Submitted on: 10/01/2025
Revised forme accepted on: 27/02/2025
Corresponding author: ml.bendiaf@univ-chlef.dz

Revue Nature et Technologie
https://journals.univ-chlef.dz/index.php/natec

Research Article

ISSN: 1112-9778 – EISSN: 2437-0312

Dynamic Adaptation for Independent Task Scheduling Using Dynamic Programming in Multiprocessor Systems

Lotfi BENDIAF[a,*], Ahmed HARBOUCHE[b] and Mohammed Amin TAHRAOUI[b]

[a] *LME Laboratory, Faculty of Exact Sciences and Informatics, Hassiba Ben Bouali University of Chlef*
[b] *LIA Laboratory, Faculty of Exact Sciences and Informatics, Hassiba Ben Bouali University of Chlef*

Citation: BENDIAF, L., HARBOUCHE, A. and TAHRAOUI, M.A. (2025, Mars 30) Dynamic Adaptation for Independent Task Scheduling Using Dynamic Programming in Multiprocessor Systems. *Revue Nature et Technologie*. 17 (1), 09-16

Abstract

Performance in IT systems is critical to ensuring that systems meet user needs and expectations. In heterogeneous computing systems (HCS), which consist of processors with varying capabilities, dynamic adaptation plays a vital role in maintaining high performance. Dynamic adaptation enables systems to adjust task allocation and resource usage in real-time to respond to changes in workloads, resource availability, and system conditions. Task scheduling is a key aspect of achieving dynamic adaptation and remains a challenging NP-hard problem in HCS. Efficient scheduling requires optimizing competing objectives, such as minimizing makespan and maximizing processor utilization, to ensure that resources are used effectively. In this work, we propose DYnamic Task Allocation using dynamic programminG (DyTAg), a task scheduling algorithm based on dynamic programming, designed to support dynamic adaptation in HCS. Dynamic programming is particularly suited to this context as it breaks the scheduling problem into smaller, manageable subproblems and solves them incrementally, enabling efficient real-time adjustments. DyTAg leverages dynamic programming to minimize makespan while maximizing resource utilization, ensuring that tasks are allocated optimally even in complex, heterogeneous environments. To evaluate its performance, DyTAg is compared against established algorithms, including Min-Min, Max-Min, and Quality of Service Guided Min-Min, using various task sets and processor configurations. The results demonstrate that DyTAg achieves superior performance, particularly in scenarios involving independent tasks and small task sets, showcasing its potential to enhance dynamic adaptation and optimize performance in heterogeneous computing systems.

Keywords: Dynamic Adaptation; Scheduling; Dynamic Programming; Multi-processor system; Heterogeneous computing environment

1. Introduction

In recent years, the field of computer science has experienced rapid progress in both hardware and software development, giving rise to innovative technologies that have greatly enhanced efficiency and usability within the Information Technology sector. In particular, innovations like multi-core processors, multi-processor systems, and distributed computing have revolutionized heterogeneous computing environments.

A major challenge in distributed systems is optimizing critical performance metrics, including makespan, resource utilization, latency, and throughput. Conventional scheduling algorithms frequently fall short when confronted with the complexities of heterogeneity and dynamic environments, such as varying workloads or node failures, making it difficult for them to sustain efficiency and scalability in practical applications.

Dynamic adaptation, which enables systems to reassign tasks and modify configurations in response to changing conditions, plays a crucial role in improving the performance of distributed systems. Achieving this, however, demands sophisticated algorithms capable of balancing workloads, adhering to task dependencies, and optimizing resource allocation in real-time.

However, while these advancements provide powerful hardware platforms that greatly enhance system performance, achieving high parallelism in HCS

requires effective management on the software side as well. Task scheduling in HCS, unfortunately, is an NP-complete problem [1, 2], meaning that finding an optimal scheduling solution is computationally intractable as the problem size grows. Different scheduling approaches yield varying execution times for the same set of tasks in an HCS, highlighting the impact of scheduling techniques on system performance.

The goal of task scheduling in HCS is to allocate tasks to processors in a way that optimizes performance in terms of Quality of Service (QoS) and resource utilization. However, as demonstrated in [3], it is often impossible to assign all priority tasks to the most efficient processor while also meeting task precedence constraints and minimizing the overall schedule length (makespan).

