UDC 004.9:004.021

M. Shmatko, O. Zhurakovska

USE OF SCHEDULE THEORY ALGORITHMS FOR TASK PLANNING AND TIME MANAGEMENT

Abstract: Good time management is a key factor in the success of work tasks and project management. In particular, scheduling tasks are an important part of workflow organization. At the moment, there are many algorithms for solving such tasks, but the algorithms must be customized to specific conditions, while ensuring the quality of the task and performance. In this article, a hybrid algorithm that combines elements of evolutionary and greedy approaches to solve the problems of schedule theory is proposed.

Keywords: schedule theory, task scheduling, evolutionary algorithm, greedy algorithm, time-management.

Introduction

Planning is a decision-making process that is regularly used in many industries and services. It is associated with the allocation of resources to perform tasks over a period of time, and its goal is to optimize one or more objectives [1].

The creation of a task schedule is the result of the planning process. An efficiently created schedule allows you to optimize time and resources, increasing the efficiency and quality of work performed. The tasks of creating schedules and optimizing them are relevant for both individual users and project teams.

The task of creating an effective schedule is used in many areas. In manufacturing industries, it helps to optimize resource and time consumption. For logistics, it helps to allocate time more efficiently, plan deliveries, etc. The most common use of scheduling ideas is in everyday life, when creating individual schedules and timetables.

To solve scheduling problems, the theory of scheduling is used, which provides a wide range of tools for solving these problems. Scheduling theory is a branch of optimization that studies methods and algorithms for efficiently allocating resources over time to achieve certain goals. It covers various aspects of planning and time management, including task scheduling, work allocation, project planning, interval scheduling, etc. [2].

The theory of schedules provides a wide range of algorithms for solving scheduling problems. The choice of an algorithm for solving a particular problem is determined by the given condition and input data, as well as the priority criteria for the algorithm's efficiency. The main criteria are speed and quality of the result. Depending on the requirements for the algorithm, both of the criteria can be selected as priorities.

The greedy scheduling algorithm is a well-known one. It works by iteratively selecting the most efficient option available based on a predefined criterion. This can be, for

 ISSN 1560-8956

example, the earliest start time or the shortest processing time. The algorithm distributes tasks between time intervals without taking into account further consequences. In terms of computing power, this approach is efficient. It is also quite simple to implement, which allows it to quickly generate realistic schedules. However, since it only makes decisions based on local current information, without any forward-looking consequences, greedy algorithms may ignore better solutions that could be obtained by considering the entire problem space [3]. Therefore, although useful in certain circumstances, they may require additional refinement or optimization to improve the results obtained.

In turn, the evolutionary or genetic algorithm was inspired by the ideas of biological evolution and genetics. To solve a problem, the algorithm imitates the processes of natural selection: in the first step, a population of potential solutions is randomly created. Then, over several “generations,” the algorithm selects and combines the best of them, i.e., those that are most adaptable or best meet the criteria. This mimics the biological processes of selection, mutation, and crossbreeding. This iterative process makes it possible to find an optimal or close to optimal schedule for the requirements [4].

This article is devoted to solving the task of scheduling. The set of tasks of the performer is given, from which it is necessary to form a schedule for the day so that the working day is completely filled with tasks. Tasks are penalized for not being added to the schedule and delayed compared to the target date. The contractor also determines the priority of the task and the importance of meeting the deadline. It is necessary to create a schedule for which the total penalty will be minimal.

Problem formulation

Let the set of tasks T=$\{T_i\}$, i= $\underline{1,m}$ be given. Given the length of the interval t, the schedule should be drawn up on the interval from 0 to t. For each task there are the duration of the task t_i (t_i=$\underline{1,3}$), priority p_i (p_i=$\underline{1,5}$), directive term d_i (d_i=$\underline{9,22}$) and weighting coefficient of delay relative to directive term α_i (α_i=$\underline{0,1}$). The final schedule may or may not include tasks. Let R be the set of tasks included in the schedule, L be the set of tasks not included in the schedule, R ∪ L = T. After the schedule is formed, the total penalty is calculated using the formula:

$$d = \sum_{i|T_i \in L} p_i + \sum_{i|T_i \in R} \alpha_i (max(0, s_i - d_i)), \qquad (1)$$

where p_k – the priority of the task l_j, s_i - task completion time in the schedule.

