How should I Play This Game? An Evolutionary Algorithm for Task Arrival Scheduling in Crowdsourced Software Development

Razieh Saremi[1],Hardik Yagnik[1], Julian Togelius[2], and Ye Yang[1]
[1]Stevens Institute of Technology, [2]New York University
[1]{rsaremi, hyagnik1, yyang4}@stevens.edu
[2]julian.togelius@nyu.edu

***Abstract*—The complexity of software tasks and the uncertainty of crowd developer behaviors make it challenging to plan for crowdsourced software development (CSD) projects. In a competitive crowdsourcing marketplace, competition on shared worker resources from simultaneously open tasks adds another layer of complexity to potential outcomes of software crowdsourcing. These lead to an increasing need for scheduling support for CSD managers to improve the visibility and predictability into crowdsourcing processes and outcomes. To that end, this paper proposes an evolutionary algorithm-based scheduling method. 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 the probability of task failure on arrival date. The proposed method recommends the best start date for the project as a whole and each individual task so as to achieve the lowest project failure ratio.**

***Index Terms*—Crowdsourcing, Task Scheduling, Task Similarity, Task Failure, Neural Network, Evolutionary Algorithm, Genetic Algorithm, TopCoder**

I. Introduction

Crowdsourced Software Development (CSD) has been increasingly adopted in modern software practices aiming at leveraging crowd-wisdom to complete software mini-tasks faster and cheaper [1], [2]. The success of CSD relies on receiving qualified submissions by a large crowd of software workers who are registering and submitting for crowdsourced tasks. More specifically, a general CSD process starts with companies distributing their project's tasks with prizes online, and then crowd software workers browsing and registering to work on selected tasks, and submitting work products once completed. Crowd submissions will be evaluated by experts and experienced developers, through a peer review process, to assure the submission quality [3] [4]. The number of submissions per task and their evaluated scores reflect the level of success in task satisfaction or completion.

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 [1], because software tasks are complex, independent, and require potential task-takers pertaining specialized skill-sets. The challenges come from multi-folds: 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/how dedicated workers will engage in their tasks; and 3) task requesters are competing shared worker resources with other open tasks in the market.

To address these challenges, existing methods have explored various methods and techniques to bridge the information gap between demand and supply in crowdsourcing-based software development. These includes: studies towards developing better understanding on worker motivation and preferences in CSD [5] [6] [7] [8], studies focusing on predicting task failure [9] [10]; studies employing modeling and simulation techniques to optimize CSD task execution processes [11], [12]; and studies for recommending the most suitable workers for tasks [13] and developing methods to create crowdsourced teams [14] [15].

A recent study shows in competitive crowdsourcing, lower level of task similarity tends to lead to higher chance of task success and workers' elasticity [16]. However, existing work lacks effective supports for analyzing the impact of task similarity and task arriving date on task failure. In this study, we approach these gaps by proposing a task scheduling method leveraging combination of evolutionary algorithm and neural network.

The objective of this study 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 motivational example to highlight the practical needs in a 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, system checks the similarity among parallel tasks to make sure they provide the efficient

workers attraction in the platform. The proposed evolutionary task scheduler then recommends a task schedule plan with lowest task failure probability per task to arrive in the platform.

The proposed system represents a task scheduling method for competitive crowdsourcing platforms based on the workflow of TopCoder[1], one of the primary software crowdsourcing platforms. The recommended schedule provides improved duration while decreasing task failure probability. Also the recommended schedule leads to an easier competition over available resource supply due to lower number of open similar tasks upon task arrival in the platform.

The remainder of this paper is structured as follows. Section II introduces a motivating example that inspires this study. Section III presents a review of related work. 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.

II. 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 total 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, which was finally completed on the third attempt, when it was listed as Task 7 with changes in the monetary prize and task type. Task 4 was cancelled and re-posted as Task 6. Task 6 was completed with no modification. Task 8 also failed at first. It was 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 successfully completed as Task 18. This means the original project plan contained 10 tasks. If all of the tasks were completed in the first posting attempt, the project could have been done during 37 days. Figure 2 represent the original plan. 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 propose a scheduling method based on a combination of evolutionary algorithms and deep learning. 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.