Consequently, numerous scheduling algorithms have been developed, each tailored to meet specific objectives [4, 5]. For instance, the authors in [6] focus on minimizing makespan, while Li *et al.* [7] propose an algorithm for scheduling preemptible tasks to optimize computational resources in Infrastructure-as-a-Service (IaaS) cloud systems.

2. Background Materials

Performance is critical for ensuring that a system can handle increasing amounts of data, traffic, or users without degrading performance, because of the strong relation between performances in IT systems and scheduling.

To achieve minimal makespan, the authors in [8] proposed a method for assigning tasks to processors based on the minimum execution cost of each task across different processors. However, in most cases, this approach resulted in an increased makespan, leading to poor QoS To address this issue, researchers such as Ezzatti *et al.* [3, 9, 10] introduced an enhanced implementation of the Min-Min heuristic that incorporates QoS constraints.

On the other hand, the authors in [11] adapted a distributed algorithm in cloud systems and proposed a workflow scheduling algorithm which considers dynamic priority of the tasks. This approach undergoes a process of min–max normalisation [12] While the focus of our research is essentially optimization, Stützle *et al.* [13] introduced the MAX–MIN Ant System (MMAS), an Ant Colony Optimization algorithm that models ants as simple agents progressively building candidate solutions. This approach is designed to address NP-hard static combinatorial optimization problems effectively.

In this paper, we propose a dynamic adaptation approach for a task scheduling algorithm in heterogeneous computing environments using dynamic programming. Our approach combines optimization of node resource utilization with QoS considerations.

2.1. Scheduling Problem Context

First of all, we start by introducing the problem's context in HCS, then its corresponding model according to scheduling criteria, namely processors' computing capacities, the total run time and system resource utilisation.

The scheduling system is represented in [20] by the quadruple $S = (T, C, R, P)$ where $T = \{t_1, \dots t_n\}$ denotes the set of tasks to be scheduled, $C = \{c_1, \dots c\}$ the set of tasks' execution costs, $R = \{r_1, \dots r_n\}$ corresponds to the task ranks, $P = \{p_1, \dots p_n\}$ set of heterogeneous processors.

Numerous methods in the literature have tackled related problem scenarios, each offering unique advantages and facing specific limitations.

2.2. Literature Approaches

Min-min heuristic uses Minimum Completion Time (MCT) as a metric, the heuristic uses as inputs the set T of all unmapped tasks. It has two phases. In the first phase, the set of minimum expected completion time M is set up from T (For each task t_i determine its minimum completion time over all processors p_j in P) where: $M = \{Min(completion_{-time}(t_i, p_j))\}$ $T = \{t_1, \dots t_n\}$ and $P = \{p_1, \dots p_m\}$, M consists of one entry for each unmapped task. In the second phase, the task with the overall minimum completion time from M is selected and assigned to the corresponding machine and the workload of the selected machine will be updated. And finally, the newly mapped task is removed from T and the process repeats until all tasks are mapped.

This algorithm complies with the QoS rules and requirements, since it matches only the tasks requesting high QoS to hosts having high QoS.

A new version of Min-Min approach was proposed by He *et al.* [3] to improve the original algorithm results. Effectively, by applying the both algorithms on the same system, traditional Min-Min and QoS Guided Min-Min, the makespan given by QoS Guided Min-Min shows a significant improvement.

The Max-min heuristic closely resembles min-min and shares the same metric MCT. It initiates by considering the set T, comprising all unmapped tasks. Subsequently, the heuristic identifies the set of minimum completion times, denoted as: $M = \left\{Min\left(completion_{time}(t_i, p_j)\right)\right\}, T = \{t_i, \dots t_n\}$ and $P = \{p_1, \dots p_m\}$. Then, the task with the highest completion time among the elements in M is chosen and assigned to the corresponding machine, resulting in an update to the workload of that machine. Finally, the newly mapped task is removed from T, and the process repeats until all tasks have been mapped [14, 15].

Table 1
Summary comparison table of the Min-Min, QoS Min-Min, and Max-Min scheduling algorithms

Algorithm	Advantages	Disadvantages	Complexity
Min-Min	- Tends to minimize the makespan by prioritizing shorter tasks on faster processors, balancing load.	- Can lead to load imbalance, as longer tasks are often delayed until the end.	- $O(n.m)$, where n is the number of tasks.
	- Simple to implement and effective for heterogeneous systems with varied task sizes.		
QoS Min-Min	- Incorporates Quality of Service (QoS) constraints, ensuring high-priority tasks are scheduled sooner.	- Additional overhead from handling QoS requirements, potentially increasing complexity.	- $O(n^2.m)$, due to QoS calculations.
	- Balances efficiency with service quality, improving task response times for critical tasks.	- May deprioritize low-QoS tasks, leading to potential delays in completing all tasks.	
Max-Min	- Prioritizes longer tasks, helping balance load by assigning longer tasks first.	- Might increase the makespan since shorter tasks are delayed, especially in heterogeneous systems.	- $O(n.m)$, similar to Min-Min
	- Useful for tasks with high variance in execution times, minimizing idle times of processors.		

3. Dynamic Adaptation

In the ever-evolving landscape of distributed systems, optimizing performance while adapting dynamically to changing conditions remains a fundamental challenge.

Distributed systems consist of interconnected and geographically dispersed nodes that collaboratively execute tasks. These systems are characterized by their heterogeneity, dynamic workloads, and varying resource availability. Ensuring optimal performance in such environments requires efficient task scheduling, resource allocation, and real-time adaptability [19].

One of the primary concerns in distributed systems is the optimization of key performance metrics, such as makespan, resource utilization, latency, and throughput. Traditional scheduling algorithms often fail to address the complexities introduced by heterogeneity and dynamic conditions, such as fluctuating workloads or node failures. As a result, they struggle to maintain system efficiency and scalability in real-world scenarios.

Dynamic adaptation, which involves the ability to reallocate tasks and adjust system configurations in response to changes, is essential for enhancing distributed system performance. However, achieving this requires advanced algorithms capable of balancing computational loads, respecting task dependencies, and optimizing resource usage in real-time.

This research addresses these challenges by developing innovative optimization techniques that incorporate dynamic adaptation to improve distributed system performance. The goal is to design methods that can efficiently manage heterogeneous resources, handle dynamic workloads, and minimize execution time while maximizing resource utilization. By tackling these issues, this work contributes to advancing the efficiency, scalability, and reliability of distributed systems in diverse applications, from cloud computing to real-time data processing.

4. Contribution

Dynamic programming methods are frequently employed to tackle discrete optimization problems, as these problems are often NP-hard. Researchers are increasingly drawn to leveraging algorithms and models derived from local search techniques to address the challenges of task scheduling. Consequently, local optimization strategies and dynamic programming approaches [17] have been developed and have proven effective in producing efficient scheduling solutions. This work aims to demonstrate the effectiveness of our approach, which utilizes dynamic programming to solve scheduling problems.

4.1. Dynamic Programming

In the context of task scheduling, dynamic programming (DP) is often used to optimize tasks allocation across processors to minimize the makespan or another objective. Here is a general mathematical model for a dynamic programming algorithm to solve a task scheduling problem on heterogeneous computing systems (HCS), where we aim to minimize the makespan See Definition 2 (*i.e.*, the maximum time to complete all tasks).

The objective is to assign each task to a processor in such a way that the makespan, or the maximum completion time among all processors, is minimized.

4.1.1. Definitions and Notation

Tasks: $T = \{ t_1, t_2, \dots, t_n \}$ represent a set of n tasks, where each task t_i has a specific workload.

Processors: $P = \{ p_1, p_2, \dots, p_m \}$ represent a set of m heterogeneous processors, each with a distinct processing speed.

Processing Times: For each task t_i and processor p_j, let $c_{i,j}$ denote the processing time required to complete task t_i on processor p_j. This processing time depends on both the task's workload and the processor's speed. We define it as:

$$c_{i,j} = \frac{w_i}{s_j} \quad (1)$$

where:
w_i is the workload of task t_i.
s_j is the speed of processor p_j.

Decision Variable: Let $x_{i,j}$ be a binary decision variable such that:

$$x_{i,j} = \begin{cases} 1 & if\ the\ task\ t_i\ is\ assigned\ to\ the\ processor\ p_j \\ 0 & otherwise \end{cases}$$

Makespan: The **makespan** C_{max}, also known as the total schedule length or the completion time, is defined as the maximum finish time (FT) of the last task in the schedule across all processors. It represents the time required to complete all tasks, as shown in Equation (1,2) [18].

$$C_{max} = \max(FT(Exit_{Task})) \quad (2)$$

where:
FT: is the finish time of task i,
C_{max}: is the overall makespan or total schedule length.
$Exit\_Task$: is the last scheduled task.