The problem has a limitation:

– $\sum_{i|T_i \in R} t_j \leq t$ - the total duration of the tasks included in the schedule should not exceed the length of the interval for which the schedule is being developed;

– $\square - \sum_{i|T_i \in R} t_j < t_{min}$ - the difference between the length of the interval for which the schedule is developed and the total duration of the tasks included in the schedule must be less than the shortest duration of the task from the set of tasks that are not included in the schedule;

- tasks cannot be interrupted by other tasks;
- $t_i \in R \,||\, t_i \in L$ - the final schedule may or may not include tasks.

You need to build a schedule for which the value of the total penalty d is minimal. At the same time, you need to take into account all available constraints.

Justification of the chosen method

Problems in the theory of schedules are often considered to be hard combinatorial problems. To solve them, there are various methods, such as greedy algorithm, genetic algorithms, simulated annealing, branch-and-bound method, metaheuristic methods, and others. These approaches cover various aspects of optimization and search for optimal solutions.

One of the most popular approaches to solving problems in the theory of scheduling is to use greedy algorithms [1]. Greedy algorithms work on the principle of choosing the most optimal step at each stage of solving the problem. In the context of schedule theory, this may mean choosing the schedule with the smallest penalty at each step of the schedule construction.

The branch-and-bound method is also used to solve such problems [1]. This method is used to accurately solve combinatorial problems by systematically searching through possible solutions and using upper bounds to eliminate the worst branches of the search tree.

Metaheuristic methods are also well-known [3]. This is a broad class of optimization algorithms that model stochastic processes to find optimal solutions. These include particle swarm algorithms, squeezing algorithms, and others.

To solve this problem, we can use an evolutionary algorithm [3]. To do this, add the parameters n, the number of iterations for the algorithm, and s, the number of initial random distributions. After that, a set of initial schedules $\square = \{\square_1, \dots, \square_\square\}$ is randomly selected. Next, new schedules are generated n times by crossover and mutation of the existing schedules in the set C. At the end of the algorithm, the schedule ci with the lowest total penalty d is selected.

The advantages of this method include efficiency at sufficiently large values of the number of iterations n. However, the method has a significant drawback - due to the large number of iterations, the algorithm takes a relatively long time to execute, and as the number of iterations decreases, the efficiency decreases.

Both of these approaches have their advantages and limitations. Greedy algorithms are fast but do not always provide the optimal solution, while genetic algorithms can be more powerful but require more computing resources.

The proposed method for solving the problem is a hybrid algorithm that combines elements of evolutionary and greedy approaches. This method was chosen because an approximate solution provided in real time will be sufficient to solve the problem. The advantage of this modified method over the evolutionary method is better performance due to fewer iterations and the efficiency of the algorithm. The thesis [5] presents a scheme of this approach, which is applied to the problem of the theory of schedules with a common directive term.

Algorithm for solving the problem

The pseudocode of the proposed hybrid algorithm is as follows:

```
Algorithm (allActivities, n, minHour, maxHour):
Schedules[] allSchedules
Schedule firstSchedule
SORT allActivities by priority
boolean flag = True
integer hour = minHour
WHILE (flag):
Activity activity
if (activity.Duration > maxHour - hour)
        break
hour += activity.Duration
ADD activity to firstSchedule
REMOVE activity from allActivities
if (allActivities.Count = 0)
        flag = False
ENDWHILE
ADD firstSchedule to allSchedules
FOR Count = 1 to n
Schedule mutadedSchedule = Mutation (allSchedules)
ADD mutatedSchedule to allSchedules
Schedule crossowerSchedule = Crossover (allSchedules)
ADD crossowerSchedule to allSchedules
ENDFOR
SORT allSchedules by penalty
RETURN allSchedules[0]
```

At the first stage, a set of schedules is created and an initial schedule is generated using a greedy algorithm. The initial set of tasks is sorted in descending order of priority, and then the tasks are added to the schedule one by one until the entire schedule interval is filled. The second stage involves mutation and crossover of the existing schedules for the number of iterations specified by the task conditions. At the third stage, the set of schedules is sorted by increasing total penalty, and the first schedule of the sorted set is returned as the solution.

