SLADE: a Smart Large-Scale Task Decomposer in Crowdsouring Systems

Yongxin Tong† Lei Chen† Jieying She† Caleb Chen Cao† H. V. Jagadish‡ Lidan Shou*
†The Hong Kong University of Science and Technology ‡University of Michigan *Zhejiang University
†{yxtong,leichen,jshe,caochen}@cse.ust.hk ‡jag@umich.edu *should@zju.edu.cn

ABSTRACT

Crowd-sourcing has been shown to be effective in a wide range of applications, and is seeing increasing use. It is generally recognized that a complex large-scale task has to be decomposed into smaller HITs (Human Intelligence Tasks) before it can be crowd-sourced. This raises the question of how to perform this task decomposition best. It turns out that creating HITs that are too large results in poor answer quality while creating HITs that are too small incurs unnecessary cost. In this paper, we propose *SLADE*, a Smart Large-scAle task DEcomposer in crowdsourcing systems, to effectively decompose a large-scale task into a set of HITs to achieve a result that is both cost-effective and accurate.

Specifically, we define the *Effective Decomposition Problem (ED Problem)* and prove its NP-hardness. Then, we investigate this hard problem from two scenarios. In the first scenario, all tasks have the same quality threshold, and we propose two efficient and effective approximation algorithms using greedy strategy and optimal priority queue structure to find a near-optimal solution. In the second scenario, quality thresholds of different tasks are different, and we extend the approximation algorithms proposed for the first scenario by using partitioning strategy. Finally, we verify the effectiveness and efficiency of *SLADE* through extensive experiments on representative crowdsourcing platforms.

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 [19]". Early success stories include Wikipedia, Yelp and Yahoo! Answers. In recent years, with the widespread usage of Web 2.0 technologies, several general-purpose platforms, such as Amazon Mechanical Turks(AMT)[1] and oDesk[2], have made crowdsourcing techniques more powerful and manageable. Meanwhile, crowdsourcing has attracted much attention from the research communities due to its success in human intrinsic applications. Especially, a wide spectrum of fundamental data-driven operations are well studied, such as max[18, 35], join[25], filtering[14, 27], entity resolution[37], and so on. Besides the carefully drilled operations, researchers and practitioners also pave the way for building crowd-powered database systems, and a couple of prototypes have been successfully developed, such as CrowdDB[13], Deco[28, 29] and Qurk[24].

The ever-increasing volume and variety of crowdsourcing tasks lead to the challenge of solving a large number of large-scale tasks with limited crowd workforce. Therefore, a common practice in existing applications is to process the tasks in a batched manner[25, 13]. Specifically, a large-scale crowdsourcing task is divided into several batches, each of which represents a *Human Intelligence Task* (HIT). Then the number of such sub-tasks (a.k.a. atomic tasks) in one HIT is termed as its *cardinality*. Furthermore, most of the atomic tasks in the aforementioned data-driven crowdsourcing applications can be conducted in the form of binary choice, and the requesters of these tasks are sensitive to false-negative results, such as in the applications of filtering or finding operations. However, existing methods focus on the higher-level strategy to transform the original applications into human tasks, but seldom provide a sophisticated plan on how to execute these large-scale tasks with under-control accuracy and cost. To further illustrate it, we show two toy examples in the following, which come from real crowd-powered applications.

EXAMPLE 1. *(Fishing-Wire Discovery) Let us take a fishing-wire discovery task as an example. To fight against over-use and our-of-report large fishing-wire, a project has been published on the Tomnod website*[3]*, where a satellite image covering more than 2 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 'fish-wire' shape (as in Figures 1(a)-(c)) in the given piece of image", which is considered as an "atomic task". They can click the tag if they think so or slip to another image if otherwise. As in the example(Figure 1), four small pieces of images a_1, a_2, a_3, a_4 are given in an HIT, and this crowdsourcing task is to find which of the small pieces of images include a fish-wires shape. Here the cardinality of this task is four. Since the project owner cannot afford to miss any dubious image, they ask multiple participants to review one image and any piece with at least one "yes" will be further scrutinised. However, the task owner needs to decide what is the best execution plan to organize these images into HITs. One possible way is to release a_1 to a_4 to be processed only once but in separate HITs (10 cents in each HIT, 40 cents in total). Another way is to group a_1 and a_2 together and a_3 and a_4 together then ask the crowd to process each HIT for twice (12 cents in each HIT, 12*2*2=48cents in total). Which is better? Is there another better choice?*

[1]https://www.mturk.com/mturk/
[2]http://www.odesk.com/
[3]http://www.tomnod.com/

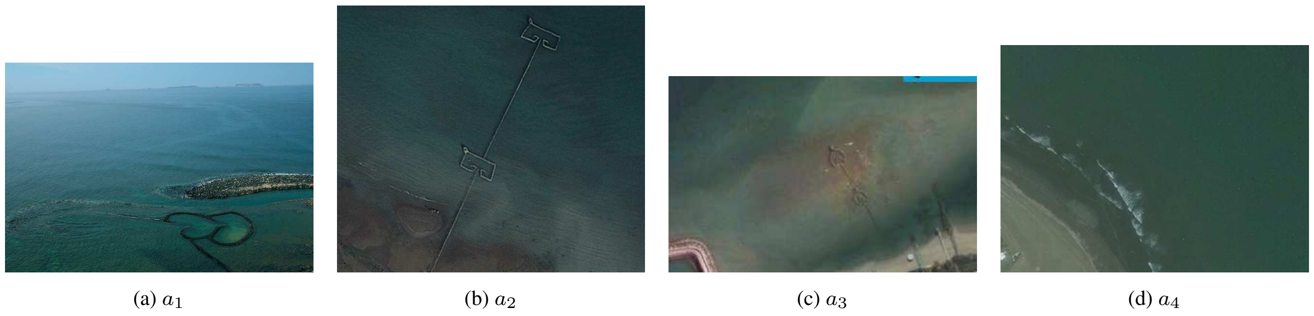

(a) a_1 (b) a_2 (c) a_3 (d) a_4

Figure 1: Fishing-Wire Discovery

EXAMPLE 2. *(Micro-Expressions Identification) The other example is from the real application of micro-expression identification. In certain campaign activities, micro-expressions are recorded and analyzed by the crowd to find participant with expression of anger or other specific intent. The crowd may receive certain basic training with targeted micro-expression and then tons of photos or videos are distributed to be screened. Please refer to Figure 4 (b) as an example, where the participants are expected to discover a targeted expression from the given batch (here the cardinality of the task is 4 since this task includes four micro-expression images). Similar to Example 1, the task requester also confronts the challenge of how to devise an optimal plan to release this task to participants.*

As discussed above, in this paper we will investigate the issue of how to provide an optimal execution plan to decompose a large-scale task into smaller HITs before it can be performed by crowds. Before introducing our solution, it is observed that the increase of *cardinality* impairs the accuracy of human answers in general due to the increase of human cognitive load[14]. On the other hand, HITs with higher rewards are more attractive. Also, there is a minimum task acceptance overhead, which makes tasks with low rewards unattractive. For these reasons, bundling atomic tasks into larger HITs will lead to lower cost and faster completion. Since our work depends crucially on these assumptions, we begin with a small pre-experiment in Section 2.

In order to solve the aforementioned challenges, we propose *SLADE*, a Smart Large-scAle task DEcomposer in crowdsourcing systems. We assume that the large-scale task can be expressed logically as a large-scale collection of atomic tasks. Each atomic task is performed by multiple workers in the crowd. SLADE converts this collection of atomic tasks into an optimal set of HITs to achieve the most cost-effective result, while satisfying a given quality (reliability) requirement. Accomplishing this is complicated by several factors. First, the smaller the HIT, the higher the confidence, and therefore the smaller the number of answers required from the crowd to meet a required reliability threshold. This complex dependence on error considerably complicates the simple tradeoff between quality and cost. Second, the atomic tasks (into which the large-scale task is logically decomposed) may not all be equal. Therefore, an HIT can include instances of an arbitrary subset of atomic tasks. Furthermore, the reliability thresholds of different atomic tasks could be different. In effect, SLADE is like a database query optimizer for a crowdsourcing system, determining a good execution plan given a logical expression to be evaluated.

