Greedy Scheduling: A Neural Network Method to Reduce Task Failure in Software Crowdsourcing

Jordan Urbaczek [1], Razieh Saremi [1][a], Mostaan Lotfalian Saremi[1][b] and Julian Togelius [2][c]
[1]*School of Systems and Enterprises, Stevens Institute of Technology, Hoboken, NJ, USA*
[2]*Tandon School of Engineering,New York University, NYC, NY,USA*
{*rsaremi, jurbacze, mlotfali*}*@stevens.edu, julian.togelius@nyu.edu*

Keywords: Crowdsourcing, Task Scheduling, Task Similarity, Task Failure, Neural Network, TopCoder

Abstract: Highly dynamic and competitive crowdsourcing software development (CSD) marketplaces may experience task failure due to unforeseen reasons, such as increased competition over shared supplier resources, or uncertainty associated with a dynamic worker supply. Existing analysis reveals an average task failure ratio of 15.7% in software crowdsourcing markets.These lead to an increasing need for scheduling support for CSD managers to improve the efficiency and predictability of crowdsourcing processes and outcomes. To that end, this research proposes a task scheduling method based on neural networks, and develop a system that can predict and analyze task failure probability upon arrival. More specifically, the model uses a range of input variables, including the number of open tasks in the platform, the average task similarity between arriving tasks and open tasks, the winner's monetary prize, and task duration, to predict the probability of task failure on the planned arrival date and two surplus days. This prediction will offer the recommended day associated with lowest task failure probability to post the task. The model on average provided 4% lower failure probability per project. The proposed model empowers crowdsourcing managers to explore potential crowdsourcing outcomes with respect to different task arrival strategies.

1 INTRODUCTION

Crowdsourced Software Development (CSD) has been used increasingly to develop software applications(Stol and Fitzgerald, 2014) (Stol and Fitzgerald, 2014). Crowdsourcing mini software development tasks leads to lower accelerated development (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. Any kind of skew in addressing these needs leads to task failure in the CSD platform. Generally, planning for CSD tasks that are complex, independent, and require a significant amount of time, effort, and expertise (Stol and Fitzgerald, 2014) is challenging. For the task provider, requesting a crowdsourcing service is even more challenging due to the uncertainty of the similarity among available tasks in the platform and the arrival of new tasks (Saremi, 2018)(Difallah et al., 2016). The availability of crowd workers' skill sets and consistency of performance history is also uncertain (Karim et al., 2016)(Zaharia et al., 2010). These factors raise the issue of receiving qualified submissions, since crowd workers may be interested in multiple tasks from different task providers based on their individual utility factors (Faradani et al., 2011).

It has been reported that crowd workers are more interested in working on tasks with similar concepts, monetary prize, technologies, complexities, priorities, and duration (Faradani et al., 2011)(Gordon, 1961)(Difallah et al., 2016)(Yang and Saremi, 2015). However, attracting workers to a large group of similar tasks may cause zero registration, zero submissions, or unqualified submissions for some tasks due to lack of availability from workers(Khanfor et al., 2017)(Khazankin et al., 2011). Moreover, lower level of task similarity in the platform leads to higher chance of task success and workers' elasticity(Saremi et al., 2020a).

For example, in Topcoder[1], a well-known Crowdsourcing Software platform, an average of 13 tasks arrive daily and are added to an average list of 200 ex-

[a] https://orcid.org/0000-0002-7607-6453
[b] https://orcid.org/0000-0001-8657-7748
[c] https://orcid.org/0000-0003-3128-4598

[1] https://www.topcoder.com/

isting tasks. There is an average of 137 active workers to take the tasks at that period, which leads to an average of 25 failed tasks each day. According to this example, there will be a long queue of tasks waiting to be taken. Considering the fixed submission date, such a waiting line may result in failed tasks. These challenges have traditionally been addressed with task scheduling methods.

The objective of this study is to provide a task schedule recommendation framework for a software crowdsourcing platform in order to improve the success and efficiency of software crowdsourcing. In this study, we first present a motivational example to explain the current task status in a software crowdsourcing platform. Then we propose a task scheduling architecture using a neural network strategy to reduce probability of task failure in the platform.

More specifically, the system uses a range of input variables, including the number of open tasks in the platform, the average task similarity between arriving tasks and open tasks, the winner's monetary prize, and task duration, to predict the probability of task failure on the planned arrival date and two surplus days. This prediction will offer the recommended the day associated with lowest task failure probability to post the task. The proposed system represents a task scheduling method for competitive crowdsourcing platforms based on the workflow of Topcoder, one of the primary software crowdsourcing platforms. The evaluation results provided on average 4% lower task failure probability.

