Task distribution algorithm in distributed team of software developers based on PSO algorithm

Shmatko Oleksandr¹, Gorbach Tetiana², Lobanov Viktor³

¹ PhD, Associate Professor, Associate Professor of the Department of software engineering and intelligent control systems; Technical University «Metinvest Polytechnic» LLC; Ukraine

² PhD, Associate Professor, Associate Professor of the Department of software engineering and intelligent control systems; National Technical University «Kharkiv Polytechnic Institute»; Ukraine

³ Student, Department of software engineering and intelligent control systems; National Technical University «Kharkiv Polytechnic Institute»; Ukraine

Abstract.
Research in the field of software engineering focuses extensively on the software development process. Nowadays, companies offering software development services are engaged in a competition to provide superior quality software products. This competition creates opportunities for enlarging their customer base, expanding the company's scale, and increasing the volume of orders. A crucial factor in achieving these objectives lies in the effectiveness of task allocation within distributed software development teams. Streamlining task allocation facilitates swift product development and ensures the delivery of high-caliber software, thereby minimizing downtime and resource expenditure. In the study titled "Task Allocation Algorithm in Distributed Software Development Teams based on PSO," the potential application of the PSO algorithm for task allocation is examined. The PSO algorithm operates by emulating the behavior of a swarm of particles in pursuit of the optimal solution, rendering it potentially beneficial for optimization tasks amidst uncertain and fluctuating work environments. The primary aim of this research is to assess the efficacy of PSO in enhancing task allocation processes. This endeavor holds promise for reducing planning duration, enhancing productivity, and elevating the quality of software development outcomes.

Keywords:
Task distribution process in software development
PSO
Software development process
**Introduction.** The process of software development is a crucial area of study within the field of software engineering. Nowadays, companies providing software development services compete to deliver higher quality software products. This presents opportunities for expanding their client base, growing the company's size, and increasing the number of orders. Quality attributes such as efficiency, design, reliability, and others contribute to the overall quality of the software development process (SDP) [1, 2]. However, it's important to minimize the time spent on planning and developing software products without compromising quality.

A significant contribution to achieving these goals lies in the quality of task allocation processes within distributed teams of software developers. Efficient task allocation facilitates rapid product development and ensures high-quality resulting software, thereby reducing downtime and the resources required to complete tasks.

The work titled "Task Allocation Algorithm in Distributed Software Development Teams based on PSO" explores the potential application of the PSO algorithm for task allocation. The PSO algorithm is based on mimicking the behavior of a swarm of particles in search of an optimal solution [3], making it potentially valuable for optimization tasks in conditions of uncertainty and changing work environments. The aim of this research is to assess the effectiveness of PSO in enhancing task allocation processes. This could lead to reduced planning time, increased productivity, and improved quality of developed software.

The main research questions in this work concern the possibility of creating an algorithm to improve and automate task allocation within distributed software development teams. It seeks to determine whether this algorithm will be more effective compared to existing methods. Additionally, the possibility of modeling and implementing software components to support this algorithm is also considered.

**Literature review.**

There are approaches to solving the problem. In the article "Task Assignment to Distributed Teams Using a Hybrid Verbal Decision Analysis Methodology" [4], the authors propose a
hybrid VDA (Verbal Decision Analysis) methodology based on the ORdinalCLASSification (ORCLASS) and ZAPROS III-i methods to assist in task allocation in Distributed Software Development (DSD) projects, considering four main task groups in the software development process: requirements, implementation, architecture/design, and testing. The authors describe these methods and provide reports on their usage to address the task allocation problem. They outline all the steps necessary for applying this methodology, including alternative identification, criteria definition, and grouping, among others. The results of their work are also summarized.

In the article "Decentralized Task Allocation Management Structure in Distributed Software Engineering" [5], the TaskAuc framework is proposed for task allocation. Auction and task allocation are performed in a decentralized manner, where all communication and coordination are conducted through blockchain. The project manager (organization representative) is responsible for task announcement through an auction process, checking proposal validity, evaluating proposals, and announcing winners using smart contracts that do not involve dependency on any third parties. The proposed framework models developers as auction participants. A proposal in this system indicates the offer submitted by a developer to demonstrate readiness to perform a specific task. On the other hand, an auction participant is a person (in this case, a developer) who submits the proposal. Project managers or organizations willing to distribute tasks are modeled as auctioneers, and developers willing to choose tasks of their choice are modeled as auction participants. The execution and procedure of this framework are described, and results are provided. It is noted that this framework does not consider the current workload of developers.

