CrowDIY: How to Design and Adapt Collaborative Crowdsourcing Workflows Under Budget Constraints

Rong Chen(✉), Bo Li, Hu Xing, and Yijing Wang

Dalian Maritime University, Dalian 116026, China
rchen@dlmu.edu.cn

Abstract. Workflow quality is a key determinant of crowdsourcing complex work, but finding ways to task design and plan has proved illusive. Instead, we formulate it as an optimization problem with budget constraints and fewer decision variables to set. We propose a two-staged approach CrowDIY that can not only estimate task attributes based on previous tasks but also optimize them with budget constraints in order to publish tasks more wisely in a timely manner. Several experimental studies have been conducted, and the results show compelling evidence that, under different conditions, the proposed approach can effectively reduce the workload of workflow design and plan, while avoiding commonly encountered trial-and-error in crowdsourcing workflows and leading up to successful complex outcomes.

Keywords: Crowdsourcing workflow · Workflow design and plan · Task publishing · Optimization

1 Motivation and Background

The dominant infrastructure in human computation systems today is *workflow*, which typically splits a business process into multiple microtasks and asks distinct workers to carry them out in pre-specified steps on services like Amazon's Mechanical Turk (MTurk), CrowdFlower and CrowdSPRING [4]. There is little doubt that crowdsourcing workflows (CWs) are powerful because they build operational knowledge into software [2], allowing people around the world to work collaboratively and contribute meaningfully.

Though CW techniques pushed the boundary of crowsourcing [8], task requesters still need to program their own workflow or intervene continuously on the execution of their manmade workflow [9]. Task requesters need to make a variety of decisions regarding the task they want to submit [12]. To understand the complexity behind practical usage, we use the example of writing short essay about Dalian – a tourist city in China. Figure 1 shows the screenshot of a crowdsourcing workflow G_1 composed of eleven tasks with indexed numbers inside circles (denoted as T_1, T_2, …, T_{11}). Tasks are of specified types: *question and answer* (QA), *choice*, *merge*, *notification*, AND- and OR-node. To design G_1 with a graphical web UI on CrowDIY (Crowdsourcing - Do It Yourselves), the requester decomposes essay writing into several steps: (1) Puts a

M. Bakaev et al. (Eds.): ICWE 2019, LNCS 11496, pp. 203–210, 2019.
https://doi.org/10.1007/978-3-030-19274-7_15

question to crowd for suggesting aspects to describe Dalian via QA node T_1, (2) Chooses three hot aspects via majority voting (T_2), (3) Asks crowd to write about the selected aspects (later bound to culture (T_3) and architecture (T_4) and transportation (T_5) at runtime), and then asks others to read and give their rates (T_6, T_7, and T_8 respectively), (4) Combine the content via a merge node T_9, (5) Makes his own decision via choice node T_{10}, and (6) T_{11} to notify the completion. Figure 1 also shows the execution status of G_1 that started from T_1, ran through task nodes (denoted in green), steps into T_9 for merging, and will end in T_{11} for notification.

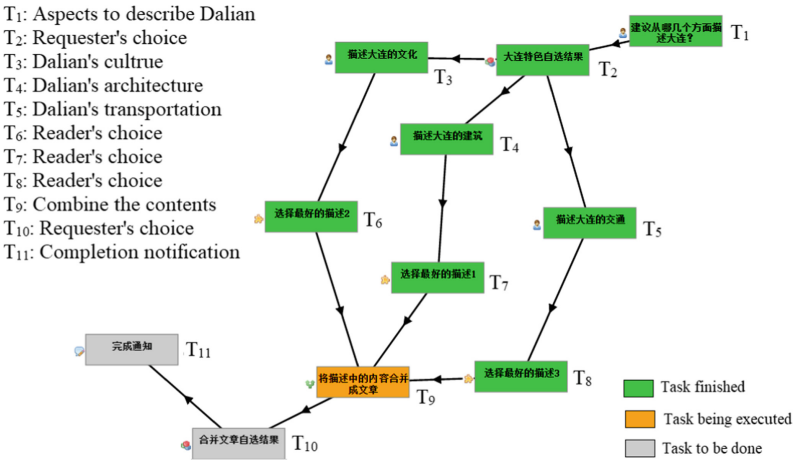

Fig. 1. The motivational example workflow G_1.

Provided with task decompositions (e.g. via a map-reduce paradigm [7], a divide-and-conquer strategy [6], and a customized policy [1]), the design and plan of workflows like G_1 are still not easy because at least information such as the task type and description, time effort, time allotted, the reward for which the worker actually booked the task, and the time it took from publishing to booking should be defined for each task at the design time. We address the problem of crowdsourcing workflow optimization (CWO), and propose a two-staged approach that can not only estimate attributes and parameters but also optimize them with budget constraints, and then publish tasks more wisely in a timely manner.