The remainder of this paper is structured as follows. Section II introduces a motivational example that inspires this study. Section III presents background and review of available works. Section IV outlines our research design and methodology. Section V presents the case study and model evaluation, and Section VI presents the conclusion and outlines a number of directions for future work.

2 MOTIVATING EXAMPLE

The motivation example illustrates a real crowdsourcing software development (CSD) project on the TopCoder platform. It was comprised of 41 tasks with a total project duration of 207 days, with an average of 8 days per task. The project experienced a 47% task failure ratio, which means 19 of the 41 tasks failed. 6 tasks failed due to client requests (i.e 14% failure) and 7 tasks failed due to failed requirements (17% failure). The remaining 8 tasks (i.e 14% failure) failed due to zero submissions.

If we ignore the task failed based on client request and failed requirements, 28 tasks remain, (see figure1). Deeper analysis reveals that most of the failed tasks entered the task pool with a similarity above 80% when compared with the available tasks.

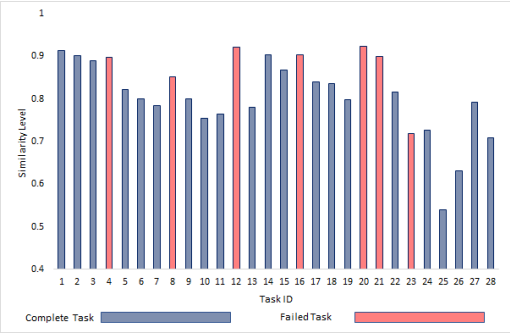

Figure 1: Overview of Tasks' Status and Similarity Level in the Platform

Also, as figure 2 illustrates, each task competes with an average of 145 similar open tasks upon arrival. The number of open tasks can directly impact the ability of a task to attract suitable workers and lead to task failure. It is reported that the degree of task similarity in the pool of tasks directly impacts the task competition level to attract registrants and task success (Saremi et al., 2020a).

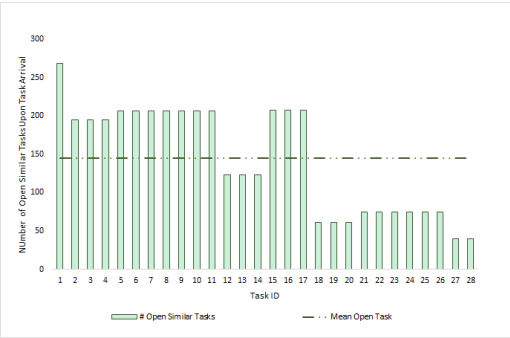

Figure 2: Number of Open Tasks in Platform upon Task Arrival

It seems task failure is a result of task competition level in the platform. This observation motivates us to investigate more and provide a task scheduling recommendation model which helps reduce the probability of task failure in the platform.

3 RELATED WORK

3.1 Task Scheduling in Crowdsourcing

Different characteristics of machine and human behavior create delays in product release(Ruhe and

Saliu, 2005). This phenomenon leads to a lack of systematic processes to balance the delivery of features with the available resources (Ruhe and Saliu, 2005). Therefore, improper scheduling would result in task starvation (Faradani et al., 2011). Parallelism in scheduling is a great method to create the chance of utilizing a greater pool of workers (Ngo-The and Ruhe, 2008; Saremi and Yang, 2015). Parallelism encourages workers to specialize and complete tasks in a shorter period. The method also promotes solutions that benefit the requester and can help researchers to clearly understand how workers decide to compete on a task and analyze the crowd workers performance (Faradani et al., 2011). Shorter schedule planning can be one of the most notable advantages of using CSD for managers (Lakhani et al., 2010).

Batching tasks in similar groups is another effective method to reduce the complexity of tasks and it can dramatically reduce costs(Marcus et al., 2011). Batching crowdsourcing tasks would lead to a faster result than approaches which keep workers separate(Bernstein et al., 2011). There is a theoretical minimum batch size for every project according to the principles of product development flow (Reinertsen, 2009). To some extent, the success of software crowdsourcing is associated with reduced batch size in small tasks. Besides, the delay scheduling method (Zaharia et al., 2010) was specially designed for crowdsourced projects to maximize the probability that a worker receives tasks from the same batch of tasks they were performing. An extension of this idea is introduced a new method called "fair sharing schedule" (Ghodsi et al., 2011). In this method, various resources would be shared among all tasks with different demands to ensure that all tasks would receive the same amount of resources. For example, this method was used in Hadoop Yarn. Later, Weighted Fair Sharing (WFS) (Difallah et al., 2016) was presented as a method to schedule batches based on their priority. Tasks with higher priority are to be introduced first.

