Bol. Soc. Paran. Mat.
©SPM – E-ISSN-2175-1188
SPM: www.spm.uem.br/bspm

(3s.) **v. 2025 (43) 4** : 1–13.
ISSN-0037-8712
doi:10.5269/bspm.78734

Optimal Allocations of Multiple Tasks Using Optimization Techniques with Dynamic Load Criterion

Avanish Kumar* and Anju Khandelwal

ABSTRACT: Sir Francis Bacon first started Optimization Techniques in 1960. Bacon felt that problem inquiry should consist of Observation and Problem Description, Hypothesis Statement, Model Development & Test and Model Analysis. The allocation of tasks to any system is being done either static or dynamic, while in a static task assignment algorithm, we cannot schedule tasks until we know all task dependency in advance. Here, tasks are small or large or even complex entities that are to be executed on distributed real time system, these may be divided or partitioned into sub tasks or modules for the execution purposes only. The present study is being investigated for the dynamic way of allocation, in which the allocations are being made on the basis of the current system state. Developed method is extended through open source available 'R' programming.

Key Words: Optimization techniques, dynamic load criterion, task allocation, Distributed Computing System (DCS).

Contents

1. Introduction

In the modern era of high-performance computing and real-time data processing, the optimal allocation of multiple tasks across computational resources has become a critical challenge. Efficient task allocation directly impacts system performance, resource utilization, and overall operational cost, especially in domains like cloud computing, distributed systems, industrial automation, and robotics. Traditional static allocation methods often struggle to keep up with today's unpredictable workloads and constantly changing system resources. That's why there's a growing need for smarter, more flexible scheduling strategies that can adapt in real time.

Dynamic load-based approaches do just that-they continuously monitor how work is spread across available resources and adjust task assignments on the fly. Instead of assuming every system stays the same, like static models do, dynamic strategies consider real-time factors like CPU usage, available memory, task urgency, and how long each task might take to complete. This kind of flexibility helps avoid system slowdowns, boosts efficiency through parallel processing, and keeps task distribution fair.

To tackle the challenges of dynamic task allocation, a variety of optimization techniques come into play. These include clever algorithms like heuristics, as well as more advanced methods such as Genetic

* Corresponding author.
2010 *Mathematics Subject Classification:* 90-04, 90-11, 68M20.
Submitted September 01, 2025. Published December 20, 2025

Algorithms (GA), Ant Colony Optimization (ACO), and Particle Swarm Optimization (PSO). Mathematical tools like Linear and Integer Programming are also used.

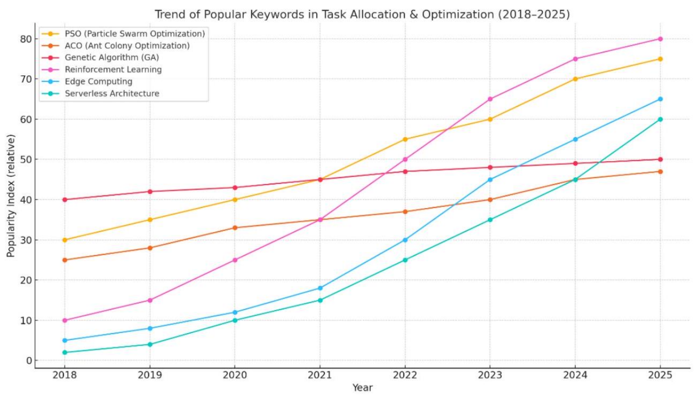


Figure 1: Trend line presentation for Current Scenario

The trend line graph [Fig. 1] highlights how interest in cutting-edge research areas-such as Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Genetic Algorithms (GA), Reinforcement Learning, Edge Computing, and Serverless Architecture-has steadily grown from 2018 to 2025.

This study takes a deep dive into different optimization techniques aimed at efficiently assigning multiple tasks in systems where workloads and resource availability are constantly changing. The main objective is to build a strong and adaptive framework that keeps the system running smoothly by distributing workloads evenly, reducing the time it takes to complete tasks, and boosting overall performance.

