

# An Evolutionary Algorithm for Task Scheduling in Crowdsourced Software Development

Razieh Saremi<sup>1</sup><sup>a</sup>, Hardik Yardik<sup>1</sup>, Julian Togelius<sup>2</sup><sup>b</sup>, Ye Yang<sup>1</sup> and Guenther Ruhe<sup>3</sup>

<sup>1</sup>*School of Systems and Enterprises, Stevens Institute of Technology, Hoboken, NJ, U.S.A.*

<sup>2</sup>*Tandon School of Engineering, New York University, NYC, NY, U.S.A.*

<sup>3</sup>*University of Calgary, Calgary, Alberta, Canada*

**Keywords:** Crowdsourcing, Task Scheduling, Task Failure, Task Similarity, Evolutionary Algorithm, Genetic Algorithm.


**Abstract:** The complexity of software tasks and the uncertainty of crowd developer behaviors make it challenging to plan crowdsourced software development (CSD) projects. In a competitive crowdsourcing marketplace, competition for shared worker resources from multiple simultaneously open tasks adds another layer of uncertainty to potential outcomes of software crowdsourcing. These factors lead to the need for supporting CSD managers with automated scheduling to improve the visibility and predictability of crowdsourcing processes and outcomes. To that end, this paper proposes an evolutionary algorithm-based task scheduling method for crowdsourced software development. The proposed evolutionary scheduling method uses a multiobjective genetic algorithm to recommend optimal task start date. The method uses three fitness functions, based on project duration, task similarity, and task failure prediction, respectively. The task failure fitness function uses a neural network to predict the probability of task failure with respect to a specific task start date. The proposed method then recommends the best tasks' start dates for the project as a whole and each individual task so as to achieve the lowest project failure ratio. Experimental results on 4 projects demonstrate that the proposed method has the potential to reduce project duration by a factor of 33-78%.


## 1 INTRODUCTION

Crowdsourced Software Development (CSD) has been increasingly adopted in modern software development practices as a way of leveraging the wisdom of the crowd to complete software mini-tasks faster and cheaper (Stol and Fitzgerald, 2014)(Saremi et al., 2017). In order for a CSD platform to function efficiently, it must address both the needs of task providers as demands and crowd workers as suppliers. Mismatches between these needs might lead to task failure in the CSD platform. In general, planning for CSD tasks is challenging (Stol and Fitzgerald, 2014), because software tasks are complex, independent, and require potential task-takers with specialized skill-sets. The challenges stem from three factors: 1) task requesters typically need to simultaneously monitor and control a large number of mini-tasks decomposed suitable for crowdsourcing; 2) task requesters generally have little to no control over how many and how dedicated workers will engage in their

tasks, and 3) task requesters are competing for shared worker resources with other open tasks in the market.

To address these challenges, existing research has explored various methods and techniques to bridge the information gap between demand and supply in crowdsourcing-based software development. Such research includes studies towards developing a better understanding of worker motivation and preferences in CSD (Faradani et al., 2011)(Difallah et al., 2016)(Yang and Saremi, 2015), studies focusing on predicting task failure (Khanfor et al., 2017)(Khazankin et al., 2011); studies employing modeling and simulation techniques to optimize CSD task execution processes(Saremi, 2018)(Saremi et al., 2019)(Saremi et al., 2021); and studies for recommending the most suitable workers for tasks (Yang et al., 2016) and developing methods to create crowdsourced teams(Wang et al., 2018) (Yue et al., 2015). The existing work lacks effective supports for analyzing the impact of task similarity and task arrival date on task failure. In this study, we approach these gaps by proposing a task scheduling method leveraging the combination of evolutionary algorithms and

<sup>a</sup> <https://orcid.org/0000-0002-7607-6453>

<sup>b</sup> <https://orcid.org/0000-0003-3128-4598>

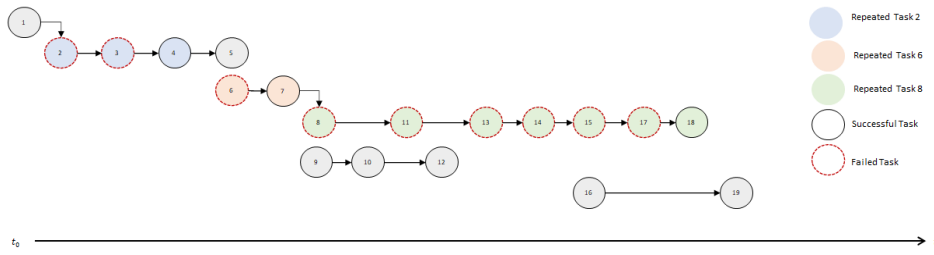


Figure 1: Overview of motivating example.

neural networks.