Another proposed crowd scheduling method is based on the quality of service (QOS) (Khazankin et al., 2011). This is a skill-based scheduling method with the purpose of minimizing scheduling while maximizing quality by assigning the task to the most available qualified worker. This scheme was created by extending standards of Web Service Level Agreement (WSLA) (Ludwig et al., 2003). The third available method method is HIT-Bundle (Difallah et al., 2016). HIT-Bundle is a batch container which schedules heterogeneous tasks into the platform from different batches. This method makes for a higher positive outcome by applying different scheduling strategies at the same time. The method was most recently applied in helping crowdsourcing-based service providers meet completion time SLAs (Hirth et al., 2019). The system works by recording the oldest task waiting time and running a stimulative evaluation to recommend the best scheduling strategy for reducing the task failure ratio.

3.2 Task Similarity in Crowdsourcing

Generally, workers tend to optimize their personal utility factor when registering for a task (Faradani et al., 2011). It is reported that workers are more interested in working on similar tasks in terms of monetary prize (Yang and Saremi, 2015), context and technology (Difallah et al., 2016), and complexity level. Context switch generates a reduction in workers' efficiency (Difallah et al., 2016). However, workers usually try to register for a greater number of tasks than they can complete (Yang et al., 2016). It is reported that a high task similarity level negatively impacts task competition level and team elasticity (Saremi et al., 2020b). A combination of these observations led to task failure due to: 1) receiving zero registrations for a task based on a low degree of similar tasks and a lack of available skillful workers (Yang and Saremi, 2015), and 2) receiving non-qualified submissions or zero submissions based on a lack of time to work on all the tasks registered by the worker(Archak, 2010).

3.3 Challenges in Crowdsourcing

Considering the highest rate for task completion and submission acceptance, software managers will be more concerned about the risks of adopting crowdsourcing. Therefore, there is a need for a better decision-making system to analyze and control the risks of insufficient competition and poor submissions due to the attraction of untrustworthy workers. A traditional method of addressing this problem in the software industry is task scheduling. Scheduling is helpful in prioritizing access to resources. It can help managers optimize task execution in the platform to attract the most reliable and trustworthy workers. Normally, in traditional methods, task requirements and phases are fixed, while cost and time are flexible. In a time-boxed system, time and cost are fixed, while task requirements and phases are flexible (Cooper and Sommer, 2016). However, in CSD all three variables are flexible. This characteristic creates a huge advantage in crowdsourcing software projects.

Generally, improper scheduling could lead to task starvation (Faradani et al., 2011), since workers with greater abilities tend to compete with low skilled

