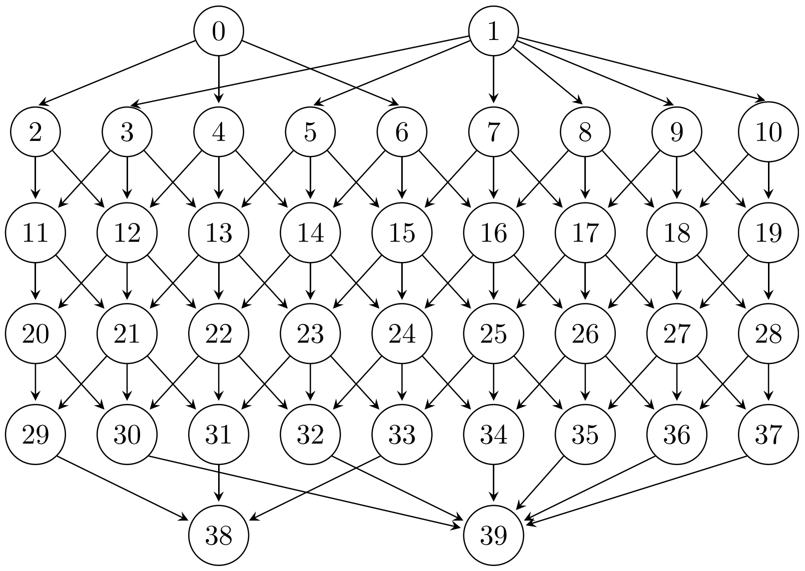

Figure 1. Task dependency graph G_{40}.

For the experiments, we have considered two types of dependent systems: a less-dependent system given by program graph G_{18} (see, Figure 2) and a highly dependent one described by G_{40} (see Figure 1). For a highly dependent system, a few faster agents do not have a significant impact on the average system exploration time, but rather cause an increase in the waiting time (see, Table 3).

We evaluated a trade-off between the number of faster agents and query budget for highly dependent (G_{40}) and less-dependent (G_{18}) systems (see Table 4). Our results show that

(1) It is better to increase the budget for a faster agent instead of increasing the number of faster agents for a less-dependent system.

(2) It is better to increase the number of faster agents in the system instead of increasing the budget for a highly-dependent system.

The experimental findings cover these points for the advantageous utilisation of faster agents vs. high budget. It is also seen that in the case of a limited total budget, favouring faster agents during budget allocation improves the performance of the system (see, Table 5) in line with the "Matthew effect" where the rich get richer and the poor get poorer (Merton, 1968).

3.2. Model specification

We consider a standard model of n agents in a system A that is required for m tasks. An agent $a_i \in A$. Tasks are formalised as a 3-tuple (T, R, D) where

- $T = \{t_1, t_2, t_3, \ldots, t_m\}$ is a set of indivisible tasks,
- $R = \{r_1, r_2, r_3, \ldots, r_m\}$ is a set of respective rewards, and
- $D = \{d_1, d_2, d_3, \ldots, d_m\}$ is a set of respective dependencies, where $d_i \subseteq T \backslash \{t_i\}$ is the set of tasks on which t_i is dependent.

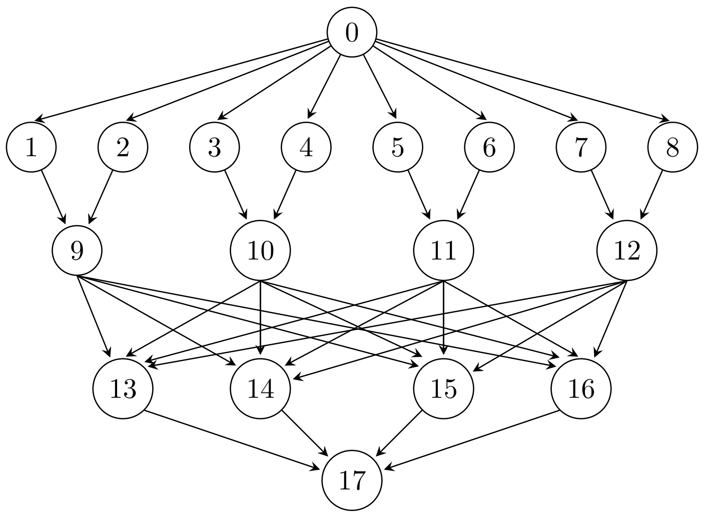

Figure 2. Task dependency graph G_{18}.

A task assignment λ is a function $\lambda : A \rightarrow 2^T$,which indicates that a subset of tasks from T is assigned to each a_i. We also require that $\lambda(a_i) \cap \lambda(a_j) = \emptyset$, if $i \neq j$, so task assignments to different agents are non-overlapping.

$\mu(a_i)$ is the set of tasks accomplished by a_i, with $\mu(a_i) \subseteq \lambda(a_i)$. If $\mu(a_i) = \lambda(a_i)$, then a_i is successful with all tasks assigned; else, it leaves some undone.

a_i can take help from an oracle by making a query. Each query to the oracle deducts a constant amount from the allocated budget B, which is a shared resource among the agents. The oracle's help is restricted based on available budget, and exhaustion of available budget can lead to failure of solution space exploration.