The objective of this work is to propose a task scheduling recommendation framework to support CSD managers to explore options to improve the success and efficiency of crowdsourced software development. To this end, we first present a motivating example to highlight the practical needs for software crowdsourcing task scheduling. Then we propose a task scheduling architecture based on an evolutionary algorithm. This algorithm seeks to reduce predicted task failure while at the same time shortening project duration and managing the level of task similarity in the platform. The system takes as input a list of tasks, their durations, and their interdependencies. It then finds a plan for the overall project, meaning an assignment of each task to a specific day after the start of the project. Also, the system checks the similarity among parallel open tasks to make sure the task will attract the most suitable available worker to perform it upon arrival. The evolutionary task scheduler then recommends a task schedule plan with the lowest task failure probability per task to arrive in the platform. We applied the proposed scheduling framework to 4 CSD projects in our database. The result confirmed that the proposed method has the potential of reducing the project duration between 33-78%.

The proposed system represents a task scheduling method for competitive crowdsourcing platforms based on the workflow of TopCoder<sup>1</sup>, one of the primary software crowdsourcing platforms. The recommended schedule provides improved duration while decreasing task failure probability and number of open similar tasks upon task arrival in the platform.

The remainder of this paper is structured as follows. Section 2 introduces a motivating example that inspires this study. Section 3 outlines our research design and methodology. Section 4 presents the case study and model evaluation. Section 5 presents a review of related works, and Section 6 presents the conclusion and outlines a number of directions for future work.

<sup>1</sup><https://www.topcoder.com/>

## 2 MOTIVATING EXAMPLE

The motivating example illustrates a real-world crowdsourcing software development (CSD) project on the TopCoder platform. It consists of 19 tasks with a project duration of 110 days. The project experienced a 57% task failure ratio, meaning 11 of the 19 tasks failed. More specifically, Task 14 and 15 failed due to client request, and Task 3 failed due to unclear requirements. The remaining eight tasks (i.e. Task 1, 2, 4, 5, 8, 11, 13, and 17) failed due to zero submissions. An in-depth analysis revealed that the eight failed tasks due to zero submissions were basically three tasks (i.e. tasks 2, 4, and 8) that were re-posted after each failure to be successfully completed as the new task.

As illustrated in Figure 1, Task 2 was cancelled and re-posted as Tasks 5 and 7. It was completed as Task 7 with changes in the monetary prize and task type. Task 4 was cancelled and re-posted as Task 6 which was completed with no modification. Task 8 also failed and re-posted six times as Tasks 11, 13, 14, 15, 17, and 18. Each re-posting modified Task 8 in terms of the monetary prize, task type, and required technology. Finally, the task was completed as Task 18. Studying task 18 revealed that it arrived with 5 similar tasks with similarity degree of 60% in the platform. Also task failure probability on the arrival day for task 18 was 17%.

This observation motivated us to develop the scheduling method presented in this paper. This method can reduce the probability of task failure in the platform, while simultaneously controlling the level of task similarity on task arrival day per project.

## 3 RESEARCH METHODOLOGY

To solve the scheduling problem, we use a genetic algorithm to schedule tasks based on the minimum probability of task failure and degree of similarity among list of parallel tasks in the project. We adapted

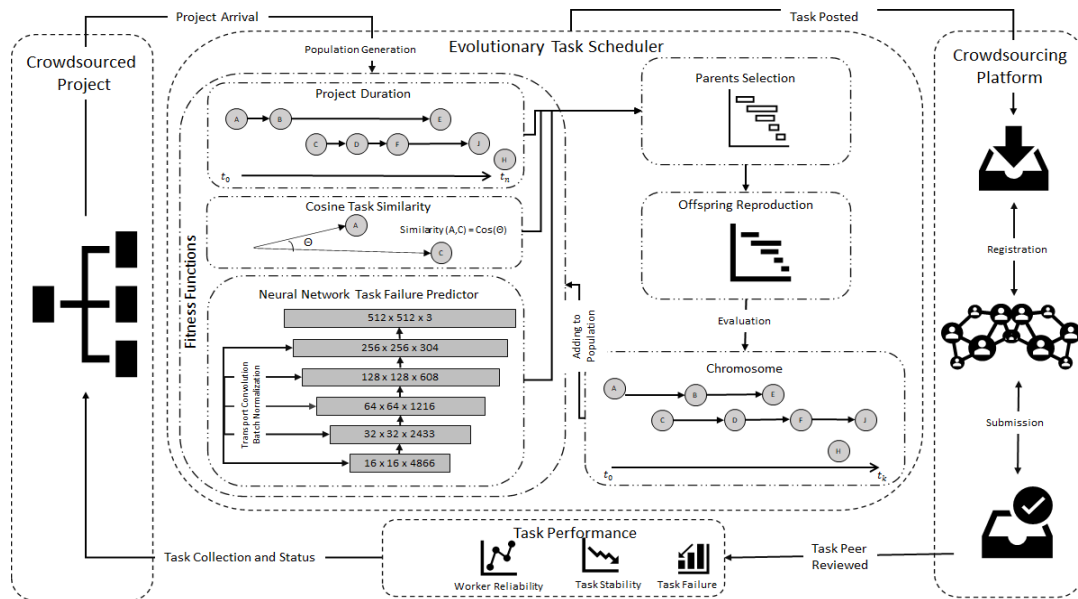


Figure 2: Overview of Evolutionary Scheduling Architecture.