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

III. Related Work

A. Task Scheduling in Crowdsourcing

Different characteristics of machine and human behavior create delays in product release [17]. This phenomenon leads to a lack of systematic processes to balance the delivery of features with the available resources [17]. Therefore, improper scheduling would result in task starvation [5]. Parallelism in scheduling is a great method to create the chance of utilizing a greater pool of workers [18] [19]. 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 [5]. Shorter schedule planning can be one of the most notable advantages of using CSD for managers [3].

Batching tasks in similar groups is another effective method to reduce the complexity of tasks and it can dramatically reduce costs [20]. Batching crowdsourcing tasks would lead to a faster result than approaches which keep workers separate [21]. There is a theoretical minimum batch size for every project according to the principles of product development flow [22]. To some extent, the success of software crowdsourcing is associated with reduced batch size in small tasks. Besides, the delay scheduling method [23] 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” [24]. 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) [7] 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) [10]. 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) [25]. The third available method method is HIT-Bundle [7]. 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 [26]. 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.

B. Task Similarity in Crowdsourcing

Generally, workers tend to optimize their personal utility factor when registering for a task [5]. It is reported that workers are more interested in working on similar tasks in

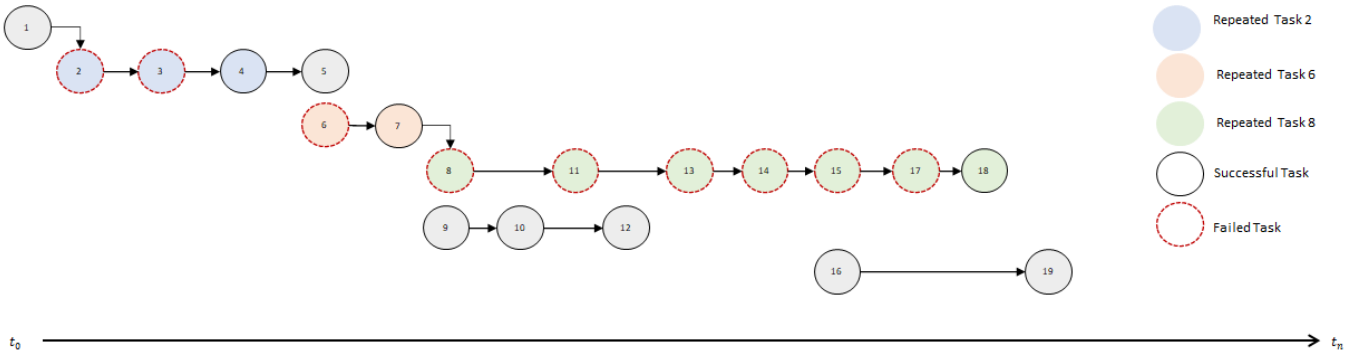

Fig. 1. Overview of Motivation Example

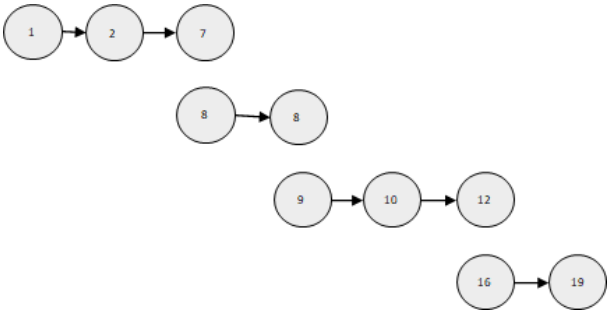

Fig. 2. Original Project Plan

terms of monetary prize [8], context and technology [7], and complexity level [6]. Context switching generates a reduction in workers' efficiency [7]. However, workers usually try to register for a greater number of tasks than they can complete [13]. It is reported that a high task similarity level negatively impacts task competition level and team elasticity [16]. Also, 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 worker availability [9] [10]. 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 [8], 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 [27].

C. 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 [28]. 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 [5], since workers with greater abilities tend to compete with low skilled workers [27]. In this case, users are more likely to choose tasks with fewer competitors [29]. 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 [30]. 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. This research presents an evolutionary based scheduling method which considers market share per arriving task in order to attract trustworthy workers.