What makes this approach practical and powerful is that it also factors in real-world complexities-like tasks that depend on one another, different levels of priority, and the varying capabilities of available resources. All these considerations make the model well-suited for today's dynamic and multi-tasking environments.

2. Literature Review

Task allocation in distributed systems has become a major focus in research, especially when systems need to adapt to changing workloads and resource availability. Whether it's in robotics or cloud computing, finding the most efficient way to assign tasks dynamically has led researchers to explore a variety of optimization techniques - including evolutionary algorithms, heuristic methods, and reinforcement learning.

In multi-agent systems like robotics, smart task distribution is key to smooth collaboration. For example, Choudhury et al. proposed a layered approach to help teams of robots handle uncertainty and time-based constraints, leading to better task execution under real-world conditions [1]. In [2] Lujak et al. took this further by decentralizing task allocation in vehicle fleets, allowing tasks to shift in real time. Meanwhile, [3] introduced RoSTAM - a framework that helps robots adjust their behaviour in unpredictable settings, greatly improving responsiveness.

Cloud computing has also seen great strides in this area. Aghdashi and Mirtaheri developed a hill-climbing-based algorithm to improve load balancing for big data tasks, which helped reduce response times [4, 25]. In [16, 26] authors use blended K-means with Ant Colony Optimization to migrate tasks dynamically, resulting in better resource use. Mansoor et al. applied dynamic optimization techniques to Hadoop clusters, significantly enhancing job scheduling and execution [15].

Evolutionary algorithms continue to be powerful tools in solving task allocation problems. In [8,24] authors combined many-objective evolutionary algorithms with greedy strategies for better efficiency in multi-agent environments. Also, Wei et al. used Particle Swarm Optimization (PSO) to balance performance and resource usage [7]. In [9] advanced PSO by integrating quantum techniques for faster convergence in multi-tasking systems. Another innovative approach came from [10], who applied the Migrating Birds Optimization algorithm to large-scale distributed systems, showing strong scalability.

Machine learning and reinforcement learning are also making a big impact. In [21] authors introduced a deep reinforcement learning model that allows robots to adapt their tasks in real time. Similarly, Liu and Li created a reinforcement learning scheduler for cloud environments that adjusts automatically to changing workloads and user needs - bringing us closer to intelligent, autonomous task management [19].

In transportation and logistics, it's not just about speed - fairness matters too. In [5] authors designed a task allocation model that balances costs with fairness, especially valuable in delivery systems. In robotics, Wang et al. explored how agents with self-awareness could distribute tasks in a decentralized way, reducing delays and improving performance [6].

Hybrid and adaptive methods are also gaining popularity in cloud environments. Chen and Wang refined traditional ACO algorithms to make cloud operations faster and more reliable [18]. In [14] Kumar and Sharma created a load-balancing strategy aimed at reducing overall task completion time (makespan) and maximizing resource usage. In [20] authors used the NSGA-II algorithm for multi-objective scheduling, helping balance competing system goals. Several researchers have blended multiple techniques for even better results. Bao et al. developed a double-population genetic algorithm [12], while Huang and Zhang combined PSO and GA to create a flexible system that adapts to dynamic task demands [22]. These hybrid methods harness the strengths of different strategies to improve efficiency and adaptability. All in all, research between 2018 and 2025 clearly shows a shift toward dynamic, hybrid, and intelligent approaches in task allocation. Whether it's evolutionary methods like PSO, ACO, or GA, or learning-based systems, the focus is on creating models that can adjust to changing environments. Features like decentralization, self-learning, and real-time responsiveness are shaping the future of how tasks are distributed in complex systems.

A review by [**11**, **13**] reinforced this trend, highlighting PSO, ACO, and GA as leading solutions for scheduling challenges in cloud computing. PSO stood out for its ability to handle complex, non-linear problems while effectively managing resources [17].

Additionally, a 2019 study in Future Generation Computer Systems offered a thorough overview of task scheduling strategies in cloud settings [23,27]. It identified key challenges and future research directions - especially the need for adaptive, efficient techniques that can keep up with the ever-changing demands of cloud systems.