2 Proposed Approach

2.1 Problem Statement

A *workflow* can be characterized by a directed acyclic graph (DAG) G = (**T**, **E**) where the nodes **T** = {T_1, T_2, …, T_n} correspond to the tasks and the edges **E** indicate the data dependencies between tasks.

Definition 1 (Total Cost). The total cost of a workflow G = (**T**, **E**) is the sum of all task rewards, defined as:

$$Cost(G) = \sum\nolimits_{\forall T_i \in T} r_i \tag{1}$$

Besides rewards, more attributes are associated with a task in a real crowdsourcing workflow; they are *type*, *level of difficulty*, *effort to complete* in terms of the number of time points, *time allotted*, *reward*, *latest booking time* for booking, *earliest publishing time* and *buffer time*. Concisely, each task T_i is characterized by $T_i = (type_i, lod_i, etc_i, ta_i, r_i, lbt_i, ept_i, bt_i)$.

Definition 2 (Sequential and Parallel Execution). A sequential execution of a workflow G = (**T**, **E**) is a sequence of tasks $sp = [T_1, T_2, \ldots, T_n]$, such that T_1 is the initial task, T_n is the final task, and for every task T_i ($1 \leq i \leq n$):

- T_i is a direct successor of one of the tasks in *sp*.
- T_i is not a direct successor of any of the tasks in *sp*.
- There is no state T_j in *sp* such that T_j and T_i belong to two alternative branches of the workflow.
  A parallel execution of a workflow G is a set $pp(G) = \{sp_1, sp_2, \ldots, sp_m\}$ of sequential executions of G such that all the parallel branches of every AND-node in $sp_j = [T_1, T_2, \ldots, T_n]$ ($1 \leq j \leq m$) are executed when that AND-node is entered. Formally,
- If T_i is the initial task of one of the parallel regions of an AND-node, then, for every other parallel region C, one of the initial tasks of C belongs to the set $\{T_1, \ldots, T_{i-1}, T_{i+1}, \ldots, T_n\}$.

The second goal of the CW research is to manage business processes in terms of time, e.g. by controlling the *estimated total execution time*, which means the longest sequential execution path that covers all parallel regions.

Definition 3 (Estimated Total Execution Time) Let $sp = [T_1, T_2, \ldots, T_n] \in pp(G)$ be any sequential execution of a workflow G. The estimated total execution time of G, denoted by *ETime*(G), is the maximum of *ETime*(*sp*):

$$ETime(sp) = lbt_1 + \sum\nolimits_{i=1}^{n} ta_i \tag{2}$$

$$ETime(G) = \max_{\forall sp \in pp(G)} ETime(sp) \tag{3}$$

The present research makes two extensions to the available CW studies: (1) A fewer task attributes (e.g. $type_i$ and lod_i) are mandatory while others are optional. The mandatory part are set manually while the optional part can be defined by functions that take mandatory lod_i and historical task data as arguments. (2) We control the execution time by minimizing the overdue risk while ensuring the deadline and the cost budget. Next we offer an overview of our approach CrowDIY before formalizing them as the CWO problem.

2.2 CWO Formulation

Throughout this paper, time-related parameters and task attributes are supposed to be characterized in terms of time points t_0 (start time), t_1, t_2, …, t_D (deadline time) such that each t_i defines a point in time that i time slices have elapsed.

Definition 4 (Overdue Risk). The overdue risk of any task T_i with respect to start time t and buffer time bt is defined as:

$$f(lod_i, t, bt) = lod_i \cdot [\alpha_2(t+bt)^2 + \alpha_1(t+bt) + \alpha_0] \tag{4}$$

with weights α_0, α_1 and $\alpha_2 \in [0..1]$.

A CWO problem is to find a solution of task attributes with minimized overdue risk while not exceeding the deadline and the cost budget. There exist two solution scenarios: *static assignment*, in which *lbt*s and *ta*s of all tasks are set while aggregating estimated execution time in design phase, and d*ynamic assignment*, in which *ept*s and *bt*s are set for initial tasks to be published while aggregating the estimated execution time of tasks not yet run.

Definition 5 (Static CWO Assignment). Let G = (**T**, **E**) be a workflow under design, R_{max} be the budget in score points, and D_{max} be the deadline in time points. A static CWO assignment is to find: for each task T_i in $sp = [T_1, T_2, \ldots, T_n] \in pp(\mathrm{G})$ ($1 \leq i \leq n$), the lbt_i and the ta_i that, minimize

$$\sum\nolimits_{\forall T_i \in sp} f(lod_i, lbt_i, ta_i)$$

subject to

$$ta_i < lbt_i - lbt_{i-1} (i \geq 2) \tag{5}$$

$$Cost(\mathrm{G}) \leq R_{max} \tag{6}$$

$$ETime(\mathrm{G}) \leq t_{D_{max}} \tag{7}$$

Note that the real execution time of tasks may be different from what was estimated. Let $\mathbf{T}_C \subseteq \mathbf{T}$ be tasks that have already completed so far, and $\mathbf{E}_C = \{<T_1, T_2> \mid \forall\, T_1, T_2 \in \mathbf{T}_C, <T_1, T_2> \in \mathbf{E}\}$ be edges that have already been covered. We separate G into two subgraphs: the completed part $\mathrm{G}_C = (\mathbf{T}_C, \mathbf{E}_C)$, and the part not completed $\overline{\mathrm{G}}_C = (\mathbf{T} - \mathbf{T}_C, \mathbf{E} - \mathbf{E}_C)$.

Definition 6 (Dynamic CWO Assignment). Let $\mathbf{T}_C = \{T_1, T_2, \ldots, T_C\}$ be tasks G = (**T**, **E**) of that have already completed at time point t_C, and $\mathrm{G} = \mathrm{G}_C \cup \overline{\mathrm{G}}_C$, and $In(\overline{\mathrm{G}}_C) = \{T_i \mid T_i$ is the initial task of any sequential execution $sp \in pp(\overline{\mathrm{G}}_C)\}$. Let R_{max} be the budget in terms of score points, and D_{max} be the deadline in terms of the number of

time points. A dynamic CWO problem is to find: for each task $T_s \in In(\overline{G}_C)$, and for each task $T_i \in sp \in pp(\overline{G}_C)$ $(i \neq s)$, the ept_s, the bt_s, the lbt_i, and the ta_i that, minimize

$$\sum\nolimits_{\forall T_i \in sp} f(lod_i, lbt_i, ta_i) + \sum\nolimits_{\forall T_s} f(lod_s, ept_s, bt_s)$$

subject to

$$t_C \leq ept_s < lbt_s (\forall T_s \in In(\overline{G}_C)) \tag{8}$$

$$ta_s \leq bt_s (\forall T_s \in In(\overline{G}_C)) \tag{9}$$

$$ta_i < lbt_i - lbt_{i-1} (i \geq 2) \tag{10}$$

$$Cost(G) + Cost(\overline{G}_C) \leq R_{max} \tag{11}$$

$$ETime(\overline{G}_C) \leq t_{D_{max}} \tag{12}$$

2.3 Solution Algorithms

Algorithm 1 depicts the overall procedure of CrowDIY, which starts from *Task*, max reward R_{max}, max deadline D_{max} to perform workflow design and revise (Step 1), planning (Steps 2–4) and publishing (Steps 7–10) remained tasks to crowd workers until all tasks are finished or the Dynamic CWO has no solution.

Algorithm 1. Crowdiy(*Task*, R_{max}, D_{max})

1: G←Design(*Task*, R_{max}, D_{max});
2: Set up G with Estimate(G);
3: CWO←Transform(t_0, G, R_{max}, D_{max});
4: < because it require, R_{min}, D_{min}>←Solve(CWO);
4: <*Task*, R_{min}, D_{min}>←Solve(CWO);
5: **if** $R_{max}<R_{min}$ or $D_{max}<D_{min}$ **then**
6: **if** not terminate **then** goto Step 1;
7: $t_C \leftarrow t_1$;
8: **do** increase t_C;
9: **if** a task is finished **then** Publish(G, $\mathbf{T}_C$, R_{max}, D_{max});
10: **until** $In(\overline{G}_C)$ empty or no solution;

Algorithm 2. Publish(G, $\mathbf{T}_C$, R_{max}, D_{max})

Input: Workflow G, tasks completed so far $\mathbf{T}_C$, max reward R_{max}, max deadline D_{max}
Output: <*Task*, R_{min}, D_{min}>

1: Let $G = G_C \cup \overline{G}_C$;
2: Let $In(\overline{G}_C)=\{T_i \mid T_i$ is the initial task of any sequential execution $sp \in pp(\overline{G}_C)\}$;
3: CWO←Transform(t_C, $\overline{G}_C$, R_{max}, D_{max});
5: <*Task*, R_{min}, D_{min}>←Solve(CWO);
6: **if** non-empty *Task* **then** publish task $T_i \in In(\overline{G}_C)$;
7: return <*Task*, R_{min}, D_{min}>;