There is a set $\mathcal{S}_j$ of possible solutions for task t_j. An agent a_i possesses a knowledge set $K(a_i)$ as key-value pairs, where t_j is a key and some specific $s_j \in \mathcal{S}_j$ is a value. The same holds for inference data as well, so for each inferred solution s_k, some task t_k is a key, and the value is an element s_k of $\mathcal{S}_k$. After a successful solution exploration for t_j, a_i adds the newly explored solution s_j and possibly inference data for the task to its knowledge set $K(a_i)$. $K(a_i) \leftarrow K(a_i) \cup \{(t_i, s_i)\} \cup \{(t_k, s_k)\}$. An agent a_i re-explores the solution for a task t_j if an available solution in $K(a_i)$ becomes invalid due to changes in the environment. An agent a_i shares its knowledge with all the other agents.

4. Cooperative exploration strategy

This section presents the details of the exploration strategy. Algorithm 1 describes the task scheduling, solution validation, and update in a knowledge set among n agents. Algorithm 2 filters out a set of available tasks for the solution space exploration by considering respective dependencies and rewards. Algorithm 3 describes the solution space exploration process by an agent.

In the algorithms, n_e is an integer having the count of available agents for solution space exploration, which is initially equal to n. The difference between n and n_e gives the count of agents that are busy in solution space exploration. T_e is the subset of T containing the filtered tasks, which do not have any unaccomplished dependency. T_e is used for task scheduling. R_e is a set of rewards for T_e. m_e is an integer giving the length of T_e, and $\mathcal{I}$ is a set that contains the inference data in key-value pairs where the inferred solution (s_k) is a value and task(t_k) is a key.

Algorithm 1 Solution Space Exploration Algorithm

Input: T: A set of tasks, R: A set of respective rewards, D: A set of respective dependencies, n: Number of Agents

Output: Knowledge Sets computed for all the agents

```
1: $n_e \leftarrow n$
2: // Get the independent set of tasks for exploration
3: $T_e \leftarrow getAvailTasks(T, R, D, n_e)$
4: // Assign the tasks to available agents
5: $taskAssignment(T_e, n_e)$
6: while true do
7:    // On receive event listener
8:    $onSolnCheckMessage()$
9:    $t_j, s_j \leftarrow$ response from an agent $a_i$
10:   if $validateSoln(t_j, s_j)$ then
11:      $allocateReward(a_i)$
12:      // Remove the dependencies from the
         dependent task on the current one
13:   $updateDependencies(t_j)$
14:   end if
15:   // On receive event listener
16:   $onTaskDoneMessage()$
17:   $t_j, s_j, \mathcal{I} \leftarrow$ response from an agent $a_i$
18:   $K(a_i) \leftarrow K(a_i) \cup \{(t_i, s_i)\}$
19:   for each $t_k, s_k \in \mathcal{I}$ do
20:      $K(a_i) \leftarrow K(a_i) \cup \{(t_k, s_k)\}$
21:   end for
22:   $n_e \leftarrow n_e + 1$
23:   go to 3
24: end while
```

Algorithm 2 Get available tasks for solution space exploration algorithm

Input: T: A set of tasks, R: A set of respective rewards, D: A set of respective dependencies, n_e: Total number of available agents for solution space exploration

Output: T_e

```
1: // Filter the tasks and respective reward by eliminating the tasks which have dependencies

2: $T_e, R_e \leftarrow getIndependentTasks(T, R, D)$
3: $m_e \leftarrow length(T_e)$
4: for $i \leftarrow 0$ to $m_e - 1$ do
5:    for $j \leftarrow 0$ to $m_e - i - 1$ do
6:       if $R_e[j] < R_e[j+1]$ then
7:          Swap $R_e[j]$ and $R_e[j+1]$
8:          Swap $T_e[j]$ and $T_e[j+1]$
9:       end if
10:   end for
11: end for
12: // Return a set of available tasks for exploration
13: if $n_e > m_e$ then
14:    return $T_e$
15: end if
16: return $T_e[0 : n_e]$
```

Algorithm 1 gets a total number of agents n, a set of tasks T with respective dependencies D, and reward R. In line 1 initially, all n agents are available for solution exploration. In line 3, we get a set of tasks from T_e that is not dependent on any other task, having the highest reward. In line 5, each task $t_j \in T_e$ is assigned to an available agent a_i. In line 8, we wait for a response from an agent a_i to validate the explored solution by a_i. In line 9, collect the explored solution for a task t_j. In lines 10–14, we do the validation for an explored solution and provide the respective reward r_j to an agent a_i based on the validation outcomes. The dependency of task t_j from all the dependent tasks on t_j is also removed. In line 16, we wait for a response from an agent a_i to get the explored solution and inference data. In line 17, we collect the explored solution (s_j) for a task t_j and inference data ($\mathcal{I}$). In line 18, we update $K(a_i)$ with the newly explored solution (s_j) as a value for a task t_j as a key. In lines 19–21, we iterate through each entry in $\mathcal{I}$, which contains inferred task (t_k) and respective solution (s_k) pairs and updates in $K(a_i)$. In line 22, we continue this process of assignment and validation for the remaining tasks.

In Algorithm 1, line 3 uses the *getAvailTasks* module, which is computed in Algorithm 2. Algorithm 2 accepts a total number of available agents for solution space exploration n_e, a set of tasks T with respective dependencies D and rewards set R, and as an output, it will return a set of tasks to be executed next. Algorithm 2, in line 2, it returns the available tasks T_e with respective rewards R_e, which does not have any dependency. In lines 4–11, it sets all the tasks in T_e in descending order based on rewards. In lines 13–16, it returns a task set T_e when the count of available tasks without any dependency is less than the total number of available agents n_e in the system. Otherwise, in line 14, it returns the top n_e number of tasks from T_e.