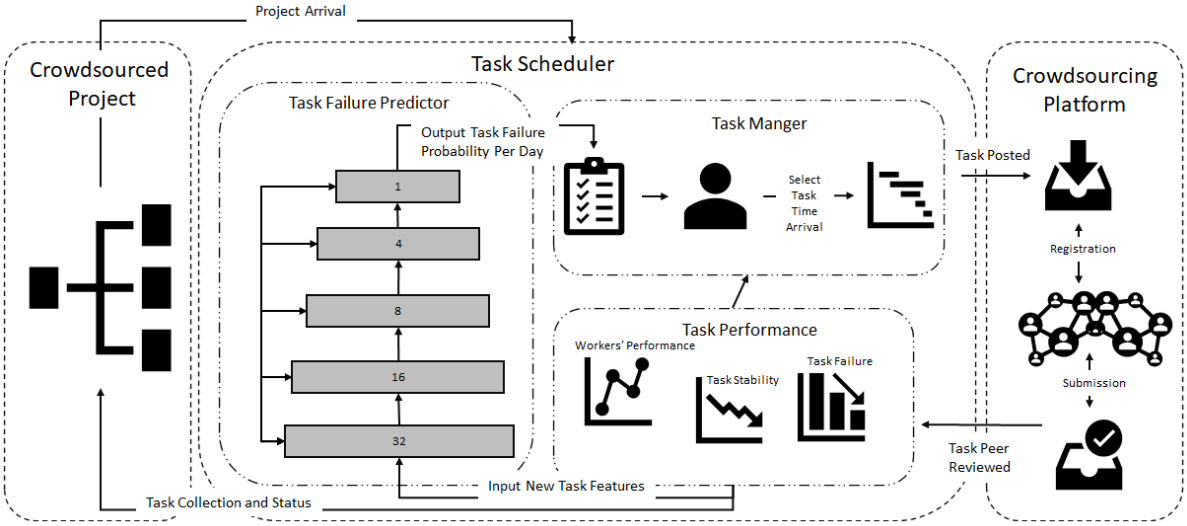

Figure 3: Overview of Scheduling Architecture

workers (Archak, 2010). In this case, users are more likely to choose tasks with fewer competitors (Yang et al., 2008). Also, workers who intentionally choose to participate in less popular tasks could potentially enhance winning probabilities, even if workers share similar expertise. It brings some severe problems in the crowd workers trust system to continue performing on a task and causes a lot of dropped and non-completed tasks. Moreover, tasks with relatively lower monetary prizes have a high probability of registration and completion, which results in only 30% of problems in the platform being solved (Rapoport and Chammah, 1966). Lower priced tasks may attract higher numbers of workers to compete and consequently increases the chance of starvation for more expensive tasks and project failure.

The above issues indicate the importance of task scheduling in the platform in order to attract the right amount of trustworthy workers and expertise that will result in a shortened task release time.

4 RESEARCH DESIGN AND METHODOLOGY

To solve the scheduling problem, we designed a model to predict the probability of task failure and recommend arrival date based on comparing predicted task failure probabilities. We utilized a neural network model to predict the probability of task failure per day. Then we add a search-based optimizer to recommend arrival day with lowest failure probability. This architecture can be operated on any crowdsourcing platform; however, we focused on TopCoder as the target platform. In this method, task arrival date is suggested based on the degree of task similarity in the platform and the reliability of available workers to make a valid submission. Figure 3 presents the overview of the task scheduling architecture. Each task is uploaded in the *task scheduler*. The *Task failure predictor* analyzes the probability of failure of an arriving task in the platform based on the number of similar tasks available that day, average similarity, task duration, and associated monetary prize. Then the model recommends a probability of task failure for the assigned date with two days surplus. In next step, the *task manager* selects the most suitable arrival date among the three recommended days and schedules the task to be posted. 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.

4.1 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 for 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. When a worker submits their final files, their submission is reviewed by experts to and labeled as a valid or invalid submission. Table1 summarizes the task features available in the data set.

Table 1: Summary of Metrics Definition

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. Range: (0, ∞))
	Task submission end date (TS)	Deadline by which all workers who registered for task have to submit their final results. Range: (0, ∞)).
	Task registration end date (TRE)	The last day that a task is available to be registered for. Range: (0, ∞)).
	Monetary Prize (P)	Monetary prize (USD) offered for completing the task and is found in task description. Range: (0, ∞)).
	Technology (Tech)	Required programming language to perform the task. Range: (0, #Tech))
	Platforms (PLT)	Number of platforms used in task. Range: (0, ∞)).
Tasks performance	Task Status	Completed or failed tasks
	# Registration (R)	Number of registrants that sign up to compete in completing a task before registration deadline. Range: (0, ∞).
	# 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].

4.2 Input to the Task Scheduler

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 (i.e number of open tasks and average task similarity) to the reported list of important factors as input of the presented model. To help in understanding of the qualities of the task scheduling tool, the input variables of the model, including average task similarity, task duration, task monetary prize, and number of open tasks are defined below. A definition for the probability of task failure in the platform, used as the reward function to train the neural network model, can also be found below.

First we need to understand the degree of task similarity among a set of simultaneously open tasks in the platform. *Def.1*: Task Similarity ($Sim_{i,j}$) is the similarity between two tasks T_i and T_j is defined as the weighted sum of all local similarities across the features listed in Table 2 :

$$Sim_{i,j} = W_1 * Dist_1(T_i, T_j) + ... + W_n * Dist_n(T_i, T_j)$$

Def.2: Task Duration (D_i) is the total available time from task (i) registration start date (TR_i) to submissions end date (TS_i):

$$D_i = \sum_{i=0}^{n} TS_i - TR_i$$

Def.3: Actual Prize (P_i) is the summation of the prize that the winner (PW_i) and runner up(PR_i) will receive after passing peer review.

$$P_i = \sum_{i=0}^{n} PW_i + PR_i$$

Def.4: 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.

$$NOT_d = \sum_{j=0}^{n} T_j$$

$$where, TRE_j >= TR_i$$

Def.5: 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.

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

$$where, TRE_j >= TR_i$$

Def.6: Task Failure Rate per day, (TF_d) is the probability that a new arriving task (T_i) does not receive a valid submission and fails given its arrival date.