dominant CSD project attributes from existing studies (Saremi et al., 2017)(Yang and Saremi, 2015)(Saremi and Yang, 2015) (Mejorado et al., 2020)(Lotfalian-Saremi et al., 2020) and defined metrics used in the scheduling framework. Then we added the neural network model from (Urbaczek et al., 2021) to predict the probability of task failure per day. In the method presented here, task arrival date is suggested based on the degree of task similarity among parallel tasks in the project and the probability of task failure in the platform based on available tasks and reliability of available workers to make a valid submission in the platform. Figure 2 presents the overview of the task scheduling architecture. List of tasks of a crowdsourced project is uploaded in the *evolutionary task scheduler*. The *task dependency* orders tasks based on the project dependency and tasks duration. In parallel, *task similarity* calculates the similarity among tasks to make sure batch of arrival tasks follow the optimum similarity level. Then, the *task failure predictor* analyzes the probability of failure of an arriving task in the platform based on the number of similar tasks available on arriving day, average similarity, task duration, and associated monetary prize. In next step, the *parent selection* selects the most suitable set of parents to use for *reproduction*, i.e. the generation of new chromosomes. In every generation, all chromosomes are evaluated by the fitness functions. The result of task performance in the platform is to be collected and reported to the client along with the input used to recommend the posting date.

### 3.1 Metrics and Task Prediction

We pose crowdsourced task scheduling as a multi-objective optimization problem with dynamic characteristics that requires agility in tracking the changes of task similarity and probability of task failure over time. Therefore, we presume that the objectives exhibit partial conflict (this was also borne out by observation) so that it will be unlikely to find a single solution that satisfies all objectives. In other words, a successful optimization finds a Pareto front of non-dominated solutions. To that end, we propose a novel evolutionary task scheduling method which combines multiple objectives. The presented method integrates artificial neural network with the non-dominated sorting genetic algorithm (NSGA-II) (Deb et al., 2002), which is a well-known method for finding sets of solutions for multi-objective problems.

#### 3.1.1 Task Dependency

A crowdsourced project ( $CP_K$ ) is a sequence of time dependent tasks that needs to be completed during specific period of time. In this study, each task can start only after all prerequisite tasks are successfully finished. The set of task dependencies is referred to as  $TD_i$ . Task (n),  $T_n$ , is parallel ( $Pl_i$ ) with task (n-1),  $T_{n-1}$ , if it starts before the submission deadline of  $T_{n-1}$ . And  $T_n$  is a sequential task ( $Sl_i$ ) for  $T_{n-1}$  if it starts after the submission deadline of  $T_n$ . The project duration ( $CPD_K$ ) is defined as the total duration between the first task's registration start date ( $TR_1$ ) and

last task's submission end date( $TS_n$ ).

$$CPD_k = (\{T_i\}, \{TD_i\}, CPD_k) \quad (1)$$

$$\text{where } \begin{cases} TD_i = \begin{cases} 1 & Pl_i = 1 \\ 0 & Sl_i = 1 \end{cases} \\ i = 1, 2, 3, \dots, n \\ CPD_k = TS_n - TR_1 \\ k = 1, 2, 3, \dots, m \end{cases}$$

### 3.1.2 Task Similarity

Task similarity is defined as the degree of similarity between a set of simultaneously open tasks. We calculate the tasks' local distance from each other to calculate a task similarity factor, based on different task features such as detailed task requirements, task monetary prize, task type, task registration and submissions date, required technology and platform.