As far as we know, this is the first work formally proposed to tackle the large-scale tasks decomposition problem. Several previous systems have been developed to generate HITs and issue them on crowdsourcing platforms. These include [13, 28, 29] or the "task dispatcher" in [14]. However, current solutions either set the *cardinality* of a HIT to a fixed value or use simple heuristics[14] to determine the value globally. None of them provides a comprehensive model to tackle such challenges, let alone an optimal solution. To sum up, we make the following contributions:

- We define the core problem in *SLADE* as *Effective Decomposition Problem (ED Problem)* to address the decomposition large-scale tasks using crowdsourcing and prove that it is NP-hard.
- We study two variants of the *Effective Decomposition Problem*. The first is called the *Homogeneous Decomposition Problem*, where all atomic tasks have the same reliability threshold. In order to solve it effectively, we propose a greedy approximation algorithm with the $\mathcal{O}(\ln n)$-approximation ratio and an faster optimal priority queue-based approximation algorithm, which has the same approximation ratio. Moreover, for the second variant where atomic tasks might have different reliability thresholds, we define the *Heterogeneous Decomposition Problem* and adapt the aforementioned approximation algorithms to solve this problem.
- *SLADE* has already been integrated into a real crowd-powered census database system. We verify the effectiveness and efficiency of the proposed methods through extensive experiments on real datasets.

The rest of the paper is organized as follows. After the preliminaries in Section 2, we introduce our problem formulation in Section 3 and analyze the complexity of the ED problem in Section 4. In Section 5, we present two approximation algorithms in the homogeneous scenario of the ED problem. Moreover, the two approximation algorithms are extended to solve the ED problem in heterogeneous scenario in Section 6. Experimental studies on both real and synthetic data sets are reported in Section 8. In Section 9, we review existing works and conclude this paper in Section 10.

2. PRELIMINARY EXPERIMENT

In this paper, we study the tradeoff between correctness and cost as a function of HIT size. In particular, our work is based on the fact that a larger *cardinality* leads to a lower confidence. To verify that this is indeed the case, we conducted a preliminary experimental study on Amazon Mechanical Turk (AMT), and demonstrate the results in Figure 2.

We adopt the classical jelly-beans-in-a-jar setting: given a sample image containing 200 dots, the worker is required to determine whether another image contains more dots or not. Each image is a sub-task in the form of binary choice, whose answer is independent of each other. We then specify the *cardinality* of a HIT in the range of 2 to 30 by aligning the target images along the question webpage. For each HIT, 10 assignments are issued to smooth the

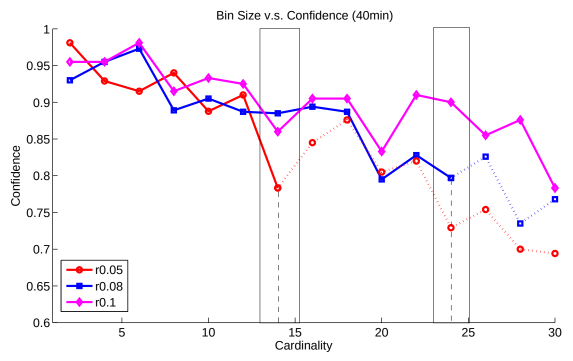


Figure 2: Cardinality vs. Cost vs. Confidence

randomness from the crowds, and three different reward levels are tested - \$0.05, \$0.08 and \$0.1. As is typical in such scenarios, we set a response time threshold, after which the HIT is considered as too slow to be of practical use. In our experiment, we used 40 minutes as the threshold. For each *cardinality* and price, the figure shows the confidences. Overtime HITs are shown in dotted lines, while timely HITs are in sold lines.

In Figure 2, it is easily noticed that the confidence declines with increasing cardinality. Then after cardinality 14 (resp. 24), the HITs with reward \$0.05 (\$0.08) are disqualified since not enough answers were obtained within the allowed 40-minute period. As the cardinality goes from 2 to 30, the valid(i.e. in-time) confidence decreases from 0.981 to 0.783, and the average cost for a sub-task decreases from \$0.025($= 0.05/2$*) to \$0.003(*$= 0.1/30$*). Therefore, in general the increase of cardinality leads to a lower confidence but cheaper average price.*

Observation: for each cardinality*, the dot from the solid line with lowest price represents* ***a most cost-effective HIT design****.*

3. PROBLEM STATEMENT

In this section, we first introduce several preliminary concepts of large-scale tasks on crowdsourcing platforms. Then, we formally define *the Effective Decomposition Problem (ED Problem)* and discuss its complexity.

3.1 Preliminaries

We first define atomic tasks on crowdsourcing platforms. An ***atomic task***, denoted by a_i, is defined as a binary choice problem. Furthermore, let T be a ***large-scale crowdsourcing task***, which consists of n independent atomic tasks, where n is called the *cardinality* of this task. For example, in the scenario of fishing-wire discovery, an atomic task is to decide whether there is a fish-wire shape in the given piece of image. This large-scale task includes more than 100,000 small pieces of satellite images, each of which needs to be checked. Furthermore, the decisions regarding different images do not influence each other. Notice that, on most popular crowdsourcing platforms such as AMT and oDesk, different crowdsourced tasks are handled independently. Thus, we also assume that all atomic tasks are independent of each other.

As introduced in Section 1, it is impossible for a single crowd worker to handle the whole large-scale task. To solve such a large-scale task through crowdsourcing, a natural idea is to decompose the large-scale task into simple ones, which are called *task bins* in this work. We define the concept of an l-cardinality task bin as follows.

Table 1: A Set including 3 Task Bins

Cardinality	1	2	3
Confidence	0.9	0.85	0.8
Incentive Cost (USD)	0.1	0.18	0.24

DEFINITION 1 (l-CARDINALITY TASK BIN). *An l-cardinality task bin is a triple, denoted as $b_l =< l, r_l, c_l >$, where (1) l is the number of atomic tasks included in the bin, namely the cardinality of this task bin; (2) r_l is the confidence, which indicates the probability that the crowds can correctly perform each atomic task in this task bin; (3) c_l is the incentive cost given to the crowds when all atomic tasks are performed in this task bin.*

Notice that an instance of an l-cardinality task bin is an instance of a combination of l different atomic tasks. In other words, one l-cardinality task bin can generate different l-cardinality task bin combinations of atomic tasks, and each combination can generate different instances. Table 1 shows a toy example of task bins, $\{b_1, b_2, b_3\}$, where the i-th column corresponds to the i-cardinality task bin. For example, the second column represents the 2-cardinality task bin b_2, where the confidence $r_2 = 0.85$ and the incentive cost $c_2 = 0.18$. In addition, given a task, $T = \{a_1, \cdots, a_5\}$, there are $\binom{5}{2}$ 2-cardinality task bin combinations, and each combination can generate an arbitrary number of instances. As discussed in Section 2, the average incentive cost and the confidence of an atomic task go down when the cardinality increases. For example, the average incentive costs of the aforementioned three task bins are 0.1, 0.09 and 0.08, respectively. Moreover, their confidences for the atomic tasks are 0.9, 0.85, and 0.8, respectively.

Furthermore, all the values of the parameters related to the task bins can be obtained by learning approaches from the popular crowdsourcing markets, such as AMT or oDesk. More details of the learning methods will be discussed in Section 7. According to the trained task bins, we can estimate the effectiveness of decomposing a large-scale task to simpler task bin instances. In fact, a task bin instance will be released as a specific task in real crowdsourcing markets. In other words, each task bin instance will be addressed by a crowd worker in real crowdsourcing markets. Since real crowd workers may be unreliable, each atomic task should be performed multiple times to guarantee the result quality with high confidence. Moreover, as discussed in Section 1, many real crowdsourcing applications require very low false negative ratio in their crowdsourcing results. Back to Example 1, in the case of discovering fishing-wires given small pieces of satellite images, the opportunity of discovering fishing-wires may be missed out due to false negatives. Therefore, we define the probability that there is no false negative result as the reliability of an atomic task, and use the confidences of the assigned task bin instances to estimate the reliability of an atomic task.

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

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

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

Intuitively, Eq. (1) represents the estimated possibility that the atomic task a_i can be accomplished by at least one assigned task bin instance.

Table 2 summarizes the commonly used notations in this paper.

Table 2: Summary of Symbol Notations

Notation	Description
a_i	an atomic task
$T = \{a_1, \cdots, a_{\lvert T\rvert}\}$	a large-scale crowdsourcing task
b_l	an l-cardinality task bin
$B = \{b_1, \cdots, b_{\lvert B\rvert}\}$	the set of task bins
$n = \lvert T\rvert$	the number of atomic tasks in T
$m = \lvert B\rvert$	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 instance
$\mathfrak{B}(a_i)$	the set of assigned task bin instances 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 per atomic task a_i
DP_T	optimal decomposition plan of T

3.2 Effective Decomposition Problem

According to the definitions of task bins and the reliability of each atomic task, we define the *Effective Decomposition Problem (ED Problem)* as follows.