Algorithm 3 Knowledge gain at agent algorithm

Input: t_j: task to implement, B: Budget to ask queries from the oracle (Global variable)

Output: t_j, s_j, $\mathcal{I}$

```
1: // Initialise the variables to default value
2: String $hint \leftarrow null$
3: boolean $isRewarded = false$
4: $s_j, \mathcal{I} \leftarrow exploreSoln(t_j, hint)$
5: **if** $s_j$ **then**
6:   // Check the reward status for the explored solution
7:   $isRewarded \leftarrow isRewardAllocated(t_j, s_j)$
8:   **if** $isRewarded$ **then**
9:     // Return task, explored solution, and inference data
10:    **return** $t_j, s_j, \mathcal{I}$
11:  **end if**
12: **else if** $B > 0$ **then**
13:   $B \leftarrow B - 1$
14:   $hint \leftarrow askHelpFromOracle()$
15:   **go to** 4
16: **end if**
17: **return** $null$
```

Algorithm 3 describes the solution space exploration by an agent (a_i). In lines 2–3, we initialise the variables with default values. In line 4, the *exploreSoln* function returns the explored solution s_j and inference data for a task t_j based on the hint if provided by an oracle. Inference data contain the set of inferred task (t_k) with respective solution (s_k) in key-value pair. The *exploreSoln* function also checks that if the number of unaccomplished tasks is less than the allocated budget, then directly takes help from the oracle instead of solution exploration to reduce the exploration time. In line 7, an agent a_i checks the status of the received reward by using the $isRewardAllocated(t_j, s_j)$ module. The $isRewardAllocated(t_j, s_j)$ module sends s_j for the validation and returns a boolean value *true/false* based on the received reward as per the outcome of the validation. In lines 8–11, we return the explored solution (s_j) and inference data ($\mathcal{I}$) for a task (t_j). In lines 12–16, an agent a_i makes a query to an oracle if it has allocated budget greater then zero and continues with exploration. In line 17, it returns *null* if solution space exploration is failed.

5. Experimental results

The cooperative solutions to exploration tasks strategy are checked for multiple scenarios: even distribution of tasks across multiple agents, average solution exploration time at the agent level, average solution exploration time at a system level, even budget distribution, uneven budget distribution, even speed allocation, variation in agents' speeds, highly dependent system, and less-dependent system.

We generate a random maze with a random target location defined during each experiment. Maze sizes are varying. Solution exploration is performed on a maximum 400×400 maze size by multiple agents in parallel. The designed model is capable of handling the task dependencies to simulate real-time scenarios. We have tested the same by using standard G_{40} (Figure 1) and G_{18} (Figure 2) dependency program graphs. An agent a_i first explores the task on its own and may take help from the oracle by utilising the allocated query budget in the case of failure. Query budget utilisation is tested by providing the shared budget among the agents. During the experiments, available tasks as per G_{40} and G_{18} were distributed among 5 different cooperative agents. The tasks were split into multiple sets for the assignment. In all the results shown, exploration and waiting time unit for time is seconds.

We have performed multiple experiments for a G_{40} program graph with 5 agents on a 400×400 maze and observed that the average solution exploration time taken is almost similar for all agents. The maze was created dynamically on each run with a random target location. Further, the same test is performed for more complex tasks as well, where the agent is unable to explore the solution independently and takes help from an oracle, subject to a query budget remaining. Naturally, the average exploration time taken for a complex task is higher in comparison to an easy task because query help is required by agents for complex tasks . There is still a chance that a task may fail even after help from an oracle because the oracle only provides a hint to explore the solution, instead of the complete solution. Not all agents in the system get complex tasks due to randomisation and dependencies.

In order to test the scalability of our model, we ran an experiment varying the number of tasks keeping the number of agents constant and also tried another one where the number of agents is varied keeping the number of tasks constant. In both cases, the system graphs are of the highly dependent type similar to G_{40}. The results clearly indicate that for a range of values, our approach shows nearly linear scaling.

Table 1 shows the system exploration time for a highly dependent system like G_{40} when varying the number of agents from 1 to 9 for a constant 200 tasks. The last column indicates that the total processing time of all the agents in the system is consistent as the number of agents is varied, indicating linear scaling.

Figure 7 shows the result of scalability experiments when the number of tasks is varied from 40 to 200 for 5 agents. It also shows linear growth for the overall system exploration time.

In some cases where the number of complex tasks was higher than the allocated budget, the number of accomplished tasks ($|\mu(i)|$) was less than the number of assigned tasks ($|\lambda(i)|$). An agent a_i also collects inference data during explorations. Based on inference data, an agent's performance in terms of time execution is shown in Figure 5.

$Expl_T(a_i)$ stands for the average exploration time taken by an agent a_i, and $TWT(a_i)$ stands for the total waiting time of an agent a_i. Table 2 shows the total waiting time of an agent a_i due to task dependencies on other tasks by the G_{40} program graph. With average exploration time in seconds, it also shows the total number of assigned tasks ($|\lambda(i)|$) to an agent a_i out of 40 tasks. During this experiment, we have observed that $|\lambda(i)|$ was equal to $|\mu(i)|$ for all agents. However, the higher value of the waiting time is seen to affect $|\lambda(i)|$. The total waiting time of individual agents impacts the system performance.