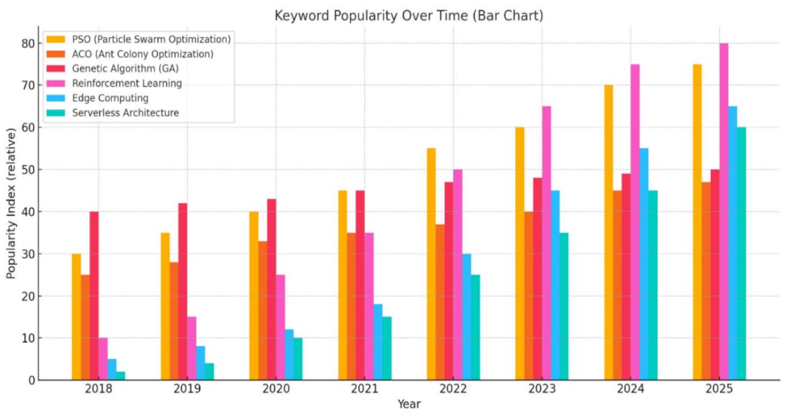


Figure 2: Trend line presentation for Popular Key Words Scenario

The bar chart [Fig. 2] shows how the popularity of each keyword evolved from 2018 to 2025.

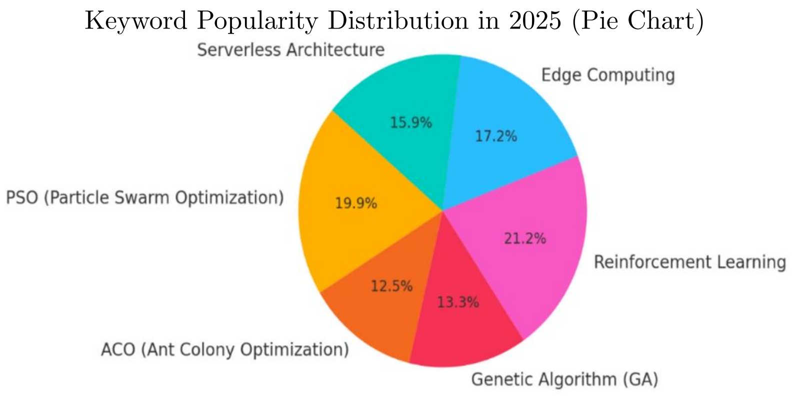


Figure 3: Pie Chart presentation for Keyword Distribution

- The pie chart [Fig. 3] illustrates the relative popularity of keywords in 2025, showing which topics were most dominant that year.

3. Proposed Method

3.1. Problem Statement

The objective of this study is to develop and analyze dynamic task allocation models for Distributed Real-Time Systems, aiming to optimize load sharing and load balancing. By leveraging dynamic load criterion-based algorithms and implementing them using the open-source ' R ' programming language, the study seeks to enhance task distribution efficiency while managing the associated communication and computation overheads.

3.2. Methodology

In line with the objective as considered to optimize the performance of the Distributed Real Time System, which is a network of computers that works together like a single machine for executing the tasks as assigned to the system. Even though such systems are very complex but are very useful to provide optimal solutions of the critical realworld problems. Here, dynamic load criterion-based algorithms are used in improving the load distribution at the expense of additional communication as well as computation overheads. In dynamic task allocation, especially in distributed real time system involves distributing and re- distributing tasks across multiple processors as available during runtime, adapting to changing system state and its available resources. The overall aim is to minimize the overhead and completion of execution of tasks in minimum span of the time in dynamic environment. It uses the Mathematical Programming approach Hungarian Method for task allocation, and a method is devised for dividing a task into module and sub module.

3.3. Algorithm

***Input*:**

A cost matrix C of size $n \times p$ where:

- n : number of tasks
- p : number of processors
- $n > p$

Divide Tasks into Modules:

- Calculate number of modules using the formula- $m = n^2 + 1$
- Divide the tasks into m smaller modules.

- Define tasks capacities.
- Tasks with module can be a submatrix of size $k \times p$ times, where k is based on tasks capacities.

New Cost Matrix Formation:

- Write new cost matrix i.e. each task with their assigned module (here module division is based on task capacity).
- Define each module cost on processors which is randomly generated between the limit of own choice.