IV. Research Methodology

To solve the scheduling problem, we used 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. Then we added a neural network model from [31] to predict the probability of task failure per day. 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 among parallel tasks in the project and the probability of task failure in the platform based on task similarity in the platform and reliability of available workers to make a valid submission. Figure 3 presents the overview of the task scheduling architecture. List of tasks

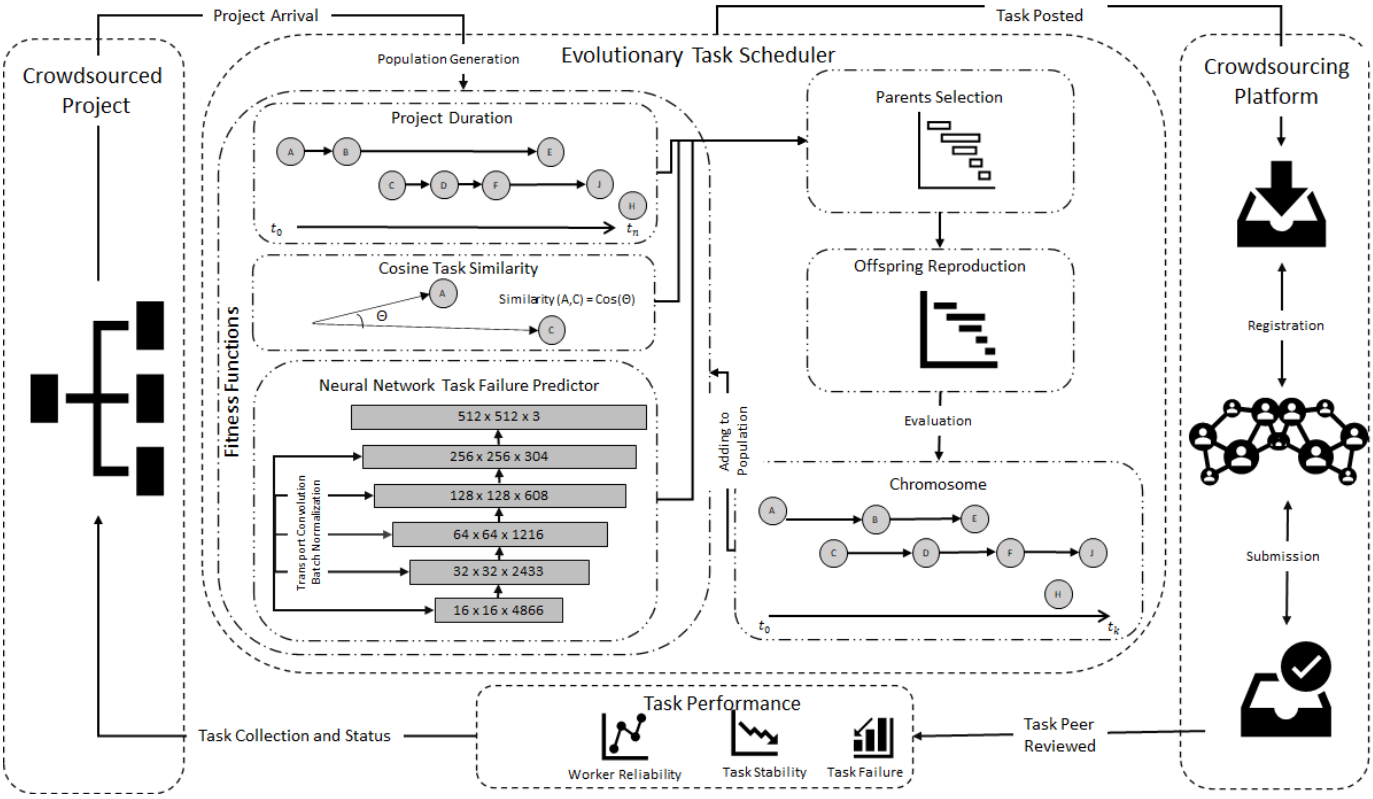

Fig. 3. Overview of Evolutionary Scheduling Architecture

TABLE I
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 (Pr)	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].

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, *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. Then the model recommends a probability of task failure for the assigned date with two days surplus. In next step, the *parent selection* selects the most suitable set of parents to use for *reproduction*, i.e. the generation of new chromosomes. 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.

A. Evolutionary Task Scheduling Conceptual Method

The crowdsourced task scheduling problem is a multi-objective optimization problem with dynamic characteristics that requires to react to changes effectively and track the changing of task similarity and probability of task failure over time. Therefore, 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) [32], which is one of the most efficient algorithm in solving static multi-objective problems.