Figure 3 shows the average solution exploration time taken by an agent a_i for a G_{18} program graph. It shows the average exploration time difference when agents are working individually or cooperatively. The experiment was performed on a 400×400 size maze, including complex tasks. To explore the complex tasks, the query budget is utilised by a_i to take help from an oracle. During the experiment, the allocated budget was insufficient to get help from an oracle for all the complex tasks. Therefore, a few complex tasks, and their dependent ones, remain unaccomplished.

Figure 4 shows that a faster agent improves the performance where some tasks were related to others' inference data and are completed due to said inference

Table 1. Scalability testing while varying the number of agents to explore the 200 tasks.

Agents	$Expl_T(HD)$	Total Processing Time
1	1032.17	1032.17
3	376.61	1129.83
5	225.10	1125.5
7	151.42	1059.94
9	120.32	1082.88

Table 2. Waiting time due to task dependencies for G_{40}.

Agents	$Expl_T(a_i)$	$\lvert\lambda(i)\rvert$	$TWT(i)$
1	30.102	8	4.17
2	28.614	9	0
3	29.912	7	15.01
4	28.015	8	9.36
5	29.721	9	4.91

Table 3. System performance when varying the count of faster agents for G_{18}.

f	$Expl_T(LD)$	WT
0	28.97	4.60
1	28.31	7.20
2	28.06	7.41
3	27.65	7.70
4	20.84	2.15
5	19.57	1.42

Table 4. System performance when varying the number of faster agents and budget for less-dependent and highly dependent systems.

f	Budget	$Expl_T(LD)$	$Expl_T(HD)$
0	20	29.73	29.87
1	20	27.30	28.52
1	40	25.95	28.13
1	60	24.54	26.94
1	**80**	**23.02**	26.33
1	100	21.73	25.54
2	100	19.41	22.41
3	100	17.70	19.38
4	100	14.79	15.86
2	20	26.48	26.96
3	20	24.51	25.79
4	**20**	23.90	**24.64**

Table 5. System performance when varying the budget for dissimilar agents for less-dependent and highly dependent systems.

Scenarios	$Expl_T(LD)$	$Expl_T(HD)$
1	19.81	21.63
2	15.97	19.75
3	13.78	17.66
4	10.92	14.24
5	10.07	13.97

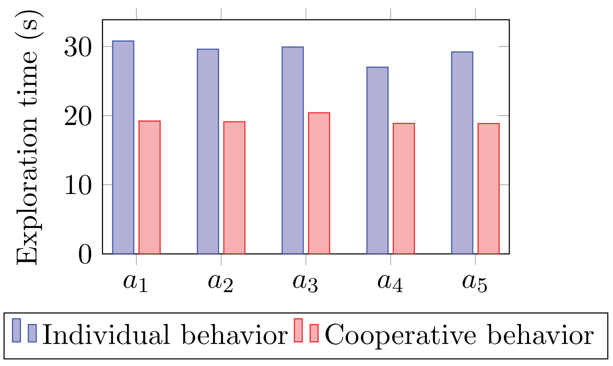

Figure 3. Average exploration time for individual vs cooperative agents for G_{18}.

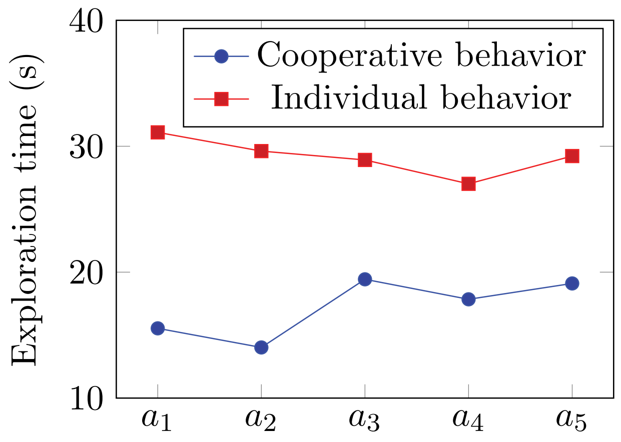

Figure 4. Cooperative faster agent improves the system performance for G_{18}.

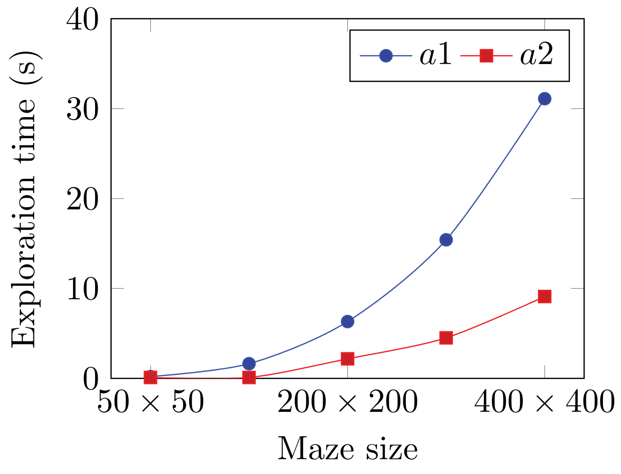

Figure 5. Solution space exploration time comparison between two agents for G_{40}.

data. Faster agents have shared the inference data with others and reduced the exploration times for other agents as well.

Figure 5 shows evaluations across several maze sizes. It shows two different agent behaviours out of 5 where agent a_1 gets new tasks and does the solution exploration, whereas agent a_2 gets the tasks for which solutions are already available due to inference data. The inference data were either collected by agent a_2 during the task exploration or received from other agents in the system. Figure 5 clearly shows that an agent a_2 takes less time for the solution space exploration compared with exploration time taken by an agent a_1. a_2's exploration time is approximately 70% less in comparison with a_1's exploration time.