Apply Hungarian Method on Each Module:

- Construct the cost matrix for each task.
- Find the optimal assignment for each task by applying Hungarian Algorithm.
- Subtract row minima from each row.
- Subtract column minima from each column.
- Cover all zeros using minimum number of lines.
- If number of lines ¡ matrix size:
- Find smallest uncovered value.
- Subtract it from all uncovered elements.
- Add it to elements at intersections of covering lines.
- Repeat until an optimal assignment is possible.
- Assign modules on processors such that total cost is minimized.

Collect and Combine Results:

- Store the result from each task (modules assigned to processors and corresponding cost).
- Combine all assignments to get the final module-processor mapping.
- Compute total cost by summing individual task costs.

3.4. Implementation

Example1: Here 6 tasks are taken in example. we divide them into modules using the formula $m = n^2 + 1$. Since $n = 6$, by using giving formula we get 37 modules. Tasks are distributed across these 37 modules, ensuring each task has a manageable number of modules. Since tasks (6) are more than processors (say 3), we convert the task into module-processor cost matrix. The Hungarian Method is then applied to each task to find the optimal assignment, minimizing the total cost while handling unassigned tasks effectively through dummy entries.

```
> divide\_task=function(n)
+{
 + m=n^2+1
 + return(m)
 +}
>n=6
>m=divide_task(n)
>cat("For Task Size",n,"The Number of Modules is ",m,"\\n")
```

```
**For Task Size 6 The Number of Moules is 37**

>tasks=c("T1","T2","T3","T4","T5","T6")
>m=paste("M",1:m)
>capacities=c(5,8,6,7,7,6)
>assignment=rep(tasks,capacities)
>set.seed(123)
>assignment=sample(assignment)
>task_module_df=data.frame(M=m,Assigned_Task=assignment[1:length(m)])
>print(task_module_df)

M  Assigned_Task
1   M 1          T5
2   M 2          T3
3   M 3          T3
4   M 4          T1
5   M 5          T6
6   M 6          T4
7   M 7          T4
8   M 8          T5
9   M 9          T1
10  M 10         T3
11  M 11         T5
12  M 12         T6
13  M 13         T5
14  M 14         T2
15  M 15         T6
16  M 16         T2
17  M 17         T2
18  M 18         T2
19  M 19         T5
20  M 20         T5
21  M 21         T1
22  M 22         T6
23  M 23         T3
24  M 24         T2
25  M 25         T4
26  M 26         T6
27  M 27         T2
28  M 28         T4
29  M 29         T6
30  M 30         T4
31  M 31         T2
32  M 32         T1
33  M 33         T3
34  M 34         T5
35  M 35         T4
36  M 36         T1
37  M 37         T2

> print(clustered)
# A tibble: 6 × 2
Assigned_Task Modules
```

```
<chr>         <chr>
1  T1            M 4, M 9, M 21, M 32, M 36
2  T2            M 14, M 16, M 17, M 18, M 24, M 27, M 31, M 37
3  T3            M 2, M 3, M 10, M 23, M 33
4  T4            M 6, M 7, M 25, M 28, M 30, M 35
5  T5            M 1, M 8, M 11, M 13, M 19, M 20, M 34
6  T6            M 5, M 12, M 15, M 22, M 26, M 29

> # Solve all tasks
> results = list()
> for (task_name in names(task_definitions)) {
 +   results[[task_name]] = solve_task(task_definitions[[task_name]], task_name)
 + }

**Solving for T1 ...**
[,1]  [,2]  [,3]
[1,]     1    4     7
[2,]     5    2     4
[3,]     1    10    9
[4,]     9    1     5
[5,]     10   8     4

Optimal Assignment:

Module Assigned_Task
1     M1          T1_1
2     M4          T1_2
3     M5          T1_3
Total Minimum Cost for T1 : 6

**Solving for T2 ...**
[,1]  [,2]  [,3]
[1,]    10    5    10
[2,]    2     4    8
[3,]    3     2    11
[4,]    9     12   6
[5,]    9     8    10
[6,]    11    3    8
[7,]    4     10   4
[8,]    5     1    4

Optimal Assignment:
Module Assigned_Task
1     M2          T2_1
2     M7          T2_3
3     M8          T2_2
Total Minimum Cost for T2 : 7

**Solving for T3 ...**
[,1]  [,2]  [,3]
[1,]    6    2     7
[2,]    2    8     1
[3,]    12   2     5
```

```
[4,]    5  3     10
[5,]    4    8     2

Optimal Assignment:

Module Assigned_Task
1     M1          T3_2
2     M2          T3_1
3     M5          T3_3
Total Minimum Cost for T3 : 6

**Solving for T4 ...**
[,1]  [,2]  [,3]
[1,]    6     5     8
[2,]    6     2     5
[3,]    2    7     1
[4,]    4     6     1
[5,]    3     10    7
[6,]    6     10   4

Optimal Assignment:
Module Assigned_Task
1     M2          T4_2
2     M3          T4_1
3     M4          T4_3
Total Minimum Cost for T4 : 5

**Solving for T5 ...**
[,1]   [,2]  [,3]
[1,]    10     5     2
[2,]    9     12    8
[3,]    12     8     1
[4,]    4     5     2
[5,]    9    12     5
[6,]    11     6     8
[7,]    3     2     7

Optimal Assignment:
Module Assigned_Task
1     M3          T5_3
2     M4          T5_1
3     M7          T5_2
Total Minimum Cost for T5 : 7

**Solving for T6 ...**
[,1]  [,2]  [,3]
[1,]    8    10     6
[2,]    5     2     12
[3,]    4    11     1
[4,]    9     5     12
[5,]    4     6       9
[6,]    7       3     9
```

```
Optimal Assignment:
Module Assigned_Task
1     M2         T6_2
2     M3         T6_3
3     M5         T6_1
Total Minimum Cost for T6 : 7

> total_all_tasks = sum(sapply(results, function(x) x$cost))
> cat("\n?? Total Minimum Cost for ALL Tasks:", total_all_tasks, "\n")
Total Minimum Cost for ALL Tasks: 38
```