*Def.1:* Task local distance ( $Dis_i$ ) is a tuple of tasks' attributes in the data set. Task local distance is:

$$Dis_i = (Prize_i, TR_i, TS_i, Type_i, Techs_i, PLT_s_i, Req_i) \quad (2)$$

*Def.2:* Task Similarity ( $Sim_{i,j}$ ) between two tasks  $T_i$  and  $T_j$  is defined as the is the dot product and magnitude of the local distance of two tasks:

$$Sim_{i,j} = \frac{\sum_{i,j=0}^n Dis_{(i,j)}}{\sum_{i=0}^n \sqrt{Dis_i} * \sum_{j=0}^n \sqrt{Dis_j}} \quad (3)$$

### 3.1.3 Task Failure Predictor

To predict probability of task failure, a fully connected feed forward neural network was trained (Urbaczek et al., 2021). It is reported that task monetary prize and task duration (Yang and Saremi, 2015)(Faradani et al., 2011) are the most important factors in raising competition level for a task. In this research, we are adding the variables considered in our observations (for example number of open tasks and average task similarity in the platform) to the reported list of important factors as input features to train the neural network model. See table 1 for the list of features. The network is configured with five layers of size 32, 16, 8, 4, 2, and 1. We applied a K-fold (K=10) cross-validation method on the train/test group to train the prediction of task failure probability in the neural network model. Also, We used early stopping to avoid over-fitting. The trained model provided a loss of 0.04 with standard deviation of 0.002. Interestingly, neural network performed better than

moving average, linear regression and support vector regression on failure prediction(Urbaczek et al., 2021).

To help in understanding of the qualities of the task failure predictor model, the main input variables are defined below, as well as the reward function to train the neural network model.

*Def.3:* Number of Open Tasks per day ( $NOT_d$ ) is the Number of tasks ( $T_j$ ) that are open for registration when a new task ( $T_i$ ) arrives on the platform.

$$NOT_d = \sum_{j=0}^n T_j \quad (4)$$

where :  $TR_j \geq TR_i$

*Def.4:* Average Task Similarity per day ( $ATS_d$ ) is the average similarity score ( $Sim_{i,j}$ ) between the new arriving task ( $T_i$ ) and currently open tasks ( $T_j$ ) on the platform.

$$ATS_d = \frac{\sum_{i,j=0}^n Sim_{i,j}}{NOT_d} \quad (5)$$

where :  $TR_j \geq TR_i$

*Def.5:* Probability of task failure per day ( $p(TF_d)$ ), is the probability that a new arriving task ( $T_i$ ) does not receive a valid submission and fails given its arrival date.

$$p(TF_d) = 1 - \frac{\sum_{i=0}^n VS_i}{NOT_i} \quad (6)$$

where :  $TR_j \geq TR_i$

To determine the optimal arrival date, we run the neural network model and evaluate the probability of task failure for three days (i.e two days surplus) from task arrival, **arrival day:** ( $p(TF_d)$ ), **one day after:** ( $p(TF_{d+1})$ ), and **two days after:** ( $p(TF_{d+2})$ ). To predict the probability of task failure in future days, there is a need to determine the number of expected arriving tasks and associated task similarity scores compared to the open tasks in that day.

*Def.6:* Considering that the registration duration (difference between opening and closing dates) for each task is known at any given point in time, the rate of task arrival per day ( $TA_d$ ) is defined as the ratio of the number of open tasks per day  $NOT_d$  over the total duration of open tasks per day.

$$TA_d = \frac{NOT_d}{\sum_{j=0}^n D_j} \quad (7)$$

By knowing the rate of task arrival per day, the number of open tasks for future days can be determined.

*Def.7:* Number of Open Tasks in the future  $OT_{fut}$  is the number of tasks that are still open given a future

date  $NOT_{fut}$ , in addition to the rate of task arrival per day  $TA_d$  multiplied by the number of days into the future  $\Delta days$ .

$$OT_{d+i} = NOT_{d+i} + TA_d * \Delta days \quad (8)$$

Also there is a need to know the average task similarity in future days.

*Def.8:* Average Task Similarity in the future  $ATS_{fut}$ , is defined as the number of tasks that are still open given a future date  $NOT_{fut}$  multiplied by the average task similarity of this group of tasks  $ATS_{fut}$ , the average task similarity of the current day  $ATS_d$  multiplied by the rate of task arrival per day  $TA_d$  and the the number of days into the future  $\Delta days$ .

$$ATS_{fut} = NOT_{fut} * ATS_{fut} + ATS_d * TA_d * \Delta day \quad (9)$$

## 3.2 Evolutionary Task Scheduling

Below, we describe the evolutionary method implemented to schedule tasks based on the metrics and failure predictor described above.

### 3.2.1 Chromosome Representation and Initial Population

The chromosome is represented as a sequence of integers where each value indicates the arrival day for the respective task. One chromosome represents the schedule for the given project. The integer values are bounded between 0 and the maximum allowed duration for the project.

### 3.2.2 Reproduction and Variation

Chromosomes are reproduced either through copying followed by mutation or through crossover. We employ standard two-point crossover: two indices are selected at random and sequences between those two points are exchanged between two parent chromosomes to create two new offspring. The probability of reproduction through crossover is 0.9. With 0.1% probability, reproduction is through copying a single parent and mutating the copy. For mutation, we use shuffling. For each value in the sequence, an index is chosen randomly for shuffling. Probability for each index to get selected for shuffling is  $1/(lengthOfSequence)$ . These parameters were decided through initial experimentation using this system as well as previous experience with similar problems.