4.1.2. Mathematical Model

Let T represent a set of tasks to be scheduled, and T_k be the subset of tasks assigned to processor P_m. T_k is the optimal subset of tasks that satisfies the recursive relation provided in Equation (3).

Recursive Relation

The recursive relation captures the trade-off between assigning tasks to different processors to balance the workload. For each task t_i and processor p_j, the recursive relationship can be expressed as:

$$DP(i,j) = \min(\max(DP(i-1,j), c_{i,j})) \quad (3)$$

where:
$DP(i-1,j)$: The makespan when assigning $i-1$ tasks among j processors.

Boundary Conditions

- Base Case:
  If there are no tasks $(n = 0)$, $DP(0,j) = 0$ for all j .
- Single Processor Case:
  If there is only one processor $(j = 1)$, $DP(i,j) = \sum_{j=1}^{i} c_{i,1}$ the cumulative time to execute all tasks sequentially on p_1 .

Objective Function

The goal is to minimize the maximum completion time across all processors, hence the makespan C_{max}:

$$C_{max} = \min_{j}(DP(i,j)) \quad (4)$$

4.2. Dynamic Adaptation vs Dynamic Programming

Dynamic adaptation in multiprocessor systems refers to the ability of the system to adjust task allocation and resource utilization in real-time to respond to changes in workload, resource availability, or system states. It is a critical feature for maintaining efficiency and performance in heterogeneous environments where tasks may have varying execution requirements and system conditions can fluctuate dynamically.

Dynamic programming (DP) serves as an effective approach to achieving dynamic adaptation in task scheduling for multiprocessor systems. DP breaks the scheduling problem into smaller, manageable subproblems and solves them incrementally, ensuring that each decision contributes to the global optimization of performance metrics, such as makespan or resource utilization. This recursive method allows the system to re-evaluate and adapt task assignments as conditions change, ensuring optimal decisions are made even in dynamic environments.

The relationship between dynamic adaptation and dynamic programming lies in DP's inherent flexibility and systematic approach to optimization. By leveraging DP, task scheduling algorithms can:

1. **Handle Dynamic Workloads**: Reassign tasks dynamically to processors based on real-time system states, ensuring balanced workloads and efficient processor utilization.
2. **Optimize in Real-Time**: Continuously adjust the scheduling solution as new tasks arrive or as resources become available or fail, maintaining system responsiveness.
3. **Integrate Task Dependencies**: Account for task precedence and succession constraints while dynamically adapting schedules, ensuring correct task execution order.
4. **Reduce Decision Complexity**: By breaking the global scheduling problem into smaller subproblems, DP simplifies the process of making optimal decisions in complex, dynamic systems.

In summary, dynamic programming provides the computational framework to implement dynamic adaptation in multiprocessor systems. Its ability to adaptively and efficiently allocate tasks in response to changing conditions makes it a foundational tool for achieving real-time optimization and robustness in these environments.

4.3. Proposed Approach

Based on the scheduling model (See Definition 1), our approach, Dynamic Tasks Scheduling Approach Knapsack based DyTAg is a scheduling algorithm that performs in a heterogeneous computing system. By means of dynamic programming, DyTAg combines between resource optimisation and completion time minimisation.

To apply dynamic programming (DP) to minimize the makespan (See Figure 1), we define a DP state that keeps track of the minimum achievable makespan up to each task assignment. The goal is to incrementally assign tasks to processors in a way that minimizes the maximum load across all processors. The method process steps are outlined as follows:

State Definition

Let $DP(i,j)$ represent the minimum possible makespan after assigning the first i tasks across j processors.

Recurrence Relation

Base Case: $DP(0,j) = 0 \; \forall \; j = 1,2,\dots,m.$ This means that if there are no tasks, the makespan is zero on all processors.

Recursive Case:

For each task T_i and each processor P_j , the DP formula to update $DP(i,j)$ is based on the workload of assigning T_i to P_j and then finding the maximum time required by any processor:

$$DP(i,j) = \min_{k=1}^{m}(\max(DP(i-1,k), DP(i,k) + c_{i,j})) \quad (5)$$

Here, the inner maximum accounts for the load balancing effect of assigning task T_i to processor P_j ,

and the outer minimum ensures that we find the minimum makespan across all valid assignments.