Table 2: Features used to measure task distance

Feature	Description of distance measure $Dist_i$
Task Monetary Prize (P)	($Prize_i$ - $Prize_j$) = $Prize_{Max}$
Task registration start date (TR)	(TR_i - TR_j) = $DiffTR_{Max}$
Task submission end date (TS)	TS_i - TS_j) = $DiffTS_{Max}$
Task Type	($Type_i == Type_j$) ? 1 : 0
Technology (Tech)	Match($Tech_i$:$Tech_j$)=$NumberOfTechs_{Max}$
Platform (PL)	($PL_i == PL_j$) ? 1 : 0
Detailed Requirement	$(Req_i * Req_j)/(\|Req_i\| * \|Req_i\|)$

$$TF_d = 1 - \frac{\sum_{i=0}^{n} VS_i}{NOT_i}$$

$$where, TRE_j >= TR_i$$

4.3 Output of the Task Scheduler

The goal of the proposed model is not only to make sure that we can predict the probability of failure for a new arriving task given the arrival date, but also recommend the most suitable posting day to decrease the task failure rate with the surplus of two days. To determine the most optimal arrival date, we run the model and evaluate the result for *arrival day*, *one day after*, and *two days after*. To predict the probability of task failure in future days, we need to determine the number of expected arriving tasks and associated task similarity scores compared to the open tasks in the future.

Def.7: Rate of Task Arrival per day (TA_d), 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 is defined as the ratio of the number of open tasks per day NOT_d over the total duration of open tasks per day D_d.

$$TA_d = \frac{NOT_d}{\sum_{j=0}^{n} D_j}$$

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

Def.8: 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 ΔT_{days}.

$$OT_{tm} = NOT_{tm} + TA_d * \Delta T_{days}$$

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

Def.9: 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 ΔT_{days}.

$$ATS_{fut} = NOT_{fut} * ATS_{fut} + ATS_d * TA_d * \Delta T_{days}$$

4.4 Task Failure Predictor

A fully connected feed forward neural network was trained to predict task failure probability based on the four features described above. The network is configured with five layers of size 32, 16, 8, 4, 2, and 1. Training was implemented in in batch sizes of 8 for 50 epochs and the mean-squared error loss function was used. Before running the model, the data set was split into a train/test group and validation group. The train/test group was 80% of the data set and the validation group was 20%. 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. For each fold, the 10% testing portion of the train/test group was cycled and the remaining 90% was used as train data. We used early stopping to avoid over fitting. The trained model provided a loss equal to 0.04 with standard deviation of 0.002.

The task manager then uses the output of the task failure predictor to recommend the task arrival day with the minimum failure probability for the schedule plan. Applying the presented model on the full data set from data set introduced in section IV-A, i.e. all 4908 tasks, yielded a reduction of 8% (i.e 75% to 67%) in the probability of task failure in the platform.

5 CASE STUDY AND MODEL EVALUATION

To study the applicability of the proposed method in the real world, we used the system to reschedule the project from the motivating example. (This example is a small part of the full data set; results on the full data set are presented above.) The reschedule result is discussed below:

5.1 Result of the Task Failure Predictor

After testing, the data from the motivation example was provided to the system for evaluation with the goal of figuring out the arrival day with the lowest failure probability per task. Figure 4 presents the initial results of the model.

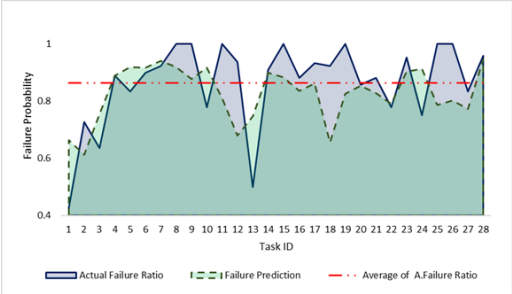

Figure 4: Comparison of initial Task Failure Prediction and Actual Failure

As shown in figure 4, the presented model's arrival date recommendations for the tasks in the sample provide an average failure prediction of 0.83, which is 0.03 lower than the actual scheduling failure. The result of initial failure probability prediction by the model in closer to the mean of actual failure, with standard deviation of 0.09. The duration of the project extended an extra day under the new scheduling recommendation, while the probability of task failure was reduced by almost 4%. Table 3 summarized the statistics of actual failure and prediction failure for the project.

Table 3: summary of actual and predicted failure probabilities of project

Statistics	Actual P(failure)	Predicted P(failure)
Min	0.42	0.61
Max	1	0.94
Mean	0.86	0.83
Median	0.90	0.84
Std	0.15	0.09

In next step, the model provides predictions of task failure probability for one and two days after the actual arrival date of the task. This result is used by the task manager to determine if the task should be posted in the future instead of today. Figure 5 illustrates the result of the failure prediction of all the three dates.