DEFINITION 3 (EFFECTIVE DECOMPOSITION (ED) PROBLEM). *Given a large-scale crowdsourcing task T including n atomic tasks $\{a_1, \dots, a_n\}$, each a_i of which with reliability threshold t_i, m task bins $\{b_1, \dots, b_m\}$, the Effective Decomposition Problem is to find a planning $DP_T = \cup\mathfrak{B}(a_i)$ among the large-scale crowdsourcing task T and the task bins to minimize the total cost, $\sum_{\beta\in\cup\mathfrak{B}(a_i),\forall a_i} c_{|\beta|}$, such that $Rel(a_i, \mathfrak{B}(a_i)) \geq t_i$, $\forall a_i \in T$.*

In particular, if all the t_i's are the same, the large-scale task T is ***homogeneous***, otherwise T is ***heterogeneous***. Moreover, each atomic task can also be assigned to different instances of an l-cardinality task bin in a decomposition plan. We illustrate the ED problem via the following example.

EXAMPLE 3. *(Effective Decomposition Problem) Given a crowdsourcing task $T = \{a_1, a_2, a_3, a_4\}$ in Figure 1, $B = \{b_1, b_2, b_3\}$, in Table 1, and all reliability thresholds of all atomic task are equal $t_i = 0.95 (1 \leq i \leq 4)$, (which means that T is homogeneous task), the optimal decomposition plan performs $\{a_1, a_2, a_3\}$, $\{a_1, a_2, a_4\}$ and $\{a_3, a_4\}$ for once respectively, which has decomposition cost of 0.66. Hence, there are 3 task bin instances. Moreover, the reliability of a_1 and a_2 in this optimal decomposition plan equals to $1 - (1 - 0.8) \times (1 - 0.8) = 0.96 > 0.95$ and the reliability of a_3 and a_4 is $1 - (1 - 0.85) \times (1 - 0.8) = 0.97 > 0.95$. Therefore, each atomic task satisfies the given reliability threshold in this decomposition plan.*

4. PROBLEM REDUCTION

In this section, in order to efficiently solve the ED problem, we first reduce the reliability constraint of the ED problem to an equivalent simple form. Then, we prove the NP-hardness of the ED problem. We also show that there exist polynomial-time solutions to a relaxed variant of the ED problem. Finally, we present a baseline algorithm for the general case of the ED problem.

4.1 Reduction of Reliability

Based on Definition 2, we equivalently rewrite the Eq. (1) as:

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

In Definition 2, our constraint of the ED problem is that the reliability of each atomic task satisfies their given reliability threshold t_i, namely $Rel(a_i, \mathfrak{B}(a_i)) \geq t_i$. According to the Eq. (2), the aforementioned constraint is equivalent to $-ln(1-Rel(a_i, \mathfrak{B}(a_i)) \geq -ln(1 - t_i)$, namely $\sum_{\forall\beta\in\mathfrak{B}(a_i)} -ln(1 - r_{|\beta|}) \geq -ln(1 - t_i)$, for an atomic task a_i. Therefore, for an arbitrary atomic task a_i, we associate each task bin instance, which is assigned to a_i, with a positive constant $-ln(1 - r_{|\beta|})$ where $r_{|\beta|}$ is the corresponding confidence of this task bin instance. Then, the reliability of an atomic task is transformed to a sum of $\sum_{\forall\beta\in\mathfrak{B}(a_i)} -ln(1 - r_{|\beta|})$.

4.2 Complexity Results

In this subsection, we first show the NP-hardness of the ED problem. Then, we also analyze that there exist polynomial-time solutions to a relaxed variant of the ED problem.

THEOREM 1. *The Effective Decomposition problem (ED problem) is NP-Hard.*

PROOF. We reduce the KNAPSACK problem, which is a well-known NP-complete problem [15], to the Effective Decomposition problem.

An instance of KNAPSACK is: given a set of m items with weights $\{w_l\}$ and values $\{v_l\}$, we want to decide whether there exists a subset of items S with total weight $\leq W$, so that the total value is equal to a given value V.

We construct an instance of the ED problem from the instance of KNAPSACK, accordingly:

(1) For m items, there are m task bins $B = \{b_1, \dots, b_m\}$, and each item corresponds to a task bin.

(2) For each task bin b_l, let $c_l = v_l$ and $r_l = 1 - e^{-\frac{w_l}{\sum_{l=1}^m w_l}}$ since $-ln(1 - r_l) = \frac{w_l}{\sum_{l=1}^m w_l}$.

(3) For an arbitrary atomic task a_i, the reliability threshold t_i satisfies $-ln(1 - t_i) = \frac{\sum_{l=1}^m w_l - W}{\sum_{l=1}^m w_l}$.

Construct a special case of the ED problem where each task bin can be used at most once and the given task only includes an atomic task a_i. To complete the proof, we prove that, the cost of the decomposition plan in the special case is just equal to $\sum_{l=1}^m v_l - V$ subject to $\sum_{l\in B-S} \frac{w_l}{\sum_{l=1}^m w_l} \geq \frac{\sum_{l=1}^m w_l - W}{\sum_{l=1}^m w_l}$ if and only if there exists a subset S ($|S| \leq W$) for KNAPSACK such that the total value is equals to V.

In addition, we prove that the general case of the ED problem is harder than the aforementioned special case. First, in the general case, the ED problem includes n atomic tasks $T = \{a_1, a_2, \cdots, a_n\}$, where the number of atomic tasks is larger or equal to that in the special case. Second, each task bin can be used multiple times in the general case. In other words, the number of possible task bin combinations in the general case is larger than that in the special case. To sum up, the special case of the ED problem can be reduced from the instance of KNAPSACK, and the general case of the ED problem is harder than its special case, hence the ED problem is NP-hard. □

Complexity Analysis of A Relaxed Variant of The ED Problem. Although the ED problem is NP-Hard, there is a relaxed variant (or called special case), 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 $1 \leq j \leq |B|$, and t_{max} is the maximum t_i ($1 \leq i \leq |T|$). In other words, in this relaxed variant of the ED problem, each atomic task satisfies its reliability threshold requirement no matter which task bin instance it is assigned to. In this case, the ED problem is simplified to the ROD CUTTING problem [7], which has an efficient dynamic programming exact solution with $\mathcal{O}(|T||B|)$ time complexity, where $|T|$ and $|B|$ are the number of atomic tasks in the large-scale task and that of distinct task bins, respectively.

4.3 Baseline Algorithm

In this subsection, we present a baseline algorithm for the general case of the ED problem.

A Baseline Algorithm. When there is at least one task bin which fails to satisfy the reliability threshold, the ED problem is NP-hard. We incorporate a column generation technique[4] into a randomized rounding algorithm[32] as baseline to solve the general case (called the hybrid randomized rounding algorithm). Note that the randomized rounding algorithm is an existing effective solution to *covering integer programming*[34], to which the integer programming form of the ED problem is equivalent. The main idea of the baseline algorithm consists of the following two steps. First, we utilize the column generation technique to solve the linear programming (LP) relaxation of the integer programming form of ED and obtain an LP optimal solution. Then, in the second step, we perform the randomized rounding algorithm based on the LP optimal solution to obtain the decomposition plan and cost. This algorithm can obtain $\ln n$-approximation ratio with the $1-\delta$ probability, where δ is a probabilistic error, and its time complexity depends on the heuristic strategy of the column generation. More details of the baseline algorithm can be found in Appendix.

5. HOMOGENEOUS DECOMPOSITION

In this section, we study the ED Problem in the homogeneous scenario, where all reliability thresholds $t_i (1 \le i \le |T|)$ **are equal** for the atomic tasks in a large-scale task T. Thus, all reliability thresholds are simplified as t in the rest of this section.

To solve this problem, we present two approximation algorithms. In Section 5.1, we design an *greedy-based algorithm*, which has $\mathcal{O}(\omega n^2 \log n)$ time complexity with a $\ln n$ approximation ratio, in which $\omega = \lceil \frac{ln(1-t)}{ln(1-r_m)} \rceil$, where t is the reliability threshold, r_m is the confidence of the task bin with the maximum cardinality m. In Section 5.2, we propose an optimal priority queue-based approximation algorithm, which is faster than the greedy-based algorithm. In particular, in some cases, the optimal priority queue-based approximation algorithm can get the exact optimal solutions.