Table 3 shows the observations when we vary the number of faster agents f out of 5 in the system where a faster agent's speed was $2\times$ compared with others. $Expl_T(LD)$ stands for the average waiting time for a less-dependent system, and WT stands for a system's average waiting time. We observe that system performance improves when $f \geq 4$ for the G_{18} program dependency graph. This shows that the system performance, which is dependent on available faster agents, varies based on the task dependencies. Fewer faster agents cannot improve the system performance due to pending exploration for parent tasks from slower agents; we just see an increase in the average waiting time of the system due to an increase in waiting times of the faster agents.

Figure 6(a) shows that increasing the speed of an agent a_i in a highly dependent system like G_40 initially improves the performance, but due to dependencies, the performance becomes constant after a specific speed increment. Varying the budget in increasing order for different speed agents improves the individual agent's performance consistently, as shown in Figure 6(b), where we have tested the performance with a budget of 20, 40, or 80 to 5 agents of different speeds.

We evaluated a trade-off between several faster agents vs. query budget for a highly dependent and less-dependent system, as shown in Table 4. $Expl_T(LD)$ stands for the average exploration time for the less-dependent system G_18, and $Expl_T(HD)$ stands for the average exploration time for the highly dependent system G_{40}. We observe that a high budget (80) for a single faster agent reduces the exploration time to 23.02. In contrast, an increment in the number of faster agents reduces the exploration time to 23.90 for a less-dependent system. Thus, it is better to increase the budget for a faster agent, instead of increasing the number of faster agents, in a less-dependent system. Similarly, an increment in the number of faster agents for a highly dependent system reduces the exploration time to 24.64. Thus, it is better to increase the number of faster agents in the system instead of increasing the budget for a highly dependent system.

Table 5 shows the exploration times for less-dependent and highly dependent systems where 5 different-speed agents are present. We also evaluate the average exploration time while allocating dissimilar budgets to an individual agent. Speed and budget combination for *Scenario* 1 is ($1\times$, 45), ($2\times$, 25), ($3\times$, 15), ($4\times$, 10), ($5\times$, 5), for

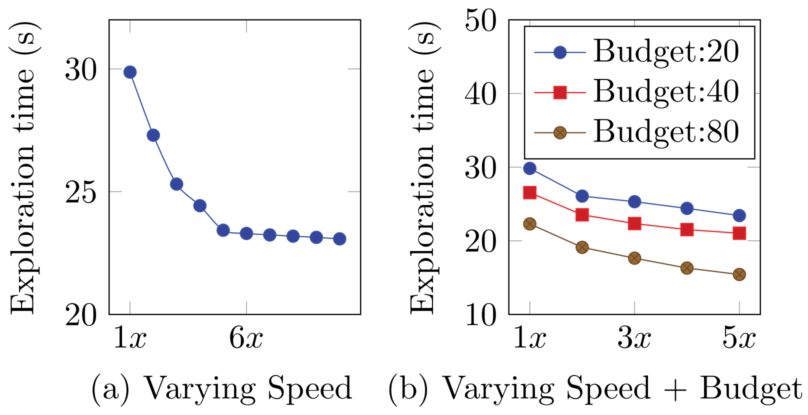

Figure 6. Varying speed and budget of an agent in a system G_{40}.

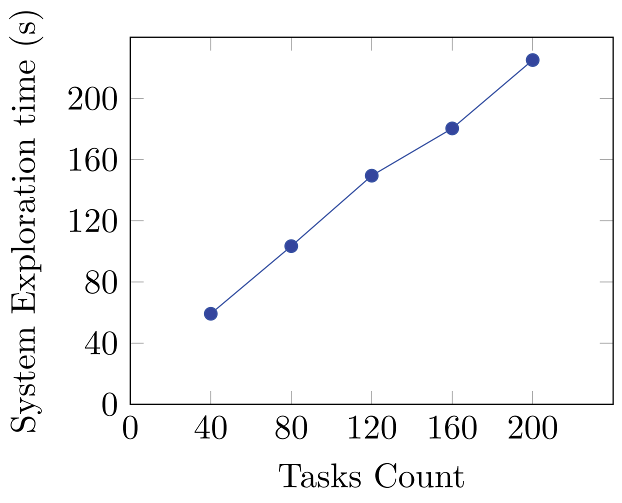

Figure 7. Scalability testing while increasing the number of tasks for 5 agents.

Scenario 2 is (1 × , 30), (2 × , 25), (3 × , 20), (4 × , 15), (5 × , 10), for *Scenario* 3 is (1 × , 20), (2 × , 20), (3 × , 20), (4 × , 20), (5 × , 20), for *Scenario* 4 is (1 × , 10), (2 × , 15), (3 × , 20), (4 × , 25), (5 × , 30), and for *Scenario* 5 is (1 × , 5), (2 × , 10), (3 × , 15), (4 × , 25), (5 × , 45). The evaluation results of all scenarios show that the system exploration time reduces when favouring faster agents, in line with the "Matthew effect" (Merton, 1968), for both less-dependent and highly dependent systems.

In summary, the following are key findings of our work:

(1) Agents' performance improves due to collection of inference data (Figure 5). This is in line with prior work that shows that using inference improves performance in goal-oriented collaborative work (C. C. Liu et al., 2016).

(2) Cooperative behavior of agents improves the agents' performance (Figure 3) as well as the system performance as a whole (Figure 4). It is well known that cooperation improves motivation (Carr & Walton, 2014), but our work suggests that it improves performance even when psychological aspects are not involved.