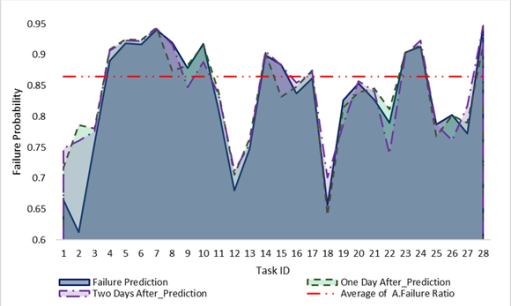

Figure 5: Details of Failure Prediction per Task for all level of predictions

Tasks 8,15,18,20,25,28 received the lowest prediction of failure probability on the second day with an average prediction of 0.81. Tasks 9, 10, 15, 22, 23, 26 received the lowest prediction of failure probability on day 3 with an average prediction of 0.8. The rest of the tasks received the lowest prediction of failure probability on the first day with an average of 0.81. However, not only was the average of all the three predictions lower than the actual failure prediction, but also most of the prediction points in all three days were lower than the average of actual failure.

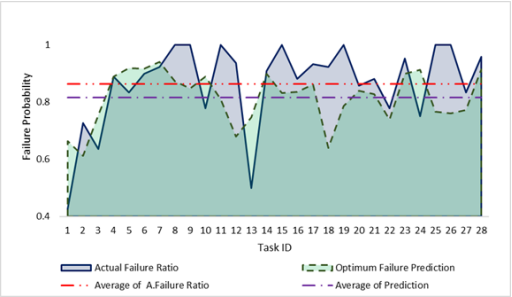

Figure 6: Comparison of Probability of Task Failure Prediction for final Schedule and Actual Schedule

With access to the 3-day outlook of task failure predictions from the model and the evaluation in figure 5, the task manager can more effectively schedule tasks by choosing the minimum task failure probability from the model results for each task. Figure 6 presents the failure probability for the project following the lowest failure prediction per task in comparison with the actual task failure. It is clear that the model's recommended schedule provides a lower

and more stationary probability of task failure with an average of 0.81, while the average probability of task failure for the actual task schedule is 0.86. The recommended task scheduling plan provides a minimum failure prediction of 0.61 and maximum of 0.94 with standard deviation of 0.09. The resulting accuracy of the model is 0.896. 8 illustrates the summary of original task timeline v.s. final task time line. 8(a) presents the original project timeline, and 8(b) shows the suggested project timeline by the presented model. 8(c)represents the duration of each task when shifted to achieve lowest failure probability.

To evaluate the model performance, we applied the Mean Square Error (MSE) metric to estimate the difference between the actual failure probability and the predicted failure probability of the same arrival day according to available data. Figure 7 presents the MSE for failure prediction per task. The average MSE is 0.09 with a minimum of 0.001 for task 3 and a maximum of 0.23 for task 1, with a standard deviation of 0.06.

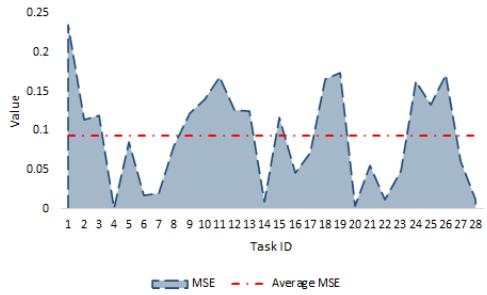

Figure 7: MSE for Probability of Task Failure per Task

5.2 Model Evaluation

To compare the performance of the proposed model, we applied a K-fold (K=10) cross-validation on the data set to predict the probability of task failure based on four different prediction approaches. The estimated probabilities of task failure are used to compute four popular performance measures that are widely used in current prediction systems for software development: 1- Mean Square Error (MSE), 2- Median of Mean Square Error ($MdMSE$), 3- Standard Deviation of mean Square Error ($StdMSE$), 4- Percentage of the estimates with Mean Square Error less than or equal to N% ($Pred(N)$).

The primary result of this analysis is shown in figure 9. It is clear that Neural Network analysis has a better predictive performance according to $Pred(0.05)$ and also has almost the lowest error rate with the MSE of 0.043%. The SVR recreation is the runner up in terms of performance with the MSE of 0.045%. Interestingly, the moving average and linear regression provided the same level of performance based on $Pred(0.05)$, while linear regression provides the lower MSE of 0.044.

