SLADE: A Smart Large-Scale Task Decomposer in Crowdsourcing

Yongxin Tong, *Member, IEEE*, Lei Chen, *Member, IEEE*, Zimu Zhou, *Student Member, IEEE*, H. V. Jagadish, *Member, IEEE*, Lidan Shou, and Weifeng Lv

Abstract—Crowdsourcing has been shown to be effective in a wide range of applications, and is seeing increasing use. A large-scale crowdsourcing task often consists of thousands or millions of atomic tasks, each of which is usually a simple task such as binary choice or simple voting. To distribute a large-scale crowdsourcing task to limited crowd workers, a common practice is to pack a set of atomic tasks into a task bin and send to a crowd worker in a batch. It is challenging to decompose a large-scale crowdsourcing task and execute batches of atomic tasks, which ensures reliable answers at a minimal total cost. Large batches lead to unreliable answers of atomic tasks, while small batches incur unnecessary cost. In this paper, we investigate a general crowdsourcing task decomposition problem, called the Smart Large-scAle task DEcomposer (SLADE) problem, which aims to decompose a large-scale crowdsourcing task to achieve the desired reliability at a minimal cost. We prove the NP-hardness of the SLADE problem and propose solutions in both *homogeneous* and *heterogeneous* scenarios. For the *homogeneous* SLADE problem, where all the atomic tasks share the same reliability requirement, we propose a greedy heuristic algorithm and an efficient and effective approximation framework using an optimal priority queue (OPQ) structure with provable approximation ratio. For the *heterogeneous* SLADE problem, where the atomic tasks can have different reliability requirements, we extend the OPQ-based framework leveraging a partition strategy, and also prove its approximation guarantee. Finally, we verify the effectiveness and efficiency of the proposed solutions through extensive experiments on representative crowdsourcing platforms.

Index Terms—Crowdsourcing, task decomposition

1 Introduction

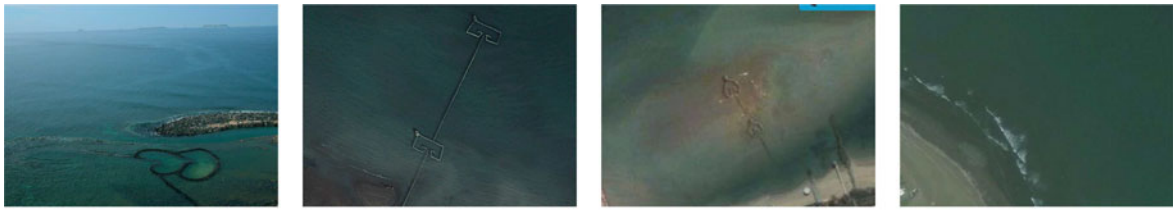
Crowdsourcing refers to the outsourcing of tasks traditionally performed by an employee to an "undefined, generally large group of people in the form of an open call [1]". Early success stories include Wikipedia, Yelp and Yahoo! Answers. In recent years, several general-purpose platforms, such as Amazon Mechanical Turks (AMT)[1] and oDesk,[2] have made crowdsourcing more powerful and manageable. Crowdsourcing has attracted extensive research attention due to its success in human intrinsic applications. Particularly, a wide spectrum of fundamental data-driven operations have been studied, such as max [2], [3], filtering [4], inference [5], [6] and so on. In addition, researchers and practitioners also pave the way for building crowd-powered databases and data mining systems, and a couple of prototypes have been successfully developed, such as CrowdDB [7], Deco [8], Qurk [9], DOCS [10] and CDB [11]. We refer readers to tutorials and surveys [12], [13], [14], [15], [16] for a full picture on crowdsourcing.

The rapid development of crowdsourcing platforms contributes to the ever-increasing volume and variety of crowdsourcing tasks. A real-world crowdsourcing task can contain thousands or millions of *atomic tasks*, where an atomic task can be considered as a unit task that requires trivial cognitive load. Despite the complexity and the variety of the crowdsourcing task goals, most atomic tasks are in the form of binary choices. According to a recent study on 27 million tasks performed by over 70,000 workers [17], boolean questions dominate the types of task operations and are widely applied in basic data-driven operations such as filtering. These large-scale crowdsourcing tasks are usually distributed to a wide range of crowd workers and are often sensitive to false negatives. To distribute a large-scale task to limited crowd workers, a common practice is to pack a set of atomic tasks into a *task bin* and send to a crowd worker in a batch [7], [18]. Furthermore, using a task bin to batch atomic tasks is also an effective way to reduce the average cost per atomic task [19]. In the following, we illustrate the adoption of task bins in large-scale crowdsourcing tasks via a real-world example in crowdsourced environment monitoring.

1. https://www.mturk.com/mturk/
2. http://www.odesk.com/

- *Y. Tong and W. Lv are with the State Key Laboratory of Software Development Environment, School of Computer Science and Engineering and International Research Institute for Multidisciplinary Science, Beihang University, Beijing 100083, China. E-mail: {yxtong, lwf}@buaa.edu.cn.*
- *L. Chen is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China. E-mail: leichen@cse.ust.hk.*
- *Z. Zhou is with the Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich 8092, Switzerland. E-mail: zimu.zhou@tik.ee.ethz.ch.*
- *H. V. Jagadish is with the Department of Electrical Engineering and Computer Science, University of Michigan, 2260 Hayward Ave, Ann Arbor, MI 48109. E-mail: jag@umich.edu.*
- *L. Shou is with the Key Laboratory of CAD & CG, Zhejiang University, Hangzhou 310027, China. E-mail: should@zju.edu.cn.*

Manuscript received 19 Apr. 2017; revised 2 Nov. 2017; accepted 3 Dec. 2017. Date of publication 24 Jan. 2018; date of current version 5 July 2018. (Corresponding authors: Lei Chen and Weifeng Lv.) Recommended for acceptance by G. Li. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TKDE.2018.2797962



(a) a_1 (b) a_2 (c) a_3 (d) a_4

Fig. 1. Fishing-line discovery.

Example 1 (Fishing-Line Discovery). The over-use and out-of-report of large fishing-lines violate the international fishing treaties, but are difficult to monitor by only a small group of people. To fight against such illegal usages of large fishing-lines, a project has been published on the Tomnod website,[3] where a satellite image covering more than two million km^2 has been transformed into a large trunk of small pieces of images. The participants are asked to decide "whether there is a 'fishing-line' shape in the given piece of image", which is considered as an "atomic task". Fig. 1 shows four example images in four atomic tasks a_1, a_2, a_3, a_4. Since the project manager cannot afford to miss any dubious image, they ask multiple participants to review the same image and any image with at least one "yes" will be further scrutinised. The project manager needs to decide plans to distribute these images. One way is to process a_1 to a_4 only once but individually (10 cents each and 40 cents in total). Another way is to group a_1 and a_2 in one task bin and a_3 and a_4 in another task bin and then ask two workers to process each task bin twice (12 cents for each task bin, 12*2*2 = 48 cents in total). Which plan is better? Is there an even better choice?

We argue that the size of the task bins (or *cardinality*) plays a crucial role in the execution plan of a large-scale crowdsourcing task in terms of *cost* and *reliability*. Decomposing a large-scale crowdsourcing task into task bins of a larger size results in a lower average cost of each atomic task in the task bins. However, it is observed that the overall reliability of a large batch of atomic tasks tends to decrease due to the increase of cognitive load [19]. Consequently, these atomic tasks have to be executed more times or dispatched to more workers to meet the reliability requirement of the large-scale crowdsourcing task, which leads to an increase in the total cost. Previous works either set the fixed cardinality of a task bin [7], [8] or adopt simple heuristics to determine a single cardinality for the entire large-scale crowdsourcing task.

To further reduce the total cost in executing a large-scale crowdsourcing task while retaining the desired reliability, we propose to harness *a set of task bin cardinalities* rather than a single one. The key insight is that with the increase of the cardinality of task bins, there is a mismatch in the drop of per atomic task reliability and the drop of per atomic task cost. For instance, it may cost 10 cents to process a_1 individually with a reliability of 0.9, while it only costs 6 cents to process a_1 in a task bin of size 2 (i.e., the cost of the task bin is 12 cents), yet with a reliability of 0.8. There is a 40 percent in per atomic task cost while only a 11 percent drop in reliability, or equivalently, approximately 1.43 task bins are needed to achieve a reliability (formally defined in Section 3.1) of 0.9 at the cost of $0.6 \times 1.43 = 0.86$ cents. With task bins of different cardinalities (and of different reliability), we then have the flexibility to optimize the total cost to satisfy a certain reliability requirement. In the above example, to fulfill a reliability requirement of 0.9 on a_1, the optimal plan is to execute a_1 individually (i.e., in a task bin of size 1) once, while for a reliability requirement of 0.95, the optimal plan is to execute a_1 in a task bin of size 2 twice.

In this paper, we propose the *Smart Large-scAle task DEcomposer (SLADE)* problem to investigate the optimal plan to decompose a large-scale crowdsourcing task into batches of task bins of varied sizes, which satisfies the reliability requirements of each atomic task at a minimal total cost. In effect, the SLADE problem is similar to the role of the query optimizer of a database that tries to find an efficient execution plan given a logical expression to be evaluated. As far as we know, this is the first work to tackle the large-scale crowdsourcing task decomposition problem.

To sum up, we make the following contributions:

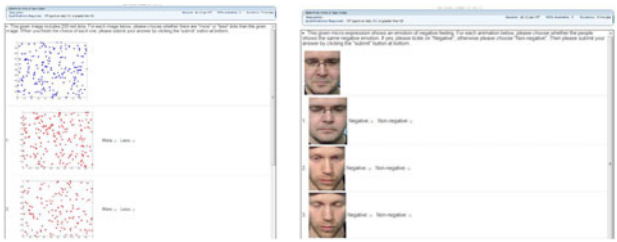
- We identify a new crowdsourcing task decomposition problem, called the *Smart Large-scAle task DEcomposer* problem, and prove its NP-hardness.
- We study two variants of the SLADE problem. The first is the *homogeneous SLADE* problem, where all atomic tasks have the same reliability requirement. We propose a greedy heuristic and an optimal priority queue-based approximation algorithm with $\log n$-approximation ratio, where n is the number of all atomic tasks. The second is the *heterogeneous SLADE* problem, where different atomic tasks may have different reliability requirements. We extend the above approximation framework to heterogeneous SLADE problem, which guarantees a slightly lower approximation ratio.
- We extensively evaluate the effectiveness and efficiency of the proposed algorithms on real datasets.

The rest of the paper is organized as follows. We present a motivation experiment in Section 2, and formally formulate the SLADE problem in Section 3. We analyze the complexity of the SLADE problem in Section 4 and propose approximation algorithms for the homogeneous SLADE problem in Section 5 and for the heterogeneous SLADE problem in Section 6, respectively. We evaluate the proposed algorithms in Section 7 and review related work in Section 8. Section 9 concludes this work.

2 Motivation Experiments

In this section, we study the tradeoff between the per atomic task reliability and the per atomic task cost as a function of the task bin size (cardinality), which motivates our SLADE problem. We conduct the motivation experiments on Amazon Mechanical Turk using the following two crowdsourcing tasks.

3. http://www.tomnod.com/



(a) (b)

Fig. 2. Screen-shots of (a) Jelly-Beans-in-a-Jar and (b) Micro-expressions identification.

Example 2 (Jelly-Beans-in-a-Jar). Given a sample image containing 200 dots, a crowd worker is asked to determine whether another image contains more dots or not. Each image is an atomic task of binary choice, whose answer is independent of each other (Fig. 2a). We then specify the *cardinality* of a task bin ranging from 2 to 30 by aligning the target images along the question webpage. For each task bin, 10 assignments are issued to smooth the randomness of workers, and three different incentive costs for one task bin are tested ($0.05, $0.08 and $0.1). As is typical in such scenarios, we set a response time threshold, after which the batch of atomic tasks is considered too slow for practical use. We used 40 minutes as the threshold.

