DOI 10.34185/1562-9945-6-155-2024-16
UDC 519.8:519.178

O. Maliienko, V. Turchyna

ANALYSIS OF THE IMPACT OF TASK PRIORITIZATION LISTS ON THE POTENTIAL FOR AVOIDING ANOMALIES IN TASK SCHEDULING

Annotation. This paper addresses relevant issues related to the anomalous deterioration in objective function values when attempting to improve the initial parameters in one of the discrete optimization problems. The primary focus is on investigating the conditions under which it is possible to prevent the occurrence of such anomalies. Contemporary scientific works devoted to schedule optimization and task prioritization management, particularly for location-allocation problems arising in the fields of computer science, engineering, and operations research, are reviewed. A priority dynamic redistribution algorithm is proposed, which allows minimizing delays and ensuring efficient resource utilization during parallel task execution. An example of applying the algorithm is provided, and its effectiveness in preventing anomalies.
Keywords: schedule theory, optimal orderings, location-allocation problems, discrete optimization, anomalies, priority dynamic redistribution, scheduling algorithms, interruptions, mathematical modeling.

In the modern world of technology, which is continuously evolving and integrating into all areas of activity, effective task management is a key aspect of achieving high levels of productivity. However, alongside the development of systems, numerous challenges arise in their improvement and optimization, which require innovative mathematical approaches and solutions. One of the most complex problems in this area is the anomalous deterioration of objective function values when attempting logical improvements to the initial system parameters.

Problem statement. The study of anomalous cases in discrete optimization problems, particularly in constructing parallel optimal orderings, is often associated with determining the conditions under which it is possible to prevent the occurrence of these anomalies [1]. Considering the generalization of the problem of minimizing the total task completion time, an interesting question arises regarding the impact of interruptions on the value of the objective function [2].

Analysis of the latest research and publications. In recent decades, a large number of scientific works have been devoted to schedule optimization and task prioritization management in various fields such as computer science, engineering, and operations research. These works emphasize the importance of ensuring efficient resource allocation and minimizing delays in task execution. In particular, significant attention is paid to algorithms that allow dy-

© Maliienko O., Turchyna V., 2024

namic resource redistribution and account for changes in the execution environment in real time.

One of the key studies is the work of Avinab Marahatta and co-authors [3], which proposes a classification scheme for dynamic task scheduling in virtualized cloud data centers, improving energy efficiency and task execution by dynamically redistributing resources and adapting to changes in system load.

In [4], a review of 67 different task scheduling algorithms is conducted, particularly those optimizing energy consumption in cloud environments. The advantages and disadvantages of various approaches, including dynamic task redistribution algorithms that consider priorities and variable execution conditions, are discussed.

The purpose of the study. The purpose of this research is to develop and justify the effectiveness of a priority dynamic redistribution algorithm that prevents the occurrence of anomalies in parallel task execution. By allowing interruptions and dynamic redistribution, the algorithm aims to optimize task execution order, minimize delays, and ensure efficient resource utilization. The task is to establish the conditions under which the proposed algorithm guarantees the execution of more important tasks first, allows more prioritized tasks to be placed in the leftmost permissible positions, and considers changes in the availability of executors and the state of task execution in real time.

Presentation of the main material of the research. When studying anomalous cases, questions often arise about determining the conditions under which it is possible to prevent the occurrence of these anomalies. In particular, the study of the impact of interruptions on the increase in the objective function value becomes interesting.

Let a directed graph $G = (V, U)$ be given, along with a priority list $L = (i_1, \dots, i_n)$ and vertex weights $T = (\tau_1, \dots, \tau_n)$. We introduce additional initial conditions: constraints on the ordering width with the set of values h_i and the allowance for interruptions during work execution [2]. The constructed parallel ordering should satisfy the priority list and have a minimal length.

Statement. Allowing interruptions for some tasks in cases where anomalies exist helps avoid their occurrence under the following conditions:

Priority: The priority list must be built according to the optimal priority list construction algorithm; high-priority tasks can interrupt the execution of lower-priority tasks.