In the article "Task Coordination in Agile Distributed Software Development Environment" [6], the NextMove approach is proposed. This method consists of two stages: task prioritization and task allocation. Tasks are prioritized based on three different criteria: task dependencies, schedules, and artifact relationships. Each task is assigned a priority rating for each of the criteria. The three ratings are weighted and summed to obtain the final priority score.
The second stage involves tasks such as matching task requirements to individual skills and balancing workload. Existing solutions offer ways to address the task allocation problem in distributed software development teams. However, they have their drawbacks. None of the options consider changes over time, such as priorities. Therefore, this work will explore the possibility of modeling a task allocation algorithm considering all needs and existing drawbacks of algorithms for this process.

Proposed Model. The problem of task allocation in software development teams involves difficulties related to effective distribution, management, and execution of tasks among team members. Team members may not understand which tasks they are supposed to perform and who is responsible for their execution. The varying levels of qualification and experience among team members can lead to uneven task distribution. During software development, unexpected changes in requirements or priorities may arise [3]. This is not an exhaustive list, but these problems underscore the relevance of this research:

- The need to optimize workflows and ensure efficiency and productivity in a virtual environment requires improvements in task management methods and software development.
- The necessity of automating the process of managing distributed tasks in a software development team to reduce costs and minimize errors.
- The need to introduce standardization for the task allocation process.

The aim of this work is to improve the software development process by implementing an effective algorithm for task allocation among the formed distributed team of developers. To achieve this goal, the following task has been formulated: modeling an algorithm for task allocation in a software development team.

To create an effective algorithm, it is also necessary to formalize the task of task allocation. To do this, attributes that may affect the process need to be identified. It should also be considered that some of these attributes may change
over time. Let’s consider the following input data [2]:
- A list of available tasks (with specified priorities, estimated complexity, and dependencies).
- A list of team members with specified and evaluated skills.
- Next, the available input data should be broken down into the following points:
  - Assigning a skill level to each team member for each task.
  - Tasks may have dependencies on other tasks.
  - Creating a list of communication costs between developers (to take into account consultations, code reviews, etc.).

Using the listed input data, we will convert them into an annotated form. This will yield the following variables:
- \( Y \) is general applicability or objective value;
- \( S \) represents the skill level of a team member for a task;
- \( X \) is a variable indicating the level of effectiveness of assigning a team member to a task;
- \( C_i \) is the cost of communication between team member \( i \) and another team member \( h \);
- \( D_i \) is a variable indicating the level of task dependence on another task \( j \);
- \( \gamma_1 \) and \( \gamma_2 \) are weighting factors to balance the importance of communication costs and levels of dependence;
- \( L \) is the complexity of the task.

\( X_{best} \) is the value for adjusting the number of tasks assigned to one developer. The fitness of each particle in the PSO algorithm is determined by the objective function [7]. The objective function should encompass key aspects of task allocation, taking into account factors such as skill utilization, task dependencies, and communication costs. The following form of the objective function has been proposed for each task (fig. 1):

\[
Y = P + X \times S - \gamma_1 \times \sum C_i - \gamma_2 \times \sum D_i - \sum X_{best} - L
\]
The part $P + X * S$ maximizes overall skill utilization, encouraging task assignments to team members with higher skill levels and also takes into account the importance of the task.

The part $y_1 * \sum_i C_i$ minimizes communication costs by penalizing assignments that involve dependencies and communication between team members.

The part $\sum_i D_i - \sum_i x_{best_i} - L$ includes task dependencies, ensuring that tasks with dependencies are assigned last, adjusting the number of tasks for one developer, and also considers the complexity of the task.

Specific weights and parameters will need to be adjusted based on the characteristics and priorities of the distributed team and tasks.

The operation principle of PSO is based on simulating the social behavior of birds and fish. It was introduced by James Kennedy and Russell Eberhart in 1995. PSO is a population-based algorithm that models the social behavior of individuals, called particles, in the search space to find optimal solutions to a specific problem [8,9].