5.3 Threats to Validity

First, the study only focuses on competitive CSD tasks on the TopCoder platform. Many more platforms do exist, and even though the results achieved are based on a comprehensive set of about 5,000 development tasks, the results cannot be claimed to be externally valid. There is no guarantee the same results would remain exactly the same in other CSD platforms.

Second, there are many different factors that may influence task similarity, task success, and task completion. Our similarity algorithm and task failure probability-focused approach are based on known task attributes in TopCoder. Different similarity algorithms and task failure probability-focused approaches may lead us to different, but similar results.

Third, the result is based on tasks only. Workers' network and communication capabilities are not considered in this research. In the future, we need to add this level of research to the existing one.

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 zero task registrations, zero task submissions or low qualified submissions due to uncertain worker behavior, and consequentially, task failure. This research presents a new scheduling method based on a neural network. The method reduces the probability of task failure in CSD platforms. The experimental result show a reduction in project failure probability of up to 4% while maintaining the same project duration.

In future research, we will focus on expanding the model to a more complicated framework that incorporates available worker similarity and considers the impact of the workers' competition performance regarding the task success to further improve efficiency in the scheduling model

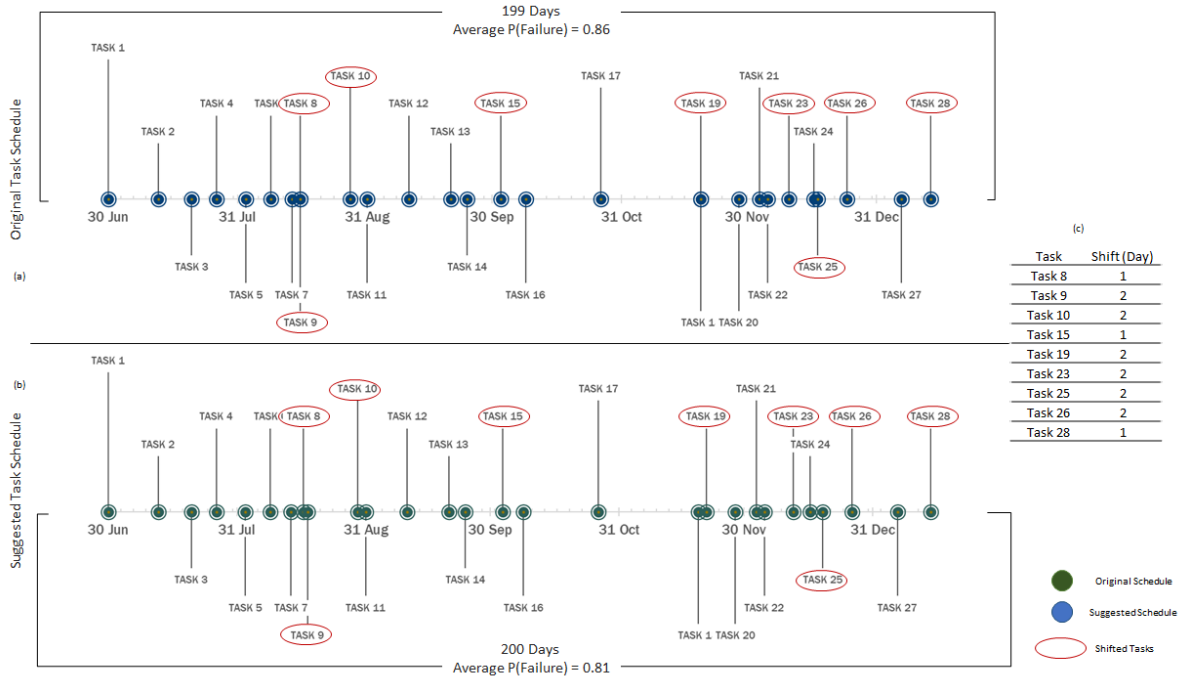

Figure 8: Project Timeline

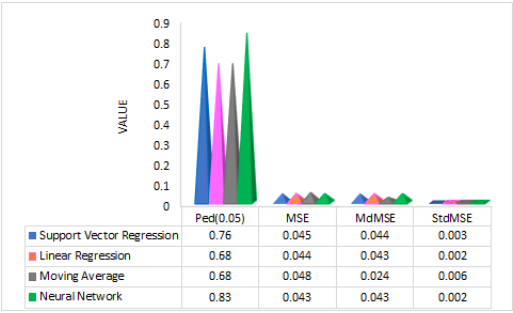

Figure 9: Performance of Task Failure Probability by Each Approach

REFERENCES

Archak, N. (2010). Money, glory and cheap talk: analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on topcoder. com. In *Proceedings of the 19th international conference on World wide web*, pages 21–30.