Solution

The solution is obtained by performing the following calculation:

$$C_{max} = \min_{j}(DP(n,j)) \quad (6)$$

where: $DP(n,j)$ gives the makespan after all n tasks are assigned across m processors.

Algorithm 1 DyTAg

Input: $T = \{t_1, t_2, \ldots, t_n\}$ (Set of tasks)
Input: $P = \{p_1, p_2, \ldots, p_m\}$ (Set of processors)
Input: $C[i][j]$ (Processing time for task t_i on processor p_j)
Output: C_{max} (Minimum achievable makespan)
Initialize DP table: Let $DP[i][j]$ represent the minimum makespan when assigning the first i tasks among j processors.
Set $DP[0][j] = 0$ for all j (no tasks assigned, so makespan is 0)
for $i = 1$ to n **do** ▷ Loop through each task
  for $j = 1$ to m **do** ▷ Loop through each processor
    $DP[i][j] = \infty$ ▷ Initialize DP value for the current state
    for $k = 1$ to j **do** ▷ Loop through all processor options up to j
      ▷ Calculate makespan by assigning task i to processor k
      makespan = $\max(DP[i-1][j], DP[i][k] + C[i][k])$
      $DP[i][j] = \min(DP[i][j], \text{makespan})$
    end for
  end for
end for
▷ The final result is the minimum makespan achievable using all tasks and processors
$C_{max} = DP[n][m]$
return C_{max}

Figure 1: DyTAg algorithm for minimizing makespan in a heterogeneous multiprocessor system

4.3 Assumptions

In our approach, we assume that:

1. All tasks are non-preemptive, meaning that once a task begins execution on a processor, it runs to completion without interruption.
2. The execution costs of tasks are known and deterministic.
3. The processors have varying speeds.
4. There are no precedence constraints between tasks.

5. Results and Discussion

The experimental section of the paper primarily focuses on the performance differences between our proposed technique and existing scheduling approaches. In addition, these tests highlight the importance of dynamic adaptation to reduce schedule length, particularly in Heterogeneous Computing Systems (HCS), where every performance metric significantly impacts the overall process. This section includes a comparison of our approach with traditional methods such as Min-Min and QoS Guided Min-Min. The impact of dynamic adaptation is also emphasized, demonstrating how real-time adjustments to task allocation and resource utilization improve system performance under varying workloads and resource conditions. Furthermore, the relevance of using dynamic programming with the DyTAg algorithm is highlighted, as it enables the system to adapt efficiently to dynamic environments by incrementally optimizing task scheduling, thus ensuring the best possible outcome in changing conditions. Finally, the results demonstrate improvements over well-established and recent approaches. Each test is conducted within a real-time context to ensure that the results closely reflect actual system environments.

5.1. Comparison Metrics

The scheduling approaches in this study are evaluated using the following metrics.

Execution Time

$\Delta time$ to calculate algorithm execution time for performance's purposes.

$$\Delta time = \text{FinishTime} - \text{BeginTime} \quad (7)$$

Makespan also defined as schedule length (See Definition)

Processors' utilisation Ratio (PU_{Ratio})

Described as the difference utilisation rate between the most utilised and the less utilised processor divided per the processor's number.

$$PU_{Ration} = \frac{C_{max},- Min(processor\_Load_j)}{|P|} \quad (8)$$

5.2. Experimental Results

In this section we aim to highlight the differences that can be noticed, this process is done by comparing DyTAg results to Min-Min and QoS guided Min-Min. Aforementioned purposes above, we consider the following example summarised below (Table 2).

Table 2
Tasks-set execution costs on three processors p_1, p_2, p_3

Processors	t_1	t_2	t_3	t_4	t_5	t_6	t_7
p_1	76	56	23	29	10	40	29
p_2	98	90	54	50	22	65	76
p_3	82	68	40	43	17	51	45

$$C_{max} = \min_{j}(DP(7,j)) \Rightarrow C_{max} = 115$$

The results are compared with Min-Min [15] and QoS guided Min-Min [3] in the following figure:

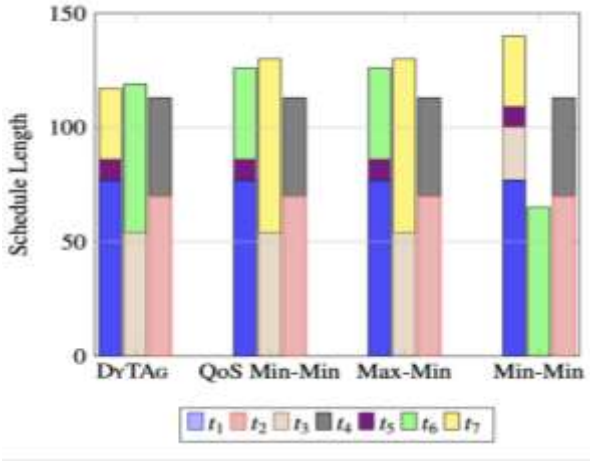

Figure 2: Min-Min, Max-Min, QoS guided Min-Min and DyTAg makespan comparison

In this comparison, the results demonstrate that the proposed approach outperforms others, with DyTAg achieving a makespan of 115 for the given set of tasks, showing its effectiveness.

This result is reached by assigning t_3, t_6 to processor p_2 and t_2, t_4 to processor p_3 and the remaining tasks t_1, t_5, t_7 to the last processor p_1. Min-Min gives a makespan of 140 while QoS Guided Min-Min and Max-Min give both a makespan of 130, which results in a gain of 15% in this test with DyTAg.

It is noted that complexities of DyTAg, Max-Min equal $O(n.m)$ and $O(n^2.m)$ respectively where N is the number of tasks, m number of processors and. This complexity disparity results to a better DyTAg's performance then other algorithms, effectively in this example the complexity of DyTAg = 833 while the complexity of Max-Min = 1029, as a result $\Delta Time$ enhanced by 19.04%

In the other hand, processors utilisation rate for our approach shows better results, PU_{Ratio} (DyTAg) = 2 while PU_{Ratio} (Min-Min) = 25, the obtained result shows that DyTAg has a better resources management with fair tasks distribution compared to the other heuristics.

6. Conclusion

In this paper, we have introduced first a task scheduling algorithm for HCS called DyTAg, which is an approach based on dynamic programming applied for task allocation.

The algorithm follows a systematic approach to task allocation, which includes the following key steps: (1) identifying and categorizing tasks based on their computational requirements, (2) selecting appropriate processors based on task characteristics and processor capabilities, (3) dynamically assigning tasks to processors while minimizing idle times, and (4) adjusting the schedule to ensure an optimal balance of load across processors. Our analysis demonstrated that DyTAg effectively reduces the overall makespan and improves task scheduling efficiency in heterogeneous environments.

The process of our approach demonstrates how it minimizes computation time, making DyTAg a more efficient task scheduling solution compared to other algorithms. The performance of DyTAg is compared with well-known algorithms such as Min-Min, QoS Guided Min-Min, and Max-Min. The comparison metrics are based on examples from related work that consider heterogeneous multiprocessor systems with various task sets. As a result, DyTAg shows improved performance in terms of makespan and processing efficiency, outperforming the Min-Min algorithm.

Future work could focus on further optimizations, such as incorporating dynamic QoS constraints or addressing challenges in real-time scheduling scenarios.

Conflicts of interest

The authors declare no conflict of interest.

References

[1] Gallet, M., Marchal, L., Vivien, F. (2009) Efficient scheduling of task graph collections on heterogeneous resources. In: 2009 *IEEE International Symposium on Parallel & Distributed Processing* [Internet]. Rome, Italy: IEEE; 2009 [cited 2023 Oct 23]. p. 1–11. Available from: http://ieeexplore.ieee.org/document/5161045/

[2] Ullman, JD. (1975) NP-complete scheduling problems. *J Comput Syst Sci*. 10 (3), 384–93. https://doi.org/10.1016/S0022-0000(75)80008-0

[3] He, X., Sun, X., Von Laszewski, G. (2003) QoS guided Min-Min heuristic for grid task scheduling. *J. Comput. Sci. Technol*. 18 (4), 442–51. [Internet] http://www.cs.iit.edu/~scs/assets/files/jcst_XHe-5-28.pdf

[4] Durillo, J. J., Prodan, R., Barbosa, J.G. (2015). *Simul. Model. Pract. Theory*. 58 (part 1), 95–111. https://doi.org/10.1016/j.simpat.2015.07.001

[5] Alkhanak, E.N., Lee, S.P., Rezaei, R., Parizi, R.M. (2016) Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues. *J. Syst. Softw*. 113: 1–26. https://doi.org/10.1016/j.jss.2015.11.023