A basic overview of how PSO works:

- Initialization: A population of particles is randomly generated in the search space. Each particle represents a potential solution to the optimization problem.
- Fitness evaluation: The fitness of each particle is assessed based on its performance in the optimization task. A fitness function quantitatively determines how well the solution satisfies the optimization criteria.
- Updating particle velocity and position: Each particle adjusts its velocity and position based on its own experience, utilizing the fitness function to calculate the next step. The equations for updating velocity and position are typically determined as follows [9] (fig 2, 3).

\[
\begin{align*}
  v_i(t + 1) &= w * v_i(t) + c_1 * r_1 * (pbest_i - x_i(t)) + c_2 * r_2 * (gbest - x_i(t)) \\
  x_i(t + 1) &= x_i(t) + v_i(t + 1)
\end{align*}
\]

Figure 2
Formula for updating particles velocity

Figure 3
Formula for updating particles position

This work is distributed under the terms of the Creative Commons Attribution-ShareAlike 4.0 International License (https://creativecommons.org/licenses/by-sa/4.0/).
Here, \( v_i(t) \) represents the velocity of particle \( i \) at time \( t \), \( x_i(t) \) denotes its position, \( w \) is the inertia weight, \( c_1 \) and \( c_2 \) are acceleration coefficients, and \( r_1 \) and \( r_2 \) are random numbers between 0 and 1.

The main steps of the Task Distribution Algorithm in a distributed software development team based on PSO are as follows:

- **Initialization:** Create a swarm of particles, each representing a potential task allocation to a specific team member.
- **Initialize the positions of particles randomly.** Evaluate the fitness of each particle based on the objective function.
- **For each iteration, update the velocities of particles based on their previous velocities, personal best positions, and best positions within the swarm.** Also, update the positions of particles based on their velocities and evaluate the fitness of each particle.
- **Check for convergence.** Verify convergence conditions (e.g., maximum number of iterations reached or satisfactory fitness achieved).
- **Result:** The best-known solution in the swarm represents the optimized task allocation.

The algorithm continues to iterate, and particles move through the search space, gradually approaching the optimal solution. PSO is particularly useful for continuous and multi-dimensional optimization tasks, and it is relatively easy to implement. However, its performance may vary depending on the choice of parameters [10].

PSO is relatively straightforward to implement and understand. It does not require complex mathematical operations, making it accessible for a wide range of applications. PSO demonstrates global exploration capabilities, allowing particles to explore the entire solution space. This is useful for searching diverse and potentially optimal tasks [11].

PSO is adaptive and can dynamically adjust to changes in the environment. This makes it well-suited for scenarios of dynamic task allocation, where the skill levels, task priorities, and communication costs may change over time. The
parallel nature of PSO ensures efficient implementation on parallel computing architectures, potentially speeding up the optimization process, especially when dealing with a large number of tasks and team members.

Despite its advantages, this algorithm also faces several challenges. PSO may struggle to reach an optimal solution, especially in multi-dimensional and complex solution spaces. Convergence is influenced by parameter settings, so finding the right balance can be challenging. The performance of PSO depends on parameter settings such as inertia weight, acceleration coefficients, and maximum velocities. Tuning these parameters for different problem cases or environments can be time-consuming [12].

Furthermore, there is still an issue with parameter initialization, such as skill levels, task priorities, and communication costs. Addressing this requires further research and development of algorithms that can provide optimal results.

**Conclusion.**

In conclusion, while PSO offers many benefits for task allocation in distributed software development teams, there are still challenges to overcome, particularly regarding parameter initialization and fine-tuning for optimal performance in various scenarios. Continued study and algorithm development are necessary to address these issues effectively. PSO's simplicity in implementation and comprehension makes it accessible across various domains as it doesn't demand intricate mathematical operations. It showcases the ability to explore the entire solution space, thus facilitating the search for diverse and potentially optimal tasks. Moreover, PSO's adaptability enables it to dynamically respond to environmental changes, rendering it suitable for scenarios involving dynamic task allocation. In such contexts, where skill levels, task priorities, and communication costs may fluctuate over time, PSO's ability to adjust proves advantageous.

However, PSO might encounter difficulty in attaining an optimal solution, particularly within multi-dimensional and intricate solution spaces. PSO's effectiveness hinges on parameter settings like inertia weight, acceleration coefficients, and maximum velocities. Adjusting these
parameters for diverse problem scenarios or environments can prove time-intensive. Moreover, there remains a challenge with parameter initialization, encompassing factors such as skill levels, task priorities, and communication costs. Rectifying this issue necessitates ongoing research and the creation of algorithms capable of delivering optimal outcomes.

References:


