Ant Colony Optimization to Reduce Schedule Acceleration in Crowdsourcing Software Development

Razieh Saremi(✉), Ye Yang, and Abdullah Khanfor

Stevens Institute of Technology, Hoboken, NJ 070390, USA
{rlotfali,yyang4,akhanfor}@stevens.edu

Abstract. The complexity of software tasks and the variety of developer skill sets requires to accomplish the tasks, provides a challenge in the planning process for software project managers. Uncertainty based on crowd workers' different time zone and first language adds a layer of complexity to the CSD task scheduling. Therefore, accessing a scheduling model which can ease task allocation to improve task success and decrease project duration is essential. Existing models are either focused on the task allocation based on workers quality, or task availability in the crowdsourced platform. To create a flexible and effective model in CSD, we present an Ant Colony Optimization algorithm. The proposed approach shows a plan based on a list of available tasks in the platform and available workers based on their performance and rating metrics. The presented model is composed of four components: task fitness, workers' attraction, task-worker availability, and task scheduler. Experimental results on 408 projects demonstrate that the proposed method reduced project duration on average 74 days.

Keywords: Crowdsourced software development · Ant Colony Optimization · Task fitness · Task similarity · Workers' availability · Topcoder

1 Introduction

There is an emergent trend in software development projects that mini-tasks can be crowdsourced to achieve accelerated development and delivery. The characteristics of crowdsourcing tasks, in general, are short, simple, repetitive, requires little time and effort, while in crowdsourced software development (CSD) tasks are more complex, independent and requires a significant amount of time, effort, and expertise to achieve the task requirements [1]. For task owners, requesting a crowdsourcing service is more challenging due to the uncertainty of the similarity among available tasks in the CSD platform and the new arrival tasks, [2, 3], as well as, crowd workers' skill sets and performance history [4, 5] in the platform. These factors raise the issue of fitness of the assigned worker to the suitable task, since crowd workers may be interested in multiple tasks from different requestors among a pool of open tasks based on their individual goals and preference which provides resource inconsistency. It is reported that workers are more interested in working on tasks with similar concepts, monetary prize, technologies, complexities, priorities, and durations [3, 5–7]. Attracting workers to a group of similar tasks may cause zero registration, zero submissions, or unqualified submissions for some tasks due to lack of time from workers [2, 8, 9]. Therefore, it is beneficial to make efficient project plans to be able to assure the availability of crowd to work on the arrival CSD tasks and to submit them on time. To receive qualified submission as a task outcome in CSD, a good understanding of task characteristics, and crowd workers sensitivity to arrival tasks is required. To address that, we presented a task scheduling approach in CSD based on ant colony optimization (ACO) algorithm [41]. The proposed method represents a plan by a task list and a planned crowd allocation matric based on the workflow of Topcoder [10], one of the primary software crowdsourcing platforms.

The paper is organized as follows: Sect. 2 introduces the background and related work; Sect. 3 presents the research design and methodology; Sect. 4 reports the experimental design of the proposed model. Section 5 Limitation and Treats to Validity; and finally, Sect. 6 gives a Conclusion and outlook to future work.

2 Background and Related Work

2.1 Task Scheduling in Software Engineering

In the traditional software development process, task scheduling is the next step occurs after defining requirement development and deciding on task decomposition [11], which takes place based on specific groups of requirements [12]. There is a limitation on literature addressing scheduling in software development even though 80% of the software projects suffering from lack of schedule planning [13]. In practice, more than half of software requirements are interdependent, and only a few of them are independent, which notably impacts scheduling complexity [12].

To overcome the scheduling challenge different traditional scheduling techniques such as program evaluation and review technique (PERT), the critical path method (CPM) [39], and the resource-constrained project scheduling problem (RCPSP) [40] model have been used in software engineering. However, due to nature of software projects that is a people-intensive activity [13], task dependencies make available human resources awaiting completion of pre-requisites. Traditional scheduling techniques do not consider human resource allocation in scheduling, or if they do, they do not cover the resources with various skills. Since human is the main resource in software projects, scheduling can be more flexible in comparison with other industries [14]. One of the approaches to address this need can be search based optimization problem.

Recently, project managers have been using global software development techniques to reduce cost and shorten release time although, different time zone and resource location may add to the scheduling challenges [15, 16]. In which, team members are distributed into different zones and countries. It seems the best practice to optimize task scheduling in this approach is following the sun [17]. Follow the sun happens when software development is distributed over a twenty-four-hour working day. In this method, tasks are distributed among workers with different time zone, and all the team members work on the same phase and tasks of the project in their own day hours.

2.2 Task Scheduling in CSD

Due to the different characteristics of the machine and human behavior, delays can occur in product release and lack of systematic processes to balance the appropriate delivery of features with the available resources [18]. Therefore, improper scheduling would result in task starvation [6]. Parallelism in scheduling is a great method to create the chance of utilizing a greater pool of workers [19, 42] as this method encourages workers to specialize and complete the task in shorter period and promote solutions in which benefits the requestor to clearly understand how workers decide to compete on a task and analyze the crowd workers performance [6]. Shorter schedule planning can be one of the most notable advantages of using CSD for managers [20].

Complex crowdsourced projects cannot be performed based on simple available parallel approaches. Complex projects have more dependencies and multiple occurrences of changing requirements [21]; they require different workers with diverse levels of expertise. Therefore, this is one of the main challenges in applying an effective method to schedule decomposed projects in crowdsourcing [22]. Since coordinating workers is difficult among a distributed global crowd, in most cases, organizational coordination techniques such as programming and feedback as general coordination methods can be applied to crowd work as well [23, 24].

Batching tasks is another effective method to reduce the complexity of tasks, and it will dramatically reduce cost [25]. Batching crowdsourcing tasks would lead to a faster result than approaches, which keep workers separate and is also quicker than the average of the fastest individual worker [26]. There is a theoretical minimum batch size for every project as one of the principles of product development flow [27]. To some extent, the success of software crowdsourcing is associated with reduced batch size in small tasks.

Besides, the delay scheduling method [5] was specially designed for crowdsourced projects to maximize the probability of a worker receiving tasks from the same batch of tasks they were performing. Extension of this idea introduced a new method called "fair sharing schedule" [28]. In this method, various resources would be shared among all tasks with different demands, which ensures that all tasks would receive the same amount of resources to be fair. For example, this method was used in Hadoop Yarn. Later, Weighted Fair Sharing (WFS) [3] was presented as a method to schedule batches based on their priority. Tasks with higher priority are introduced first. Another proposed crowd scheduling method is based on quality of service (QOS) [9], 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) [29]. The third available method would be a game with a purpose [30], in which a task will not be started unless a certain number of workers registered for it. The most recent method is HIT-Bundle [3] a batch container which schedules heterogeneous tasks into the platform from different batches. This method makes for a higher outcome by applying different scheduling strategies at the same time.

2.3 Workers' Motivation

The diversity of software workers in crowdsourcing, makes motivational factors be typically divided into two categories: intrinsic factors and extrinsic factors. First, intrinsic clusters comprise enjoyment factors and community values that are associated with age, location, personal career, society and even task identity [31]. In addition, participation motivation categories [32] such as learning, and self-marketing are considered as some of the main intrinsic factors for workers which are mentioned in the concept of "activation enabling". Second, Extrinsic clusters include financial and social aspects, which are the direct effect of the educational background, household income, and task award or payment [31]. It is reported that [32], motivations such as organizer appreciation, prizes, and knowledge experts are the highest ranked among workers. Another high-ranking motivation factor is the presence of requesters with a prestige brand name, such as Google, or NASA, which attracts software workers to apply for tasks, as it can potentially be used to strengthen resumes and affect software workers ratings either indirectly or directly [33].

Different studies on motivation patterns of crowdsourcing workers reported that monetary prize is one of the top motivating factors to attract and involve potential workers in task competition in crowdsourcing market [1, 6, 34]. The monetary prize typically correlates with the degree of task complexity and required competition levels as well as task priority in the project development [6, 34]. The second motivation factor which affects monetary prize is the worker's skill level [31]. Higher skilled level workers generally have higher motivation level of winning a prize and gaining communal identification while the top motivational factors of occasional workers are typically a pas-time human capital investment and skill variety. Therefore, task requesters usually struggle to distribute awards for the tasks with a good scheduling plan in the hope of getting the best possible task submissions in terms of the tasks' type based on workers historical activities and submissions [6].

2.4 Challenges in CSD

Considering the highest rate for task completion and accepting submissions, software managers will be more concerned about the risks of adopting crowdsourcing. Therefore, there is a need for better decision-making system to analyze and control the risk 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 the resources. It can help managers to 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 [17]. However, in CSD all three variables are flexible. This factor creates a huge advantage in crowdsourcing software projects.

Generally, improper scheduling could lead to task starvation [6], since workers with high abilities tend to compete with low skilled workers [34]. Hence, users are more likely to choose tasks with fewer competitors [33]. Also, workers intentionally choosing less popular tasks to participate could potentially enhance winning probabilities, even if workers share similar expertise. It brings some severe problems in the CSD trust system and causes a lot of dropped and none- completed tasks. Moreover, tasks with relatively lower monetary prizes have a high probability to be chosen and be solved, which results in only 30% of problems in platform being solved [24]. This may attract higher numbers of workers to compete and consequently makes the higher 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 trustable and expert workers as well as shorten the release time.

3 Research Design and Methodology

To solve the scheduling problem, we present an ant colony optimization approach. We extend the model proposed in [35] to be adopted to CSD domain. A scheduling approach with an event-based scheduler and an ant colony optimization algorithm is introduced in [35]. In each colony, ants provide some pheromone (chemical tracking) to react to workers and tasks history as the pheromone (workers and tasks history) to present a better scheduling method. The presented approach represents a task list and a planned worker allocation matrix. To create the final task scheduler, it is essential to understand required pheromones in the model. This model contains three heuristic pheromones of time box, task scheduling batching, and worker availability.

3.1 Time Box

A project manager is creating an ordered list of decomposed tasks per project. In CSD planning, each task can be associated with multiple workers, the minimum slack time (MINSK) process [36] per task is used to create the time pheromone. This heuristic is usually used for sequential task scheduling adoption. The task with smaller MINSK implies that it is either more urgent or simpler to perform. The MINSK for a task can be estimated as follow:

Definition 1: MINSK is the amount of time a task may be delayed from its posting date without delaying the submission due date is total slack of task (i),TST$_i$:

$$TST_i = LED_i - EED_i \, or \, TST(i) = LST_i - EST_i \tag{1}$$

In which:

EST$_i$, the task registration date in the platform,
LST$_i$, latest task registration date in the platform,
EED$_i$, the task submissions date in the platform,
LED$_i$, the worker submissions date in the platform.

Definition 2: Amount of time a task can be delayed from its earliest starting time without delaying the starting time of any of its immediate sequential tasks is called free slack of task (i), FST_i:

$$FST_i = Min\left(EED_i - ESD_i\right) \tag{2}$$

Definition 3: Task with minimum slack are bottlenecks. Bottleneck task (i), BNT_i, calculates as diverse ratio of minimum slack time per task (TST_i). The smaller the BNT_i, is the more urgent the task is, in terms of schedule as early as possible.

$$BNT_i = \frac{1}{TST_i} \tag{3}$$

To build a feasible list of tasks by workers, each worker creates an eligible list of tasks that satisfy their skillset. The list maker includes the following steps:

Step a: add the task that the worker is interested in the eligible list of tasks.
Step b: for k = 1 to n:
Step b-1: select a task from eligible list and put it in the kth position of the task list following random number (v).

Definition 4: The probability of task T_i is chosen to be registered by workers based on time box pheromone is:

$$TBTR_i = \frac{T_i * BNT_i}{n},\ where\ n\ is\ total\ number\ of\ task\ per\ project \tag{4}$$

This means that the task has a probability of being taken with the maximum pheromone value, so the algorithm can strongly exploit the past search of tasks.

Step b-2: update the eligible list by removing the selected task from the eligible list and adding a new feasible task that satisfy the workers' skillset.

After step b-1 and b-2 repeated n times a feasible task list is built.

3.2 Task Similarity Batching

Task Similarity Analysis

To analyze task similarity in the platform there is a need to understand the tasks' local distance from each other and task similarity factor based on it.

Definition 5: Task local distance (Dis_j) it is a tuple of all tasks' attributes in the data set. In respect to introduce variables in Table 1, task local distance is:

$$Disi = (Award, Registration\ date, Submission\ Date, Task\ type, Technology, Task\ requirement) \tag{5}$$

Table 1. Description of top 50% of task type in CSD platform [10]

Task type	Description
First2Finish	The first person to submit passing entry wins
Assembly competition	Assemble previous tasks
Bug hunt	Find and fix available bugs
Code	Programming specific task
UI prototype	User Interface prototyping is an analysis technique in which users are actively involved in the mocking-up of the UI for a system
Architecture	This contest asks competitors to define the technical approach to implement the requirements. The output is a technical architecture document and finalized a plan for assembly contests
Test suit	Competitors produce automated test cases to validate the quality, accuracy, and performance of applications. The output is a suite of automated test cases

Definition 6: Task Similarity Factor (TSi,j) is dot product and magnitude of local distance of two tasks:

$$TSi,j = \frac{\sum_{i,j=0}^{n} Disi(Tj, Ti)}{\sum_{i=0}^{n} \sqrt{Disi(Ti)} * \sum_{j=0}^{n} \sqrt{Djsj(Tj)}} \tag{6}$$

Task Similarity Pheromone

The similarity matrix specifies the causal similarity among tasks. Before creating the heuristics, it is important to understand the fitness of task (i), FT_i, in the platform.

Definition 7: the fitness of task (i), FT_i, in the platform to receive registration ix:

$$FTi = \frac{\sum_{i=1}^{n} (TSi * MaxRi * TDi)}{\sum_{i=1}^{n} TDi} \tag{7}$$

Now, to create the pheromone, we assumed there is a minimum number of registrations for tasks, which helps a task get a higher chance of being successfully complete. If a task leaves the pool of open tasks and enter the pool of registered tasks, there will be minimum 1 registration associated with the task. This is called the initial task registration (ITR).

Definition 8: Probability of task (i) from similarity batch (j) is registered by a worker is:

$$TR(i,j) = FTi * ((1 - TSi) * N + TSi * ITR) \tag{8}$$

While N is the total available task in the platform in the same timeframe.

This heuristic helps project managers to choose the best so far task in their list and have a higher chance for receiving a qualified submission.

3.3 Workers Availability

Last heuristics is helping to make a suitable pheromone for worker availability. The heuristic of worker (j) attraction on registering for task (i) is define as Worker Attraction (i, j), AW_{ij}:

$$WA_{ij} = \begin{cases} MinRej + AveRej * WERTj & WRDj = TPDi \\ MinRej + AveRej * WERTj + (WRDij - TDi) * MaxRej & TPDi < WRDj \le TSDi \end{cases} \quad (9)$$

In which:

Definition 9: Minimum reliability of worker (j), $MinRe_j$, represent the lowest reliability associate to worker j before taking the task (i) and after taking the task (i).

$$MinRe_j = Min\left\{OldReliability_j, NewReliability_j\right\} \quad (10)$$

Definition 10: Average reliability of worker (j), $AveRe_j$, represent the average reliability associate to worker j before taking the task (i) and after taking the task (i).

$$AveRe_j = \frac{OldReliability_j + NewReliability_j}{2} \quad (11)$$

Definition 11: Maximum reliability of worker (j), MaxRej, represent the highest reliability associate to worker j before taking the task (i) and after taking the task (i).

$$MaxRe_j = Max\left\{OldReliability_j, NewReliability_j\right\} \quad (12)$$

Definition 12: Worker (j) earliest registration time, WERTj, represent the minimum worker task registration time, WTRTj associate to worker j in worker's task registration history.

$$WERTj = Min\{WTRT1, WTRT2, \ldots, WTRTj\} \quad (13)$$

Definition 13: Registration date of worker (j) for task (i), WRDij, is the date worker j registered for task i.

Definition 14: The pheromone heuristic of worker availability is the probability that worker (k) will submit the taken task (i):

$$TS(i,k) = WAij * Prob(WRi) * PLSi * M \quad (14)$$

While M is total registered task in the platform in the same time frame.

The upper bound estimation of workers' availability is 1. This means that all the workers in the platform have required skillset for performing the arrival task.

The scheduling problem in CSD is tracible as both task list and the workers' allocation have to be optimally decided. However, there are heuristics that can help us to build the task list and select workers' allocation, it is difficult to create an effective heuristic that can predict the exact workers who registers for the task. Therefore, we coupled the presented AOC model with a search-based procedure.

Table 2. Workers availability in CSD platform [38]

Workers' belt	Rating range (X)	% workers
Gray	X < 900	90.02%
Green	900 < X < 1200	2.88%
Blue	1200 < X < 1500	5.39%
Yellow	1500 < X < 2200	1.54%
Red	X > 2200	0.16%

4 Experimental Studies

4.1 Data Set

The gathered dataset contains 403 individual projects including 4907 component development tasks and 8108 workers from Jan 2014 to Feb 2015, extracted from Topcoder website [10]. After removing workers who never registered or submitted any task in their membership history, the number of active workers reduced to 5062 workers.

In the experiment phase we test the proposed method on 408 projects and 4900 decomposed tasks performed by 5700 workers.

Based on the Topcoder definition we clustered tasks under 14 task type [37], however we have considered 7 highest common task type in this research. Table 1 summarized the description of the considered task types per project decomposed tasks.