[6] Panda, S.K. and Jana, P.K (2016) Uncertainty-Based QoS Min–Min Algorithm for Heterogeneous Multi-cloud Environment. *Arab J. Sci. Eng*. 41 (8), 3003–25. https://doi.org/10.1007/s13369-016-2069-7

[8] Li, J., Qiu, M., Ming, Z., Quan, G., Qin, X., Gu Z. (2012) Online optimization for scheduling preemptable tasks on IaaS cloud systems. *J. Parallel. Distrib. Comput*. 72 (5), 666–77. https://doi.org/10.1016/j.jpdc.2012.02.002

[9] Min-You Wu, Wei Shu and Zhang, H. (2000) Segmented min-min: a static mapping algorithm for meta-tasks on heterogeneous computing systems. In: "Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat NoPR00556)" [Internet]. Cancun, Mexico: IEEE Comput. Soc; 2000 [cited 2023 Oct 23]. p. 375–85. Available from: https://doi.org/10.1109/HCW.2000.843759

[10] Ezzatti, P., Pedemonte, M., Martín, A. (2013) An efficient implementation of the Min-Min heuristic. *Comput. Oper. Res*., 40 (11), 2670–6. https://doi.org/10.1016/j.cor.2013.05.014

[11] Pedemonte, M., Ezzatti, P., Martín A. (2026) "Accelerating the Min-Min Heuristic". In: Wyrzykowski R, Deelman E, Dongarra J, Karczewski K, Kitowski J, Wiatr K, editors. "Parallel Processing and Applied Mathematics" [Internet]. Cham: Springer International Publishing; 2016 [cited 2023 Oct 23]. p. 101–10. (Lecture Notes in Computer Science; vol. 9574). Available from: http://link.springer.com/10.1007/978-3-319-32152-3_10

[12] Gupta I., Kumar M.S., Jana P.S. (2018) Efficient Workflow Scheduling Algorithm for Cloud Computing System: A Dynamic Priority-Based Approach. *Arab J. Sci. Eng*., 43 (12), 7945–60. https://doi.org/10.1007/s13369-018-3261-8

[13] Etminani, K. and Naghibzadeh, M. (2007) "A Min-Min Max-Min selective algorihtm for grid task scheduling". In: 2007 3rd IEEE/IFIP International Conference in Central Asia on Internet [Internet]. Tashkent: IEEE; 2007 [cited 2023 Oct 23]. p. 1–7. Available from: http://ieeexplore.ieee.org/document/4401694/

[14] Stützle, T. and H.H. Hoos. (2000) Max-Min Ant System. *Future Gener Comput Syst*., 16 (8), 889–914. https://doi.org/10.1016/S0167-739X(00)00043-1

[15] Freund, R.F., Gherrity, M., Ambrosius, S., Campbell, M., Halderman, M., Hensgen, D. *et al.* (1998) Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In: Proceedings Seventh Heterogeneous Computing Workshop (HCW'98) [Internet]. Orlando, FL, USA: IEEE Comput. Soc; 1998 [cited 2023 Oct 23]. p. 184–99. Available from: doi: 10.1109/HCW.1998.666558

[16] Braun, T.D, Siegal, H.J., Beck, N., Boloni, L.L., Maheswaran, M., Reuther, A.I. *et al.* (1999) A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In: Proceedings Eighth Heterogeneous Computing Workshop (HCW'99) [Internet]. San Juan, Puerto Rico: IEEE Comput. Soc; 1999 [cited 2023 Oct 23]. p. 15–29. Available from: doi: 10.1109/HCW.1999.765093

[17] Zouache, D., Moussaoui, A., Ben Abdelaziz F. (2018) A cooperative swarm intelligence algorithm for multi-objective discrete optimization with application to the knapsack problem. *Eur. J. Oper. Res*. 264 (1), 74–88. https://doi.org/10.1016/j.ejor.2017.06.058

[18] Galimyanova, N.N. (2008) Experimental investigations of combined algorithms of branch and bound method and dynamic programming method for knapsack problems. *J. Comput. Syst. Sci. Int*., 47 (3), 422–8. https://doi.org/10.1134/S106423070803012X

[19] Saho, R.M. and Kumari Padhy, S. (2022) A novel algorithm for priority-based task scheduling on a multiprocessor heterogeneous system. *Microprocess. Microsyst*., 95, 104685. https://doi.org/10.1016/j.micpro.2022.104685