1) Task Dependency: A crowdsourced project (P_K) is a sequence of time dependent tasks that needs to be completed during specific period of time. In this study, the time sequence of an arriving task is considered as a measurement of the series of parallel or sequential tasks that followed finish-to-start task dependency techniques TD_i. Task (n), T_n, is parallel (P_i) with task (n-1), T_{n-1}, if it starts before T_{n-1}. And T_n is a sequential task (S_i) for T_{n-1} if it starts after T_n is completed. The project duration (PD_K) is defined as the total duration of sequential tasks per project.

$$P_k = (\{T_i\}, \{TD_i\}, PD_k)$$

$$where \begin{cases} \begin{cases} TD_i = \begin{cases} 1 & P_i = 1 \\ 0 & S_i = 1 \end{cases} \\ \text{i= 1, 2, 3, \ldots, n} \end{cases} \\ \begin{cases} PD_k = \sum_{i=0}^{k} S_i \\ \text{K= 1, 2, 3, \ldots, m} \end{cases} \end{cases}$$

2) Task Similarity: Task similarity level is defined as the degree of task similarity between a set of simultaneously open tasks. To analyze task similarity, we calculate the tasks' local distance from each other to analyzed task similarity factor, based on textual task requirements. More specifically, the calculation follows the following detailed definitions.

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 II :

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

3) Task Failure Predictor: To predict probability of task failure in the platform, a fully connected feed forward neural network was trained. It is reported that task monetary prize and task duration [8] [5] 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 features to train the neural network model. 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.

TABLE II
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) = $Diff_TS_{Max}$
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	($Req_i * Req_j$)/($\|Req_i\| * \|Req_i\|$)

To help in understanding of the qualities of the task failure predictor model, the input variables of the model, including task duration, task monetary prize, number of open tasks, and average task similarity 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.

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)(i.e first place) and runner up(PR_i)(i.e second place) will receive after passing peer review.

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

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: 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.

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

$$where : TRE_j >= TR_i$$

To determine the optimal arrival date, we run the neural network 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, there is a 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}.

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

B. Evolutionary Task Scheduling Design

TABLE III
CHROMOSOME

Tasks	A	B	C	D	E	F	G	H	I
Arrival Day	1	7	11	17	33	11	61	28	22

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. So here, one chromosome represents the schedule for the given project. The integer values are bounded between 0 and the maximum allowed duration for the project.

2) Reproduction and variation: Chromosomes are reproduced either through simple copying or through crossover, followed by mutation. We employ standard two-point crossover: Randomly two indices are selected 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.

For mutation, we use a shuffling strategy. For each value in the sequence, an index is chosen randomly for shuffling. Probability for each index to get selected for shuffling is $1/(lengthOfSequence)$; and, the probability of performing variation operation on a given sequence is 0.1.

During initialization of chromosomes, as well as during reproduction, we ensure that every chromosome adheres to the task dependency constraint. A crowd project contains a list of time related tasks. The task dependency constraint is that tasks in the project follow their required dependencies.

$$TR_j = \begin{cases} TR_i & S_{i,j} = 0 \\ TS_i + 1 & S_{i,j} = 1 \end{cases}$$

3) Fitness Functions: We use three fitness functions that respectively seeks to minimize project duration, to minimize probability of task failure in the platform, and to 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.

$$PD_k = \sum_{i=0}^{k} S_i$$

- *Task Similarity*: It is reported that tasks with similarity of 60% lead to the highest level of competition with lower level of task failure [16]. Similarity fitness function makes sure that tasks that are not following the same similarity level and are parallel do not arrive at the same time.

$$\begin{cases} P_{i,j} = 1 & TR_j = \begin{cases} TR_i & Sim_{i,j} = 0.6 \\ TRE_i & Sim_{i,j} \neq 0.6 \end{cases} \\ P_{i,j} = 0 & TR_j = TS_i + 1 \end{cases}$$

- *Task Failure Probability*: The system recommend the arrival date with the lowest probability of task failure based on the result of neural network.

$$TR_i = min(p(TF_d), p(TF_{tm}), p(TF_{tm} + 1))$$

4) Selection Operator: For parent selection operation a crowding-distance based tournament strategy has been used. 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 selection method from NSGA-II.