During initialization of chromosomes, as well as during reproduction, we ensure that every chromosome adheres to the task dependency constraint. The

task dependency constraint is that tasks in the crowd project follow their required dependencies.

$$TR_j = \begin{cases} TR_i & Sl_{i,j} = 0 \\ TS_i + 1 & Sl_{i,j} = 1 \end{cases} \quad (10)$$

### 3.2.3 Fitness Functions

We use three fitness functions that seek to, respectively, minimize project duration, minimize probability of task failure in the platform, and manage task similarity in the project. Each fitness function is described below:

- *Project duration:* The first fitness function measures the complete duration of the project for suggested schedule.
- *Task Similarity:* Tasks with cosine similarity of 60% lead to the highest level of competition with lower level of task failure (Saremi et al., 2020). The similarity fitness function assures that parallel tasks that are not 60% similar do not arrive at the same time.

$$\begin{cases} Pl_{i,j} = 1 & TR_j = \begin{cases} TR_i & Sim_{i,j} = 0.6 \\ TRE_i & Sim_{i,j} \neq 0.6 \end{cases} \\ Pl_{i,j} = 0 & TR_j = TS_i + 1 \end{cases} \quad (11)$$

When two task in the project are arriving *parallel* that are not following 60% rule, task (j),  $T_j$ , with longer registration duration, will be postponed to arrive after task (i),  $T_i$  registration end date.

$$TR_j = TRE_i \quad (12)$$

$$\text{where : } \begin{cases} Pl_{i,j} = 1 \\ TRE_i \leq TRE_j \end{cases}$$

- *Task Failure Probability:* Tasks following the dependency requirement function and meeting the task similarity are entering the task failure probability function to be assigned to the task registration start date with *lowest* failure probability based on the result of the neural network.

$$TR_i = \min(p(TF_d), p(TF_{d+1}), p(TF_{d+2})) \quad (13)$$

### 3.2.4 Selection Operator

For parent selection operation we use a crowding-distance based tournament strategy. A crowding-distance is a measure of how close a chromosome is to its neighbors. For selecting individuals for the next generation from the pool of current population and offspring we use the standard selection method from NSGA-II (Deb et al., 2002).

Table 1: Summary of Task Features in the Data Set.

| Type              | Metrics                           | Definition   |
|-------------------|-----------------------------------|--|
| Tasks Attributes  | Task registration start date (TR) | The first day of task arrival in the platform and when workers can start registering for it. (mm/dd/yyyy)                          |
|                   | Task submission end date (TS)     | Deadline by which all workers who registered for task have to submit their final results. (mm/dd/yyyy)                             |
|                   | Task registration end date (TRE)  | The last day that a task is available to be registered for. (mm/dd/yyyy)   |
|                   | Task Duration ( $D_i$ )           | total available days from task (i) registration start date ( $TR_i$ ) to submissions end date ( $TS_i$ )                           |
|                   | Monetary Prize (MP)               | Monetary prize (USD) offered for completing the task and is found in task description. Range: (0, $\infty$ )                       |
| Tasks Performance | Technology (Tech)                 | Required programming language to perform the task. Range: (0, # Tech)  |
|                   | Platforms (PLT)                   | Number of platforms used in task. Range: (0, $\infty$ ).   |
|                   | Task Status                       | Completed or failed tasks  |
|                   | # Registrants (R)                 | Number of registrants that sign up to compete in completing a task before registration deadline. Range: (0, $\infty$ ).            |
|                   | # Submissions (S)                 | Number of submissions that a task receives before submission deadline. Range: (0, # registrants].                                  |
|                   | # Valid Submissions (VS)          | Number of submissions that a task receives by its submission deadline that have passed the peer review. Range: (0, # registrants]. |