Bernstein, M. S., Brandt, J., Miller, R. C., and Karger, D. R. (2011). Crowds in two seconds: Enabling realtime crowd-powered interfaces. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 33–42.

Cooper, R. G. and Sommer, A. F. (2016). The agile–stage-gate hybrid model: A promising new approach and a new research opportunity. *Journal of Product Innovation Management*, 33(5):513–526.

Difallah, D. E., Demartini, G., and Cudré-Mauroux, P. (2016). Scheduling human intelligence tasks in multi-tenant crowd-powered systems. In *Proceedings of the 25th international conference on World Wide Web*, pages 855–865.

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*.

Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., and Stoica, I. (2011). Dominant resource fairness: Fair allocation of multiple resource types. In *Nsdi*, volume 11, pages 24–24.

Gordon, G. (1961). A general purpose systems simulation program. In *Proceedings of the December 12-14, 1961, eastern joint computer conference: computers-key to total systems control*, pages 87–104.

Hirth, M., Steurer, F., Borchert, K., and Dubiner, D. (2019). Task scheduling on crowdsourcing platforms for enabling completion time slas. In *2019 31st International Teletraffic Congress (ITC 31)*, pages 117–118. IEEE.

Karim, M. R., Messinger, D., Yang, Y., and Ruhe, G. (2016). Decision support for increasing the efficiency of crowdsourced software development. *arXiv preprint arXiv:1610.04142*.

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)*, pages 495–504. IEEE.

Khazankin, R., Psaier, H., Schall, D., and Dustdar, S. (2011). Qos-based task scheduling in crowdsourcing environments. In *International Confer-*

ence on Service-Oriented Computing, pages 297–311. Springer.

Lakhani, K. R., Garvin, D. A., and Lonstein, E. (2010). Topcoder (a): Developing software through crowdsourcing. *Harvard Business School General Management Unit Case*, (610-032).

Ludwig, H., Keller, A., Dan, A., King, R. P., and Franck, R. (2003). Web service level agreement (wsla) language specification. *Ibm corporation*, pages 815–824.

Marcus, A., Wu, E., Karger, D., Madden, S., and Miller, R. (2011). Human-powered sorts and joins. *arXiv preprint arXiv:1109.6881*.

Ngo-The, A. and Ruhe, G. (2008). Optimized resource allocation for software release planning. *IEEE Transactions on Software Engineering*, 35(1):109–123.

Rapoport, A. and Chammah, A. M. (1966). The game of chicken. *American Behavioral Scientist*, 10(3):10–28.

Reinertsen, D. G. (2009). Celeritas publishing.

Ruhe, G. and Saliu, M. O. (2005). The art and science of software release planning. *IEEE software*, 22(6):47–53.

Saremi, R. (2018). A hybrid simulation model for crowdsourced software development. In *Proceedings of the 5th International Workshop on Crowd Sourcing in Software Engineering*, pages 28–29.

Saremi, R., Lotfalian Saremi, M., Desai, P., and Anzalone, R. (2020a). Is this the right time to post my task? an empirical analysis on a task similarity arrival in topcoder. In Yamamoto, S. and Mori, H., editors, *Human Interface and the Management of Information. Interacting with Information*, pages 96–110, Cham. Springer International Publishing.

Saremi, R., Saremi, M. L., Desai, P., and Anzalone, R. (2020b). Is this the right time to post my task? an empirical analysis on a task similarity arrival in topcoder. *arXiv preprint arXiv:2004.12501*.

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)*, pages 28–34. IEEE.

Saremi, R. L., Yang, Y., Ruhe, G., and Messinger, D. (2017). Leveraging crowdsourcing for team elasticity: an empirical evaluation at topcoder. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 103–112. IEEE.

Stol, K.-J. and Fitzgerald, B. (2014). Two's company, three's a crowd: a case study of crowdsourcing software development. In *Proceedings of the 36th International Conference on Software Engineering*, pages 187–198.

Yang, J., Adamic, L. A., and Ackerman, M. S. (2008). Crowdsourcing and knowledge sharing: strategic user behavior on taskcn. In *Proceedings of the 9th ACM conference on Electronic commerce*, pages 246–255.

Yang, Y., Karim, M. R., Saremi, R., and Ruhe, G. (2016). Who should take this task? dynamic decision support for crowd workers. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10.

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)*, pages 1–10. IEEE.

Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., and Stoica, I. (2010). Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*, pages 265–278.