5.1 A Greedy-Based Algorithm

Although the baseline algorithm utilizes the hybrid randomized rounding algorithm to solve the ED problem in polynomial time, there still exist two challenges. The first challenge is efficiency. The running time of the hybrid randomized rounding algorithm is inefficient since its heuristic strategy adopted in the column generation phase depends on the solution structure of the corresponding linear programming relaxation. The second challenge is approximation quality. Though the hybrid randomized rounding algorithm can return a $\ln n$ approximation ratio with a high probability, in fact, as shown in our experimental evaluation (Section 8), it usually results in larger decomposition cost than our proposed two algorithms do. Thus, to address the two challenges, we propose a more efficient greedy-based approximation algorithm also with a $\ln n$ approximation ratio.

The main idea of our greedy-based algorithm is to make locally optimal choice in each iteration. In particular, to efficiently find the local optimal choice in each iteration, we design a ranking schema, which can guarantee to obtain the same local optimal

```
Algorithm 1: A Greedy-Based Algorithm
   Input: A large-scale task $T = \{a_1, \dots, a_{|T|}\}$, a set of task
          bins $B = \{b_1, \dots, b_{|B|}\}$, a reliability threshold $t$
   Output: An approximate decomposition plan, $DP_T$, and an
           approximate decomposition cost, $Cost_T$
1  $DP_T \leftarrow \emptyset$;
2  Initialize $\theta = \{\theta_1, \cdots, \theta_n\}$, where each $\theta_i \leftarrow -\ln(1-t)$;
3  Rank $T = \{a_{i_1}, \cdots, a_{i_{|T|}}\}$ in non-ascending order of $\theta_i$;
4  while $\theta_{i_1} > 0$ do
5      $j^* \leftarrow argmin_{j \in B} \frac{c_j}{min\{j \times (-\ln(1-r_j)), \sum_{k=1}^{j} \theta_{i_k}\}}$;
6      $DP_T \leftarrow DP_T \cup \{\{a_{i_1}, \cdots, a_{i_{j^*}}\}\}$;
7      $Cost_T \leftarrow Cost_T + c_{j^*}$;
8      for $k \leftarrow 1$ to $j^*$ do
9          $\theta_{i_k} \leftarrow \theta_{i_k} - (-\ln(1 - r_{j^*}))$;
10     Rank $T = \{a_{i_1}, \cdots, a_{i_{|T|}}\}$ in non-ascending order of $\theta_i$;
11 return $DP_T$ and $Cost_T$
```

choice by only checking m task bin instances instead of enumerating all $\sum_{l=1}^{m} \binom{n}{l}$ task bin instances, where n is the number of atomic tasks in the large-scale task, $m = |B|$, which is the size of the set of task bins B. The correctness of ranking schema is discussed in Lemma 1.

The pseudo code of the greedy-based algorithm is shown in Algorithm 1. Line 2 initializes the residual θ of each atomic task to $-\ln(1-t)$. Then, it ranks n atomic tasks in terms of their 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 instance with the minimum $\frac{c_j}{min\{j \times (-\ln(1-r_j)), \sum_{k=1}^{j} \theta_{i_k}\}}$. After choosing the locally optimal task bin b_{j^*}, the algorithm allocates the first j^* ranked atomic tasks in T into the final decomposition plan and adds c_{j^*} into the final decomposition cost in lines 6 and 7, respectively. Then, the residuals of the first j^* ranked atomic tasks are reduce by $-\ln(1 - r_{j^*})$ each in lines 8-9. After the subtraction step, the algorithm re-ranks all atomic tasks in T in non-ascending order of the residual values in line 10. Finally, this algorithm stops when all atomic tasks satisfy the reliability threshold requirement.

EXAMPLE 4 (GREEDY-BASED ALGORITHM). *Back to our running example, given the task having 4 atomic tasks, the set of task bins in Table 1, and the reliability threshold $t = 0.95$, the execution process of Algorithm 1 is shown as follows. This algorithm first initializes each $\theta_i = 2.996$ where $1 \le i \le 4$. Since all θ_i's are the same, it keeps the original order of atomic tasks as <$a_1, \cdots, a_4$>. Then, this algorithm selects the task bin instance {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 this algorithm re-ranks T to <$a_2(2.996)$, $a_3(2.996)$, $a_4(2.996)$, $a_5(2.996)$, $a_1(0.693)$>, where the value in each bracket is the current residual of the corresponding atomic task. The algorithm continues to perform similar iterations until it stops. The final decomposition plan is: $\{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}, \{a_1, a_2, a_3\}, \{a_4\}$, and the final decomposition cost is 0.74.*

In terms of Algorithm 1, the ranking order determines the choice of j^* in each iteration. Lemma 1 guarantees that the locally optimal choice generated from the m task bin instances corresponding to the first m ranked atomic tasks is equivalent to that from the $\sum_{l=1}^{m} \binom{n}{l}$ task bin instances. Especially, in practice, $m \ll n$.

Table 3: The optimal priority queue 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

LEMMA 1. *The local optimal choice from the m task bin instances in line 5 of Algorithm 1 is equivalent to that from all $\sum_{l=1}^{m}\binom{n}{l}$ task bin instances.*

PROOF. According to line 5 of Algorithm 1, there are two cases when finding the local optimal choice. The first case is that each residual is greater than $-ln(1-r_1)$, where r_1 is the largest confidence of all task bins. In this case, the local optimal choice is equivalent to the j^*-cardinality task bin subject to minimizing $\frac{c_j}{-\ln(1-r_j)}$, which is not affected by the combinations of different atomic tasks. The second case is that at least one residual is lower than $-\ln(1-r_j)$. The local optimal choice should have the minimum $\frac{c_j}{\sum_{i=1}^{j}\theta_i}$, namely the maximum j residuals. The ranking order just provides the maximum j residuals. To sum up, the lemma holds. □

In addition, the approximation ratio of Algorithm 1 is $\ln n$ and is proven in the following theorem.

THEOREM 2. *The approximation ratio of Algorithm 1 is $\ln n$.*

PROOF. According to Lemma 1, we know that the local optimal choice from m task bin instances in the current first m atomic tasks is equivalent to that from $\sum_{l=1}^{m}\binom{n}{l}$ task bin instances, which is identical with the local optimal choice of the classical greedy solution of covering integer programming [10] with the approximation ratio $\ln n$. Hence, Algorithm 1 has the approximation ratio $\ln n$ as well. In other words, $\sum_{i=1}^{n} w_j \leq OPT \cdot \sum_{i=1}^{n}\frac{1}{i} = \ln n \cdot OPT$ if OPT is the optimal decomposition cost. □

Computational Complexity Analysis: According to the property of the set of task bins, the task bin with the largest cardinality has the smallest confidence. Thus, given an arbitrary atomic task, the upper bound on the number of iterations in Algorithm 1 is $n\omega = n\lceil\frac{\ln(1-t)}{\ln(1-r_m)}\rceil$, where t is the reliability threshold, r_m is the confidence of the m-cardinality task bin, and m is the size of the set of task bins. Moreover, in each iteration, this algorithm needs to rank all atomic tasks w.r.t. their current residuals. In other words, it spends $\mathcal{O}(n\log n)$ time to rank in each iteration. Hence, the total computational complexity of Algorithm 1 is $\mathcal{O}(n\omega(m+n\log n)) = \mathcal{O}(\omega n^2\log n)$ since $m \ll n$ in practice.

5.2 An Optimal-Priority-Queue-based Algorithm

To further enhance the efficiency of the greedy-based algorithm, we propose a new approximation algorithm based on a specific data structure, called the optimal priority queue. The new approximation algorithm not only return better decomposition plans in practice but also has smaller time complexity. In particular, by means of the optimal priority queue, we can get the exact optimal solution in some cases. In this subsection, we first introduce how to construct the optimal priority queue in the pre-processing phase and provide two pruning methods to reduce the redundant search space. Then, based on the optimal priority queue, we devise an faster approximation algorithm with the $\ln n$ approximation ratio as well.

5.2.1 Constructing the Optimal Priority Queue

In this subsection, we first define the optimal priority queue and then introduce how to build this structure. First, we define two basic concepts, the unit cost of an atomic task and lowest common multiple in a combination of task bins. Let a combination of task bins be denoted by $Comb = \{n_{k_1} \times b_{k_1}, \cdots, n_{k_l} \times b_{k_l}\}$, where $n_{k_i} \times b_{k_i}$ means that the k_i-cardinality task bin is repeatedly used n_{k_i} times for an atomic task. The *unit cost* of $Comb$ is $UC = \sum_{i=1}^{l}\frac{c_{k_i}}{k_i}n_{k_i}$. Moreover, the *lowest common multiple*, denoted as LCM, of $Comb$ is $k_1 \times \cdots \times k_l$. For example, given a type of combination of task bins in Table 1 $Comb = \{3 \times b_1, 2 \times b_2, 1 \times b_3\}$, $UC = 3 \times 0.1 + 2 \times \frac{0.18}{2} + 1 \times \frac{0.24}{3} = 0.56$, and $LCM = 1 \times 2 \times 3 = 6$. According to the two concepts, we define the optimal priority queue.

DEFINITION 4 (OPTIMAL PRIORITY QUEUE). *Given a set of task bins, $B = \{b_1, \cdots, b_m\}$ and an reliability threshold t, the optimal priority queue OPQ is a priority queue consisting of combinations of task bins and satisfies the following conditions: (1) the elements in the optimal priority queue is ranked in descending order of their corresponding LCM values; (2) for any element OPQ_i (with LCM_i and UC_i) in the optimal priority queue, there does NOT exist another element OPQ_j (with LCM_j and UC_j) such that $LCM_i \geq LCM_j$ and $UC_i \geq UC_j$; (3) each combination in this optimal priority queue satisfies the reliability threshold requirement.*

EXAMPLE 5. *(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 when the reliability threshold is 0.95. In Table 3, each column corresponds to a combination of task bins. For example, for the first column $\{2 \times b_3\}$, $UC = 2 \times \frac{0.24}{3} = 0.16$ and $LCM = 3$. In addition, if an atomic task is assigned into each task bin instance of the $Comb$ in the first column, its quality 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, an atomic task satisfies the reliability threshold no matter which task bin instances of the $Comb$ in Table 1 it is assigned into. Moreover, Table 1 records the elements of OPQ in descending order of LCM.*

Algorithm 2: Building Optimal Priority Queue Algorithm

```
Input: A set of task bins $B = \{b_1, \ldots, b_{|B|}\}$, 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($i, q, S, B, t$)
5  for $k \leftarrow i$ to $|B|$ 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 $S$;
```

To obtain the optimal priority queue, we design a depth-first-search-based algorithm (Algorithm 2) to enumerate all possible combinations of task bins and check which combinations need to be stored in the optimal priority queue. This algorithm performs the depth-first-search enumeration starting 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 firstly check whether the new combination cannot be pruned by Lemma 2 in line 7 and satisfies the reliability threshold requirement in line 8. If so, algorithm inserts the current combination into the optimal priority queue, otherwise it continues to enumerate until it 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 2. *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. First, according to the definition of optimal priority queue, $Comb_2$ does not appear in the optimal priority queue if $Comb_1.UC < Comb_2.UC$ and $Comb_1.LCM \leq Comb_2.LCM$. Furthermore, due to $Comb_2.UC < Comb_2^s.UC$ and $Comb_2.LCM \leq Comb_2^s.LCM$, where $Comb_2^s$ is an arbitrary superset of $Comb_2$, all combinations that are supersets of $Comb_2$ do not exist in the optimal priority queue as well. Hence, the lemma is correct. □

EXAMPLE 6. *(Construction of the 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, this 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 the $\{b_1 + b_2\}$, which is updated as OPQ_1 because $-\ln(1-0.9) - \ln(1-0.85) = 4.20 > 2.996$, 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 since $2 \times (-\ln(1-0.85)) = 3.794 > 2.996$, its $LCM = 2$ and its $UC = 0.18 < 0.19$. The final OPQ is shown in Table 3.*

5.2.2 Optimal-Priority-Queue-based Algorithm

Based on the optimal priority queue, we propose an enhanced approximation algorithm, called the optimal-priority-queue-based algorithm (OPQ Algorithm for short) in this subsection. The main idea of the OPQ Algorithm is to repeatedly utilize the optimal combinations in the optimal priority queue to approximate the global optimal solution as much as possible. If the number of atomic tasks in a large-scale task can be exactly divided by the LCM of the first element whose LCM is not greater than n in the queue, the decomposition plan is globally optimal according to Lemma 3. Otherwise, we prove that the enhanced approximation algorithm still has the $\ln n$ approximation guarantee.

LEMMA 3. *Given the number of atomic tasks n and the optimal combination of task bins $Comb^*$, i.e. the first element in the optimal priority queue whose $LCM \leq n$, with lowest common multiple LCM^* and unit cost UC^*, the global optimal cost is $n \times UC^*$ if n can be divided by LCM^* exactly.*

PROOF. Since $Comb^*$ is the optimal combination of task bins, the global cost for each atomic task is UC^*. In addition, n can be divided by LCM^*. Thus, the total optimal decomposition cost is $n \times UC^*$. □

The pseudo code of the OPQ Algorithm is shown in Algorithm 3. Lines 1 initializes the optimal priority queue using Algorithm 2. Then, the algorithm iteratively makes assignment using the combinations of task bins in OPQ in lines 2-11. Particularly, the algorithm uses the first element OPQ_1 in OPQ, whose values of

Algorithm 3: Optimal-Priority-Queue-based Algorithm

```
   Input: A large-scale task $T = \{a_1, \ldots, a_{|T|}\}$, a set of task
          bins $B = \{b_1, \ldots, b_{|B|}\}$, 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  while $n > 0$ do
3      while $OPQ_1.LCM > n$ do
4          Remove $OPQ_1$ from $OPQ$;
5      $LCM^* \leftarrow OPQ_1.LCM$;
6      $UC^* \leftarrow OPQ_1.UC$;
7      $k \leftarrow \lfloor \frac{n}{LCM^*} \rfloor$;
8      $DP_T \leftarrow$ Assignment($T, OPQ_1, k \times LCM^*$);
9      $Cost_T \leftarrow k \times LCM^* \times UC^*$;
10     Remove the first $k \times LCM^*$ atomic tasks from $T$;
11     $n \leftarrow n \mod LCM^*$;
12 return $DP_T$ and $Cost_T$
```

LCM and UC are stored in LCM^* and UC^* respectively, to make assignment for the first $\lfloor \frac{n}{LCM^*} \rfloor \times LCM^*$ atomic tasks in T in lines 3-9 in each iteration. The remaining $n \mod LCM^*$ atomic tasks are precessed in subsequent iterations.

We explain Algorithm 3 in the following example.

EXAMPLE 7. *(Optimal-Priority-Queue-based Algorithm) Back to our running example, given a set of task bins in Table 1, a reliability threshold $t = 0.95$, the algorithm first finds and loads the optimal priority queue, which is shown in Table 3. Then in the first iteration, the algorithm uses $OPQ_1 = \{2 \times b_3\}$ to make assignment for $\{a_1, a_2, a_3\}$. In the second iteration, the algorithm uses $\{2 \times b_1\}$ to make assignment for the remaining a_4. To sum up, the final decomposition plan is: $2 \times \{a_1, a_2, a_3\}$ and $2 \times \{a_4\}$. The decomposition cost is $1 \times 3 \times 0.16 + 1 \times 1 \times 0.2 = 0.68$. We observe that the decomposition cost is smaller than that of the greedy-based algorithm, which is 0.76.*

In addition, the approximation ratio of Algorithm 3 (the optimal-priority-queue-based algorithm) is shown in the following theorem.

THEOREM 3. *The approximation ratio of Algorithm 3 is* $\ln n$.

PROOF. According to Algorithm 2, we can get the optimal combinations of task bins. Thus, the maximum lowest common multiple of the optimal combinations can be upper bounded by $\frac{ln(1-t)}{ln(1-r_m)}$, where t is the reliability threshold, r_m is the minimum confidene in the given task bins. Let OPT_{LP} denote the optimal solution of the corresponding linear programming for our integer programming. According to Algorithm 3, the worst case of approximation quality is the case $n = m \cdot \omega - 1$ where $\omega = \lceil \frac{ln(1-t)}{ln(1-r_m)} \rceil$. In that case, $OPT_{frac} = \frac{(m\omega-1)c_l}{l}$, which is the fractional optimal solution based on the optimal combinations of task bins and is lower than the optimal integer solution. Moreover, the worst cost of Algorithm 3 is $Cost_w = (m\omega - 1)c_1$. Thereby, the worst approximation ratio between OPT_{frac} and Algorithm 3 is $\frac{Cost_w}{OPT_{frac}}$=$\ln n$. Because the OPT_{frac} is the lower bound of the corresponding integer programming, the approximation ratio also holds in the integer programming. □

Computational Complexity Analysis: According to Algorithm 3, the time complexity of this algorithm is $\mathcal{O}(\alpha(\lfloor \frac{n}{OPQ_1.LCM} \rfloor + 1))$, where α is cost to make assignment for $OPQ_1.LCM$ atomic tasks using the task bin instances in OPQ_1, which is small in practice.

Algorithm 4: Adapted Optimal-Priority-Queue-based Algorithm

```
Input: A large-scale task $T = \{a_1, \ldots, a_{|T|}\}$, a set of task
       bins $B = \{b_1, \ldots, b_{|B|}\}$, reliability thresholds
       $\{t_1, \cdots, t_{|T|}\}$
Output: An approximate decomposition plan, $DP_T$, and an
        approximate decomposition cost, $Cost_T$
1 Rank $T$ in non-ascending order of $t_i$, $\{t_1^{new}, \cdots, t_{|T|}^{new}\}$;
2 while $T$ is not $\emptyset$ do
3     Find the optimal priority queue $OPQ$ using $t = t_1^{new}$ of
      the new first ranked atomic task in $T$;
4     $OPQ^* \leftarrow$
      $\arg\min_{\{OPQ_i \in OPQ | OPQ_i.LCM \le |T|\}} OPQ_i.UC$;
5     $DP_T \leftarrow$ Assignment($T$, $OPQ^*$, $OPQ^*.LCM$);
6     $Cost_T \leftarrow OPQ^*.LCM \times OPQ^*.UC$;
7     Remove the first $OPQ^*.LCM$ atomic tasks from $T$ and
      keep the order of the remaining tasks in $T$;
8 return $DP_T$ and $Cost_T$
```

6. HETEROGENEOUS DECOMPOSITION

In this section, we study the ED Problem in the heterogeneous scenario, where the atomic tasks in a large-scale task may have different reliability thresholds. In the following, we will introduce how to adapt our proposed approximation algorithms in the homogeneous scenario to solving the heterogeneous scenario.

First, the following lemma shows that the greedy-based algorithm (Algorithm 1) still works by changing the reliability thresholds of atomic tasks only.

LEMMA 4. *Algorithms 1 still works in the heterogeneous scenario by changing the reliability thresholds t_i's of different atomic tasks, respectively.*

PROOF. In terms of the integer programming form in Definition 3, different atomic tasks a_i might have different reliability thresholds t_i in the heterogeneous scenario. In other words, each atomic task may have a different reliability threshold in Definition 3. For Algorithm 1, different reliability thresholds t_i only affect the original residuals θ_i for each atomic task in line 2 in Algorithm 1. Thus, the algorithm still works in the heterogeneous scenario. ☐

Moreover, the optimal priority queue-based algorithm (Algorithm 3) can be also extended to the heterogeneous scenario by the following method. The main idea is to partition the whole set of atomic tasks into groups and run Algorithm 3 for each group. More specifically, we first rank the atomic tasks in non-ascending order of their reliability thresholds. Then for the first-ranked atomic task, we run Algorithm 3 to build an optimal priority queue based on its reliability threshold. We next select the element OPQ_i in the queue with the least UC. Let its LCM be LCM^* and its UC be UC^*. We then make assignment for the first LCM^* ranked atomic tasks using OPQ_i and remove these atomic tasks from T after such assignment. This process is repeated for the remaining atomic tasks.

The adapted optimal priority queue-based algorithm is presented in Algorithm 4. In line 1, we first rank the atomic tasks in non-ascending order of their reliability thresholds. Then in lines 2-7, we iteratively make assignment based on the optimal priority queue-based algorithm. In line 3, we obtain the optimal priority queue using the threshold of the first ranked atomic task in the updated T, which changes at each new iteration. We find the element with the least UC in line 4. We then make assignment for the first $OPQ^*.LCM$ ranked atomic tasks in T in line 5 and update $Cost_T$ accordingly in line 6. In line 7, we remove the first $OPQ^*.LCM$ atomic tasks from T, which have already been assigned, and keep the order of the remaining atomic tasks unchanged. Notice that the first ranked atomic task in T has been updated after this line.

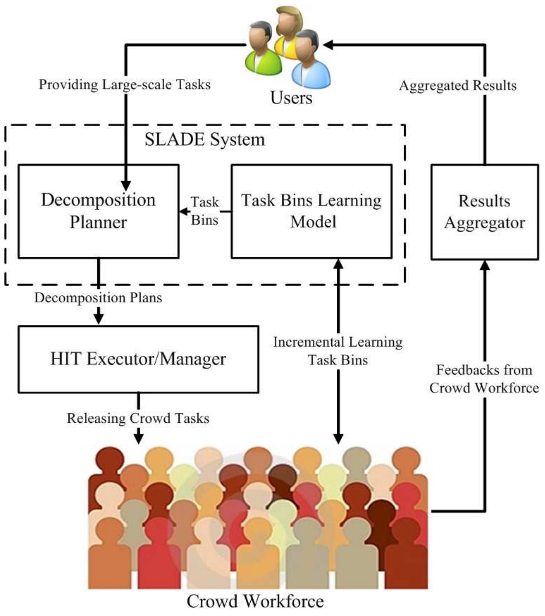


Figure 3: Architecture of SLADE Module

Computational Complexity Analysis: In Algorithm 4, there are at most $|T|$ iterations. Therefore, the time complexity of this algorithm is $\mathcal{O}(|T|(\log|T| + \alpha + \gamma))$, where α is the cost to find the optimal priority queue and γ is the cost to make assignment for $OPQ^*.LCM$ atomic tasks using the task bin instances in OPQ^*, which are both small since $|B|$ is relatively small in practice.

Besides the heterogeneous scenario, the difficulty of the tasks is another factor that may influence the performance of the crowds. We discuss more details in Section 8.5.

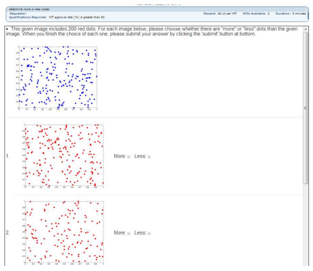
7. SYSTEM IMPLEMENTATION

After introducing various solutions for SLADE, we are ready to introduce the whole architecture of SLADE, which is shown in Figure 3. The crowd-powered system receives large-scale tasks from users, then the SLADE module provides an optimized crowd-execution plan based on the task-bin parameters. Afterwards, the atomic tasks are delivered to lower-level HIT executor/manager to be processed by crowd workforce followed by their specific result aggregator modules. The SLADE module, shown in the dashed box, consists of two models: "Decomposition Planner" and "Task Bins Learning Model". We briefly introduce each in turn.

Decomposition Planner. Effective decomposition plans are generated in terms of task bins, which are trained by the task bin learning model.

Task Bins Learning Model. It plays an important role since the quality of the learned task bins will directly affect the quality of the final decomposition plans. In fact, the set of task bins serves as a real-time probe to the running labor market like AMT or oDesk, monitoring the quality of the current worker flow, whose quantity and response time fluctuate over time [12]. Moreover, in the vision of building up a stable Crowd-powered database system, the set of task bins keeps tracking the state of the computing component, providing information to the upper-level management component.

To obtain the value of the set of tasks bins, when a batch of



determine more or less nodes

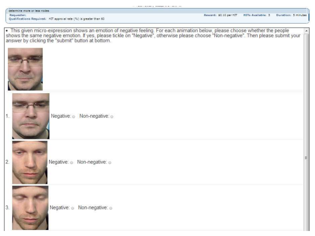
Requester:

Qualifications Required: HIT approval rate (%) is greater than 60

Reward: $0.10 per HIT HITs Available: 2 Duration: 5 minutes

• This given image includes 200 red dots. For each image below, please choose whether there are "more" or "less" dots than the given image. When you finish the choice of each one, please submit your answer by clicking the 'submit' button at bottom.

(a) Interface of *Jelly*



determine more or less nodes

Requester:

Qualifications Required: HIT approval rate (%) is greater than 60

Reward: $0.10 per HIT HITs Available: 3 Duration: 5 minutes

• This given micro-expression shows an emotion of negative feeling. For each animation below, please choose whether the people shows the same negative emotion. If yes, please tickle on "Negative", otherwise please choose "Non-negative". Then please submit your answer by clicking the "submit" button at bottom.

(b) Screen-shots of Experiments HIT Interface

Figure 4: Screen-shots of Experiments HIT Interface

atomic tasks arrives, this model regularly issues testing HITs with different cardinalities. The testing HITs have equivalent difficulty and nature as the real tasks, but the ground truth is known in advance to calculate the confidence. A crowd-powered database system always has response time requirement, which is inversely relevant to the HIT reward. Thus, the cost of each cardinality is obtained as the minimum cost that meets the response time requirement. After obtaining the answers from the testing HITs, the confidence can be easily obtained by any sort of regression or simple counting methods.

Discussion: The architectural novelty of SLADE is that it strategically instantiates the "query executor" as seen in traditional databases. More specifically, such a "query executor" can be considered as a middleware among the existing crowd-powered database prototypes: the "Optimizer" in CrowdDB [13], the "Statistics Manager" and "HIT Compiler" in Qurk [25], the "Planner" in Deco[27], and so on. These previous works provide such an "optimizer" based on certain heuristics. The incorporation of SLADE completes the effort of providing a User-transparent Crowd-powered Database, i.e. the database user operates on with normal or slightly adapted SQL that does not need to know any detail and optimization of the crowds. To sum up, the SLADE system bridges the gap between the requirements of large-scale tasks from customers and crowd workers effectively.

8. EXPERIMENTAL STUDY

In this section, we present a series of empirical studies on the performance of the proposed algorithms. All experiments are conducted on an Intel(R) Core(TM) i7 3.40GHz PC with 4GB main memory and Microsoft Windows 7 OS. Moreover, all the algorithms are implemented and compiled using Microsoft's Visual C++ 2010.

Our empirical studies are conducted on two real datasets gathered from running tasks on Amazon MTurk. The first data set is gathered from the *jelly-beans-in-a-jar* experiments described in Section 2, which we label as *"Jelly"* in the following section. For the second dataset, we issue a set of micro-expression identification tasks on Amazon MTurk, in order to gather a second series of *task bins*. The experiments are set based on the common observation that human is good at perceiving emotions of others from their micro-expressions. We use the Spontaneous Micro-expression Database(SMIC)[30] to test human cognitive ability: given a sample portrait whose micro expression is denoted as positive or negative, the worker is required to decide the emotion of another target portrait. The experimental interface can be found on Figure 4. We vary the cardinality from 1 to 30 with task prices \$0.05, \$0.1 and \$0.2. We then gather the cardinality-cost information to form a second *task bin* input and label it as "SMIC" in this section.

Moreover, the default values of the parameters in our experiments are listed as follows. We set the default value of maximum cardinality ($|B|$) to 20, and the reliability threshold t to 0.9 for all tasks. The number of atomic tasks in the given large-scale task is set to 10,000. In heterogeneous scenarios, the default values are generated according to the Normal distribution with parameters μ and σ set to 0.1 and 0.03, respectively.

To better study the inherent computational characteristics of our proposed algorithms, we conduct the empirical study in three aspects: the intrinsic traits of task bins, the performance of our algorithms for approximation calculation of cost and the performance of the algorithms for the ED problem. In the following figures, *Baseline*, *IGreedy*, and *OPQ* represent the baseline algorithm, the greedy-based algorithm, and the optimal-priority-queue-based algorithm, respectively.

8.1 Characteristics of Task Bins

In this subsection, in parallel to the experimental study described in Section 1, we explore the crowdsourcing platform properties on the SMIC dataset. The results are presented in Figure 5a.

Cardinality vs. Confidence. Similarly, we vary the cardinality from 1 to 30 and evaluate the confidence at each setting. We again notice a general descending tendency, which is expected, since larger cardinality leads to more workloads for crowd workers and higher possibility of making mistakes. However, the confidence in general is only 0.7, much lower than that of the *Jelly* data sets. This is caused by the intrinsic discrepancy between such two types of atomic tasks.

Cost vs. Confidence. Figure 5a also reports the relationship between the total cost and the confidence on the SMIC dataset. It is observed that the increase of cost does not exhibit obvious enhancement on confidence. The reason is that the increase of cost mainly attracts more workers (which may reduce the overall latency), but cannot change the nature of the task.

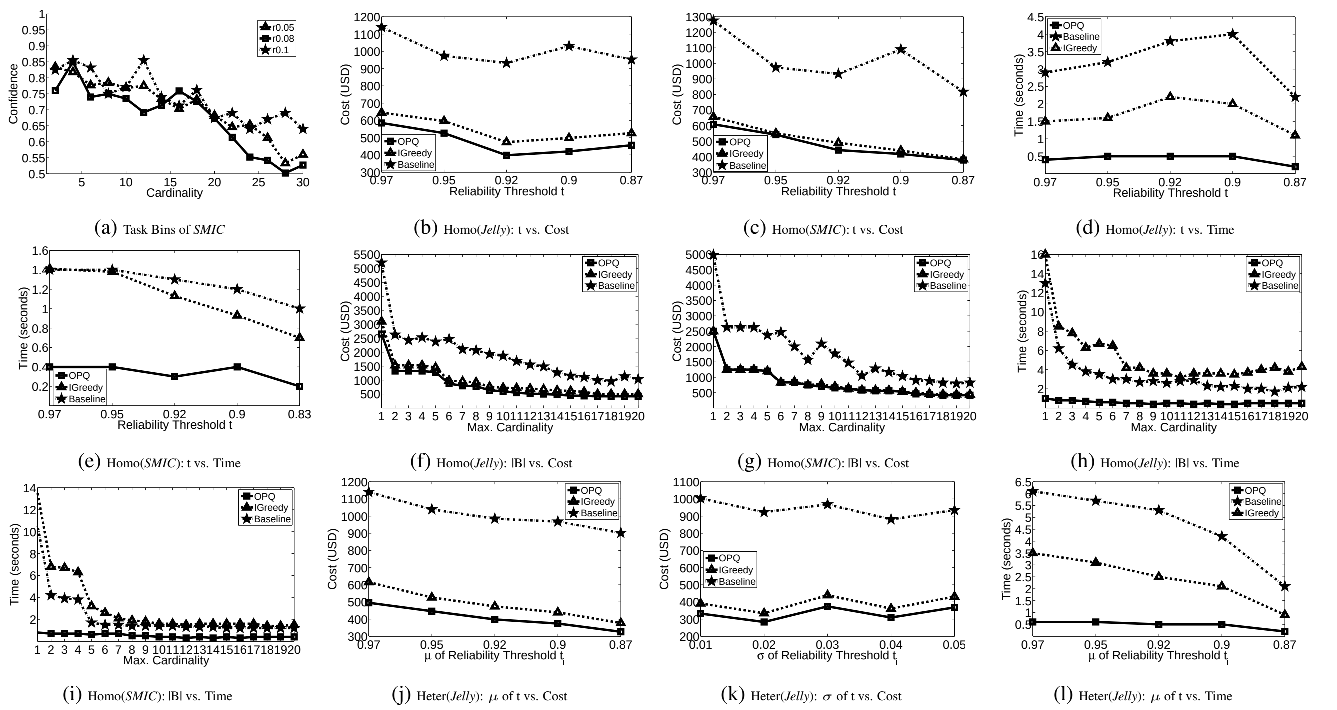
To sum up, the increase of cardinality leads to a decrease in confidence.

8.2 Algorithms in the Homogeneous Scenario

As mentioned in Section 4, the ED problem is NP-hard and it is impossible to obtain the exact optimal decomposition cost in polynomial time unless P=NP. Hence, the approximation quality of the proposed algorithms can be measured by the decomposition cost. In this subsection, we evaluate the cost of the decomposition plan of the three proposed algorithms in the homogeneous scenario, varying the reliability threshold and the maximum cardinality in the set of task bins, respectively.

Varying t. Figures 5b and 5c report the decomposition cost of our proposed three algorithms, where the reliability threshold varies from 0.97 to 0.87, and other parameters are set to their default values. In Figure 5b and Figure 5c, we observe that, with lower reliability threshold t, the overall trends of decomposition cost of all the three algorithms decrease. This is because we need fewer crowd workers to satisfy the lower reliability requirement. Then we present the running time of our proposed algorithms in homogeneous scenario in Figure 5d and 5e over both the *Jelly* and the *SMIC* data sets. In accordance to a varying reliability threshold t, in Figures 5d and 5e, we observe that when the reliability threshold is low (e.g., $t = 0.9$), the running time of *baseline* and *IGreedy* decrease significantly but that of *OPQ* changes slightly. In other words, the running time of *OPQ* is not sensitive to the reliability threshold. This is because, *OPQ* finds the optimal combination using the optimal-priority-queue structure in advance.

Varying $|B|$. Figures 5f and 5g show the decomposition cost of our proposed three algorithms over the two datasets, when the maximum cardinality $|B|$ varies from 1 to 20, and other parame-



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

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

(i) Homo(*SMIC*): |B| vs. Time (j) Heter(*Jelly*): μ of t vs. Cost (k) Heter(*Jelly*): σ of t vs. Cost (l) Heter(*Jelly*): μ of t vs. Time

Figure 5: Characteristics of Task Bins; Cost and Running Time of Proposed Algorithms

ters are set to their default values. Figures 5f and 5g show that the *Baseline* algorithm is significantly affected with the increase of $|B|$. That is reasonable since the *Baseline* algorithm obtains the solution via the randomized rounding method, which is easily affected by a random noise when $|B|$ is small. Hence, the decomposition cost depends on varied $|B|$. Furthermore, the other two algorithms are not sensitive to varied $|B|$, especially when $|B| \geq 6$. Then we test the efficiency of the proposed algorithms with a varying maximum cardinality $|B|$. The results from two datasets are presented in Figure 5h and Figure 5i. The results show that the *OPQ* outperforms the others due to its advantage in data structure design.