Also, Topcoder categorized workers under 5 group belts based on workers rating [38]. Table 2 summaries worker distribution respectively.

4.2 Experimental Setting

The main objective in this research is to minimize the project duration and maximizing the task success rate. Therefore, we need to update the set up the experiment based on workers' arrival in the platform to schedule the task arrival. The parameters of the proposed ACO are set as follows:

The start number of arrival ants follows the average of competition level per tasks and equal to 18 [7] and the minimum number of workers registering per task is 1. The umber of task arrival in the platform follows the task arrival schedule based on the actual data gathered from Topcoder. Moreover, the assigned random number in choosing the eligible list of tasks is used as the degree of aggressiveness in the algorithm. The larger the random number is, the stronger the algorithm exploits the past search experience and exhibits a fast converging manner.

4.3 Results and Analysis

Task Attraction

According to the result, tasks with duration of 3 weeks or less provides higher level of average attraction for workers to perform and make a submission. Interestingly tasks with 27–29 days duration provide the highest level of attraction for workers with degree of attraction of 99%. Figure 1 illustrates the overall result of task attraction after applying ACO model.

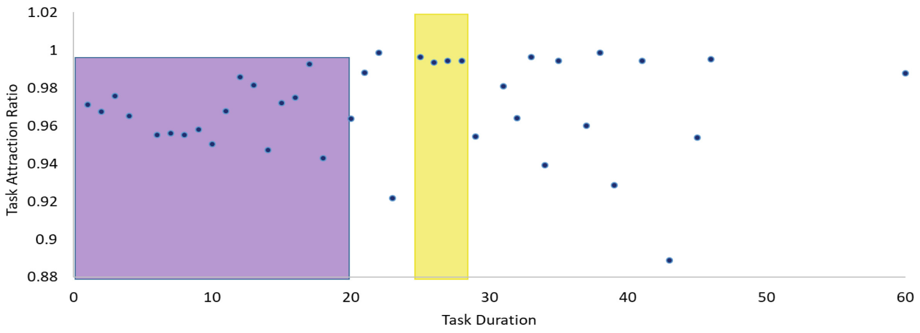

Fig. 1. Task attraction in the CSD based on ACO

Task Fitness

Deeper investigation of task fitness in the platform shows that tasks with duration less than 10 days and similarity degree of 60% and above would provide the average task fitness of 17%. Task with duration between 10 and 20 days reduce the task fitness to 10%. And interestingly task with duration more than 20 days dropped the task fitness dramatically. Figure 2 shows the details of task fitness based on different task duration and similarity degree of 60% and above.

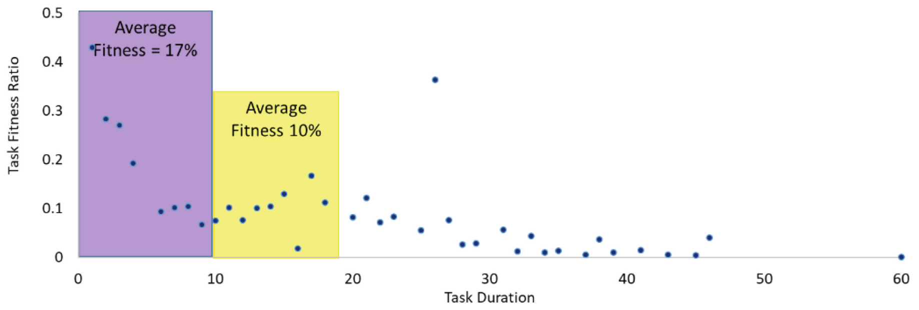

Fig. 2. Task fitness in CSD based on ACO

Workers' Attraction

Applying worker reliability pheromone provides the result that tasks with duration less than 10 days can attract on average 13 workers to perform them. And tasks wit duration between 25–35 days can attract on average 9 workers to perform them. While tasks with duration between 10–25 days and more than 35 days can attract less than 8 workers to perform them. Figure 3 presents the distribution of workers' attraction and task duration based on worker availability pheromone.

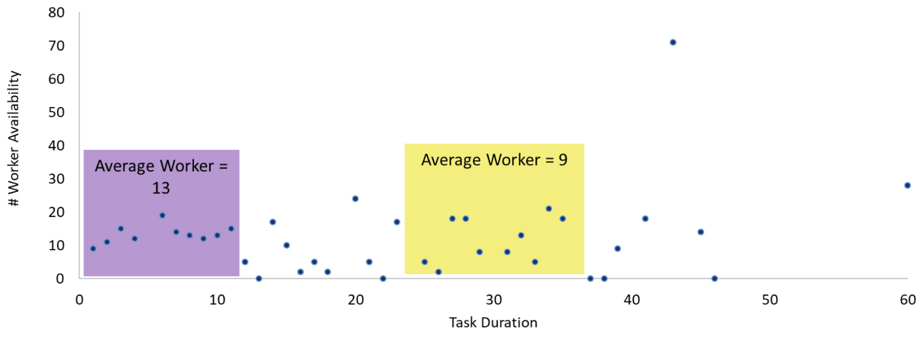

Fig. 3. Distribution of workers' attraction in CSD based on ACO

Task-Worker Attraction

Applying all the 3 pheromones to the dataset provides the task-worker attraction metrics. As it is shown in Fig. 4, 20 new arrival task per week in the platform results in to attracting on average 90 workers to perform the tasks, while 20–40 task arrival will lead to attracting on average 168 active worker. 40–60 tasks provide on average 200 available worker and 80 tasks brings 348 available workers in the platform.

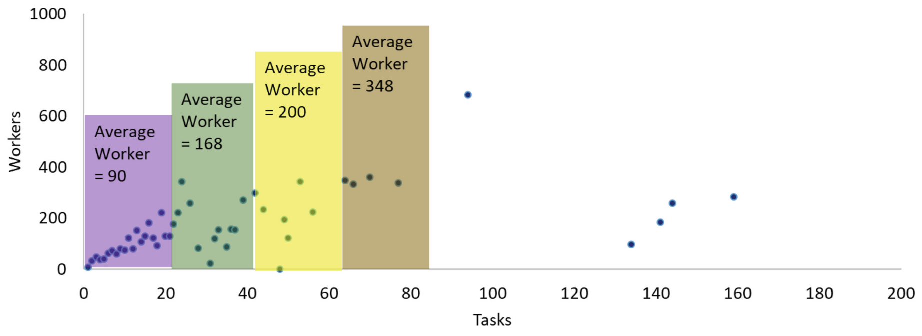

Fig. 4. Task-worker attraction in CSD based on ACO

Schedule Acceleration

Applying overall ACO scheduling model reduced the project duration to on average 74 days. As it is presented in Fig. 5, projects with decomposed number of 50 tasks can be done with in 4 weeks. Projects that are decomposed between 50–100 tasks need 6 weeks duration to be complete. Projects with number of tasks between 100–150 require 9 weeks of schedule duration. And all the projects with more than 150 tasks can be completed with on average of 12 weeks duration. 83% of the projects can be scheduled to be complete with in 10 weeks or less.

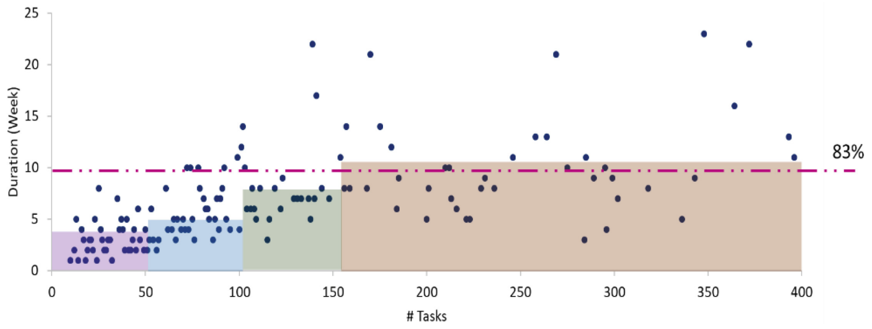

Fig. 5. Schedule acceleration in CSD based on ACO

Table 3 summarized the overall status of the projects in our dataset. As it is presents, the suggested ACO scheduling method decreased the task failure ratio for average of 2%, increased the Workers reliability in making a submission for 1.5% and Workers trust-ability in not only making a submission but make a valid submission on average 12%.