Design($Task$, R_{max}, D_{max}) means that the requester can design a CW via the Web UI in several steps: decompose complex tasks into small ones by calling divide(Task), place a choice node for selecting answers, manage task dependencies and structure (AND-node or OR-node), later combine the results into a coherent solution via merge node, and finalize with a notification node. Design can be extended recursively or revised repeatedly by Algorithm 1 (from step 1 to 6). As described by Algorithm 2, Transform(t_C, $\overline{G}_C$, R_{max}, D_{max}) instantiates constraints Eqs. (8)–(12) and the overdue risk function Eq. (4) for $\overline{G}_C$ at current time t_C.

3 Evaluation and Results

We implemented the solution method in a crowdsourcing workflow system CrowDIY in Python [5], running on the Django Web Framework with SQlite and other tools for solving the CWO problem and generating workflow. To solve the CWO problem, CrowDIY integrates Gurobi, Cplex and Choco through constraint programing in Java in order to find static and dynamic CWO assignments. We set weights of Eq. (4) with $\alpha_0 = 0.25$, $\alpha_1 = 0.4$, and $\alpha_2 = 0.5$.

Workflows were generated with JGraphT–a Java library of graph theory data structures and algorithms [10], and mandatory attributes such as node type and level of difficulty are generated uniformly in random. We vary the number of workflows from 1 to 500 while the number of tasks in every workflow is in [6..20]. We assumed that there were 3000 workers and task attributes were generated. For every task type, we generated other task attributes that are linearly dependent on task difficulty as we did in case studies. Also 300 workers were assigned the least time allotted to finish a task and the minimum acceptable reward, which were generated using the normal distribution based on the average reward, average allotted time and their allowable deviation parameters. So we prepared a large number of different workflows with randomizing workflow structures and diverse deadlines, and tasks in them have various allotted times and booking times.

The number (#W) of workflows ranges from 1 to 500, and each is compared with the reference case #W = 1. First, we guess the max deadline D_{max} for every workflow in every case. If the manmade D_{max} does not make sense, there is no solution to the CWO formulation of the workflow under consideration. So we can count the number of trial-and-error (#E) of CWO solving. If D_{max} makes sense, then we guess the max reward R_{max}. If the manmade R_{max} works, constraint solvers return the overdue risk and their execution time (#T) in seconds. In particular, #OR indicates the multiple of 329.7 or 722.1, namely the overdue risk of the reference case #W = 1. If R_{max} is implausible, "no solution" means that, at design time the workflow is found more likely to "fail" because it requires the deadline extension. So we compare the time extension (#X) in time points raised by failed workflows. The more time extension failed workflows require, the better solution the constraint solver can ensure. All metrics we used are reported on average for all the workflows we prepared.

The first experiment is to find the best solver for workflow plan (i.e. static CWO assignment). The comparison results were summarized in Table 1. It can be seen that as #W grows, their performance present the trend of linear growth under four metrics. Also we can see that the performance of Gurobi and Cplex are similar in #E, #OR, and #X. But Gurobi is much better than Cplex in terms of #T. So we choose Gurobi to conduct the rest experiments.

Table 1. Results from comparative constraint solvers.

	Gurobi				Cplex				Choco			
#W	#E	#OR **329.7**	#T	#X	#E	#OR **329.7**	#T	#X	#E	#OR **722.1**	#T	#X
1	0	1	2.3	0	0	1	2.9	0	0	1	4.1	0
10	0	5.5	2.9	0	0	5.5	4.8	0	0	5.5	6.0	0
50	0	13.6	3.5	0	0	13.6	18.1	0	0	13.7	26.4	0
100	1	23.7	7.5	1	1	23.7	23.5	1	1	23.4	65.6	1
200	3	27.8	11.0	7	4	27.9	59.0	7	1	28.0	130.0	3
300	5	31.0	10.2	10	5	31.1	91.5	11	1	31.3	146.6	3
400	6	34.1	15.0	12	7	34.1	116.3	13	1	34.2	272.8	4
500	9	38.0	17.1	17	10	38.1	165.0	18	2	38.1	299.8	4

The second experiment is to verify whether buffer time influences the final outcome of all workflows in the task publishing algorithm with a linear dependence $bt_s = x \cdot ta_s$ ($x \in \{0.2, 0.5, 1, 2, 3, 4, 5, 6\}$). It can be seen from Fig. 4 that the optimal results have achieved the minimum value when buffer time is almost equal to its allotted time. In case of smaller buffer time, for example $x = 0.2$ and $x = 0.5$ (x-axis), more tasks were not booked on time, so the reward to workers should be raised. At the same time, lack of time also increase the possibility of missing deadlines. That is why three metrics (#OR, #X and #E) have higher values. If the buffer time is larger, for instance coefficient $x \in [2..6]$, the values of three metrics are higher than the optimal results, but still much lower than the buffer time. This is because tasks can be booked earlier by workers.