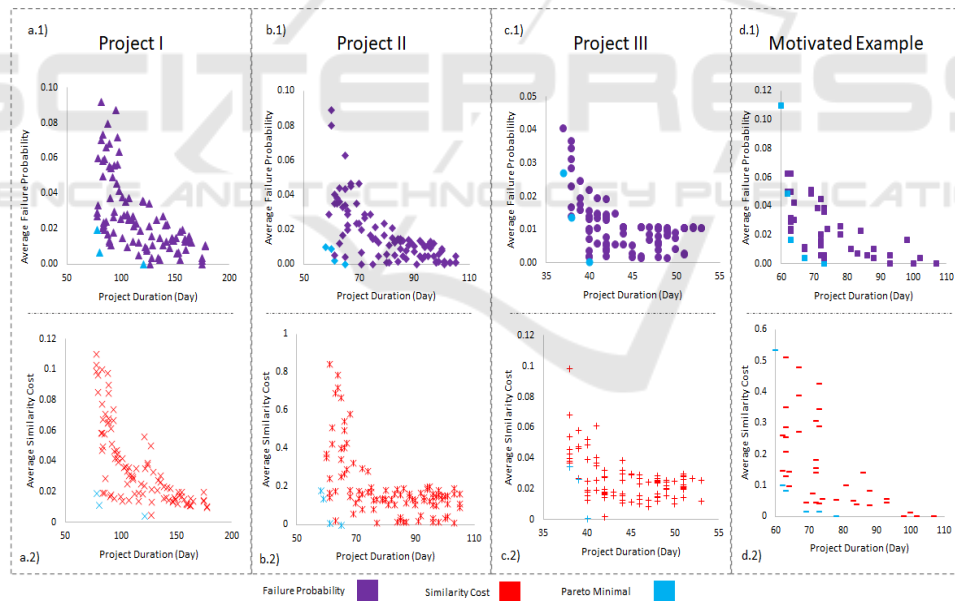


Figure 3: Average Task failure Probability (a.1, b.1, c.1, d.1) and Task Similarity Cost(a.2, b.2, c.2, d.2) of Recommended Schedules per project.

### 3.2.5 Population and Termination Criterion

The presented algorithm used a population of 200, and the run length was experimentally determined to be 200 generations, after which no more non-dominated solutions generally emerged. Task execution follows the project schedule provided for a

project in our data set and the initial chromosome follows the original task dependencies with 0 days delay (i.e the earliest project schedule).

## 4 EXPERIMENT DESIGN

To evaluate the evolutionary task scheduling method this section presents the experiment design and evaluation base line for this study.

### 4.1 Research Questions

To investigate the impact of probability of task failure in platform and task similarity level with in the project, the following research questions were formulated and studied:

**RQ1 (Overall Performance):** *How does the evolutionary task scheduling method help to reduce project duration and task failure potential?*

**RQ2 (Task Similarity):** *What is the effect of introducing a task similarity objective on the algorithm's ability to find short and robust schedules?*

### 4.2 Data Set

The gathered data set contains 403 individual projects including 4,908 component development tasks and 8,108 workers from Jan 2014 to Feb 2015, extracted from TopCoder website. Tasks are uploaded as competitions in the platform and crowd software workers register and complete the challenges. On average, most of the tasks have a life cycle of 14 days from the first day of registration to the submission's deadline. Table 1 summarizes the features available in the data set and used in fitness functions.

## 5 RESULTS

This section presents the evaluation results of applying the proposed scheduling framework to 4 CSD projects. Table 2 summarize the projects overall view. As figure 3-a represents, the framework recommends a schedule with duration of 121 days with 0.0% probability of task failure and 0.4% similarity cost for project I. This provides almost 70% schedule acceleration. The result of scheduling for project II, figure 3-b, is a project duration with Pareto minimum of 65 days, and 0% failure probability and task similarity cost.

The recommended schedule results in 78% acceleration. Applying the presented framework to the project III, 3-c. lead to 55% shorter schedule acceleration. the project duration shortens to 40 days, with probability of task failure as low as 0.02% and task similarity cost of 0.04%. Finally, scheduling the motivating example with the presented system recom-

Table 2: Summary of Scheduled Projects.

| Project ID  | # Tasks | Original (day) | Final (day) | Recom (day) | Schedule Acceleration |
|-------------|---------|----------------|-------------|-------------|-----------------------|
| Project I   | 25      | 70             | 393         | 121         | 70%                   |
| Project II  | 24      | 58             | 203         | 65          | 78%                   |
| Project III | 23      | 31             | 88          | 40          | 55%                   |
| Motivating  | 11      | 37             | 110         | 73          | 33%                   |

mended a schedule with 73 days duration, 0.07% average probability of task failure and 1.3% average task similarity cost. This means a 33% schedule acceleration.

In order to answer the research questions in part 4.1 and have a better understanding of how the presented framework part 3.2 is working, we explain scheduling of motivating example using the evolutionary scheduling introduced in details in parts 5.1, 5.2 and 5.3.

### 5.1 Project Schedule Acceleration

To answer RQ1 we eliminate the second fitness function (*task similarity*) and scheduled the project focusing on minimizing task failure probability. Figure 4-a illustrates the scheduling result. As it is shown in figure 4-a, project duration has decreased from 110 days to 60 days. The probability of task failure has dropped to 10%. While the scheduling method providing 23 days delay in compare with the shortest possible project plan (37 days), the recommended schedule finishes 50 days earlier than the original project duration with 47% higher chance of success. According to the recommended project timeline, project faced the maximum task failure probability of 33% on task 7 and the minimum failure probability of 0.0 on tasks 2,5,6,8,9 and 10.