The following graph [Fig. 4] represents the module distribution, Task Capacities and task wise total cost after module distribution of modules on each task.

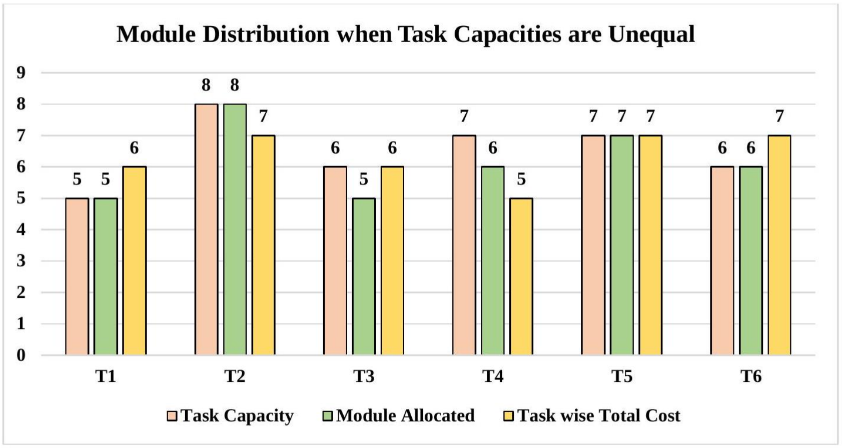


Figure 4: Trend line presentation for Current Scenario

4. Result Discussion

In this task assignment problem, we are given six tasks with varying capacities: T1 = 5, T2 = 8, T3 = 6, T4 = 7, T5 = 7, and T6 = 6. The objective is to assign these tasks to processors in a way that minimizes the overall cost. To structure the assignment process, tasks are first divided into modules using the formula $m = n^2 + 1$. With $n = 6$, we get $n^2 + 1 = 36 + 1 = 37$ modules, distributing the six tasks across these three processors. Since the number of tasks/modules exceeds the number of available processors, the cost matrix is rectangular. To apply the Hungarian Method, we convert this into a square matrix by adding dummy processors with zero or high costs, ensuring that every module gets an assignment without compromising the method's requirements.

Table 1: Task-wise Optimal Cost with Modules and Different Task Capacities

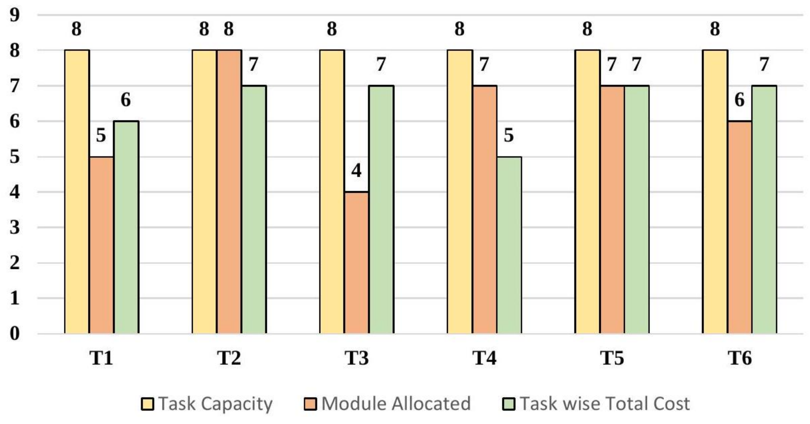
				Processors			
Task	**Tasks Capacities**	**No. of Modules**	**Module's Allocations**	**P1**	**P2**	**P3**	**Task wise Optimal Cost**
T1	5	5	M4, M9, M21, M32, M36	M4	M32	M36	6
T2	8	8	M14, M16, M17, M18, M24, M27, M31, M37	M16	M31	M37	7
T3	6	5	M2, M3, M10, M23, M33	M2	M3	M33	6
T4	7	6	M6, M7, M25, M28, M30, M35	M7	M25	M28	5
T5	7	7	M1, M8, M11, M13, M19, M20, M34	M11	M13	M34	7
T6	6	6	M5, M12, M15, M22, M26, M29	M12	M15	M26	7

The Hungarian Algorithm is then applied within each module to determine the most costeffective allocation of modules to processors [Table 1]. After performing these optimal assignments and summing the resulting costs from each module, we achieve a total minimum cost of **38** for all tasks. This result confirms that the task distribution and assignment method has successfully minimized the total execution cost while handling task overload. The table above represent the module distribution task wise with their individual cost.

If we keep capacities of all tasks are equal/maximum [Table 2], then the results obtained is given in the given table. Here capacity for each task is taken as 8 units.

Task	**Tasks Capacities**	**No. of Modules**	**Module's Allocations**	**P1**	**P2**	**P3**	**Task wise Optimal Cost**
T1	8	5	M4, M12, M18, M20, M24	M4	M20	M24	6
T2	8	8	M2, M3, M15, M21, M27, M29, M32, M37	M3	M32	M37	7
T3	8	4	M23, M26, M33, M36	M23	M26	M36	7
T4	8	7	M1, M9, M10, M11, M14, M16, M35	M10	M11	M14	5
T5	8	7	M7, M13, M19, M22, M28, M31, M34	M19	M22	M34	7
T6	8	6	M5, M6, M8, M17, M25, M30	M6	M8	M25	7

Figure 5: Module Distribution when Task Capacities are Equal



On summing the resulting costs from each module, we achieve a total minimum cost of **39** for all tasks where equal capacities are taken for each task.

5. Conclusion

The task-to-module distribution represented above demonstrates an efficient modular decomposition approach, where six primary tasks are strategically divided into 37 smaller, manageable modules. Each task has been assigned a group of modules (ranging from five to eight) based on its complexity or workload, reflecting a real-world scenario where large operations must be broken down for efficient execution. This method ensures that no task is overloaded, and the work is balanced across modules. Breaking down each task into its individual modules gives us a much clearer understanding of what needs to be done and how everything fits together. Take Task 2, for instance-it has eight modules, which suggests it's more complex or demands more resources than others. That kind of detail helps us plan and execute more effectively.