(3) Increasing speeds of agents improves the system performance up to a certain point only. Due to dependency on other tasks it may not improve the performance further (Figure 6a). In some cases it may increase the waiting time of an agent. Where an agent will wait for other task to be available for exploration (Table 3). This is in line with Amdahl's Law for parallel processing (Hill & Marty, 2008) which also holds that increasing the speed of a single component in a multi-processor system does not improve system performance beyond a point.

(4) Increasing speed and budget for an agent, linearly improves the system performance (Figure 6b).

(5) Constraints evaluation for highly dependent and less dependent system shows that its better to increase number of faster agent in a highly dependent system, while it is better to increase budget in a less-dependent system (Table 4).

(6) Increasing budget for a faster agent gives the better system performance (Table 5). This is in line with the "Matthew Effect" Merton (1968) that also holds that it is better to reward the higher-performing, rather than to spread resources equitably.

6. Conclusions

In this paper, we have evaluated trade-offs between agents' constraints of speed and query budget for a system where agents are dissimilar in speed but similar in function and can solve problems directly as well as by querying. As shown in our experimental results, favouring faster agents during budget allocation with a fixed total budget reduces the exploration time efficiently, in line with the "Matthew effect". The experimental findings showed that allocating more budget to a faster agent offers better performance in a less-dependent system, while in a highly dependent system, increasing the number of faster agents offers a better performance.

Given the large number of systems where solutions to complex problems can be computed cooperatively by several agents, or gained by query or inference subject to constraints, weaver that this work can be used to formulate a set of guidelines for improving the performances of such systems given necessary trade-offs.

Currently, a static reward value is used for task prioritisation. The limitation of the system is that the reward is not reducing or expiring over a period of time, which is not in line with the hard real-time applications like flight control systems, nuclear power plants, stock exchange, and medical and automotive equipment (Anceaume et al., 1999). Future work concerns the adaption of the proposed solution for hard real-time application where time plays a critical role for the explored solution. One objective is to consider the varying reward, which is reducing over a period of time.

Disclosure statement

No potential conflict of interest was reported by the author (s).

ORCID

Karishma http://orcid.org/0000-0003-3842-7408
Shrisha Rao http://orcid.org/0000-0003-0625-5103

References

Agrawal, P., & Rao, S. October 2014. Energy-aware scheduling of distributed systems. *IEEE*, Trans. *Autom. Sci. Eng. 11* (4): 1163–1175. https://doi.org/10.1109/TASE.2014.2308955)

Anceaume, E., Cabillic, G., Chevochot, P., & Puaut, I. (1999). A flexible run-time support for distributed dependable hard real-time applications. In *Proceedings 2nd ieee international symposium on object-oriented real-time distributed computing (isorc'99)(cat. no. 99-61702)* (pp. 310–319).

Ashutosh, K., Consul, S., Dedhia, B., Khirwadkar, P., Shah, S., & Kalyanakrishnan, S. (2020). Lower bounds for policy iteration on multi-action mdps. In *2020 59th ieee conference on decision and control (cdc)* (pp. 1744–1749).

Burgard, W., Moors, M., Stachniss, C., & Schneider, F. (2005). Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, *21*(3), 376–386. https://doi.org/10.1109/TRO.2004.839232

Cares, J. R. (2002). The use of agent-based models in military concept development. *Proceedings of the winter simulation conference*. (Vol. 1 , pp. 935–939). IEEE.

Carr, P. B., & Walton, G. M. (2014). Cues of working together fuel intrinsic motivation. *Journal of Experimental Social Psychology*, *53*(53), 169–184. https://doi.org/10.1016/j.jesp.2014.03.015

Choi, H., Crump, C., Duriez, C., Elmquist, A., Hager, G., Han, D., Hearl, F., Hodgins, J., Jain, A., Leve, F., Li, C., Meier, F., Negrut, D., Righetti, L., Rodriguez, A., Tan, J., & Trinkle, J. (2021). On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward. *Proceedings of the National Academy of Sciences*, *118* (1), e1907856118. https://doi.org/10.1073/pnas.1907856118 .

David, L., Cottet, F., & Nissanke, N. (2001). Jitter control in on-line scheduling of dependent real-time tasks. In *Proceedings 22nd ieee real-time systems symposium (rtss 2001) (cat. no. 01pr1420)* (pp. 49–58). IEEE.

Dawson, S., Wellman, B. L., & Anderson, M. (2010). Using simulation to predict multi-robot performance on coverage tasks. *2010 ieee/rsj international conference on intelligent robots and systems* (pp. 202–208). IEEE.

Dear, R. G., & Sherif, J. S. Using simulation to evaluate resource utilization strategies. (2000). *SIMULATION*, *74*(2), 75–83. https://doi.org/10.1177/003754970007400202

Dorri, A., Kanhere, S. S., & Jurdak, R. (2018). Multi-agent systems: A survey. *IEEE Access*, *6*, 28573–28593. https://doi.org/10.1109/ACCESS.2018.2831228

Eramo, V., Listanti, M., Lavacca, F. G., Iovanna, P., Bottari, G., & Ponzini, F. (2016). Trade-off between power and bandwidth consumption in a reconfigurable xhaul network architecture. *IEEE Access*, *4*, 9053–9065. https://doi.org/10.1109/ACCESS.2016.2639578

Fiscko, C., Kar, S., & Sinopoli, B. (2021). Efficient solutions for targeted control of multi-agent mdps. *2021 American control conference (acc)* (pp. 690–696). IEEE.