### 5.2 Task Similarity

To answer RQ2 we have added task similarity fitness function to the evolutionary scheduling. In this part we analyze the recommended schedule with 0 similarity cost form the solution space. As it is presented in figure 4-b, evolutionary scheduling recommends a plan with duration of 79 days and average probability of task failure of 5%. Compare with the schedule from RQ1, evolutionary scheduling provides 19 days longer project plan, while it provides 5% lower task failure probability. Also, evolutionary scheduling takes similarity among the project tasks in to account. This creates easier competition for attracting resources. Also, the method recommended to postpone the start date of the project for 47 day, in order to have 0 days overlap due to task similarity.

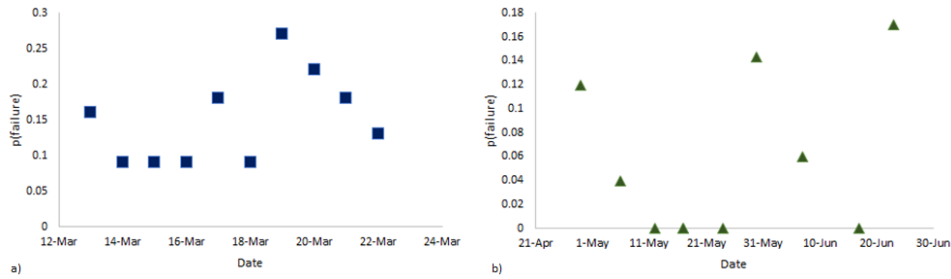


Figure 4: A) Overall Performance of scheduling framework, b) Schedule Acceleration with Task Similarity.

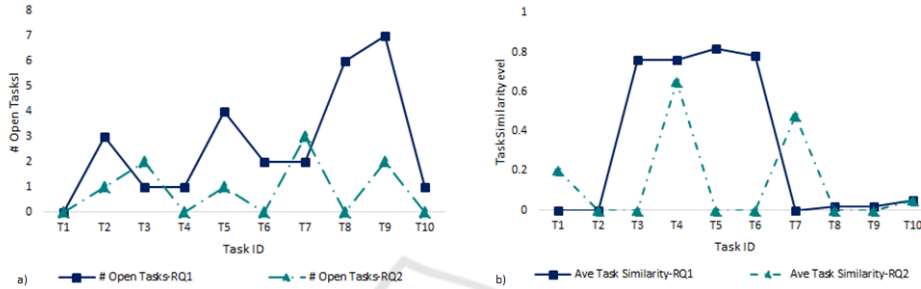


Figure 5: A) Number of Open Tasks on Task Arrival Day, b) Average Task Similarity on Task Arrival Day.

### 5.3 Comparison between Scheduling with and without Task Similarity

To have a better understanding of the impact of the proposed evolutionary scheduling we investigate the number of open tasks, number of open tasks and average task similarity level on the recommended schedule.

Figure 5-a presents the number of open task per arrival task based on different scheduling plan in RQ1 and RQ2. Scheduling under RQ1 conditions lead to competition level with on average 3 other tasks per day. The least competition to attract the resources happened for  $T_1$  with 0 open tasks in the arrival date and  $T_9$  faced the hardest competition with 7 other tasks available in the platform. Clearly, taking task similarity into account creates less competition. As it is shown scheduling under RQ2 provides task arriving with average one open task per arrival date, with minimum 0 open task for  $T_1, T_4, T_6, T_8, T_{10}$ , and maximum 3 open tasks when  $T_7$  arrived.

Another measure to investigate is average task similarity on the recommended task arrival date. As it is shown in figure 5-b, in RQ1 tasks are arriving to the platform with on average 32% task similarity.  $T_3, T_4, T_5$ , and  $T_6$  are competing with tasks with more than 75% similarity. While adding task similarity fitness function in RQ2 provides task arrival with on average 14% task similarity in the platform.  $T_4$  faced the highest task similarity level of 65%.

### 5.4 Discussion and Findings

Our evolutionary scheduling method generated plans with average probability of task failure of 5%. This result not only is 53% lower than the original plans, but it is also lower than reported failure ratio in the platform (i.e 15.7%) (Mejorado et al., 2020)(Yang et al., 2016). Figure 3-d-1. shows the average probability of task failure per recommended schedule by evolutionary scheduler. According to the figure 3-d-1 longer project duration provides lower probability of task failure. The goal is to reduce schedule duration while reducing task failure, so the optimum solutions have duration of around 73 days (33% schedule acceleration) which provides task failure probability of 0.07%.

Investigating the number of open tasks and the average task similarity on recommended arriving day per task support that the evolutionary task scheduling method assures lower competition over shared supplier resources per arrival task(i.e 1 and 14% respectively).