Analysis of the results

In this section, we study the efficiency and speed of the proposed hybrid algorithm in comparison with the evolutionary algorithm. The input data are presented in table 1.

Table 1.

Input data

Name	Description	The value of the variable
T	Set of tasks	A randomly generated set of tasks of length m
m	Dimensionality of the set of tasks	10-100
t_i	Duration of the task i, i= $\underline{1, m}$	Random value (t_i=$\underline{1,3}$)
α_i	Weighting coefficient of delay relative to the target date	Random value (α_i=$\underline{0,1}$)
p_i	Priority of the task i, i= $\underline{1, m}$	Random value (p_i=$\underline{1,5}$)
d_i	Directive term of the task i, i= $\underline{1, m}$	Random value (d_i=$\underline{9,22}$)
n	Number of iterations of algorithms	25-150
s	The number of initial random schedules for the evolutionary algorithm	25-150

Let's solve the problem with the given input data using evolutionary and hybrid algorithms. Let n = 25 for both algorithms, s = 25, m = $\underline{10, 40}$. For each value of the problem dimension, 100 experiments are performed for both methods. The average running time of the algorithm and the average total penalty are calculated as the arithmetic mean of all experiments. The results are shown in fig. 1 and fig. 2.

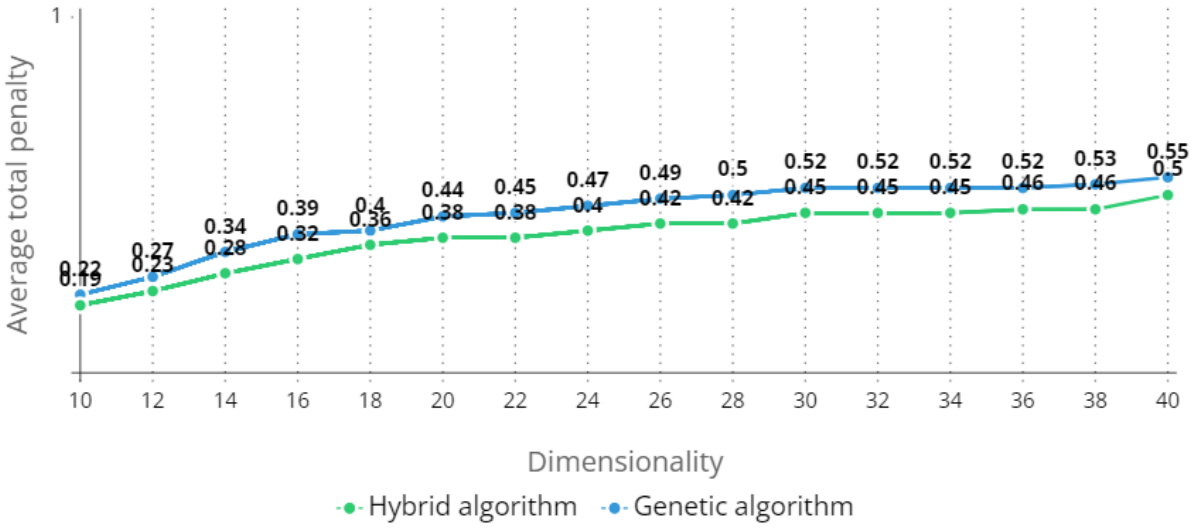

Figure 1. Comparison of work efficiency

ISSN 1560-8956

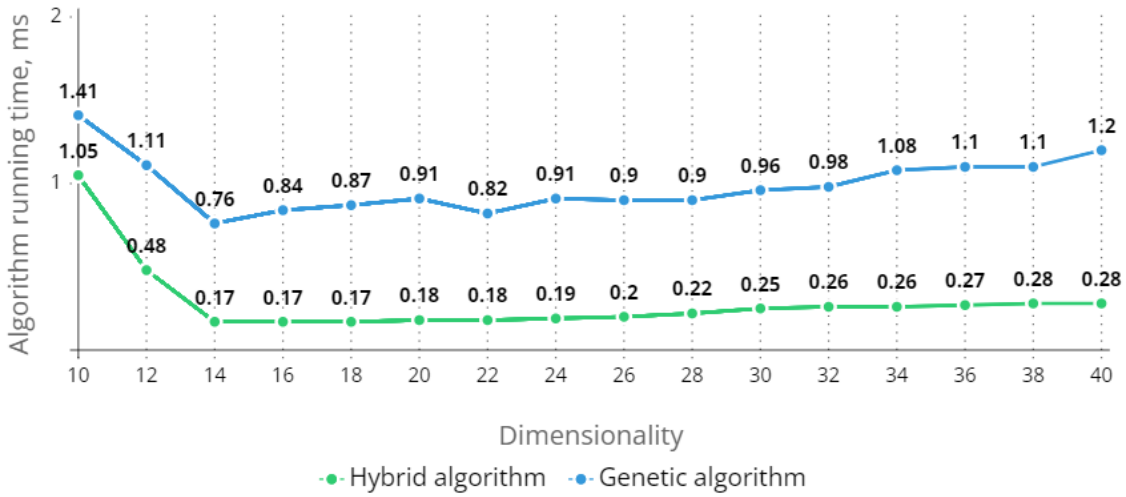

Figure 2. Performance comparison

From the results obtained, we can conclude that the hybrid algorithm shows better results in terms of efficiency, but the difference in performance is significant in favor of the hybrid algorithm. As the problem dimension increases, the average total penalty increases for both algorithms, but the performance does not decrease significantly.

Let's increase the number of iterations for both algorithms (n = 50, s = 50). The results are shown in fig. 3 and fig. 4.

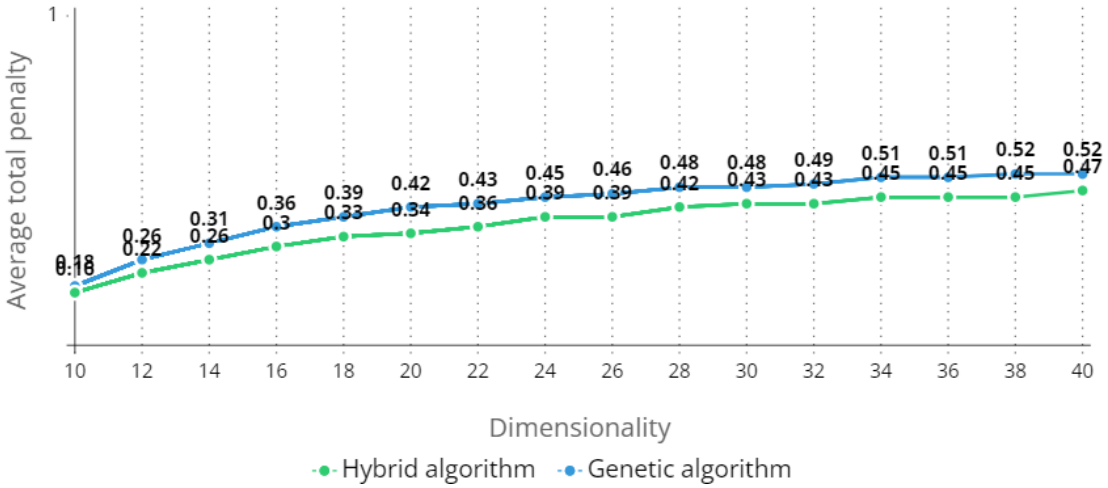

Figure 3. Comparison of work efficiency

From the results obtained, we can conclude that as the number of algorithm iterations increases, the efficiency of the algorithms improves. The difference in the efficiency of the algorithms remains constant and small, but the difference in speed is significant.

Letm = 15, □ = $\underline{10,150}$, □ = $\underline{10,150}$. The results are shown in fig.5 and fig.6.