Ghassemi, P., DePauw, D., & Chowdhury, S. (2019). Decentralized dynamic task allocation in swarm robotic systems for disaster response: Extended abstract. *2019 international symposium on multi-robot and multi-agent systems (mrs)* (pp. 83–85). IEEE.

Gruler, A., Fikar, C., Juan, A. A., Hirsch, P., & Contreras-Bolton, C. (2017). Supporting multi-depot and stochastic waste collection management in clustered urban areas via simulation–optimization. *Journal of Simulation*, *11*(1), 11–19. https://doi.org/10.1057/s41273-016-0002-4

Guo, M., & Dimarogonas, D. V. (2017). Task and motion coordination for heterogeneous multiagent systems with loosely coupled local tasks. *IEEE Transactions on Automation Science and Engineering*, *14*(2), 797–808. https://doi.org/10.1109/TASE.2016.2628389

Gupta, S., & Pujari, S. (2009). A multi-agent system (mas) based scheme for health care and medical diagnosis system. *cinternational conference on intelligent agent multi-agent systems* (pp. 1–3). IEEE.

Hill, M. D., & Marty, M. R. (2008). Amdahl's law in the multicore era. *Computer*, *41*(7), 33–38. https://doi.org/10.1109/MC.2008.209

Hongwei, A., Xiong, L., & Xie, X. (2010). Multi-agent interactions centric virtual battlefield simulation model. *2010 2nd international conference on advanced computer control* (Vol. 3 , pp. 315–319). IEEE.

Hubmann, C., Schulz, J., Becker, M., Althoff, D., & Stiller, C. (2018). Automated driving in uncertain environments: planning with interaction and uncertain maneuver prediction. *IEEE Transactions on Intelligent Vehicles*, *3*(1), 5–17. https://doi.org/10.1109/TIV.2017.2788208

Ismail, S., Shaikh Ali, S. H., & Abu Bakar, M. H. (2018). Agent-based self-regulated learning simulation adopting the concept of gusc model. *2018 international symposium on agent, multi-agent systems and robotics (isamsr)* (pp. 1–6). IEEE.

Jianqiang Wang, Wenjuan Zhou, Shiwei Li, Danlei Shan. (2019). Impact of personalised route recommendation in the cooperation vehicle-infrastructure systems on the network traffic flow evolution. *Journal of Simulation*, 13(4), 239–253. doi:10.1080/17477778.2018.1515579

Kurniawati, H., Hsu, D., & Lee, W. S. (2008). Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *In Robotics: Science and systems* (Vol. 2008).

Lee, S., Jain, S., & Son, Y.-J. (2022, January). A hierarchical decision-making framework in social networks for efficient disaster management. *ACM Transactions on Modeling and Computer Simulation*, *32*(1), 1–26. https://doi.org/10.1145/3490027

Lee, Y. C., & Zomaya, A. Y. (2011, August). Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Transactions on Parallel and Distributed Systems*, *22*(8), 1374–1381. https://doi.org/10.1109/TPDS.2010.208

Li, S., Maddah-Ali, M. A., Yu, Q., & Avestimehr, A. S. (2018, January). A fundamental tradeoff between computation and communication in distributed computing. *IEEE Transactions on Information Theory*, *64*(1), 109–128. https://doi.org/10.1109/TIT.2017.2756959

Littman, M. L., Dean, T. L., & Kaelbling, L. P. (2013). On the complexity of solving markov decision problems. *arXiv preprint arXiv*, 1302.4971. https://arxiv.org/abs/1302.4971

Liu, M., Chang, W., Li, C., Ji, Y., Li, R., & Feng, M. (2020). Discrete interactions in decentralized multiagent coordination: A probabilistic perspective. *IEEE Transactions on Cognitive and Developmental Systems*, 13(4), 1010–1022, Dec. 2021. doi:10.1109/TCDS.2020.3040769

Liu, C., Hamrick, J. B., Fisac, J. F., Dragan, A. D., Hedrick, J. K., Sastry, S. S., & Griffiths, T. L. (2016). Goal inference improves objective and perceived performance in human- robot collaboration. In C. M. Jonker, S. Marsella, J. Thangarajah, & K. Tuyls (Eds.) *Proceedings of the 2016 international conference on autonomous agents & multiagent systems*. may 9-13, 2016. ACM. 940–948.

Liu, F., & Liu, Z. (2018). A neighborhood-based value iteration algorithm for pomdp problems. *2018 ieee 30th international conference on tools with artificial intelligence (ictai)* (pp. 808–812). IEEE.

Lu, Y., Nolte, T., Bate, I., & Norström, C. (2010). Timing analyzing for systems with task execution dependencies. *2010 ieee 34th annual computer software and applications conference* (pp. 515–524). IEEE.

Lu, Y., Nolte, T., Kraft, J., & Norstrom, C. (2010). Statistical-based response-time analysis of systems with execution dependencies between tasks. *2010 15th ieee international conference on engineering of complex computer systems* (pp. 169–179). IEEE.

Mahela, O. P., Khosravy, M., Gupta, N., Khan, B., Alhelou, H. H., Mahla, R., & Siano, P. (2020). Comprehensive overview of multi-agent systems for controlling smart grids. *CSEE Journal of Power and Energy Systems*, 8(1), 115–131, Jan. 2022. doi:10.17775/CSEEJPES.2020.03390

Merton, R. K. (1968). The matthew effect in science: The reward and communication systems of science are considered. *Science*, *159*(3810), 56–63. https://doi.org/10.1126/science.159.3810.56

Mukhopadhyay, S., & Jain, B. (2001). Multi-agent markov decision processes with limited agent communication. *Proceeding of the 2001 ieee international symposium on intelligent control (isic '01) (cat. no. 01ch37206)* (pp. 7–12). IEEE.