Interruption: An interrupted task can be resumed at any step, but according to the priority list.

Dynamic Actualization: At each step of task execution, work can be interrupted and tasks can be redistributed.

Proof.

Priority. Let a priority list be given, built according to the optimal priority list construction algorithm. We will prove that this condition guarantees the execution of more important tasks first. Suppose that a high-priority task (for example, task *i*) can interrupt a lower-priority task (for example, task *j*). According to this condition, task i is always executed before task *j*

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

if they are simultaneously available for execution. This means that a high-priority task will never be blocked by a lower-priority task, which prevents delays in the execution of critically important tasks.

Interruption. Consider the condition that an interrupted task can be resumed at any step, but according to the priority list. We will prove that this allows more prioritized tasks to be placed in the leftmost permissible positions. Let task i be interrupted by task j with a higher priority. Since task i can be resumed later, but only after all higher-priority tasks have been completed, this ensures that task i does not occupy the executor at a moment when it is possible to execute task j. Thus, interruptions allow more prioritized tasks to be placed in the leftmost permissible positions in the schedule, reducing the overall execution time and preventing anomalies.

Dynamic Actualization. Consider the condition that at each step of task execution, work can be interrupted and tasks can be redistributed. We will prove that this condition accounts for changes in the availability of executors and the state of task execution in real time. Suppose task i is being executed, but at the current step, a higher priority task j appears. Dynamic actualization allows the immediate suspension of task i and the assignment of resources to execute task j. Thus, interruptions and task redistributions account for changes in the availability of executors and the state of task execution in real time, allowing for a prompt response to changing conditions and avoiding delays. This also helps prevent anomalies by ensuring efficient resource use and task execution in order of importance.

Condition 1 guarantees the execution of more important tasks first.

Condition 2 allows more prioritized tasks to be executed in the leftmost permissible positions [5].

Condition 3 accounts for changes in the availability of executors and the state of task execution in real time. We will illustrate the fulfillment of these conditions with the following example.

Example. Graph G is given (fig. 1),

$$L = (1,2,3,4,5,6,7,8,9,10),\ \mathrm{T} = (4,4,2,2,1,3,4,2,1,7),\ h = 3.$$

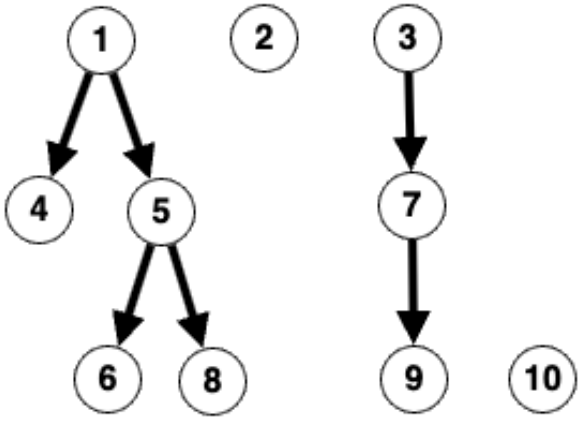

Figure 1 - Graph G

Table 1

Optimal ordering

1	1	1	1	4	4	8	8						
2	2	2	2	5	6	6	6						
3	3	7	7	7	7	9	10	10	10	10	10	10	10

The length is $l = 14$.

Table 2

Optimal ordering with interruptions

1	1	1	1	4	4	7	9						
2	2	2	2	5	7	8	8						
3	3	7	7	6	6	6	10	10	10	10	10	10	10

The length of the ordering did not change, $l = 14$.

Apply the dynamic resource redistribution algorithm to solve this problem. Modify L, using the optimal priority list construction algorithm.

Find the total task completion time for each of the graph paths:

$l_{1-4} = 5, l_{1-6} = 8, l_{1-8} = 7, l_2 = 4, l_{3-9} = 7, l_{10} = 7.$

Update the priority list: $L' = (1,5,6,8,3,7,9,10,4,2).$

Table 3

Optimal ordering using list L' and the possibility of interruptions

1	1	1	1	5	6	6	6	10	2	2	2
3	3	7	7	7	8	8	10	4			
10	10	10	10	10	7	9	4	2			

The length of the ordering is $l = 12$, which is 14.3% shorter than the initial result.

Let's take a closer look at the fulfillment of conditions 1-3 on the constructed ordering from table 3.

Condition 1: task i_{10} is interrupted by task i_7, and task i_7 is interrupted by task i_8.

Condition 2. Find $\underline{S}$ (table 4), $\bar{S}$ (table 5) and the sets of allowable places for vertices i_5 and i_7.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Table 4

Ordering $\underline{S}$

1	1	1	1	4	4		
2	2	2	2	5	6	6	6
3	3	7	7	7	7	9	
10	10	10	10	10	10	10	
					8	8	

Table 5

Ordering $\bar{S}$

	10	10	10	10	10	10	10
	3	3	7	7	7	7	9
						8	8
1	1	1	1	5	6	6	6
						4	4
				2	2	2	2

$\mu_5 = [5], \mu_7 = [3,7]$. Task i_5 is performed in the only allowable place. Task i_7 starts execution at the leftmost allowable place due to the impossibility of executing higher priority tasks at time t_3, and after the interruption, it starts execution at the leftmost allowable place according to the priority list.

Condition 3: task i_7 is interrupted by task i_8, while on the next step, task i_7 interrupts task i_{10}.

The pseudocode of the algorithm for solving this problem in Python:

```
#Priority Dynamic Redistribution Algorithm
#Initialization: Mark all executors as free
executors = [free for _ in range(total_executors)]
tasks = priority_queue  #Assume this is a list of tasks sorted by priority

while tasks:
    #Assign work: Assign available task i to a free executor
    for task in tasks:
        if task.available:
            for executor in executors:
                if executor == free:
                    executor = task
                    task.assigned = True
                    break

    #Redistribution: Check for higher priority tasks
    for executor in executors:
```

```
            if executor != free:
                current_task = executor
                for task in tasks:
                    if task.priority < current_task.priority and
task.available:
                        #Interrupt current task and assign higher
priority task
                        executor = task
                        task.assigned = True
                        current_task.interrupted = True
                        break

        #Update the list of tasks: remove completed tasks, update
availability
        tasks = [task for task in tasks if not task.completed]
        #Mark all free executors
        for i in range(len(executors)):
            if executors[i].completed:
                executors[i] = free
    #End of Algorithm
```

Let's consider the operation of the algorithm step by step.

Initialization: All executors are initially marked as free.

Job Assignment: Iterate through the list of tasks (assumed to be sorted by priority) and assign available tasks to free executors.

Redistribution: For each executor who is currently working on a task, check if a higher priority task is available. If a higher priority task is found, interrupt the current task and assign the higher priority task to the executor.

Task Update: Remove completed tasks from the list and update the availability of tasks and executors.

Free Executors: Mark executors as free if they have completed their assigned tasks.

This pseudocode provides a clear and structured way to manage tasks based on their priority and dynamically reassign executors to higher priority tasks as they become available.

Conclusions. The results obtained during the research demonstrate that using the priority dynamic redistribution of tasks significantly reduces the risk of anomalies in parallel ordering problems. The proposed algorithm ensures the execution of the most important tasks first and allows them to be placed in the leftmost allowable positions. This ensures optimal resource utilization and minimizing total execution time. Dynamic task distribution updating allows for real-time adjustments to executor availability and task execution possibilities, enabling timely responses to changes in conditions to avoid delays. Thus, applying the proposed approach provides an effective solution for parallel ordering problems, particularly in real-time conditions.

Moreover, it is worth noting several important questions that require further research.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Clarification of algorithm efficiency criteria. It is necessary to explore in more detail which efficiency criteria should be considered when evaluating the performance of the priority dynamic redistribution in real-world systems. For example, it is essential to analyze how the algorithm handles different types of anomalies and which parameters have the most significant impact on its effectiveness.