5) Termination criterion: We ran the evolutionary algorithm for a fixed number of iterations/generations. We get multiple solutions, also called Pareto Optimal, in almost all cases, since we want to optimize multiple objectives.

V. Experiment Design

To evaluate the conceptual model introduced in Section IV, this section presents the experiment design and evaluation base line for this study.

A. 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 in this paper:

RQ1 (Project Schedule Acceleration): How to provide a crowdsourced task scheduling to improve project duration and reduce probability of task failure?

This research question aims to investigate the affect of the proposed method on project duration while decreasing task failure probability in the CSD platform.

RQ2 (Task Similarity): How does similarity among tasks impact on project duration while reducing probability of task failure?

Understanding impact of task inter-dependencies on project duration helps to provide better strategies in task planning.

RQ3 (Scheduling Trade-Off): What is the trade off between RQ1 and RQ2?

Providing an easier competition level per task in the platform, directly reduces the probability of task failure. Therefore providing a task schedule which not only minimize project duration and probability of task failure but also grantees an easier competition per task is the main goal of a project manager.

B. 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. TableI summarizes the task features available in the data set.

C. Implementation of The Evolutionary Task Scheduling

There are three steps in implementing the evolutionary task scheduler: initial population and chromosome representation, fitness function, and task scheduler.

1) Initial Population and Chromosome Representation: Task population is equal to the all the tasks decomposed form the crowd project. Task execution follows the project schedule provided by the project manager. The the initial chromosome follows the original task dependencies introduced by the project manager with 0 days delay. This research used tasks in the project form motivation example as task population and the earliest project schedule was used as initial chromosome. Table IV shows the task dependency in the example project.

TABLE IV
Task Dependency in the Motivation Example

Task ID (T_i)	Sequential (S_i)	Parallel (P_i)	Duration(day) (D_i)
T_1			3
T_2	T_1		4
T_3	T_2		5
T_4	T_2	T_3	2
T_5	T_4		5
T_6	T_4	T_5	5
T_7	T_6		6
T_8	T_7		30
T_9	T_7	T_8	5
T_{10}	T_9		6

2) Fitness Function: In order to manage fitness function in the experiment we followed below steps:

- *Project duration*: The first fitness function measures the complete duration of the project for suggested schedule.
- *Task Similarity*: The similarity function designed to calculate the cosine similarity among the tasks in the project. 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.

$$TR_j = TRE_i$$

$$where: \begin{cases} P_{i,j} = 1 \\ TRE_i <= TRE_j \end{cases}$$

- *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 date with *lowest* failure probability.

VI. Results

The aim of the proposed scheduling method is to enables managers with the task execution plan with lowest task failure probability and easiest competition level per task. In order to answer the research questions in part V-A we scheduled the motivation example with the evolutionary scheduling introduced in part IV-A.

A. Project Schedule Acceleration

In order to have a better understanding of shortening task scheduling while minimizing task failure probability, we studied schedule acceleration of motivation project under evolutionary scheduling. To answer this question we eliminate the second fitness function (Task Similarity) and scheduled the project with focusing on minimizing task failure probability. Figure 4-a illustrates the scheduling result.

As it is shown 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),

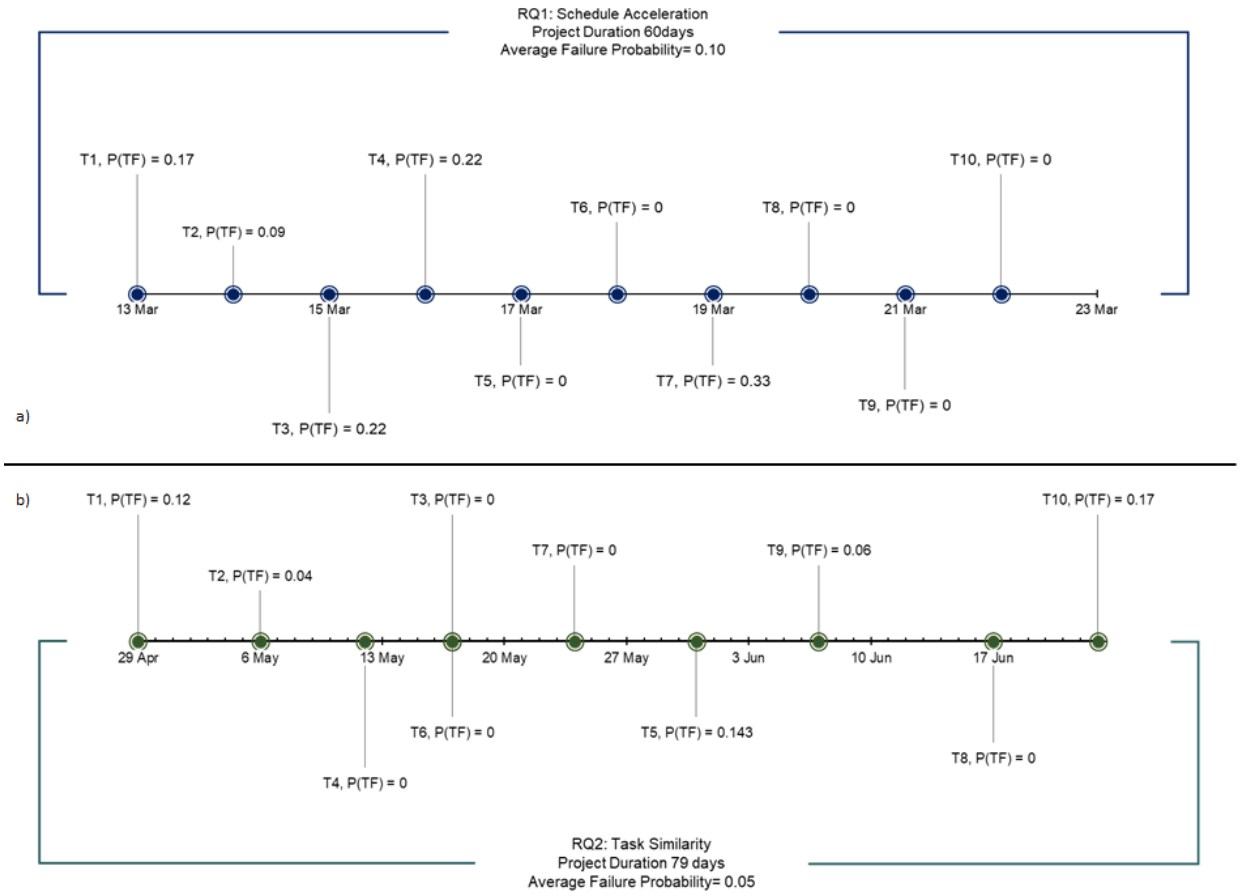

Fig. 4. Schedule Acceleration Timeline

the re commanded schedule finishes 50 days earlier than the original project duration with 47% higher chance of success.

Moreover, 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.

B. Task Similarity

To successfully crowdsource a software project, not only is it important to achieve a task scheduling plan to provide lowest probability of task failure, but also it is vital to provide easy competition level per task. 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 79days 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 to attract resources.

Moreover, the method recommended to postpone the start date of the project for 47 day, April 29th, in order to have 0 days overlap due to task similarity.

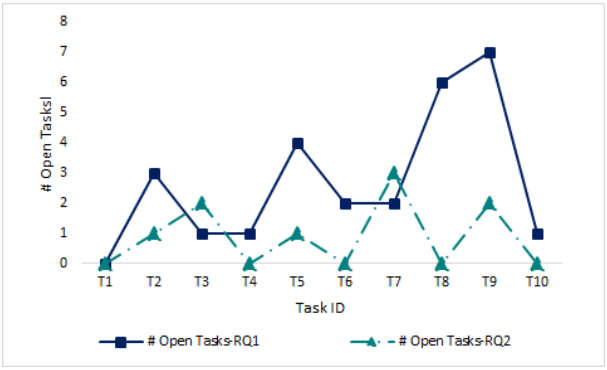

Fig. 5. Number of Open Tasks on Task Arrival Day

C. Scheduling Trade Off

To have a 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 arrival date.

Figure 5 presents the number of open task per arrival task based on different scheduling plan in RQ1 and RQ2. Scheduling under decreasing task failure while improving the project duration lead to competition level with on average 3 other tasks per day. The easiest 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 in to account creates easier competition level per arriving tasks. As it is shown the result of 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 6 the scheduling with focus on improving project duration and decreasing task failure probability. Tasks are arriving to the platform with on average 32% task similarity. T_3,T_4,T_5, andT_6 are competing with tasks with more than 75% similarity. While adding task similarity fitness function to scheduling process provides task arrival with on average 14% task similarity in the platform.T_4 faced the highest task similarity level of 65%.