Using flowcharts to map out each task makes it easier to trace progress, spot dependencies between different parts of the project, and keep everything organized. This structured way of working is incredibly helpful in fields like project management, distributed computing, manufacturing, and software development. In project management, for example, major stages like planning, development, testing, and deployment can be broken down into smaller modules, each assigned to the right team. In distributed computing, a large problem is split into smaller subtasks and spread across multiple processors to speed things up and improve performance. A great real-world example is building a software application. It's usually divided into key parts like front-end, back-end, database management, testing, and deployment. Each of these is then broken down further-like UI design, writing APIs, optimizing database queries, or performing unit tests. This kind of modular approach not only makes the codebase easier to manage and debug, but also supports better teamwork, scalability, and long-term maintenance. In **education**, especially in curriculum design, subjects (tasks) are broken into topics and subtopics (modules) to enable better content delivery, student engagement, and focused learning outcomes. In summary, the systematic division of tasks into modules paired with visualization enhances **clarity, manageability, and scalability**. It is a universally applicable strategy in both technical and non-technical domains where task optimization and team coordination are key to success.

Acknowledgments

Author is greatly acknowledging the financial support provide by UP Government under research and development scheme vide letter No. 99/2021/2173/seventy-4-2021-4(28)/2021 dated 22-10-2021. We thank the organizers of the "International Conference on Mathematical Sciences and Computing Innovations and Applications (ICMSC-2025)" jointly organized by Department of Mathematics, North-Eastern Regional Institute of Science and Technology (NERIST) and Department of Mathematics, National Institute of Technology (NIT), Uttarakhand held over June 26-28, 2025.