Table 3. Summary of project level task scheduling in CSD

Number of task per project	Number of project	Duration (Week)		Failure ratio		Worker reliability		Worker trust-ability	
		Original	Suggested	Original	Suggested	Original	Suggested	Original	Suggested
50	194	10	4	16%	15%	13%	13%	69%	72%
100	68	11	5	17%	16%	14%	14%	68%	81%
150	36	12	5	18%	16%	13%	15%	72%	85%
200	12	11	4	20%	20%	13%	14%	67%	83%
250	11	9	4	23%	16%	13%	14%	73%	85%
300	10	19	6	18%	16%	14%	15%	44%	69%
350	6	12	5	22%	19%	12%	12%	57%	79%
400	4	30	10	17%	15%	10%	12%	57%	73%
450	1	1	1	18%	14%	13%	22%	80%	85%
500	7	30	9	18%	13%	13%	15%	51%	70%
550	1	9	2	20%	16%	12%	16%	84%	95%
600	0	0	0	0%	0%	0%	0%	0%	0%
650	1	22	9	25%	17%	12%	14%	85%	89%
700	5	4	1	18%	17%	14%	15%	66%	79%
750	1	2	2	15%	14%	12%	13%	53%	77%
800	0	0	0	0%	0%	0%	0%	0%	0%

5 Limitation and Threats to Validity

There are several threats to validity. First the presented model is based on the data gathered during Jan 2014–Feb 2015. There is a chance that data set from different time line may lead to slightly different result.

Second, the proposed method does not consider worker's multi-tasking factors, i.e., how many tasks can a worker take, at maximum, during the same period of time. Though this is one of the important factors influencing software project scheduling decisions, we don't see there is a pattern in the upper bound limits for worker's multi-tasking.

Third, the data preparation and similarity analysis are complicated by the temporal nature of task attributes and descriptions. However, based on the utility factor of each individual workers, different sets of task attribute can be used to analyze similarity among tasks.

Lastly, the data sets used in this study for both training and testing are mostly unbalanced data sets in terms of the number of samples belonging to different classes for both tasks and workers. The issue is not addressed in the presented model and will be considered in our future work.

6 Conclusion

CSD provides software organizations access to infinite worker resource supply from the Internet. Assigning tasks to a pool of unknown workers from all over the glob is challenging. A traditional approach to solve such challenge is scheduling. Generally, dependencies among tasks and workers make task scheduling an NP-hard problem. Improper task scheduling in CSD may cause to zero task registration, zero task submissions or low qualified submissions due to uncertain workers behavior. This research addressed a new scheduling model for solving the CSD planning problem has been developed. The proposed model adopted ant colony heuristics to solve the complicated planning problem via CSD. The experimental experience lead to reducing project duration for average of 74 days (i.e. 2.5 months) and provide a more stable workload assignment to available workers in the platform.

In the future research we will focus on the trust network among available workers and apply the proposed scheduling model based on the recognized trust network in the platform, to understand the impact of assigned workers to the same task on workers' decision making and consequently task failure ratio.

References

1. Stol, K.-J., Fitzgerald, B.: Two's company, three's a crowd: a case study of crowdsourcing software development. In: The 36th International Conference on Software Engineering (2014)
2. Saremi, R.: A hybrid simulation model for crowdsourced software development. In: Proceedings of the 5th International Workshop on Crowd Sourcing in Software Engineering (2018)