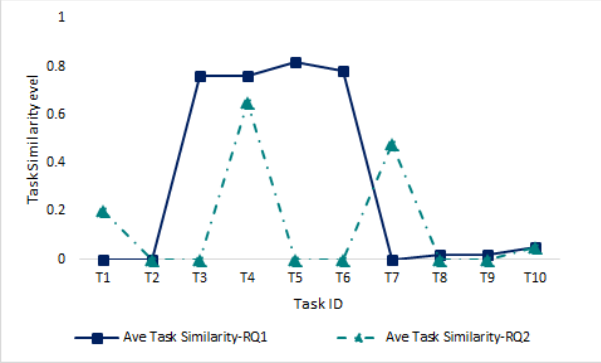

Fig. 6. Average Task Similarity on Task Arrival Day

D. Discussion and Finding

The recommended plan that provides schedule acceleration of 27% (i.e 79/110). Also, the result of evolutionary scheduling provided a plan with average probability of task failure of 5%. This result not only is 53% lower than the original plan of the project but also it is lower than reported failure ratio in the platform (i.e 15.7%) [33] [13]. Figure 7 shows the average probability of task failure per recommended schedule by evolutionary scheduler. According to the figure 7 longer project duration provides lower probability of task failure. The goal is to reducing schedule acceleration while reducing task failure, hence, the optimum solutions occurs under 5 recommended schedule with duration around 70 days which provides task failure probability of 4%(shown in green) .

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

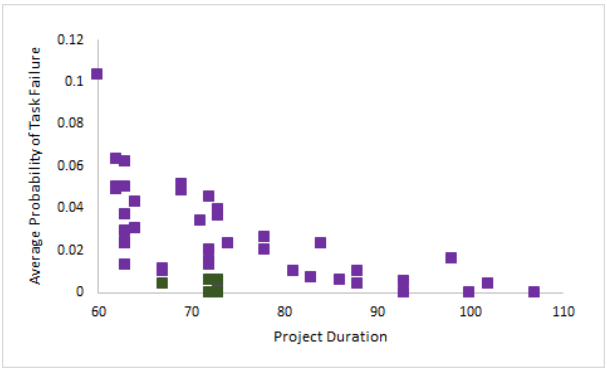

Fig. 7. Average Task failure Probability per Recommended Task Schedule

1 and 14% respectively). Figure 8 illustrates the average task similarity cost per recommended task schedule. Task similarity cost is the ratio of duration added to the recommended task schedule due to considering task similarity fitness function. As it is shown in figure 8 the lowest cost happens when either project duration is between 70 days to 80 days or more than 100 days. Since the main objective is reducing project during the optimum solution is project duration of 70-80 days(shown in green).

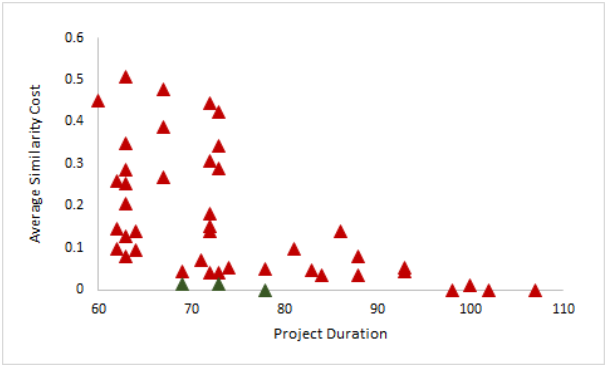

Fig. 8. Average Task Similarity Cost per Recommended Task Schedule

E. 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.

VII. 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 proposes an evolutionary task scheduling method. 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 recommends the best start date for the project as a whole and each individual task so as to achieve the lowest project failure ratio.

In future, we would like to expand our model to provide a platform level scheduler which not only recommends the best scheduling plan but also provides required task decomposition action to achieve the lowest task failure.

References

[1] K.-J. Stol and B. Fitzgerald, "Two's company, three's a crowd: a case study of crowdsourcing software development," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 187–198.

[2] R. L. Saremi, Y. Yang, G. Ruhe, and D. Messinger, "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)*. IEEE, 2017, pp. 103–112.