Task similarity cost is the ratio of duration added to the recommended task schedule due to considering task similarity fitness function. As shown in figure 3-d-2 the lowest similarity cost is when project duration is between 70 to 80 days or more than 95 days. The main objective is project duration, so the optimum solution 70-80 days (Pareto minimum shown in blue).



## 6 CONCLUSION AND FUTURE WORK

CSD provides software organizations access to an infinite, online worker resource supply. Assigning tasks to a pool of unknown workers from all over the globe is challenging. A traditional approach to solving this challenge is task scheduling. Improper task scheduling in CSD may cause Task failure due to uncertain worker behavior. The proposed approach recommends task scheduling plans based on a set of task dependencies in a crowdsourced project, similarities among tasks, and task failure probabilities based on recommended arrival date. The proposed evolutionary scheduling method utilizes a genetic algorithm to optimize and recommend the task schedule. The method uses three fitness functions, respectively based project duration, task similarity, and task failure prediction. The task failure fitness uses a neural network to predict probability of task failure on arrival date. The proposed method empowers crowdsourcing managers to explore potential outcomes with respect to different task arrival strategies. This includes the probability of task failure, number of open similar task in terms of context, prize, duration and type on the arrival day, and different schedule acceleration. The experimental results on 4 different projects demonstrate that the proposed method reduced project duration on average 59%

## REFERENCES

- Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*.
- Difallah, D. E., Demartini, G., and Cudré-Mauroux, P. (2016). Scheduling human intelligence tasks in multi-tenant crowd-powered systems. In *International Conference on World Wide Web*.
- Faradani, S., Hartmann, B., and Ipeirotis, P. G. (2011). What's the right price? pricing tasks for finishing on time. In *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*.
- Khanfor, A., Yang, Y., Vesonder, G., Ruhe, G., and Messinger, D. (2017). Failure prediction in crowdsourced software development. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*.
- Khazankin, R., Psai, H., Schall, D., and Dustdar, S. (2011). Qos-based task scheduling in crowdsourcing environments. In *International Conference on Service-Oriented Computing*.
- Lotfalian-Saremi, M., Saremi, R., and Martinez-Mejorado, D. (2020). How much should i pay? an empirical analysis on monetary prize in topcoder. In *HCI*.
- Mejorado, D. M., Saremi, R., Yang, Y., and Ramirez-Marquez, J. E. (2020). Study on patterns and effect of task diversity in software crowdsourcing. In *International Symposium on Empirical Software Engineering and Measurement*, pages 1–10.
- Saremi, R. (2018). A hybrid simulation model for crowdsourced software development. In *International Workshop on Crowd Sourcing in Software Engineering*.
- Saremi, R., Lotfalian Saremi, M., Desai, P., and Anzalone, R. (2020). Is this the right time to post my task? an empirical analysis on a task similarity arrival in topcoder. In *HCII*.
- Saremi, R., Yang, Y., and Khanfor, A. (2019). Ant colony optimization to reduce schedule acceleration in crowdsourcing software development. In *International Conference on Human-Computer Interaction*.
- Saremi, R., Yang, Y., Vesonder, G., Ruhe, G., and Zhang, H. (2021). Crowdsim: A hybrid simulation model for failure prediction in crowdsourced software development. *arXiv preprint arXiv:2103.09856*.
- Saremi, R. L. and Yang, Y. (2015). Empirical analysis on parallel tasks in crowdsourcing software development. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*.
- Saremi, R. L., Yang, Y., Ruhe, G., and Messinger, D. (2017). Leveraging crowdsourcing for team elasticity: an empirical evaluation at topcoder. In *International Conference on Software Engineering*.
- Stol, K.-J. and Fitzgerald, B. (2014). Two's company, three's a crowd: a case study of crowdsourcing software development. In *International Conference on Software Engineering*.
- Urbaczek, J., Saremi, R., Saremi, M. L., and Togelius, J. (2021). Greedy scheduling: A neural network method to reduce task failure in software crowdsourcing. In *International Conference on Enterprise Information Systems*.
- Wang, H., Ren, Z., Li, X., and Jiang, H. (2018). Solving team making problem for crowdsourcing with evolutionary strategy. In *2018 5th International Conference on Dependable Systems and Their Applications*.
- Yang, Y., Karim, M. R., Saremi, R., and Ruhe, G. (2016). Who should take this task? dynamic decision support for crowd workers. In *International Symposium on Empirical Software Engineering and Measurement*.
- Yang, Y. and Saremi, R. (2015). Award vs. worker behaviors in competitive crowdsourcing tasks. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*.
- Yue, T., Ali, S., and Wang, S. (2015). An evolutionary and automated virtual team making approach for crowdsourcing platforms. In *Crowdsourcing*.