Example 3 (Micro-Expressions Identification). Some campaign activities record photos or videos and ask the crowd to find the participants with certain expressions. The crowd may receive basic training on the targeted micro-expression and then photos or videos are distributed to be screened. As shown in Fig. 2b, a crowd worker is expected to label the emotion of another target portrait as positive or negative given a sample portrait. The images are from the Spontaneous Micro-expression Database (SMIC) [20]. We also vary the cardinality from 2 to 30 with the incentive cost per task bin as $0.05, $0.1 and $0.2, respectively. Similarly, we set a time threshold of 30 minutes.

Fig. 3 characterizes the relationships among the cardinality, confidence and cost of a task bin on both the Jelly-Beans-in-a-Jar (*Jelly*, Fig. 3a) and the Micro-Expressions Identification (*SMIC*, Fig. 3b) tasks. Here confidence refers to the average probability that the crowds can correctly complete each atomic task in this task bin. We also conduct experiments on the *Jelly* dataset with different difficulty (Fig. 3c). The difficulty of a *Jelly* task is indicated by the number of dots in the given sample image. We specify the difficulty level as 1 for 50 dots, level 2 for 200 dots, and level 3 for 400 dots (labeled as Diff. 1/2/3).

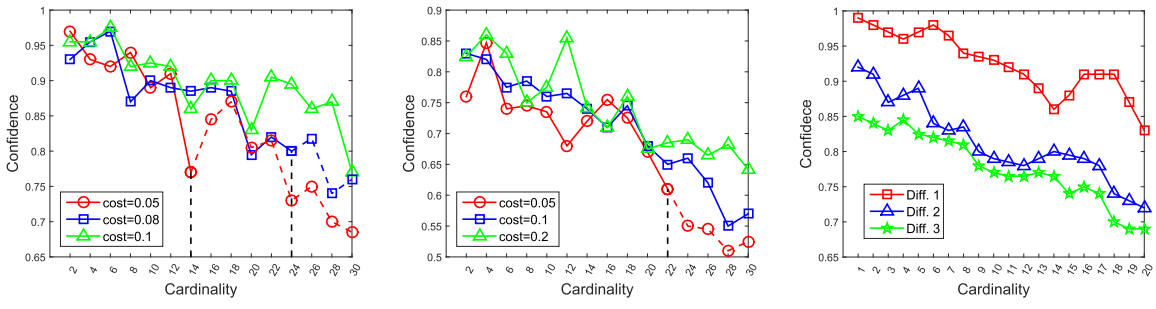
Take Fig. 3a as an illustration. Overtime task bins (not finished within 40 minutes) are shown in dotted lines, while the rest are in sold lines. We see that the confidence declines with the increase of cardinality. After cardinality of 14 (resp. 24), the task bins with cost $0.05 ($0.08) are disqualified since no enough answers are obtained within 40 minutes. As the cardinality goes from 2 to 30, the confidence decreases from 0.981 to 0.783, and the average cost per atomic task decreases from $0.025 ($= 0.05/2$) to $0.003 ($= 0.1/30$).

We make the following observations: (1) There is a mismatch in the drop of confidence and the drop in cost. Specifically, the confidence only decreases from 0.981 to 0.783 while the average cost per atomic task decreases from 0.025 to 0.003. The moderate drop in confidence may be explained by the preference of performing a sequence of similar atomic tasks, which reduces cognitive load of task-switching [21]. It indicates the potential of total cost saving to apply task bins rather than dispatch each atomic task individually to each crowd worker. (2) The decreasing trends of confidence vary for different costs (see the curves for the cost of 0.05, 0.08, and 0.1). Thus it is more flexible to achieve certain accuracy requirement by using a combination of task bins of different sizes and confidence. (3) While the confidence of crowd workers tend to be less sensitive to the drop in cost (i.e., reward to workers), the quantity of crowd workers is notably sensitive to the drop in cost (e.g., the maximal size for in-time responses at a cost of 0.05 is only half of that at a cost of 0.1 (14 versus 30).

The above observations hold for different types of tasks (Fig. 3b) and for the same tasks of different difficulty levels (Fig. 3c). The difference lies in the absolute values, e.g., the general confidence is only 0.7 for the *SMIC* tasks. It is essential to adopt a set of task bins as probes to evaluate the difficulty of different types of atomic tasks so as to select a proper set of task bins. We refer readers to [19] for further discussions on the difficulties and task designs of atomic tasks. In this paper, we focus on how to batch atomic tasks that are homoplasmic and thus with the same difficulty. Packing tasks that vary significantly in difficulty is out of the scope of this work.

3 Problem Statement

In this section, we first introduce several important concepts of large-scale crowdsourcing tasks and then formally define the *SLADE Problem* and discuss its complexity.



(a) (b) (c)

Fig. 3. Relationships among the cardinality, confidence, and cost of a task bin tested on (a) Jelly-Beans-in-a-Jar, (b) Micro-Expressions Identification tasks, and on (c) Jelly-Beans-in-a-Jar tasks of different difficulties.

TABLE 1
A Set including Three Task Bins

Task Bins	b_1	b_2	b_3
Cardinality l	1	2	3
Confidence r_l	0.9	0.85	0.8
Incentive Cost (USD) c_l	0.1	0.18	0.24

3.1 Preliminaries

We focus on large-scale crowdsourcing tasks consisting of *atomic tasks*. An atomic task, denoted by a_i, is defined as a binary choice problem. Due to their trivial cognitive load and simple structure, atomic tasks of boolean questions dominate the types of task operations adopted in the marketplace [17]. We further define a *large-scale crowdsourcing task* T as a set of n independent atomic tasks, i.e., $T = \{a_1, a_2, \ldots, a_n\}$ $(n = |T|)$. Large-scale crowdsourcing tasks are common in real-world crowdsourcing. For example, in the fishing-line discovery application, an atomic task is to decide whether there is a fishing-line shape in a given image, while the whole task consists of over 100,000 satellite images to be checked. Note that typical tasks posted on popular crowdsourcing platforms such as AMT and oDesk are large-scale tasks consisting of simple atomic tasks that can be handled independently by each crowd worker, e.g., the decision of one image will not affect that of another image. We therefore omit the task coordination among co-workers and refer readers to [22] for the high-level discussions on complex task decomposition.

As discussed in Section 2, *batched* atomic tasks hold promise to reduce the total cost of a large-scale crowdsourcing task with achieving the same accuracy requirement. The aim of this study is to explore the design space of cost-effective batched atomic task decomposition plans for a large-scale crowdsourcing task. Formally, we define a batch of atomic tasks as *l-cardinality task bins* as follows.

Definition 1 (l-Cardinality Task Bin). *An l-cardinality task bin is a triple, denoted as $b_l = \langle l, r_l, c_l \rangle$, where (1) the cardinality l is the maximum number of different atomic tasks that can be included in the task bin; (2) r_l is the confidence, which indicates the average probability that the crowds can correctly complete each atomic task in this task bin; (3) c_l is the incentive cost given to the crowds who complete all the atomic tasks in this task bin.*

An l-cardinality task bin is similar to a container, which can contain at most l atomic tasks. Different combinations of atomic tasks can be contained in a task bin, and the atomic tasks contained in an l-cardinality task bin are given to one crowd worker in a bundle. Table 1 shows an example of task bins, $\{b_1, b_2, b_3\}$, where the ith column corresponds to the i-cardinality task bin. For example, the second column represents the 2-cardinality task bin b_2, with the confidence $r_2 = 0.85$ and the cost $c_2 = 0.18$. Based on the observations in Section 2, Table 1 assumes the average cost and the confidence of an atomic task drop with the increase of the task bin cardinality. For example, the average costs of the three task bins are 0.1, 0.09 and 0.08, respectively, while their confidences are 0.9, 0.85, and 0.8, respectively.

In practice, the choices of task bin cardinalities and the corresponding confidences and costs can be learned from historical records. In fact, popular marketplaces such as like AMT and oDesk use a set of different task bins as real-time probes to monitor the quality of the current work flow [23]. To obtain the parameters of the set of tasks bins, when a batch of atomic tasks arrives, one can regularly issue testing task bins with different cardinalities. The atomic tasks in testing task bins are the same as the real tasks, yet the ground truth is known to calculate the confidence. A database system, although crowd-powered, always has a response time requirement, which is inversely proportional to the incentive cost of each task bin. Thus the cost for each cardinality is calculated as the minimum cost that meets the response time requirement. After obtaining the answers from the testing task bins, the confidence can be obtained by regression or counting methods.

Each atomic task is usually performed by multiple crowd workers to guarantee the quality of the task [19]. For batched atomic tasks, each atomic task is assigned to multiple task bins for the same purpose.

Since many real-world crowdsourcing applications require low false negative ratios, e.g., discovering fishing-lines from satellite images, we define the *reliability of an atomic task* as the probability of no false negatives. We can link the reliability of an atomic task to the confidences of the task bins where the atomic task is assigned.

Definition 2 (Reliability). *Given an atomic task a_i and the set of assigned task bins $\mathfrak{B}(a_i)$, the reliability, denoted by $Rel(a_i, \mathfrak{B}(a_i))$, of a_i in $\mathfrak{B}(a_i)$ is as follows:*

$$Rel(a_i, \mathfrak{B}(a_i)) = 1 - \prod_{\beta \in \mathfrak{B}(a_i)} (1 - r_{|\beta|}), \tag{1}$$

where $|\beta|$ is the cardinality of the task bin β, and $r_{|\beta|}$ is the confidence of the task bin β.

Equation (1) represents the estimated possibility that a_i can be correctly completed by at least one assigned task bin.

Table 2 summarizes the notations used in this paper.

3.2 SLADE Problem

According to the definitions of task bins and the reliability of each atomic task, we define the *SLADE Problem* as follows.

Definition 3 (SLADE Problem). *Given a large-scale crowdsourcing task T consisting of n atomic tasks $\{a_1, \ldots, a_n\}$, the corresponding reliability thresholds $\{t_1, \ldots, t_n\}$ for each atomic task, and a set of task bins $B = \{b_1, \ldots, b_m\}$, the SLADE Problem is to find a planning $DP_T = \{\tau_i, b_i\}_{i=1}^{m}$, which means task bin b_i is used for τ_i times, to minimize the total cost, $\sum_{i=1}^{m} \tau_i c_i$, such that $Rel(a_i, \mathfrak{B}(a_i)) \geq t_i, \forall a_i \in T$.*

In particular, if the reliability threshold t_i of each atomic task is the same, the variant of the SLADE problem is called the *homogeneous SLADE problem*, which is studied in Section 5. When the reliability thresholds of the atomic tasks are different, the variant is called the *heterogeneous SLADE problem*, which is discussed in Section 6. Moreover, each atomic task can be assigned to multiple l-cardinality task bins in a decomposition plan, i.e., each atomic task can be dispatched to and processed by multiple crowd workers to improve the reliability (will be defined shortly) of each atomic task. We illustrate the (homogeneous) SLADE problem via the following example.

TABLE 2
Summary of Symbol Notations

Notation	Description
a_i	an atomic task
$T = \{a_1, \ldots, a_n\}$	a large-scale crowdsourcing task
b_l	an l-cardinality task bin
$B = \{b_1, \ldots, b_m\}$	the set of task bins
$n = \|T\|$	the number of atomic tasks in T
$m = \|B\|$	the number of task bins in B
r_l	the confidence for each atomic task in a b_l
c_l	the incentive cost of a b_l
β	an arbitrary task bin
$\mathfrak{B}(a_i)$	the set of assigned task bins for a_i
$Rel(a_i, \mathfrak{B}(a_i))$	the reliability of a_i in the set $\mathfrak{B}(a_i)$
$R(a_i, \mathfrak{B}(a_i))$	the equivalent reduction of $Rel(a_i, \mathfrak{B}(a_i))$
t_i	the reliability threshold of a_i
DP_T	optimal decomposition plan of T

Example 4 (Homogeneous SLADE Problem). Given a crowdsourcing task $T = \{a_1, a_2, a_3, a_4\}$, where each atomic task a_i is the same as in Fig. 1, a set of task bins $B = \{b_1, b_2, b_3\}$ in Table 1, and the reliability thresholds of each atomic task $t_i = 0.95, 1 \leq i \leq 4$, the homogeneous SLADE problem can be illustrated in Fig 4. A feasible decomposition plan, P_1, is to adopt four 2-cardinality task bins, i.e., $\{a_1, a_2\}$, $\{a_1, a_2\}$, $\{a_3, a_4\}$ and $\{a_3, a_4\}$, respectively. In P_1, the reliability of a_i, $(1 \leq i \leq 4$ is $1 - (1 - 0.85) \times (1 - 0.85) = 0.98 > 0.95$, with a total cost of $0.18 \times 4 = 0.72$. Another feasible decomposition plan, P_2, is to use two 3-cardinality task bins and one 2-cardinality task bin, i.e., $\{a_1, a_2, a_3\}$, $\{a_1, a_2, a_4\}$ and $\{a_3, a_4\}$, respectively. The reliability of all the atomic tasks in P_2 also exceeds 0.95, while P_2 costs only $0.24 \times 2 + 0.18 = 0.66$. Fig. 4 illustrates the two decomposition plans. In fact, P_2 is the optimal decomposition plan since it has the lowest total cost among all the feasible decomposition plans. Note that the cost saving scales up with the amount of the atomic tasks in T. For a large-scale crowdsourcing task T with thousands or millions of atomic tasks, the optimal decomposition plan can reduce substantial incentive costs while retaining the desired reliability for each atomic task.

4 Problem Reduction

In this section, we first reduce the reliability constraint of the SLADE problem to an equivalent simple form. Then, we prove the NP-hardness of the SLADE problem. We also show that there are polynomial-time solutions to a relaxed variant of the SLADE problem. Finally, we reduce the SLADE problem to the *covering integer programming* (CIP) problem [24], and apply the existing solution of the CIP problem as the baseline algorithm for our SLADE problem.

4.1 Reduction of Reliability

We equivalently rewrite Equation (1) in Definition 2 as:

$$R(a_i, \mathfrak{B}(a_i)) = -ln(1 - Rel(a_i, \mathfrak{B}(a_i)) = \sum_{\beta \in \mathfrak{B}(a_i)} -ln(1 - r_{|\beta|}). \quad (2)$$

In Definition 2, the constraint of the SLADE problem is that the reliability of each atomic task satisfies a given reliability threshold t_i, namely $Rel(a_i, \mathfrak{B}(a_i)) \geq t_i$. Base on Equation (2), this constraint is equivalent to $-ln(1 -$

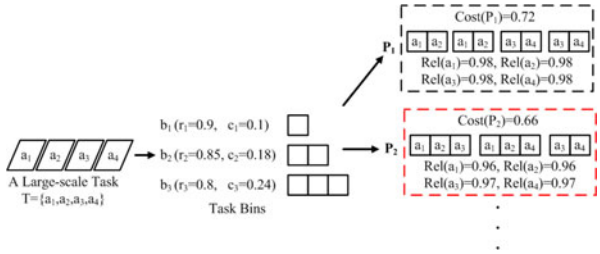


Fig. 4. Illustration of the SLADE problem.

$Rel(a_i, \mathfrak{B}(a_i)) \geq -ln(1 - t_i)$, namely $\sum_{\beta \in \mathfrak{B}(a_i)} -ln(1 - r_{|\beta|}) \geq -ln(1 - t_i)$, for an atomic task a_i. Thus the reliability of an atomic task is transformed to a sum of $\sum_{\beta \in \mathfrak{B}(a_i)} -ln(1 - r_{|\beta|})$.

4.2 Complexity Results

We first show the NP-hardness of the SLADE problem and then demonstrate that there are polynomial-time solutions to a relaxed variant of the SLADE problem.

Theorem 1. *The SLADE problem is NP-Hard.*

Proof. To complete the proof, we reduce the Unbounded Knapsack Problem(UKP) [25] to the SLADE problem. Then the hardness of the SLADE problem follows.

An instance of UKP is: given a set of m items with weights $\{w_1, \ldots, w_m\}$ and values $\{v_1, \ldots, v_m\}$, and each item can be used unbounded multiple times. The decision problem is to decide whether there exists a set $N = \{n_1, \ldots, n_m\}$ (denoted as the number that each item is used) such that the total weight is no more than a specific weight threshold, i.e., $\sum_{i=1}^{m} n_i \cdot w_i \leq W$ and the total value is no less than a given value threshold, i.e., $\sum_{i=1}^{m} n_i \cdot v_i \geq V$. Without loss of generality, we can assume that $v_i > 0$ for every item.

An instance of SLADE problem can be constructed from the above instance of UKP as follows:

- Construct m task bins $B = \{b_1, \ldots, b_m\}$. Each item in UKP corresponds to a task bin.
- For each task bin b_i, let $c_i = w_i$ and $r_i = 1 - e^{-v_i}$.
- For the crowdsourcing task T in SLADE problem, there is only one atomic task a_1 with the reliability threshold $t_1 = 1 - e^{-V}$.

Let $DP_T = \{\tau_i, b_i\}_{i=1}^{m}$ be the decomposition plan of the SLADE instance. To complete the proof, we prove that the decomposition plan DP_T spends no more than W subject to $\sum_{i=1}^{m} [\tau_i \cdot -\ln(1 - r_i)] \geq -\ln(1 - t_1)$ if and only if $N = \{\tau_1, \ldots, \tau_m\}$ is a feasible solution of UKP.

Since there is only one atomic task in T, then the reduction of reliability defined by Equation (2) is equal to

$$\sum_{i=1}^{m} [\tau_i \cdot -\ln(1 - r_i)] = \sum_{i=1}^{m} [\tau_i \cdot -\ln(1 - 1 + e^{-v_i})] = \sum_{i=1}^{m} \tau_i \cdot v_i.$$

Besides, $-\ln(1 - t_1) = -\ln(1 - (1 - e^{-V})) = V$. Therefore, a feasible decomposition plan DP_T of the SLADE problem should satisfy $\sum_{i=1}^{m} \tau_i \cdot v_i \geq V$. And we also know that the cost of this plan is $\sum_{i=1}^{m} \tau_i \cdot c_i = \sum_{i=1}^{m} \tau_i \cdot w_i$, which should be no more than W.

Therefore, as long as DP_T is a feasible plan of the SLADE problem, $N = \{\tau_1, \ldots, \tau_m\}$ must be a feasible solution of UKP and vice versa.

To sum up, the decision version of SLADE problem can be reduced from an instance of UKP and UKP is NP-Complete. Hence, the decision version of SLADE problem is NP-Complete and the SLADE problem is NP-Hard. □

Complexity of a Relaxed Variant of the SLADE Problem. Although the SLADE problem is NP-Hard, there is a relaxed variant which can be solved in polynomial time. The relaxed variant requires that the confidences of all task bins are always greater than the maximum reliability threshold of all atomic tasks, namely $r_j \geq t_{\max}$ where $1 \leq j \leq m$, and $t_{\max}$ is the maximum t_i $(1 \leq i \leq n)$. That is, each atomic task satisfies its reliability threshold requirement no matter which task bin it is assigned to. This relaxed variant of the SLADE problem can be simplified to the ROD CUTTING problem [26], which has an efficient dynamic programming exact solution with $\mathcal{O}(nm)$ time complexity, where n and m are the number of atomic tasks in the large-scale task and the number of distinct task bins, respectively.

4.3 Baseline Algorithm

In this subsection, we first reduce the SLADE problem to the CIP problem [24] and present a baseline algorithm using existing solutions of the CIP problem.

The CIP problem is shown as follows. Given a matrix U of integer non-negative coefficients $u_{i,j} \in \mathbb{N}$ $(i \in I = \{1, \ldots, |I|\}, j \in J = \{1, \ldots, |J|\})$, and positive vectors C and V, the CIP problem is to find a vector $Y \in \mathbb{N}$ such that

$$\begin{aligned} \min \quad & \sum_{j \in J} c_j y_j \\ \text{s.t.} \quad & \sum_{j \in J} u_{ij} y_j \geq v_i \qquad \forall i \in I \\ & y_j \in \mathbb{N}, \qquad \forall j \in J, \end{aligned} \tag{3}$$

where $y_j \in Y$, $c_j \in C$ and $v_i \in V$ [24].

We can reduce the SLADE problem to the CIP problem in two steps.

- **Step 1.** For the n atomic tasks in T and an l-cardinality task bin $b_l \in B$, there are $\binom{n}{l}$ distinct combination instances, which consist of the set C_l $(l \in \{1, \ldots, m\})$ and $|C_l| = \binom{n}{l}$. Thus, let $|J| = \sum_{l=1}^{m} |C_l|$, and for $j \in [1 + \sum_{i=1}^{l-1} C_i, \sum_{i=1}^{l} C_i]$, each $c_j = c_l$ and $u_{ij} = -\ln(1 - r_l)$ if the task a_i is batched into an l-cardinality task bin in the jth instance of J.
- **Step 2.** For each atomic task a_i with the reliability threshold t_i in the SLADE problem, we have $v_i = -\ln(1 - t_i)$ $(v_i \in V)$ and $|I| = n$ in the CIP problem.

Finally we come up with a *baseline algorithm* for the SLADE problem as follows.

- Transform the SLADE problem to the CIP problem by the aforementioned reduction process.
- Solve the CIP problem via existing methods [24].
- Return the results of the reduced CIP problem, which is equivalent to the planning of the SLADE problem.

Note that the baseline algorithm cannot give the optimal solution, as the CIP problem [24] is NP-hard. Existing solutions are only approximate. Although the baseline algorithm can be applied to both homogenous and heterogenous SLADE problems and can be easily implemented, the reduction step will generate exponential $(\sum_{l=1}^{m} \binom{n}{l})$ combination instances. Thus, the baseline algorithm is impractical for large-scale crowdsourcing tasks, where there can be thousands or millions of atomic tasks. Accordingly, we only generate part of the combination instances for performance evaluation. To address the scalability issue, we propose a greedy heuristic algorithm and an optimal priority queue-based approximation framework in the next two sections.

5 HOMOGENEOUS SLADE

In this section, we study the homogeneous SLADE problem, where all reliability thresholds $t_i (1 \leq i \leq n)$ *are equal*. Thus, all reliability thresholds are simplified as t $(t_i = t, \forall i)$ in the rest of this section. In Section 5.1, we first present a greedy heuristic algorithm, called Greedy, which is simpler and more efficient than the baseline algorithm but has no approximation guarantee. Then we propose an optimal priority queue-based (OPQ) algorithm in Section 5.2, which is not only faster than the Greedy algorithm but also guarantees $\log n$ approximation ratio, where n is the number of atomic tasks in a specific large-scale crowdsourcing task. In particular, in some cases, the OPQ-Based algorithm can even return the exact optimal solution.

5.1 Greedy Algorithm

To obtain a decomposition plan that satisfies the reliability threshold and has low total cost, we need to consider both the incentive cost (cost for short) and the confidence of the assigned task bins to each atomic task. A task bin with smaller cost will result in lower total cost, while a task bin with higher confidence can possibly reduce the number of task bins used in the decomposition plan. Therefore, the greedy algorithm is to consider the *cost-confidence ratio* of each task bin and its corresponding atomic tasks and include the task bin and its corresponding atomic tasks with the lowest ratio into the decomposition plan until the all atomic tasks satisfy the reliability threshold constraint.

Specifically, the cost-confidence ratio for an l-cardinality task bin and its corresponding atomic tasks is defined as:

$$ratio = \frac{c_l}{\min\{l \times (-\ln(1 - r_l)), \sum_{k=1}^{l} \theta_{i_k}\}}. \tag{4}$$

In Equation (4), c_l is the cost of the l-cardinality task bin b_l, and r_l is the confidence of the atomic tasks in b_l. As explained in Section 4.1, $-\ln(1 - r_l)$ is the contributed reliability per atomic task in b_l. Thus, $l \times (-\ln(1 - r_l))$ is the total contributed reliability for the atomic tasks in b_l. We further define the threshold residual θ_{i_k} of the i_k-th $(1 \leq k \leq l)$ atomic task, which is its reliability threshold subtracting its current total reliability contributed by the assigned task bins. It is possible that the total threshold residual of the assigned l atomic tasks in b_l is smaller than $-\ln(1 - r_l)$, thus the cost-confidence ratio should be $\frac{c_l}{\min\{l \times (-\ln(1 - r_l)), \sum_{k=1}^{l} \theta_{i_k}\}}$. Based on the cost-confidence ratio, the main idea of the greedy algorithm is to choose the locally optimal task bin b_{l^*} and assign l^* atomic tasks with the highest l^* threshold residuals in each iteration, and then the algorithm maintains the threshold residual of each atomic task and ranks all the

atomic tasks according to their current threshold residuals. Finally, the algorithm terminates when every threshold residual becomes zero.

Algorithm 1. Greedy

Input: A large-scale task $T = \{a_1, \ldots, a_n\}$, a set of task bins $\{b_1, \ldots, b_m\}$, a reliability threshold t
Output: An approximate decomposition plan DP_T, and an approximate total cost $Cost_T$

1 $DP_T \leftarrow \emptyset$;
2 Initialize $\theta = \{\theta_1, \ldots, \theta_n\}$ for each atomic task, where each $\theta_i \leftarrow -\ln(1-t)$;
3 Rank $T = \{a_{i_1}, \ldots, a_{i_n}\}$ in non-ascending order of θ_i;
4 **while** $\theta_{i_1} > 0$ **do**
5 $\quad l^* \leftarrow argmin_{l \in B} \frac{c_l}{\min\{l \times (-\ln(1-r_l)), \sum_{k=1}^{l} \theta_{i_k}\}}$;
6 $\quad DP_T \leftarrow DP_T \cup \{\{a_{i_1}, \ldots, a_{i_{l^*}}\}\}$;
7 $\quad Cost_T \leftarrow Cost_T + c_{l^*}$;
8 $\quad$ **for** $k \leftarrow 1$ to l^* **do**
9 $\quad\quad \theta_{i_k} \leftarrow \theta_{i_k} - (-\ln(1-r_{l^*}))$;
10 $\quad$ Rank $T = \{a_{i_1}, \ldots, a_{i_n}\}$ in non-ascending order of θ_i;
11 **return** DP_T and $Cost_T$

The procedure of the greedy algorithm is illustrated in Algorithm 1. Initially, the decomposition plan DP_T is empty in line 1. Line 2 initializes the threshold residual θ_i of each atomic task to $-\ln(1-t)$. Then it ranks n atomic tasks in terms of their threshold residuals in line 3. Lines 4-10 iteratively perform the greedy strategy. As long as at least one atomic task fails to satisfy the reliability threshold requirement, the algorithm chooses the task bin with the minimum $\frac{c_l}{\min\{l \times (-\ln(1-r_l)), \sum_{k=1}^{l} \theta_{i_k}\}}$. After choosing the locally optimal task bin b_{l^*}, the algorithm allocates the first l^* ranked atomic tasks in T to the final decomposition plan and adds c_{l^*} to the incentive cost in lines 6 and 7, respectively. Then, the threshold residuals of the first l^* ranked atomic tasks are reduced by $-\ln(1-r_{l^*})$ each in lines 8-9. Afterwards the algorithm re-ranks all the atomic tasks in T in a non-ascending order of their threshold residuals in line 10. Finally, the whole procedure terminates when the threshold residual of each atomic task is zero.

Example 5 (Greedy Algorithm). Back to our running example. Given a crowdsourcing task with 4 atomic tasks, the set of task bins in Table 1, and the reliability threshold $t = 0.95$, Algorithm 1 executes as follows. It first initializes each $\theta_i = 2.996$ where $1 \leq i \leq 4$. Since all θ_i's are the same, the initial order of the atomic tasks is $\langle a_1, a_2, a_3, a_4 \rangle$. Then the algorithm selects the first task bin $\{a_1\}$ in the first round because the ratio $\frac{0.1}{-\ln(1-r_1)} = 0.043$ is the smallest in line 5. Then, $\theta_1 = 2.996 - 2.303 = 0.693$, and the algorithm re-ranks T as $\langle a_2, a_3, a_4, a_1 \rangle$, based on the corresponding threshold residuals of $2.996, 2.996, 2.996, 0.693$. The algorithm continues similar iterations till all the threshold residuals become zero. The final decomposition plan is: $\{a_1\}$, $\{a_2\}, \{a_3\}, \{a_4\}, \{a_1, a_2, a_3\}, \{a_4\}$, with a total cost of 0.74.

Computational Complexity Analysis. Note that the task bin with the maximum cardinality has the smallest confidence for each atomic task in this task bin. Therefore given an arbitrary atomic task and a reliability threshold t in the homogeneous SLADE problem, the upper bound on the number of iterations in Algorithm 1 is $n\lceil\frac{\ln(1-t)}{\ln(1-r_m)}\rceil$, where r_m is the confidence of the m-cardinality task bin, n is the total number of atomic tasks, and m is the maximum cardinality of all the task bins. Furthermore, the algorithm needs to rank all the atomic tasks according to their current threshold residuals, which costs $\mathcal{O}(n \log n)$ time per iteration. Hence the total computational complexity of Algorithm 1 is $\mathcal{O}(n\lceil\frac{\ln(1-t)}{\ln(1-r_m)}\rceil\ (m + n \log n)) = \mathcal{O}(n^2 \log n)$ since $\lceil\frac{\ln(1-t)}{\ln(1-r_m)}\rceil$ is a constant and $m \ll n$ in practice.

5.2 Optimal-Priority-Queue-Based (OPQ) Algorithm

In this subsection, we introduce an approximation algorithm based on a specific data structure, called the optimal priority queue. This approximation algorithm not only returns decomposition plans with lower total cost in practice but also has a lower time complexity. In particular, with the optimal priority queue data structure, we can even obtain the exact optimal solution in certain cases. In the following, we first introduce how to construct the optimal priority queue and then devise a faster approximation algorithm that guarantees $\log n$ approximation ratio.

5.2.1 Constructing the Optimal Priority Queue

Before introducing the optimal priority queue data structure, we first define two basic concepts, the *lowest common multiple* in a combination of task bins and the *unit cost* of an atomic task using this combination. Denote a combination of task bins as $Comb = \{n_{k_1} \times b_{k_1}, \ldots, n_{k_l} \times b_{k_l}\}$, where $n_{k_i} \times b_{k_i}$ means that an atomic task is assigned n_{k_i} times to k_i-cardinality task bins. The *least common multiple*, denoted as LCM, of $Comb$ is $lcm(k_1, k_2, \ldots, k_l)$, which represents the number of atomic tasks in T to be assigned using this combination. The *unit cost* of $Comb$ is $UC = \sum_{i=1}^{l} \frac{c_{k_i}}{k_i} n_{k_i}$.

Example 6 (Combination ($Comb$)). Given the set of task bins in Table 1, we can construct an arbitrary combination of task bins e.g., $Comb = \{3 \times b_1, 2 \times b_2, 1 \times b_3\}$. For $Comb$, its lowest common multiple is $LCM = 1 \times 2 \times 3 = 6$, and the unit cost of an atomic task using $Comb$ is $UC = 3 \times 0.1 + 2 \times \frac{0.18}{2} + 1 \times \frac{0.24}{3} = 0.56$, meaning that we can assign 6 atomic tasks to $Comb$, with an "averaged" incentive cost of 0.56 per atomic task and a total cost of $0.56 \times 6 = 3.36$ for the 6 atomic tasks. Fig. 5 illustrates the above $Comb$ and how 6 atomic tasks are assigned in this combination, where each atomic task is assigned to six task bins (three 1-cardinality bins, two 2-cardinality bins and one 3-cardinality bins). For example, as shown in the last row in Fig. 5, the atomic task a_1 is assigned into the six task bins $(\{a_1\}, \{a_1\}, \{a_1\}, \{a_1, a_2\}, \{a_1, a_2\}, \{a_1, a_2, a_3\})$.

Definition 4 (Optimal Priority Queue). *Given a set of task bins $B = \{b_1, \ldots, b_m\}$ and a reliability threshold t, an optimal priority queue OPQ is a priority queue consisting of the combinations of task bins ($Comb$'s) and satisfies the following conditions: (1) the elements in the optimal priority queue is ranked in an descending order of their corresponding LCM values; (2) for any element OPQ_i (with $OPQ_i.LCM$ and $OPQ_i.UC$) in the optimal priority queue, there is NO element OPQ_j (with $OPQ_j.LCM$ and $OPQ_j.UC$) such that $OPQ_i.LCM \geq OPQ_j.LCM$ and $OPQ_i.UC \geq OPQ_j.UC$; (3) all the*

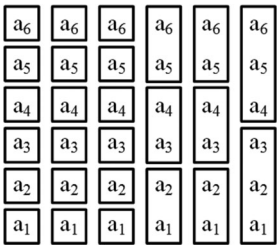


Fig. 5. Illustration of an arbitrary combination of task bins $Comb = \{3 \times b_1, 2 \times b_2, 1 \times b_3\}$.

combinations of task bins in this optimal priority queue satisfy the reliability threshold requirement for each atomic task.

Example 7 (Optimal Priority Queue). Back to our running example, given the set of task bins in Table 1, the optimal priority queue is shown in Table 3 with a reliability threshold of 0.95 for a_i, $1 \leq i \leq 4$. In Table 3, each column corresponds to a combination of task bins. For example, for the first column $\{2 \times b_3\}$, $OPQ_1.UC = 2 \times \frac{0.24}{3} = 0.16$ and $OPQ_1.LCM = 3$. In addition, if an atomic task is assigned to the $Comb$ in the first column, its reliability is $2 \times (-\ln(1-0.8)) = 3.22 > -\ln(1-0.95) = 2.996$. Thus, the atomic task satisfies the reliability threshold requirement. In fact, the $Comb$ in the first column is the optimal decomposition plan for $OPQ_1.LCM = 3$ atomic tasks. We describe an optimal priority queue based approximate algorithm for arbitrary numbers of atomic tasks in Section 5.2.2.

To obtain the optimal priority queue, we design a depth-first-search-based enumeration algorithm (Algorithm 2). The algorithm starts depth-first-search enumeration from one b_1 instance, removes unnecessary elements and returns the optimal priority queue in lines 1-3. In the depth-first-search enumeration process in lines 5-13, each recursion operation first checks whether the new combination cannot be pruned by Lemma 1 in line 7 and satisfies the reliability threshold requirement in line 8. If yes, the algorithm inserts the current combination into the optimal priority queue. Otherwise the algorithm continues until a combination of task bins satisfies the conditions in lines 7 and 8.

In Algorithm 2, the pruning rule in line 7 significantly reduces the redundant enumeration space as shown below.

Lemma 1. *Given two combinations of task bins $Comb_1$ and $Comb_2$, $Comb_2$ and all combinations that are supersets of $Comb_2$ can be safely pruned in the enumeration process if $Comb_1.UC < Comb_2.UC$ and $Comb_1.LCM \leq Comb_2.LCM$.*

Proof. According to the definition of the optimal priority queue, this pruning rule deletes the combinations which violate the requirement of monotonicity, i.e., condition (2). Hence, the lemma is correct. □

Example 8 (Building Optimal Priority Queue). Back to the set of task bins in Table 1 and $t = 0.95$. Algorithm 2 first enumerates the combinations based on b_1 until the combination $\{2 \times b_1\}$ since $2 \times (-\ln(1-0.9)) = 4.605 > -ln(1-0.95) = 2.996$. Then, the algorithm inserts the combination $\{2 \times b_1\}$ as OPQ_1, which is the first element in the optimal priority queue OPQ. After that, it recursively enumerates $\{b_1 + b_2\}$, which is updated as OPQ_1 because $-\ln(1-0.9) - \ln(1-0.85) = 4.20 > 2.996$, i.e., its $LCM = 2 > 1$ and its $UC = 0.19 < 0.2$. $\{2 \times b_1\}$ then becomes OPQ_2. Note that $\{b_1 + b_2\}$ is removed from OPQ when the combination $\{2 \times b_2\}$ is enumerated because $2 \times (-\ln(1-0.85)) = 3.794 > 2.996$, its $LCM = 2$ and its $UC = 0.18 < 0.19$. The final OPQ is shown in Table 3.

TABLE 3
The Optimal Priority Queue (OPQ) of Table 1 (t = 0.95)

$Comb$	$\{2 \times b_3\}$	$\{2 \times b_2\}$	$\{2 \times b_1\}$
UC	0.16	0.18	0.2
LCM	3	2	1

Algorithm 2. Building Optimal Priority Queue

Input: A set of task bins $B = \{b_1, \ldots, b_m\}$, a reliability threshold t
Output: An optimal priority queue OPQ
1 Enumerate$(1, 0, \emptyset, B, t)$;
2 Remove any OPQ_i with $OPQ_i.LCM \geq OPQ_j.LCM$ and $OPQ_i.UC \geq OPQ_j.UC$ for some j;
3 **return**OPQ;
4 **SubFunction:Enumerate(**p, q, S, B, t**)**
5 **for** $k \leftarrow p$ to m **do**
6 Add b_k into S;
7 **if** $\forall i$: $S.LCM < OPQ_i.LCM$ or $S.UC < OPQ_i.UC$ **then**
8 **if** $q - \ln(1-r_k) \geq -\ln(1-t)$ **then**
9 Insert S into OPQ;
10 Remove any OPQ_i with $OPQ_i.LCM = S.LCM$ and $OPQ_i.UC > S.UC$;
11 **else**
12 Enumerate$(k, q - \ln(1-r_k), S, B, t)$;
13 Remove b_k from S;

5.2.2 OPQ-Based Algorithm

Based on the optimal priority queue, we propose an enhanced approximation algorithm, called the optimal-priority-queue-based algorithm (OPQ-Based for short). Its main idea is to repeatedly utilize the optimal combinations in the optimal priority queue to approximate the global optimal solution. Given the number of atomic tasks in T, denoted by n, and the lowest common multiple of the first element in the optimal priority queue, denoted by LCM, the decomposition plan is globally optimal if $n \equiv 0, \pmod{LCM}$. Otherwise, we prove that the enhanced approximation algorithm still has a $\log n$ approximation ratio guarantee.

The pseudo code of the OPQ-Based Algorithm is shown in Algorithm 3. Line 1 initializes the optimal priority queue using Algorithm 2. Then the algorithm iteratively assigns atomic tasks to combinations of task bins in OPQ in lines 4-17. Specifically, the algorithm assigns the first $\lfloor \frac{n}{OPQ_1.LCM} \rfloor \times OPQ_1.LCM$ atomic tasks to the first element OPQ_1 in OPQ in each iteration. The remaining $n \bmod OPQ_1.LCM$ atomic tasks are processed in subsequent iterations in lines 13-17. We record the previous assignment in lines 16-17 to avoid the condition where the cost incurred in lines 8-10 in the current iteration is greater than the previous one. Once the condition holds, we simply use OPQ_{prev} to make

assignments for the remaining tasks. Since n is smaller than $OPQ_{prev}.LCM$, the algorithm will terminate. We explain Algorithm 3 in the following example.

Algorithm 3. OPQ-Based

Input: A large-scale task $T = \{a_1, \ldots, a_n\}$, a set of task bins $\{b_1, \ldots, b_m\}$, a reliability threshold t
Output: An approximate decomposition plan DP_T, and an approximate decomposition cost $Cost_T$

1 Initialize the optimal priority queue OPQ;
2 $Cost_{prev} \leftarrow \infty$;
3 **while** $n > 0$ **do**
4 **while** $OPQ_1.LCM > n$ **do**
5 Remove OPQ_1 from OPQ;
6 $k \leftarrow \lfloor \frac{n}{OPQ_1.LCM} \rfloor$;
7 **if** $k \times OPQ_1.LCM \times OPQ_1.UC > Cost_{prev}$ **then**
8 $DP_T \leftarrow$ Assignment($T, OPQ_{prev}, OPQ_{prev}.LCM$);
9 $Cost_T \leftarrow Cost_T + OPQ_{prev}.LCM \times OPQ_{prev}.UC$;
10 $n \leftarrow n - OPQ_{prev}.LCM$;
11 **else**
12 $DP_T \leftarrow$ Assignment($T, OPQ_1, k \times OPQ_1.LCM$);
13 $Cost_T \leftarrow Cost_T + k \times OPQ_1.LCM \times OPQ_1.UC$;
14 Remove the first $k \times OPQ_1.LCM$ atomic tasks from T;
15 $n \leftarrow n \bmod OPQ_1.LCM$;
16 $OPQ_{prev} \leftarrow OPQ_1$;
17 $Cost_{prev} \leftarrow OPQ_1.LCM \times OPQ_1.UC$;
18 **return** DP_T and $Cost_T$

Example 9 (OPQ-Based Algorithm). Given the set of task bins in Table 1 and a reliability threshold $t = 0.95$, the algorithm first finds the optimal priority queue as in Table 3. Then in the first iteration, the algorithm uses $OPQ_1 = \{2 \times b_3\}$ to assign a_1, a_2 and a_3. In the second iteration, it uses $\{2 \times b_1\}$ to assign a_4. The final decomposition plan is $2 \times \{a_1, a_2, a_3\}$ and $2 \times \{a_4\}$ with the total cost of $1 \times 3 \times 0.16 + 1 \times 1 \times 0.2 = 0.68$ (the cost of one combination of $\{2 \times b_3\}$ plus the cost of one combination of $\{2 \times b_1\}$), which is lower than 0.76 using the greedy algorithm.

Lemma 2. *OPQ_1 yields the lowest unit cost ($OPQ_1.UC$) for one atomic task in all the combinations of task bins ($Comb$).*

Proof. Note that for any $Comb$ which is used to accomplish one atomic task, its $Comb.UC > OPQ_1.UC$ if its $Comb.LCM > OPQ_1.LCM$, since this $Comb$ must be visited in Algorithm 2 and replaced by OPQ_1. Suppose its $Comb.LCM < OPQ_1.LCM$, we consider two cases. If this $Comb$ remains in OPQ, it becomes OPQ_i and its $OPQ_i.UC$ is still greater $OPQ_1.UC$ due to the smallest index of OPQ_1. If not, there must be an $OPQ_i \in OPQ\,(i \geq 1)$ such that $OPQ_i.UC < Comb.UC$, and the result still holds. □

Lemma 3. *When the total number of tasks n is equal to $OPQ_1.LCM$, OPQ_1 achieves an optimal solution.*

Proof. From lemma 2 we know the lowest unit cost is $OPQ_1.UC$. Since we need to finish (at least) n atomic tasks, the lemma follows straightaway. □

By induction, we have the corollary below.

Corollary 1. *When n is equal to $k \times OPQ_1.LCM (k \in N^+)$, using OPQ_1 for n times is an optimal solution.*

The following theorem shows the approximation ratio of Algorithm 3 (the OPQ-Based algorithm) for an arbitrary n.

Theorem 2. *The approximation ratio of Algorithm 3 is $\log n$, where n denotes the number of atomic tasks in T.*

Proof. We denote the index of combinations of task bins in OPQ in Algorithm 3 by $j_1, j_2, \ldots, j_r$, where r is the number of iterations of the algorithm. Then the number of atomic tasks assigned in each iteration will be $k_1 \times OPQ_1.LCM, \ldots, k_l \times OPQ_{j_r}.LCM$. We assume $j_1 = 1$ for a large-scale crowdsourcing task T, i.e., $n \geq OPQ_1.LCM$. Lines 8-11 in Algorithm 3 indicate that for any s, t, $1 \leq s \leq t \leq r$, we have $k_s \times OPQ_{j_s}.LCM \times OPQ_{j_s}.UC \geq k_t \times OPQ_{j_t}.UC \times OPQ_{j_t}.UC$. Then we have

$$\begin{aligned} OPT &\geq n \times OPQ_1.UC \\ &\geq k_1 \times OPQ_1.LCM \times OPQ_1.UC \\ &\geq k_s \times OPQ_{j_s}.LCM \times OPQ_{j_s}.UC, s = 1, 2, \ldots, r. \end{aligned} \tag{5}$$

The first inequality holds because the optimal value of the linear programming relaxed from the original problem is a lower bound of OPT. We sum up the costs incurred in each iteration ($k_s \times OPQ_{j_s}.LCM \times OPQ_{j_s}.UC$), then we have $Cost_T \leq r \times OPT$. Next we give an upper bound of the total number of iterations r. We consider some iteration s. Here we use n to denote the number of remaining tasks in this iteration. If $OPQ_{j_s}.LCM \geq n/2$, the remainder will be $n - OPQ_{j_s}.LCM < n/2$. If $OPQ_{j_s}.LCM < n/2$, the remainder is less than $OPQ_{j_s}.LCM < n/2$. In total, at most $\log n$ iterations, the algorithm terminates. The approximation ratio will be $\log n$. □

Computational Complexity Analysis: According to Algorithm 3, the time complexity of this algorithm is $\mathcal{O}(\alpha \log n)$, where α is the cost to make assignment for $OPQ_1.LCM$ atomic tasks, which is small in practice.

6 Heterogeneous SLADE

In this section, we study the heterogeneous SLADE problem, where the atomic tasks in a large-scale crowdsourcing task can have different reliability thresholds. In the following, we will introduce how to extend our proposed algorithms, Greedy and OPQ-Based in the homogeneous scenario to solve the heterogeneous SLADE problem.

First, we shows that the Greedy algorithm (Algorithm 1) still works by only changing the reliability thresholds of the atomic tasks. In fact, for Algorithm 1, different reliability thresholds t_i only affect the original threshold residual θ_i of each atomic task in line 2 in Algorithm 1. Thus the algorithm still works in the heterogeneous SLADE problem.