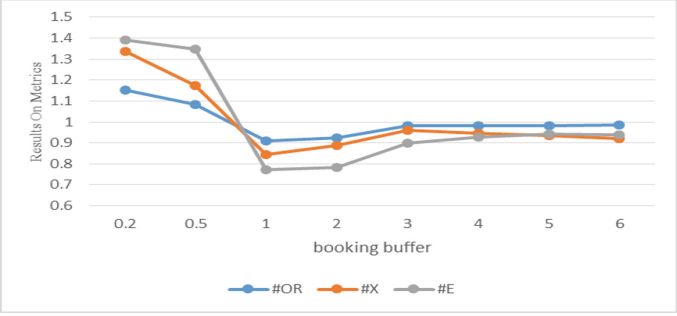

Fig. 4. Finding the most appropriate buffer time.

4 Conclusion and Future Work

The present approach eases the complexity behind collaborative crowdsourcing process, but dynamic approach to publishing cannot guarantee the time constraints because there is a lot of uncertainty in crowdsourcing, especially the anonymous people with uncertain skills and commitments. What merits future investigation includes advancing the training of Estimator and the control of workflow, and exploiting statistical sampling of people from the crowd after they contributed meaningfully in previous tasks [3, 11].

Acknowledgments. This work is supported by the National Natural Science Foundation of China (No. 61672122, No. 61602077), the Natural Science Foundation of Liaoning Province of China (No. 2015020023), the Educational Commission of Liaoning Province of China (No. L2015060) and the Fundamental Research Funds for the Central Universities (NO. 3132016348).

References

1. Bernaschina, C., Catallo I., Fraternali P., Martinenghi, D., Tagliasacchi, M.: Champagne: a web tool for the execution of crowdsourcing campaigns. In: International Conference on World Wide Web (Companion), pp. 171–174. ACM, New York (2015)
2. Bigham, J.P., Bernstein, M.S., Adar, E.: Human-computer interaction and collective intelligence. In: Handbook of Collective Intelligence, pp. 57–84. MIT Press (2015)
3. Chen, R., Chen, S.-F., Zhang, X.-Y.: A two-staged task assignment algorithm for worker recommendation in a crowdsourcing environment. In: International Conference on Industrial Engineering and Engineering Management, Singapore, pp. 2034–2038 (2017)
4. Doan, A., Ramakrishnan, R., Halevy, A.Y.: Crowdsourcing systems on the world-wide web. Commun. ACM **54**, 86–96 (2011)
5. Huang, Y.-T.: Design and implementation of a workflow system for crowdsourcing. Master thesis, Dalian Maritime University (2017). (in Chinese)
6. Kittur, A., Smus, B., Khamkar, S., Kraut, R.E.: CrowdForge: crowdsourcing complex work. In: Annual ACM Symposium on User Interface Software and Technology, pp. 43–52. ACM, New York (2011)
7. Kulkarni, A., Can, M., Hartmann, B.: Collaboratively crowdsourcing workflows with turkomatic. In: ACM Conference on Computer Supported Cooperative Work, pp. 1003–1012. ACM, New York (2012)
8. Little, G., Chilton, L.B., Goldman, M., Miller, R.C.: TurKit: human computation algorithms on mechanical turk. In: Annual ACM Symposium on User Interface Software and Technology, pp. 57–66. ACM, New York (2010)
9. Retelny, D., Bernstein, M.S., Valentine, M.A.: No workflow can ever be enough: how crowdsourcing workflows constrain complex work. In: ACM Human-Computer Interaction, CSCW, vol. 1, Article 89, 23 p. ACM (2017)
10. JGraphT. https://jgrapht.org. Accessed 10 Jan 2019
11. Gadiraju, U., Kawase, R.: Improving reliability of crowdsourced results by detecting crowd workers with multiple identities. In: Cabot, J., De Virgilio, R., Torlone, R. (eds.) ICWE 2017. LNCS, vol. 10360, pp. 190–205. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60131-1_11
12. Catallo, I., Martinenghi, D.: The dimensions of crowdsourcing task design. In: Cabot, J., De Virgilio, R., Torlone, R. (eds.) ICWE 2017. LNCS, vol. 10360, pp. 394–402. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60131-1_25