Ndoye, F., & Sorel, Y. (2013). Monoprocessor real-time scheduling of data dependent tasks with exact preemption cost for embedded systems. *2013 ieee 16th international conference on computational science and engineering* (pp. 714–721). IEEE.

Qin, L., Ouyang, F., & Xiong, G. (2018). Dependent task scheduling algorithm in distributed system. *2018 4th international conference on computer and technology applications (iccta)* (pp. 91–95). IEEE.

Rigney, D. (2010). *The matthew effect: how advantage begets further advantage*. Columbia University Press.

Salmerón-Garcı, J., Inigo-Blasco, P., Dı, F., Cagigas-Muniz, D., et al. (2015). A tradeoff analysis of a cloud-based robot navigation assistant using stereo image processing. *IEEE Transactions on Automation Science and Engineering*, *12* (2), 444–454. https://doi.org/10.1109/TASE.2015.2403593

Shani, G., Pineau, J., & Kaplow, R. (2013). A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, *27*(1), 1–51. https://doi.org/10.1007/s10458-012-9200-2

Shi, J., Ueter, N., von der Brüggen, G., & Chen, J.-J. (2019). Multiprocessor synchronization of periodic real-time tasks using dependency graphs. *2019 ieee real-time and embedded technology and applications symposium (rtas)* (pp. 279–292). IEEE.

Squazzoni, F., & Gandelli, C. (2012). Saint matthew strikes again: an agent-based model of peer review and the scientific community structure. *Journal of Informetrics*, *6* (2), 265–275. https://doi.org/10.1016/j.joi.2011.12.005.

Tavanpour, M., Kazi, B. U., & Wainer, G. (2020). Discrete Event Systems Specifications Modelling and Simulation of Wireless Networking Applications. *Journal of Simulation*, 16(1), 1–25. doi:10.1080/17477778.2020.1750313

Topcuoglu, H., Hariri, S., & Min-You, W. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, *13*(3), 260–274. https://doi.org/10.1109/71.993206

Viseras, A., Xu, Z., & Merino, L. (2020). Distributed multi-robot information gathering under spatio-temporal inter-robot constraints. *Sensors*, *20* (2), 484. https://doi.org/10.3390/s20020484.

Vlassis N, Nikos, Spaan, Matthijs .(2004). A fast point-based algorithm for pomdps. *In bene- learn 2004: proceedings of the annual machine learning conference of Belgium and the Netherlands* (pp. 170–176).

Waeber, R., Frazier, P. I., & Henderson, S. G. (2012, August). A framework for selecting a selection procedure. *ACM Transactions on Modeling and Computer Simulation*, *22*(3), 1–23. https://doi.org/10.1145/2331140.2331144

Wilsdorf, P., Pierce, M. E., Hillston, J., & Uhrmacher, A. M. (2019). Round- based Super-Individuals—Balancing Speed and Accuracy. In *Proceedings of the 2019 ACM SIGSIM Conference On Principles of Advanced Discretesimulation(SIGSIM-PADS '19)* (pp. 95–98). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3316480.3322894

Xiao, J., & Peng, J. (2019, July). Trade-offs between computation, communication, and synchronization in stencil-collective alternate update. *CCF Transactions on High Performance Computing*, *1*(2), 144–160. https://doi.org/10.1007/s42514-019-00011-x

Xu, H., & Yang, Y. (2009). Research and design on dynamic multi-agent cooperative processing model. *2009 international conference on web information systems and mining* (pp. 432–436). IEEE.

Yang, C., Yu, Z., Liu, Y., Wang, L., & Guo, B. (2019). Dynamic allocation for complex mobile crowdsourcing task with internal dependencies. In *2019 ieee smartworld, ubiquitous intelligence computing, advanced trusted computing, scalable computing communications, cloud big data computing, internet of people and smart city innovation (smart- world/scalcom/uic/atc/cbdcom/iop/sci)* (pp. 818–825). IEEE.

Zhang, J., Wei, L., Liu, M., & Deng, Y. (2021). A competition model for modeling and describing matthew effect in computational social systems. *2021 11th international conference on intelligent control and information processing (icicip)* (pp. 438–443). IEEE.

Zhao, Y., Chen, L., Li, Y., & Tian, W. (2014). Efficient task scheduling for many task computing with resource attribute selection. *China Communications*, *11*(12), 125–140. https://doi.org/10.1109/CC.2014.7019847

Zhao, L., Du, M., & Chen, L. (2018). A new multi-resource allocation mechanism: A tradeoff between fairness and efficiency in cloud computing. *China Communications*, *15*(3), 57–77. https://doi.org/10.1109/CC.2018.8331991

Zomaya, A. Y., & Lee, Y. C. (2012). Comparison and analysis of greedy energy-efficient scheduling algorithms for computational grids. *Energy-efficient distributed computing systems* (pp. 189–214). Wiley-IEEE Computer Society.

ZongZhang, Zhang, David Hsu, & d Wee Sun, Lee. (2014). Covering number for efficient heuristic-based pomdp planning. In *Proceedings of the 31st International Conference on Machine Learning*, 32 (ICML'14). JMLR.org, I–28–I–36.