The OPQ-Based algorithm (Algorithm 3) can be also extended to the heterogeneous scenario using the following partition method, and we call the extended algorithm OPQ-Extended. The main idea is to partition the whole set of atomic tasks into groups and run Algorithm 3 for each group. Specifically, we first use quantiles of $2^{\alpha+i}$ to divide the range of the thresholds into different intervals. α will be defined in line 4 of Algorithm 4. Since the upper bound of an interval can bound the thresholds of the atomic tasks that fall into this interval, we construct some optimal

TABLE 4
The Optimal Priority Queue OPQ^0 ($t = 0.632$)

$Comb$	$\{1 \times b_3\}$	$\{1 \times b_2\}$	$\{1 \times b_1\}$
UC	0.08	0.09	0.1
LCM	3	2	1

TABLE 5
The Optimal Priority Queue OPQ^1 ($t = 0.86$)

$Comb$	$\{1 \times b_1\}$
UC	0.1
LCM	1

priority queues based on the upper bounds of the divided intervals. Then for each interval, we can perform Algorithm 3 to obtain an approximate decomposition plan.

Algorithm 4. Building Optimal Priority Queue Set

Input: A large-scale task $T = \{a_1, \ldots, a_n\}$, a set of task bins $B = \{b_1, \ldots, b_m\}$, reliability thresholds $\{t_1, \ldots, t_n\}$
Output: A set of optimal priority queues $OPQS$

1 Initialize $OPQS \leftarrow \emptyset$, $\theta = \{\theta_1, \ldots, \theta_n\}$, where each $\theta_i \leftarrow -\ln(1 - t_i)$;
2 $\theta_{\min} \leftarrow \min(\theta_1, \ldots, \theta_n)$;
3 $\theta_{\max} \leftarrow \max(\theta_1, \ldots, \theta_n)$;
4 $\alpha \leftarrow \lfloor \log \theta_{\min} \rfloor$, $i \leftarrow 0$;
5 **while** $2^{\alpha+i} < \theta_{\max}$ **do**
6 **if** $2^{\alpha+i+1} > \theta_{\max}$ **then**
7 $\tau \leftarrow \theta_{\max}$;
8 **else**
9 $\tau \leftarrow 2^{\alpha+i+1}$;
10 $OPQ^i \leftarrow Algorithm2(B, 1 - e^{\tau})$;
11 $OPQS \leftarrow OPQS \cup OPQ^i$;
12 $i \leftarrow i + 1$;
13 **return** $OPQS$

The Algorithm 4 shows the process that builds a set of optimal priority queues, denoted by $OPQS$, based on the range of the thresholds $[\theta_{\min}, \theta_{\max}]$. Specifically, the algorithm iteratively builds an optimal priority queue OPQ^i for the interval with the upper bound $2^{\alpha+i+1}$. In line 7, it ensures that the upper bound of the final interval is $\theta_{\max}$. In each iteration, the algorithm increases i by 1 in line 12 and thus proceeds to the next interval. The algorithm will terminate until the upper bound is greater than $\theta_{\max}$. It finally returns the set of optimal priority queues $OPQS$.

Example 10 (Building Optimal Priority Queue Set). Back to our running example of four atomic tasks a_1, a_2, a_3 and a_4, we set their reliability thresholds to 0.5, 0.6, 0.7 and 0.86. Thus the corresponding the values of $\theta_i = -\ln(1 - t_i)$ are $\theta_1 = 0.69$, $\theta_2 = 0.92$, $\theta_3 = 1.61$ and $\theta_4 = 1.97$, respectively. The parameter α is initialized as $\lfloor \log 0.69 \rfloor = -1$. In the first ($i = 0$) iteration, since $2^{-1+0+1} = 2^0 = 1 < 1.97 = \theta_{\max}$, therefore $\tau = 1$ and OPQ^0 is generated with threshold $1 - e^{-1} = 0.632$ using Algorithm 2. Table 4 shows the optimal priority queue OPQ^0 generated after this iteration. Then in the second ($i = 1$) iteration, note that $2 > 1.97 = \theta_{\max}$, $\tau = \theta_{\max} = 1.97$. Hence OPQ^1 is generated with the threshold $1 - e^{-1.97} \approx 0.86$, which is shown in Table 5. Finally, $OPQS = \{OPQ^0, OPQ^1\}$ in this example.

The basic idea of the optimal priority queue-extended (OPQ-Extended) algorithm is to partition all atomic tasks into different groups and run the OPQ-Based algorithm (Algorithm 3) based on the building optimal priority queue set algorithm (Algorithm 4) for each group. Algorithm 5 illustrates the procedure. First, all atomic tasks are divided into different groups in lines 5-7. For each atomic task, the algorithm finds the upper bound of the interval in which θ_i lies, and assigns it to the corresponding set. Then, for each set of atomic tasks S_i, we perform the OPQ-Based algorithm (Algorithm 3) using the corresponding OPQ^i in lines 8-16. Finally, we merge the decomposition plan for each set to generate the global decomposition plan in line 17.

Algorithm 5. OPQ-Extended

Input: A large-scale task $T = \{a_1, \ldots, a_n\}$, a set of task bins $B = \{b_1, \ldots, b_m\}$, reliability thresholds $\{t_1, \ldots, t_n\}$
Output: An approximate decomposition plan DP_T, and an approximate decomposition cost $Cost_T$

1 Initialize $\theta = \{\theta_1, \ldots, \theta_n\}$, where each $\theta_i \leftarrow -\ln(1 - t_i)$;
2 $\alpha \leftarrow \lfloor \log \theta_{\min} \rfloor$, $\beta \leftarrow \lceil \log \frac{\theta_{\max}}{\theta_{\min}} \rceil$;
3 Initialize $OPQS = \{OPQ^0, \ldots, OPQ^{\beta-1}\}$ using Algorithm 4;
4 Set $S_0, \ldots, S_{\beta-1}$ all $\emptyset$;
5 **foreach** $a_i \in T$ **do**
6 Find the lowest j s.t. $\theta_i \leq 2^j$;
7 Assign a_i into $S_{j-\alpha-1}$;
8 **foreach** task set S_i **do**
9 **if** i equals $\beta - 1$ **then**
10 $DP_{S_i} \leftarrow Algorithm3(B, t_{\max}, OPQ^i)$;
11 $Cost_{S_i} \leftarrow Algorithm3(B, t_{\max}, OPQ^i)$;
12 **else**
13 $DP_{S_i} \leftarrow Algorithm3(B, 1 - e^{-2^{\alpha+i+1}}, OPQ^i)$;
14 $Cost_{S_i} \leftarrow Algorithm3(B, 1 - e^{-2^{\alpha+i+1}}, OPQ^i)$;
15 $DP_T \leftarrow DP_T \cup DP_{S_i}$;
16 $Cost_T \leftarrow Cost_T + Cost_{S_i}$;
17 **return** DP_T and $Cost_T$

Example 11 (OPQ-Extended Algorithm). Back to our running example of four atomic tasks a_1, a_2, a_3 and a_4 with their reliability thresholds to 0.5, 0.6, 0.7 and 0.86, the optimal priority queue set $OPQS = \{OPQ^0, OPQ^1\}$ is generated in Example 10. Based on $OPQS$, the four atomic tasks are divided into two sets S_0 and S_1. Specifically, a_1, a_2 and a_3, a_4 are assigned to S_0 and S_1, respectively. Algorithm 3 returns the decomposition plan $DP_{S_0} = \{a_1, a_2\}$ for S_0 using the combination of $\{1 \times b_2\}$ in OPQ^0. Similarly, $DP_{S_0} = \{\{a_3\}, \{a_4\}\}$ for S_1 using the combination of $\{1 \times b_1\}$ in OPQ^1. Finally the global decomposition plan is $\{\{a_1, a_2\}, \{a_3\}, \{a_4\}\}$ with a total cost of 0.38.

Theorem 3. *The approximation ratio of Algorithm 5 is* $2\lceil \log \frac{\theta_{\max}}{\theta_{\min}} \rceil \log n$.

Proof. For any atomic task $a \in S_i$, the transformed threshold of a should be in the range of $[2^{\alpha+i}, 2^{\alpha+i+1})$. We define $Cost_{[2^{\alpha+i}, 2^{\alpha+i+1})}$ as the cost incurred by the algorithm when making assignments for all the atomic tasks in S_i. Let $OPT_{2^{\alpha+i+1}}$ be the minimum cost when the transformed

threshold is homogeneously $2^{\alpha+i+1}$. Following Theorem 2, we immediately have $Cost_{[2^{\alpha+i},2^{\alpha+i+1})} \leq \log n * OPT_{2^{\alpha+i+1}}$. Note that adopting the decomposition plan of $OPT_{2^{\alpha+i}}$ twice can be regarded as a feasible decomposition plan for the atomic tasks with the homogeneous transformed threshold of $2^{\alpha+i+1}$. This indicates that $OPT_{2^{\alpha+i+1}} \leq 2OPT_{2^{\alpha+i}}$. When we use $OPT_{[2^{\alpha+i},2^{\alpha+i+1})}$ to describe the minimum cost where the transformed thresholds of atomic tasks in S_i are heterogeneously in the range of $[2^{\alpha+i}, 2^{\alpha+i+1})$, we have $OPT_{2^{\alpha+i}} \leq OPT_{[2^{\alpha+i},2^{\alpha+i+1})}$. Summing up over i, the cost incurred by the algorithm $Cost$ is no greater than $2\lceil \log \frac{\theta_{\max}}{\theta_{\min}} \rceil \log n\, OPT$, where OPT is the optimal solution, because $i = 0, \ldots, \lceil \log \frac{\theta_{\max}}{\theta_{\min}} \rceil - 1$. □

Computational Complexity Analysis: In Algorithm 5, there are at most $\lceil \log \frac{\theta_{\max}}{\theta_{\min}} \rceil$ iterations. Thus for each iteration, the time complexity is $\mathcal{O}(\lceil \log \frac{\theta_{\max}}{\theta_{\min}} \rceil (\log n + \alpha + \gamma))$, where α is the cost to find the optimal priority queue and γ is the cost to make assignment for S_i.

7 Experimental Study

This section presents the performance evaluation. All experiments are conducted on an Intel(R) Core(TM) i7 3.40 GHz PC with 4 GB main memory and Microsoft Windows 7 OS. All the algorithms are implemented and compiled using Microsoft's Visual C++ 2010.

Our empirical studies are conducted on two real datasets gathered by running tasks on Amazon MTurk. The first dataset is gathered from the *jelly-beans-in-a-jar* experiments (labelled as *"Jelly"*) and the second is from the *micro-expression identification* experiments (labelled as *"SMIC"*). The detailed settings of the two experiments are presented in Section 2. We set the default value of maximum cardinality ($|B|$) to 20, and the number of atomic tasks to 10,000. In homogenous scenarios, the reliability threshold t is set to 0.9 for all atomic tasks. In heterogeneous scenarios, the default reliability thresholds are generated according to the Normal distribution with parameters μ and σ set to 0.9 and 0.03, respectively. The experiments with reliability thresholds generated according to heavy tailed and uniform distributions are also conducted. As the results are similar, we omit them due to the limited space.

In the following evaluations, *Baseline*, *Greedy*, and *OPQ-Based* represent the baseline algorithm, the greedy algorithm, and the optimal-priority-queue-based algorithm, respectively. *OPQ-Extended* is the extended OPQ-Based algorithm for heterogeneous scenarios. We mainly evaluate the effectiveness and the efficiency of the algorithms.

7.1 Evaluations in the Homogeneous Scenario

This subsection presents the performance of the three algorithms in the homogeneous scenario.

Varying t. Fig. 6a and 6b report the decomposition cost with various reliability thresholds. The decomposition costs of all the three algorithms decrease with a lower reliability threshold t, because fewer crowd workers (and thus task bins) are needed to satisfy the lower reliability requirement. Fig. 6c and 6d show the running time of the three algorithms with the same sets of reliability thresholds. The running time of *OPQ-Based* is insensitive to the reliability threshold, while those of *baseline* and *Greedy* drop dramatically with low reliability thresholds. This is because *OPQ-Based* finds the optimal combination using the optimal-priority-queue structure in advance.

Varying $|B|$. Fig. 6e and 6f show the decomposition cost when the maximum cardinality $|B|$ varies from 1 to 20. With the increase of the maximum cardinality $|B|$, all algorithms tend to gain lower cost, as they can choose from more kinds of task bins. We also see that the decomposition cost of *Baseline* is significantly affected by $|B|$. This is reasonable since *Baseline* obtains the solution via the randomized rounding method, which is easily affected by a random noise when $|B|$ is small. Conversely, the other two algorithms are less sensitive to $|B|$, especially when $|B| \geq 6$. Then we test the efficiency of the proposed algorithms with the same set of $|B|$. The results are shown in Fig. 6g and 6h. *OPQ-Based* outperforms the others due to the optimal priority queue data structure design.

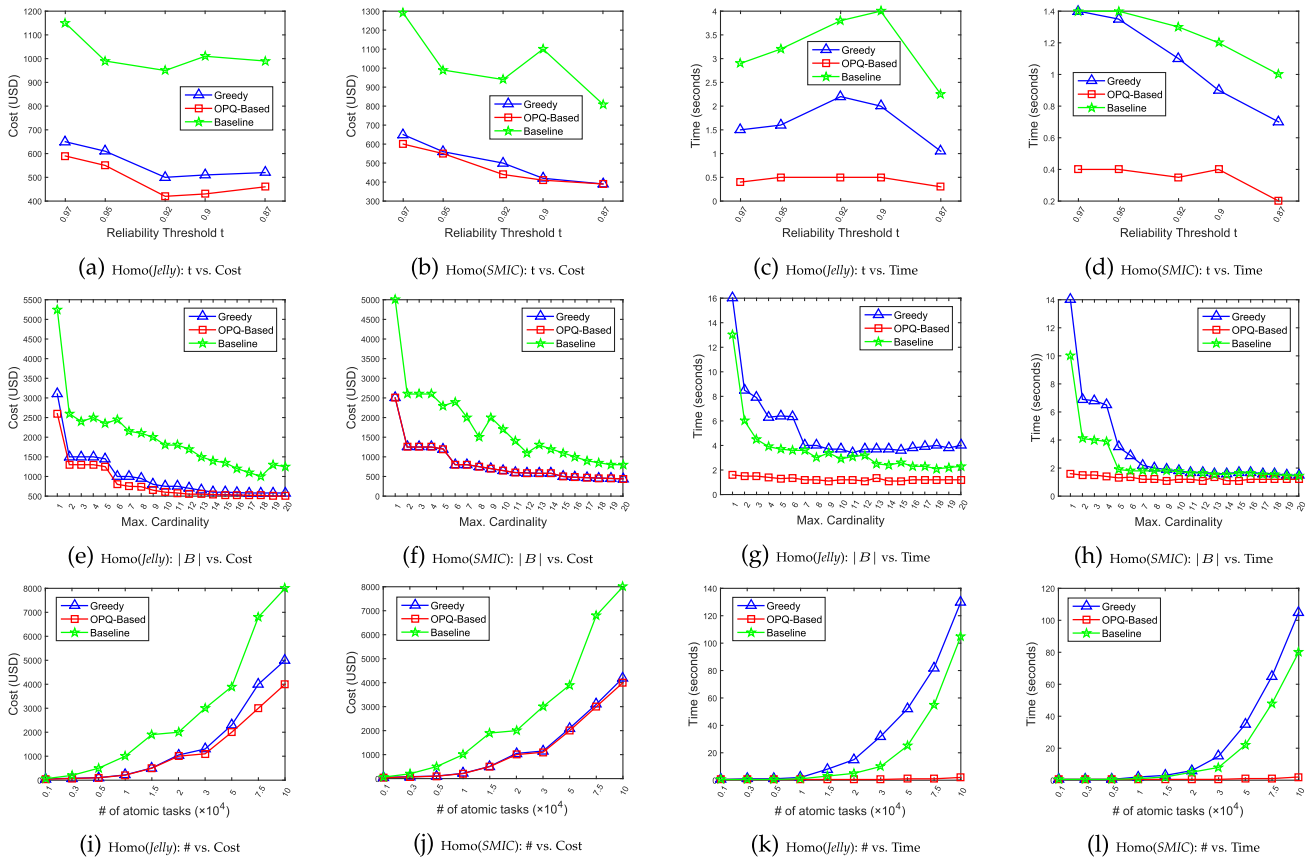
Scalability. We first study the decomposition cost of the three algorithms, by setting the number of atomic tasks, i.e., parameter #, from 1,000 to 10,000. Fig. 6i and 6j compare the decomposition cost of the three algorithms. As expected, when the # of atomic tasks increases from 1,000 to 100,000, the decomposition cost of the three algorithms all increases. This is because more atomic tasks lead to more crowd workers and thus more total cost. *OPQ-Based* has the smallest decomposition cost on the two datasets. This is because *OPQ-Based* first finds the optimal combinations for an atomic task and provides the decomposition plan in terms of the optimal combinations. This also verifies the better approximation ratio of *OPQ-Based* in practice. *Greedy* is more effective than *Baseline* in some cases. This is because *Baseline* utilizes a randomized rounding method, which may not be effective in certain cases. Fig. 6k and 6l plot the running time of the three algorithms with the same set of atomic task quantities. *OPQ-Based* is the fastest, and *Baseline* is much slower than *OPQ-Based* but faster than *Greedy*. This is because *OPQ-Based* pre-computes the optimal combinations for an atomic task while *Greedy* adopts the iterative strategy based on the local optimal solutions.

Conclusion. OPQ-Based is both more effective and efficient than the other two. *Baseline* is the least effective and *Greedy* is the least efficient.

7.2 Evaluations in the Heterogeneous Scenario

This subsection presents the performance of the algorithms for the heterogeneous scenario, where different atomic tasks may have different reliability thresholds. We generate the reliability thresholds following the Normal distribution. As with the evaluations for the homogeneous scenario, the experimental results on *Jelly* and *SMIC* are similar in the heterogenous scenario. Hence we only present the results on the *"Jelly"* dataset in the heterogenous scenario.

Varying Standard Deviation σ. Fig. 7a and 7b show the performance by varying the standard deviation σ of the reliability thresholds. With increasing σ, the decomposition costs of the three algorithms decrease. However, the change is not monotonous. It depends on two factors. First, as σ increases, the number of distinct reliability thresholds increases. Yet the decomposition cost with more distinct reliability thresholds might not be greater than that with fewer distinct reliability thresholds. The decomposition cost depends on the

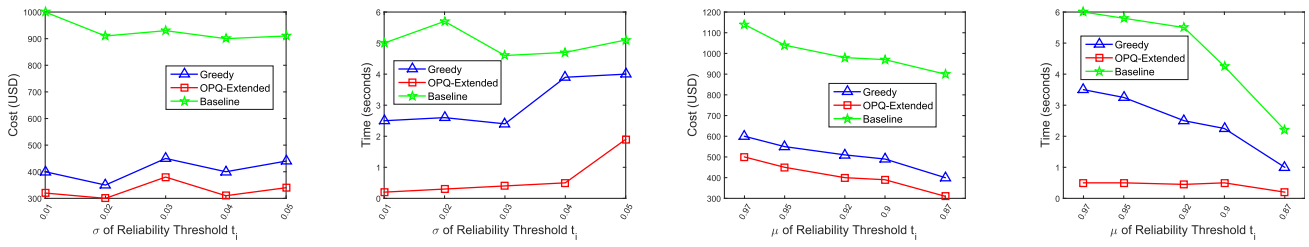


(a) Homo(*Jelly*): t vs. Cost (b) Homo(*SMIC*): t vs. Cost (c) Homo(*Jelly*): t vs. Time (d) Homo(*SMIC*): t vs. Time

(e) Homo(*Jelly*): $|B|$ vs. Cost (f) Homo(*SMIC*): $|B|$ vs. Cost (g) Homo(*Jelly*): $|B|$ vs. Time (h) Homo(*SMIC*): $|B|$ vs. Time

(i) Homo(*Jelly*): # vs. Cost (j) Homo(*SMIC*): # vs. Cost (k) Homo(*Jelly*): # vs. Time (l) Homo(*SMIC*): # vs. Time

Fig. 6. Results of homogeneous scenarios.



(a) Heter(*Jelly*): σ of t vs. Cost (b) Heter(*Jelly*): σ of t vs. time (c) Heter(*Jelly*): μ of t vs. Cost (d) Heter(*Jelly*): μ of t vs. Time

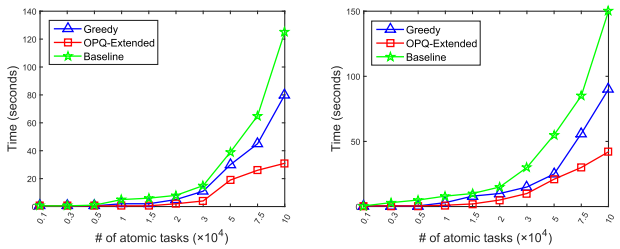
Fig. 7. Results of heterogeneous scenarios.

values of the reliability thresholds rather than the number of distinct reliability thresholds. Thus, the change of decomposition cost is not monotonous. Second, as σ increases, the likelihood of larger reliability thresholds also increases. The increase of the reliability threshold is $ln(1-\Delta t)$, where Δt is the increase ratio of the reliability threshold t. Thus, the trend of decomposition cost must decrease when the standard deviation of reliability thresholds increases. Fig. 7b shows that the running time of the three algorithms increases when σ increases. Due to the increase of σ, the number of distinct reliability thresholds increases. Hence, the three algorithms need more search space to find their approximate optimal solutions. Particularly, the running time of *OPQ-Extended* increases notably because it has to build a priority queue for each type of reliability threshold. With more distinct reliability thresholds, *OPQ-Extended* needs more running time.

Varying mean μ. Fig. 7c and 7d show the results by varying the mean μ of reliability thresholds. With decreasing μ, the decomposition cost of the three algorithms decreases. In most cases, *OPQ-Extended* has the lowest decomposition cost. This makes sense because *OPQ-Extended* first discovers the optimal combination for an atomic task and provides the decomposition plan based on the optimal combination.

Scalability. We study the scalability of the proposed algorithms in Fig. 8a and 8b over both the *Jelly* and *SMIC* datasets, by varying the parameter # from 1,000 to 100,000. The overall tendency resembles the cases in homogeneous scenarios. But with larger number of the atomic tasks, *OPQ-Based* takes longer running time compared to that in homogeneous scenarios. This is because *OPQ-Based* has to construct optimal priority queues for various distinct reliability threshold values in heterogeneous scenarios.

Conclusion. In the heterogeneous scenarios, when the number of distinct reliability thresholds increases, the three algorithms spend more running time. When the number of lower reliability thresholds increases, the decomposition cost will decrease.



(a) Heter(*Jelly*): # vs. Time (b) Heter(*SMIC*): # vs. Time

Fig. 8. Scalability results of heterogeneous scenarios.

8 Related Work

Human computation has been practiced for centuries. Specifically, whenever a "human" serves to "compute", a human computation is observed. This leads to a history of Human Computation even longer than that of electronic computer. However, with the emergence of Internet web service, especially the one that facilitates online labor recruiting and managing like Amazon MTurk and oDesk, human computation starts to experience a new age where the source of human is broadened to a vast pool of crowds, instead of designated exerts or employees. This type of outsourcing to crowds, i.e., crowdsourcing, is now receiving countless success in many areas such as fund raising, logistics, monitoring and so on. The practice introduced in this paper is within the collection of data-driven applications, where database services and data mining services adopt online crowds as a Human Processing Unit (HPU) to tackle human intrinsic tasks [14], [15], [16].