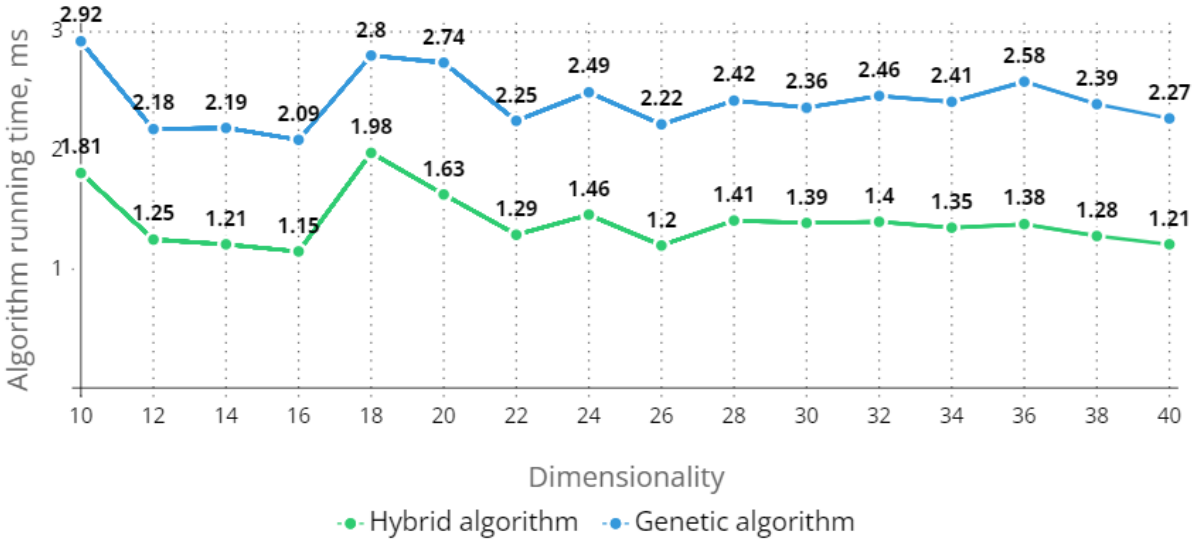

Figure 4. Performance comparison

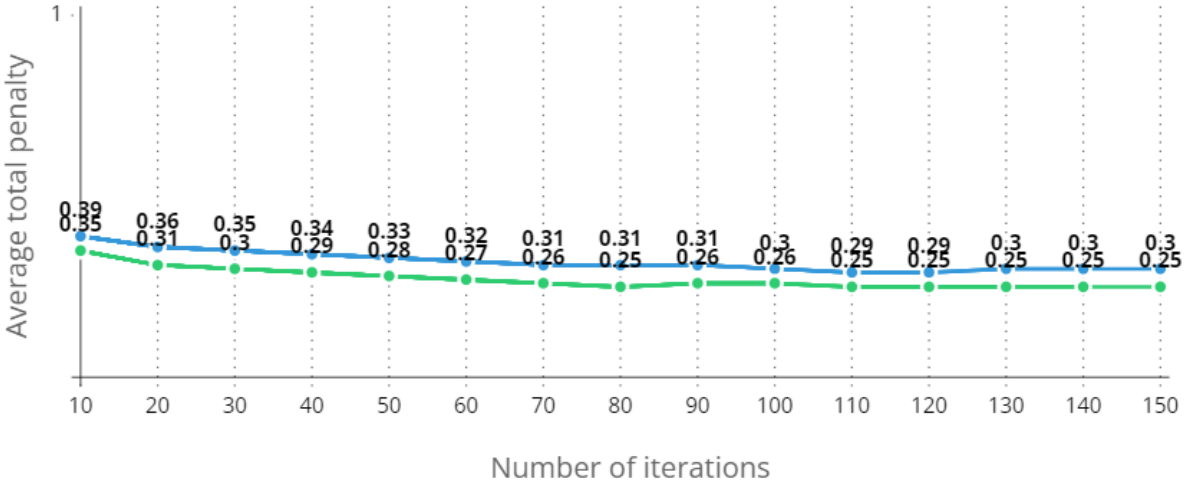

Figure 5. Comparison of work efficiency

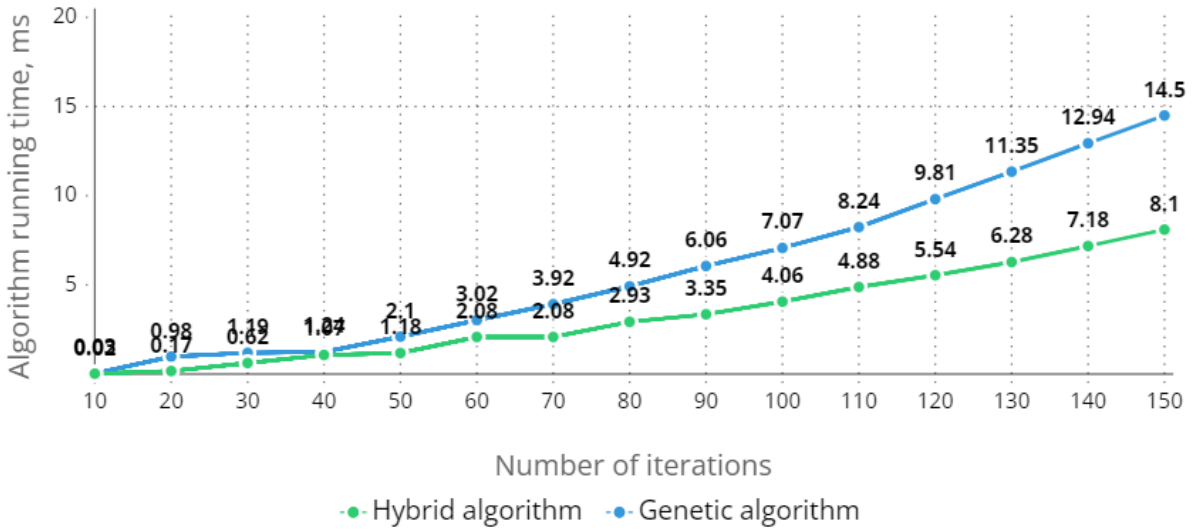

Figure 6. Performance comparison

ISSN 1560-8956

From the results obtained, we can conclude that although the efficiency of both algorithms increases with the number of iterations, this improvement is relatively small (30% for the genetic algorithm and 40% for the hybrid algorithm with an increase in the number of iterations by 1400%). Instead, with an increase in the number of iterations, the performance of the genetic algorithm decreases more significantly than for the hybrid algorithm.

Conclusions

Task planning and time management are important aspects of project management and workflow organization. The use of algorithms allows you to improve and automate the processes of solving scheduling tasks.

The proposed hybrid algorithm, which combines elements of evolutionary and greedy algorithms, allows solving the tasks of scheduling more efficiently and quickly compared to the use of an evolutionary algorithm. In today's world, the performance of the chosen algorithm is an important criterion for assessing the quality of work, so the proposed approach is relevant.