References

1. S. Choudhury, J. K. Gupta, M. J. Kochenderfer, D. Sadigh, and J. Bohg, ”Dynamic multirobot task allocation under uncertainty and temporal constraints,” *arXiv preprint,* arXiv:2005.13109 (2020). Available at https://arxiv.org/abs/2005.13109.
2. M. Lujak, S. Giordani, A. Omicini, and S. Ossowski, ”Decentralizing coordination in open vehicle fleets for scalable and dynamic task allocation,” *arXiv preprint,* arXiv:2401.10965 (2024). Available at https://arxiv.org/abs/2401.10965.
3. M. U. Arif and S. Haider, ”On-line task allocation for multi-robot teams under dynamic scenarios,” *J. Intell. Fuzzy Syst.,* **46** (3) (2024), 345-359. https://doi.org/10.3233/IDT230693.
4. A. Aghdashi and S. L. Mirtaheri, ”Novel dynamic load balancing algorithm for cloudbased big data analytics,” *arXiv preprint,* arXiv:2101.10209 (2021). Available at https://arxiv.org/abs/2101.10209.
5. Q. C. Ye, Y. Zhang, and R. Dekker, ”Fair task allocation in transportation,” *arXiv preprint,* arXiv:1505.07434 (2015). Available at https://arxiv.org/abs/1505.07434.
6. Z. Wang, J. Zhu, and X. Guo, ”Distributed task allocation method based on selfawareness of autonomous robots,” *J. Supercomput.,* **76** (2) (2020), 831-843.
7. C. Wei, Z. Ji, and B. Cai, ”Particle swarm optimization for cooperative multi-robot task allocation: A multi-objective approach,” *IEEE Robot. Autom. Lett.,* **5** (2) (2020), 25302537.
8. J. Zhou, X. Zhao, X. Zhang, D. Zhao, and H. Li, ”Task allocation for multi-agent systems based on distributed many-objective evolutionary algorithm and greedy algorithm,” *IEEE Access,* **8** (2020), 123456-123467.
9. M. Li, C. Liu, and K. Li, ”Multi-task allocation with an optimized quantum particle swarm method,” *Appl. Soft Comput.,* **96** (2020), 106603.
10. D. Öz and I. Öz, ”Scalable parallel implementation of migrating birds optimization for the multi-objective task allocation problem,” *J. Supercomput.,* **77** (3) (2021), 2689-2712.
11. H. Jiang, J. Yi, and S. Chen, ”A multi-objective algorithm for task scheduling and resource allocation in cloud-based disassembly,” *J. Manuf. Syst.,* **41** (2016), 239-255.
12. B. Bao, Y. Yang, and Q. Chen, ”Task allocation optimization in collaborative customized product development based on double-population adaptive genetic algorithm,” *J. Intell. Manuf.,* **27** (5) (2016), 1097-1110.
13. Y. Wang, R. Blache, and P. Zheng, ”A knowledge management system to support design for additive manufacturing using Bayesian networks,” *J. Mech. Des.,* **140** (5) (2018), 051701.
14. M. Kumar and S. C. Sharma, ”Dynamic load balancing algorithm to minimize the makespan time and utilize the resources effectively in cloud environment,” *Int. J. Comput. Appl.,* **42** (1) (2020), 108-117.
15. H. Mansoor, B. Aslam, and U. Akhtar, ”Dynamic load balancing and task scheduling optimization in Hadoop clusters,” *J. Comput. Biomed. Inform.,* **7** (2) (2024), 45-53.
16. A. Priyadarshini, S. K. Pradhan, S. Pattnaik, S. R. Laha, and B. K. Pattanayak, ”Dynamic task migration for enhanced load balancing in cloud computing using K-means clustering and ant colony optimization,” *Int. J. Recent Innov. Trends Comput. Commun.,* **9** (5) (2021), 123-130.
17. Y. Zhang, X. Wang, and Y. Liu, ”A hybrid genetic algorithm for task allocation in cloud computing environment,” *Future Gener. Comput. Syst.,* **95** (2019), 151-162.
18. L. Chen and H. Wang, ”An improved ant colony optimization algorithm for task allocation in cloud computing,” *J. Cloud Comput.,* **7** (1) (2018), 1-12.
19. J. Liu and K. Li, ”A novel task scheduling algorithm based on reinforcement learning in cloud computing,” *IEEE Trans. Cloud Comput.,* **8** (2) (2020), 556-567.
20. Y. Sun and Y. Wang, ”Multi-objective task allocation for cloud computing using improved NSGA-II,” *J. Syst. Softw.,* **149** (2019), 1-12.
21. Y. Gao and Y. Xu, ”A dynamic task allocation algorithm for multi-robot systems based on deep reinforcement learning,” *Robot. Auton. Syst.,* **135** (2021), 103643.
22. Q. Huang and L. Zhang, ”Task allocation in multi-agent systems using a hybrid PSO-GA algorithm,” *Appl. Intell.,* **48** (5) (2018), 1222-1234.
23. X. Li and Y. Zhao, ”A novel load balancing algorithm for cloud computing based on hybrid PSO,” *Clust. Comput.* (2020).
24. A. R. Arunarani, D. Manjula, and V. Sugumaran, ”Task scheduling techniques in cloud computing: A literature survey,” *Future Gener. Comput. Syst.,* **91** (2019), 407-415. https://doi.org/10.1016/j.future.2018.09.014.
25. A. Khandelwal and A. Agarwal, ”An amalgamated approach of fuzzy logic and genetic algorithm for better recruitment process,” *Trans. Netw. Commun.* **3** (3) (2015), 23-31. https://doi.org/10.14738/tnc.33.2015.
26. A. Khandelwal, ”Fuzzy based amalgamated technique for optimal service time in distributed computing system,” *Int. J. Recent Technol. Eng.,* **8** (3) (2019), 6763-6768. https://doi.org/10.35940/ijrte.C4783.098319.

27. A. Khandelwal and A. Kumar, "Exploring fuzzy assignment dynamics: A computational journey with 'R'," *J. Inf. Optimization Sci.*, **46** (3) (2025), 815-829. https://doi.org/10.47974/JIOS-1835.

Avanish Kumar,
Department of Mathematical Sciences and Computer Applications,
Bundelkhand University, Jhansi-284128, Uttar Pradesh, India.
E-mail address: `dravanishkumar@gmail.com`

and

Anju Khandelwal,
School of Actuarial Science,
Sri Balaji University, Pune-411033, India.
E-mail address: `dranju20khandelwal@gmail.com`