[3] K. R. Lakhani, D. A. Garvin, and E. Lonstein, "Topcoder (a): Developing software through crowdsourcing," *Harvard Business School General Management Unit Case*, no. 610-032, 2010.

[4] A. Kittur, J. V. Nickerson, M. Bernstein, E. Gerber, A. Shaw, J. Zimmerman, M. Lease, and J. Horton, "The future of crowd work," in *Proceedings of the 2013 conference on Computer supported cooperative work*, 2013, pp. 1301–1318.

[5] S. Faradani, B. Hartmann, and P. G. Ipeirotis, "What's the right price? pricing tasks for finishing on time," in *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[6] G. Gordon, "A general purpose systems simulation program," in *Proceedings of the December 12-14, 1961, eastern joint computer conference: computers-key to total systems control*, 1961, pp. 87–104.

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

[8] Y. Yang and R. Saremi, "Award vs. worker behaviors in competitive crowdsourcing tasks," in *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2015, pp. 1–10.

[9] A. Khanfor, Y. Yang, G. Vesonder, G. Ruhe, and D. Messinger, "Failure prediction in crowdsourced software development," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2017, pp. 495–504.

[10] R. Khazankin, H. Psaier, D. Schall, and S. Dustdar, "Qos-based task scheduling in crowdsourcing environments," in *International Conference on Service-Oriented Computing*. Springer, 2011, pp. 297–311.

[11] R. Saremi, "A hybrid simulation model for crowdsourced software development," in *Proceedings of the 5th International Workshop on Crowd Sourcing in Software Engineering*, 2018, pp. 28–29.

[12] R. Saremi, Y. Yang, and A. Khanfor, "Ant colony optimization to reduce schedule acceleration in crowdsourcing software development," in *International Conference on Human-Computer Interaction*. Springer, 2019, pp. 286–300.

[13] Y. Yang, M. R. Karim, R. Saremi, and G. Ruhe, "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*, 2016, pp. 1–10.

[14] H. Wang, Z. Ren, X. Li, and H. Jiang, "Solving team making problem for crowdsourcing with evolutionary strategy," in *2018 5th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE, 2018, pp. 65–74.

[15] T. Yue, S. Ali, and S. Wang, "An evolutionary and automated virtual team making approach for crowdsourcing platforms," in *Crowdsourcing*. Springer, 2015, pp. 113–130.

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

[17] G. Ruhe and M. O. Saliu, "The art and science of software release planning," *IEEE software*, vol. 22, no. 6, pp. 47–53, 2005.

[18] A. Ngo-The and G. Ruhe, "Optimized resource allocation for software release planning," *IEEE Transactions on Software Engineering*, vol. 35, no. 1, pp. 109–123, 2008.

[19] R. L. Saremi and Y. Yang, "Empirical analysis on parallel tasks in crowdsourcing software development," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*. IEEE, 2015, pp. 28–34.

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

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

[22] D. G. Reinertsen, "Celeritas publishing," 2009.

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

[24] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *Nsdi*, vol. 11, no. 2011, 2011, pp. 24–24.

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

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

[27] N. Archak, "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*, 2010, pp. 21–30.

[28] R. G. Cooper and A. F. Sommer, "The agile–stage-gate hybrid model: A promising new approach and a new research opportunity," *Journal of Product Innovation Management*, vol. 33, no. 5, pp. 513–526, 2016.

[29] J. Yang, L. A. Adamic, and M. S. Ackerman, "Crowdsourcing and knowledge sharing: strategic user behavior on taskcn," in *Proceedings of the 9th ACM conference on Electronic commerce*, 2008, pp. 246–255.

[30] A. Rapoport and A. M. Chammah, "The game of chicken," *American Behavioral Scientist*, vol. 10, no. 3, pp. 10–28, 1966.

[31] J. Urbaczek, R. Saremi, M. L. Saremi, and J. Togelius, "Scheduling tasks for software crowdsourcing platforms to reduce task failure," *arXiv preprint arXiv:2006.01048*, 2020.

[32] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[33] D. Martinez Mejorado, R. Saremi, Y. Yang, and J. E. Ramirez-Marquez, "Study on patterns and effect of task diversity in software crowdsourcing," *arXiv*, pp. arXiv–2006, 2020.