Experiments were conducted to compare the efficiency and speed of the hybrid and evolutionary algorithms. The results obtained allowed us to conclude that the evolutionary algorithm demonstrates lower performance than the hybrid algorithm with the same input data. With a significant increase in the number of iterations of the hybrid algorithm, the efficiency of the work is significantly improved, while the running time of the algorithm remains relatively short compared to the evolutionary algorithm.

It is worth noting that the choice of an algorithm for a scheduling problem depends on the conditions of the problem and the input parameters. Algorithms may be better suited for a certain type of task, but may be inefficient or not suitable for other tasks.

It should also be noted that with minor changes in the initial conditions of the problem, the proposed hybrid algorithm can be modified to solve the problem.

REFERENCES

1. Scheduling Theory, Algorithms, and Systems / Michael L. Pinedo – 2012. – 5 Edition – P. 1-3.

2. Theory of Scheduling / Michael L. Pinedo - 2016. - P. 1-10.

3. Introduction to Algorithms / Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein - 2022 - P. 341-351.

4. Genetic Algorithm in Search, Optimization and Machine Learning / David E. Goldberg – 1989 – P. 1-2.

5. *Shmatko M., Zhurakovska O*. Using theory of scheduling algorithms for tasks planning and time management. Proceedings of the V International Scientific and Practical Conference of Young Scientists and Students “Software Engineering and Advanced Information Technologies (SoftTech-2023)”, Kyiv, 19-21 Dec., 2023. P.385-387.