Adaptation of the algorithm to other problem types. It is worth considering the possibility of adapting the proposed algorithm to other types of discrete optimization problems where anomalies arise. Specifically, this includes location-allocation problems with complex constraints, which occur in various applied fields such as logistics, medicine and finance.

Investigation of the impact of unforeseen events. It is important to study how the proposed algorithm responds to sudden changes in task execution conditions, such as the emergence of new tasks or executor failures. Such studies could enhance the algorithm and make it more adaptable to unforeseen events.

Integration with other optimization algorithms. Such as genetic algorithms, machine learning, or deep analysis methods. This would enable the creation of hybrid algorithms that could more effectively handle complex tasks.

Analysis of implementation costs. It is crucial to explore the implementation costs of the proposed approach in real-world systems. This includes computational resources needed to run the algorithm and the time required to adapt the algorithm to specific operating conditions.

These further research directions will contribute both to new theoretical results for solving some classical discrete optimization problems and to the potential for effective application to important practical tasks.

REFERENCES

1. Graham R. (1969) Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics. Vol. 17. PP. 416–429. DOI: 10.1137/0117039

2. Малієнко О.О., Коваленко Є.О. (2024) Дослідження впливу переривань на виникнення аномалій у задачах паралельного упорядкування. Комбінаторні конфігурації та їхні застосування. Вип. 26. С. 103-107.

3. Marahatta A., Pirbhulal S., Zhang F., Parizi R. M., Choo K.-K. R., Liu Z. (2021) Classification-Based and Energy-Efficient Dynamic Task Scheduling Scheme for Virtualized Cloud Data Center. IEEE Transactions on Cloud Computing. Vol. 9. PP. 1376-1390 DOI: 10.1109/TCC.2019.2918226

4. Ghafari, R., Kabutarkhani, F. H., Mansouri, N. (2022) Task scheduling algorithms for energy optimization in cloud environment: a comprehensive review. Cluster Comput. Vol. 25. PP. 1035-1093 DOI: 10.1007/s10586-021-03512-z

5. Турчина В.А., Федоренко Н.К. (2011) Алгоритми побудов усіх паралельних упорядкувань заданої довжини. Питання прикладної математики і математичного моделювання. С. 268-274.

Received 19.10.2024.
Accepted 25.10.2024.

Аналіз впливу списку пріоритетів виконання завдань на можливість уникнення аномалій у задачах упорядкування

У даній роботі розглянуто актуальні проблеми, пов'язані з аномальним погіршенням значень цільової функції при спробах покращення початкових параметрів в одній із задач дискретної оптимізації. Основна увага приділена дослідженню умов, за яких можливе запобігання виникненню таких аномалій. Розглядаються сучасні наукові роботи, присвячені оптимізації розкладів і управлінню пріоритетами завдань, зокрема для задач розподілу та розміщення, що виникають в галузях комп'ютерних наук, інженерії та операційних досліджень. Запропоновано алгоритм пріоритетного динамічного перерозподілу, що дозволяє мінімізувати затримки та забезпечити ефективне використання ресурсів при паралельному виконанні завдань. Наведено приклад застосування алгоритму та доведено його ефективність у запобіганні виникненню аномалій.

Малієнко Ольга Олександрівна – аспірант Дніпровського національного університету імені Олеся Гончара.

Турчина Валентина Андріївна – кандидат фізико-математичних наук, доцент, завідувач кафедри обчислювальної математики та математичної кібернетики Дніпровського національного університету імені Олеся Гончара.

Olha Maliienko – Postgraduate Student of Oles Honchar Dnipro National University.

Valentyna Turchyna – Candidate of Physical and Mathematical Sciences, Associate Professor, Head of the Department of Computational Mathematics and Mathematical Cybernetics of Oles Honchar Dnipro National University.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)