3. Difallah, D.E., Demartini, G., Cudré-Mauroux, P.: Scheduling human intelligence tasks in multi-tenant crowd-powered systems. ACM (2016)
4. Yang, Y., Karim, M.R., Saremi, R., Ruhe, G.: Who Should take this task? – Dynamic decision support for crowd workers. In: Proceedings of ESEM (2016)
5. Gordon, G.: A general purpose systems simulation program. In: EJCC, Washington D.C. (1961)
6. Faradani, S., Hartmann, B., Ipeirotis, P.G.: What's the right price? Pricing tasks for finishing on time. In: Proceedings of the Human Computation (2011)
7. Yang, Y., Saremi, R.L.: Award vs. worker behaviors in competitive crowdsourcing tasks. In: ESEM, Benjin, Chaina, Imperical Software Engineering and Measures (2015)
8. Khanfor, A., Yang, Y., Vesonder, G., Ruhe, G.: Failure prediction in crowdsourced software development. In: 24th Asia-Pacific Software Engineering Conference (2017)
9. Khazankin, R., Psaier, H., Schall, D., Dustdar, S.: QoS-based task scheduling in crowdsourcing environments. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) ICSOC 2011. LNCS, vol. 7084, pp. 297–311. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25535-9_20
10. Topcoder Website. http://www.Topcoder.com
11. Mao, K., Capra, L., Harman, M., Jia, Y.: A survey of the use of crowdsourcing in software engineering. Technical report RN/15/01, Department of Computer Science, University College London (2013)
12. Ma, C.: The Business of Software. Free Press, New York (2004)
13. Regnell, B., Brinkkemper, S.: Market-driven requirements engineering for software products. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements. Springer, Heidelberg (2005). https://doi.org/10.1007/3-540-28244-0_13
14. Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Nattoch Dag, J.: An industrial survey of requirements interdependencies in software product release planning. In: RE 2010. IEEE Computer Society (2001)
15. Vathsavayi, S., Sievi-Korte, O., Koskimies, K., Systä, K.: Planning global software development projects using genetic algorithms. In: Ruhe, G., Zhang, Y. (eds.) SSBSE 2013. LNCS, vol. 8084, pp. 269–274. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39742-4_23
16. Gao, C., Zhang, H., Jiang, S.: Constructing hybrid software process simulation models. In: 11th International Conference on Software and Systems Process (ICSSP) (2015)
17. Cooper, R.: Agile–stage-gate hybrids. In: Research-Technology Management (2016)
18. Saliu, M.O., Ruhe, G.: The art and science of software release planning. In: IEEE Software (2005)
19. Ngo-The, A., Ruhe, G.: Optimized resource allocation for software release planning. Trans. Softw. Eng. (2009)
20. Lakhani, K.R., Jeppesen, L.B., Lohse, P.A., Panetta, J.A.: The value of openness in scientific problem solving. HBS Working Paper Number: 07-050, Harvard Business School (2007)
21. Kittur, A., et al.: The future of crowd work. In: CSCW (2013)
22. LaToza, T.D., Ben Towne, W., van der Hoek, A., Herbsleb, J.D.: Crowd development. In: The 6th International Workshop on Cooperative and Human Aspects of Software Engineering (2013)
23. March, J.G., Simon, H.A.: Organizations. Wiley, New York (1958)
24. Ross, J., Irani, L., Silberman, M.S., Zaldivar, A., Tomlinson, B.: Who are the crowdworkers? Shifting demographics in Amazon mechanical turk. ACM (2010)
25. Marcus, A., Wu, E., Karger, D., Madden, S., Miller, R.: Human-powered sorts and joins. VLDB Endow. (2009)

26. Bernstein, M., Brandt, J., Miller, R.C., Karger, D.R.: Crowds in two seconds: enabling realtime crowd-powered interfaces. In: The 24th Annual ACM Symposium on User Interface Software and Technology (2011)
27. Reinertsen, D.G.: The Principle Product Development Flow, Second Generation Lean Product Development. Celeritas Publishing, Redondo Beach (2009)
28. Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., Stoica, I.: Dominant resource fairness: fair allocation of multiple resource types. USENIX Association (2011)
29. IBM. http://www.research.ibm.com/wsla/
30. Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., Stoica, I.: Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. ACM (2010)
31. Kaufmann, N., Schulze, T., Veit, D.: More than fun and money: worker Motivation in crowdsourcing - a study on mechanical turk. In: 17th AMCIS (2011)
32. Krcmar, H., Bretschneider, U., Huber, M., Leimeister, J.M.: Leveraging crowdsourcing: activation-supporting components for IT-based ideas competition. J. Manag. Inf. Syst. **26**, 197–224 (2009)
33. Yang, J., Adamic, L.A., Ackerman, M.S.: Crowdsourcing and knowledge sharing: strategic user behaviour on Taskcn. In: 9th ACM Conference on Electronic Commerce (2008)
34. Archak, N.: Money, glory and cheap talk: analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on Topcoder.com. In: 19th International Conference on World Wideweb (2010)
35. Chen, W.-N., Zhang, J.: Ant colony optimization for software project scheduling and staffing with an event-based scheduler. IEEE Trans. Softw. Eng. **39**(1), 1–17 (2013)
36. Ozdamar, L.: A genetic algorithm approach to a general category project scheduling problem. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **29**(1), 44–59 (1999)
37. Saremi, R., Yang, Y.: Empirical analysis on parallel tasks in crowdsourcing software development. In: 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW) (2015)
38. Saremi, R., Yang, Y., Ruhe, G., Messinger, D.: Leveraging crowdsourcing for team elasticity: an empirical evaluation at Topcoder. In: ICSE-SEIP (2017)
39. Shtub, A., Bard, J.F., Globerson, S.: Project Management: Processes, Methodologies, and Economics, 2nd edn. Prentice Hall, Upper Saddle River (2005)
40. Brucker, P., Drexl, A., Mohring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: notation, classification, models and methods. Eur. J. Oper. Res. **112**, 3–41 (1999)
41. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybern. Part B Cybern. **26**(1), 29–41 (1996)
42. Saremi, R.L., Yang, Y.: Empirical analysis on parallel tasks in crowdsourcing software development. In: 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), pp. 28–34, November 2015