In data-driven applications, human cognitive abilities are mainly exploited in two types: voting among many options, and providing contents according to certain requirements. Most basic queries in database [7] and data mining [27], [28] can be decomposed into simple voting as human tasks: max discovery [2], [3], task assignment[29], [30], [31], [32], [33], filtering [4], [34] and jury selection [35], [36], [37], join [5], [18], entity resolution [5], [38], and data cleaning [39] into two-option (Yes or No)or multiple voting (connecting same entities). Meanwhile, to break the close world assumption in traditional databases, human are enrolled to provide extraneous information to answer complex queries, such as item enumeration [40].

Moreover, several recent works have also been developed to optimize the performance of crowdsourcing platforms for different aspects [19], [22]. In particular, [19] proposes a difficulty control framework for the tasks based on majority voting aggregation rules. CrowdForge [22] is a prototype to decompose complex task like article writing, science journalism to small tasks. Note that most of the aforementioned work focus on higher-level query transformation from a specific type of task into the form of task bins and the corresponding aggregation rules, but our paper is the first work that focuses on providing a comprehensive instruction to build the in-effect "query optimizer" module in crowd-powered databases.

9 Conclusion

In this paper, we propose a general crowdsourcing task decomposition problem, called the Smart Large-scAle task DEcomposer Problem, which is proven to be NP-hard. To solve the SLADE Problem, we study it in homogeneous and heterogeneous scenarios, respectively. In particular, we propose a series of efficient approximation algorithms using the greedy strategy and the optimal priority queue data structure to discover near-optimal solutions. Finally, we verify the effectiveness and efficiency of the proposed algorithms through extensive empirical studies over representative crowdsourcing platforms.

Acknowledgments

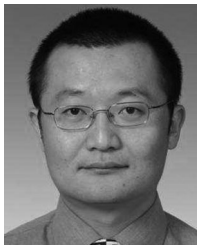
The authors are grateful to anonymous reviewers for their constructive comments on this work. Yongxin Tong and Weifeng Lv are supported in part by the National Grand Fundamental Research 973 Program of China under Grant 2015CB358700, NSFC Grant No. 61502021 and 61532004, and SKLSDE (BUAA) Open Program SKLSDE-2016ZX-13. Lei Chen is supported in part by the Hong Kong RGC Project 16202215, Science and Technology Planning Project of Guangdong Province, China, No. 2015B010110006, NSFC Grant No. 61729201, 61232018, Microsoft Research Asia Collaborative Grant, Huawei Grant, and NSFC Guang Dong Grant No. U1301253. H. V. Jagadish is supported in part by NSF Grant IIS-1250880. Lidan Shou is supported in part by NSFC Grant No. 61672455.

References

[1] J. Howe, *Crowdsourcing: Why the Power of the Crowd is Driving the Future of Business*. New York, NY, USA: Crown, 2009.
[2] S. Guo, A. G. Parameswaran, and H. Garcia-Molina, "So who won?: Dynamic max discovery with the crowd," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 385–396.
[3] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis, "Max algorithms in crowdsourcing environments," in *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 989–998.
[4] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom, "Crowdscreen: algorithms for filtering data with humans," in *Proc. ACM SIGMOD Int. Conf Manage. Data*, 2012, 361–372.
[5] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "Crowder: Crowdsourcing entity resolution," *Proc. VLDB Endowment* 2012, pp. 1483–1494.
[6] Y. Zheng, G. Li, Y. Li, C. Shan, and R. Cheng, "Truth inference in crowdsourcing: Is the problem solved?" *Proc. VLDB Endowment*, 2017, pp. 541–552.
[7] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "CrowdDB: Answering queries with crowdsourcing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 61–72.
[8] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom, "Deco: A system for declarative crowdsourcing," *Proc. VLDB Endowment*, 2012, pp. 1990–1993.
[9] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller, "Demonstration of qurk: A query processor for humanoperators," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 1315–1318.
[10] Y. Zheng, G. Li, and R. Cheng, "DOCS: domain-aware crowdsourcing system," in *Proc. VLDB Endowment*, 2016, pp. 361–372.
[11] G. Li, et al., "CDB: Optimizing queries with crowd-based selections and joins," in *Proc. ACM Int. Conf. Manage. Data*, 2017, pp. 1463–1478.
[12] G. Li, Y. Zheng, J. Fan, J. Wang, and R. Cheng, "Crowdsourced data management: Overview and challenges," in *Proc. ACM Int. Conf. Manage. Data*, 2017, pp. 1711–1716.
[13] Y. Tong, L. Chen, and C. Shahabi, "Spatial crowdsourcing: challenges, techniques, and applications," in *Proc. VLDB Endowment*, 2017, pp. 1988–1991.
[14] G. Li, J. Wang, Y. Zheng, and M. J. Franklin, "Crowdsourced data management: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 9, pp. 2296–2319, Sep. 2016.

[15] H. Garcia-Molina, M. Joglekar, A. Marcus, A. G. Parameswaran, and V. Verroios, "Challenges in data crowdsourcing," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 4, pp. 901–911, Apr. 2016.
[16] A. I. Chittilappilly, L. Chen, and S. Amer-Yahia, "A survey of general-purpose crowdsourcing techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 9, pp. 2246–2266, Sep. 2016.
[17] A. Jain, A. D. Sarma, A. Parameswaran, and J. Widom, "Understanding workers, developing effective tasks, and enhancing marketplace dynamics: A study of a large crowdsourcing marketplace," *Proc. VLDB Endowment*, 2017, pp. 829–840.
[18] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller, "Human-powered sorts and joins," *Proc. VLDB Endowment*, 2011, pp. 13–24.
[19] J. Gao, X. Liu, B. C. Ooi, H. Wang, and G. Chen, "An online cost sensitive decision-making method in crowdsourcing systems," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 217–228.
[20] T. Pfister, X. Li, G. Zhao, and M. Pietikainen, "Recognising spontaneous facial micro-expressions," in *Proc. IEEE Int.l Conf. Comput. Vis.*, 2011, pp. 1449–1456.
[21] P. D. Adamczyk and B. P. Bailey, "If not now, when?: the effects of interruption at different moments within task execution," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2014, pp. 271–278.
[22] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut, "Crowdforge: crowdsourcing complex work," in *Proc. 24th Annu. ACM Symp. User Interface Softw. Technol.*, 2011, pp. 43–52.
[23] S. Faradani, B. Hartmann, and P. G. Ipeirotis, "What's the right price? pricing tasks for finishing on time," in *Proc. 11th AAAI Conf. Human Comput.*, 2011, pp. 26–31.
[24] V. V. Vazirani, *Approximation Algorithms*. New York, NY, USA: Springer, 2003.
[25] M. Silvano and T. Paolo, *Knapsack Problems: Algorithms and Computer Implementations*. Hoboken, NJ, USA: Wiley, 1990.
[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (3rd Version)*. Cambridge, MA, USA: MIT Press, 2009.
[27] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart, "Crowd mining," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 241–252.
[28] R. Gomes, P. Welinder, A. Krause, and P. Perona, "Crowdclustering," in *Proc. 24th Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 558–566.
[29] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng, "QASCA: A quality-aware task assignment system for crowdsourcing applications," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1031–1046.
[30] C.-J. Ho and J. W. Vaughan, "Online task assignment in crowdsourcing markets," in *Proc. 26th AAAI Conf. Artif. Intell.*, 2012, pp. 45–51.
[31] T. Song, et al., "Trichromatic online matching in real-time spatial crowdsourcing," in *Proc. IEEE 33rd Int Conf. Data Eng.*, 2017, pp. 1009–1020.
[32] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing," in *Proc. IEEE 32nd Int. Conf. Data Eng.*, 2016, pp. 49–60.
[33] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, "Online minimum matching in real-time spatial data: Experiments and analysis," in *Proc. VLDB Endowment* 2016, pp. 1053–1064.
[34] A. G. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom, "Optimal crowd-powered rating and filtering algorithms," in *Proc. VLDB Endowment*, 2014, pp. 1053–1064.
[35] C. C. Cao, J. She, Y. Tong, and L. Chen, "Whom to ask? Jury selection for decision making tasks on micro-blog services," *Proc. VLDB Endowment*, 2012, pp. 1495–1506.
[36] C. C. Cao, Y. Tong, L. Chen, and H. V. Jagadish, "Wisemarket: A new paradigm for managing wisdom of online social users," in *Proc. 19th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, 2013, pp. 455–463.
[37] Y. Zheng, R. Cheng, S. Maniu, and L. Mo, "On optimality of jury selection in crowdsourcing," in *Proc. 18th Int. Conf. Extending Database Technol.*, 2015, pp. 193–204.
[38] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng, "Leveraging transitive relations for crowdsourced joins," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 229–240.
[39] Y. Tong, C. C. Cao, C. J. Zhang, Y. Li, and L. Chen, "CrowdCleaner: Data cleaning for multi-version data on the web via crowdsourcing," in *Proc. IEEE 30th Int. Conf. Data Eng.* 2014, pp. 1182–1185.
[40] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar, "Crowdsourced enumeration queries," in *IEEE 29th Int. Conf. Data Eng.*, 2013, pp. 673–684.

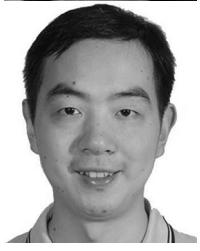
Yongxin Tong received the PhD degree in computer science and engineering from the Hong Kong University of Science and Technology, in 2014. He is currently an associate professor in the School of Computer Science and Engineering, Beihang University. His research interests include crowdsourcing, uncertain data mining and management, and social network analysis. He is a member of the IEEE.



Lei Chen received the BS degree in computer science and engineering from Tianjin University, China, in 1994, the MA degree from the Asian Institute of Technology, Thailand, in 1997, and the PhD degree in computer science from the University of Waterloo, Canada, in 2005. He is now an professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His current research interests include data-driven crowdsourcing, uncertain and probabilistic databases, web data management, and multimedia. Currently, he serves as an editor-in-chief for the *International Journal on Very Large Data Bases*, an associate editor-in-chief for the *IEEE Transaction on Data and Knowledge Engineering* and a trustee board member of VLDB Endowment.

Zimu Zhou received the BE degree from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2011 and the PhD degree in computer science and engineering from the Hong Kong University of Science and Technology. He is currently a post-doctoral researcher in the Computer Engineering and Networks Laboratory, ETH Zurich. His research interests include wireless networks, mobile computing, and crowdsourcing. He is a student member of the IEEE and ACM.

H. V. Jagadish received the PhD degree from Stanford University, in 1985. He is currently the Bernard A Galler Collegiate professor of electrical engineering and computer science at the University of Michigan. His research interests include databases and big data. He is a member of the IEEE.



Lidan Shou received the PhD degree in computer science from the National University of Singapore. He is a professor in the College of Computer Science, Zhejiang University, China. Prior to joining the faculty, he worked in the software industry for more than two years. His research interests include spatial database, data access methods, visual and multimedia databases, and web data mining.

Weifeng Lv received the PhD degree in computer science from Beihang University. He is a professor, the dean of the School of Computer Science and Engineering, and vice director of the State Key Laboratory of Software Development Environment. His research interests include large-scale software development methods and massive data oriented software support technology. He is the leader of the Group of National Smart City Standard.