8.3 Algorithms in the Heterogeneous Scenario

In this subsection, we evaluate the performance of our proposed algorithms in the heterogeneous scenario, where different atomic tasks might have different reliability thresholds. We generate different reliability thresholds following the Normal distribution. Then, we verify the decomposition cost while varying means and standard deviations of reliability thresholds in terms of the Normal distribution, respectively.

Varying the *standard deviation* σ of the distribution of reliability thresholds. We first test the decomposition cost of our proposed three algorithms, by changing the standard deviation σ of reliability thresholds from 0.01 to 0.05 in Figures 5k. With increasing standard deviation of the distribution of the reliability threshold, decomposition costs of the three algorithms decrease. However, the change is not monotonous. It depends on the following two factors. On one hand, with the increase of standard deviation of reliability thresholds, the number of distinct reliability thresholds increases. However, 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 values of the reliability thresholds rather than the number of distinct reliability thresholds. Thus, the change of decomposition cost is not monotonous. On the other hand, with the increase of standard deviation of reliability thresholds, the likelihood of larger values of reliability thresholds also increases, and the incremental effect of the reliability threshold is $ln(1-\Delta t)$, where Δt is the increasable ratio of the reliability threshold t. Thus, the trend of decomposition cost must decrease when the standard deviation of reliability thresholds increases significantly.

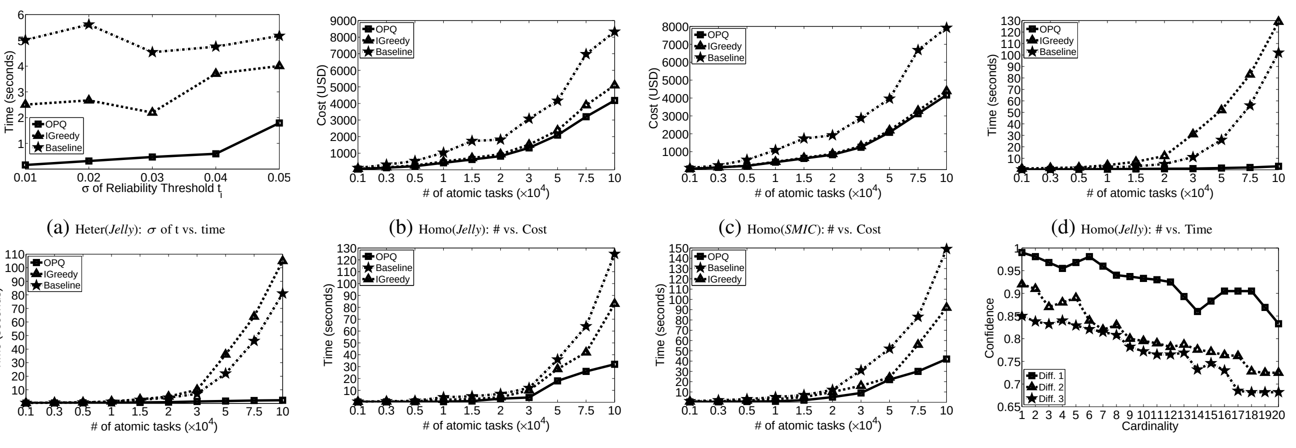
In addition, Figure 6a shows that the running time of the three algorithms increases with the increase of standard deviation of reliability thresholds. Due to the increase of standard deviation of reliability thresholds, the number of distinct reliability thresholds increases. Hence, the three algorithms need more search space to find their approximation optimal solution with more different reliability thresholds. In particular, the running time of *OPQ* increases more significantly because *OPQ* has to build a priority queue for each type of reliability threshold. Thus, with larger number of distinct reliability thresholds, *OPQ* needs more running time.

Varying the *mean* of the distribution of reliability thresholds: We also verify the running time of our proposed algorithms in Figure 5j, and the decomposition cost in Figure 5l, by varying the mean μ of reliability thresholds from 0.97 to 0.87 in Figures 5l. With decreasing the mean of the distribution of the reliability threshold, the general trends of decomposition cost of the three algorithms decrease. In most cases, *OPQ* has the smallest decomposition cost. This makes sense because *OPQ* algorithm first discovers optimal combination for an atomic task and provides the decomposition plan based on the optimal combination. Furthermore, the *Baseline* algorithm has no monotonous change because the vary of mean might lead to more randomized reliability.

To sum up, in the heterogeneous scenario, when increasing the number of distinct reliability thresholds, the three algorithms have to spend more running time. Moreover, when the number of lower reliability thresholds is increased, the decomposition cost must decrease.

8.4 Scalability of Algorithms

In this subsection, we specifically test the scalability performance of our proposed algorithms. The experimental results are presented



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

(e) Homo(*SMIC*): # vs. Time (f) Heter(*Jelly*): # vs. Time (g) Heter(*SMIC*): # vs. Time (h) Diff(*Jelly*): Card vs. Confidence

Figure 6: Cost and Running Time of Proposed Algorithms; Study of Difficulty's Effects

from Figure 6b to Figure 6g.

Homogeneous Scenario. We first test the decomposition cost of the three algorithms, by setting the number of atomic tasks, i.e. parameter $\#$, from 1,000 to 10,000. Figure 6b and Figure 6c compare the decomposition cost of *Baseline*, *IGreedy*, and *OPQ* over two databsets, for different $\#$ of atomic tasks in a large-scale task. When the $\#$ of atomic tasks increases from 1,000 to 100,000, the decomposition cost of the three algorithms always increases. This is expected, since more atomic tasks lead to more cost spent on crowd workers.

In both Figure 6b and Figure 6c, we observe that *OPQ* has the smallest decomposition cost. This is because the *OPQ* algorithm first discovers 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* algorithm in practice. *IGreedy* can have better approximation effect than that of *Baseline* in some cases. This is because the *Baseline* algorithm utilizes a randomized rounding method, which may lead to a worse approximation effect. In Figure 6d and Figure 6e we report the running time for the three algorithms, with the increase of $\#$ from 1,000 to 100,000, where other parameters are set to default values. In these two figures, *OPQ* is the fastest algorithm, and *Baseline* is much slower than *OPQ* and faster than *IGreedy*. That is reasonable because *OPQ* pre-computes the optimal combinations for an atomic task. However, *IGreedy* has to use the iterative strategy based on the local optimal solutions.

Heterogeneous Scenario. We present the running time of our proposed algorithms in the heterogeneous scenario in Figure 6f and 6g over both the *Jelly* and *SMIC* datasets, by varying the $\#$ from 1,000 to 100,000. The overall tendency resembles the cases in homogeneous scenarios. But when the number of the atomic tasks becomes large, *OPQ* consumes more running time compared to that in homogeneous scenarios. In heterogeneous scenarios, the *OPQ* algorithm has to construct optimal priority queues for various distinct reliability threshold values.

8.5 Effects of Atomic Task Difficulty

In this subsection, we briefly discuss the influence of the intrinsic difficulty of an atomic task on confidence. We conduct a series of experiments on the *Jelly* dataset, and the results are presented in Figure 6h. The intrinsic difficulty of a *Jelly* experiment is represented by the number of dots in the given sample image: we specify difficulty level 1 with a dots number of 50, level 2 with a number of 200, and level 3 with a number of 400. The performances are labeled as *Diff. 1/2/3* in Figure 6h. It is easily noticed that lower difficulty, i.e. less dots in the sample image, generates an obviously higher confidence. In addition, the trade-off between increasing cardinality and decreasing confidence is still observed as all the three curves decline gradually.

The experiment results are in accordance with our observation in Section 1, that a different set of Task Bins should be probed and learned for a different type of atomic task, in terms of both intrinsic difficulty or task design.

The intrinsic difficulty of a crowdsourcing task is widely studied, and many sophisticated models have been proposed[14]. However this is not the focus of this paper and we leave it as a future work.

9. RELATED WORK

In this section, we review the related work of data-driven crowdsourced human computation and optimization.

Human computation has been practiced for centuries. Specifically, whenever a "human" serves to "compute", a human computation is observed. This leads to the 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 recruitment and management such as Amazon MTurk (AMT) 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 experts 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 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.

In data-driven applications, human cognitive abilities are mainly exploited in two aspects: voting among multiple options, and providing contents according to certain requirements. Most basic queries in database [13] and data mining [2, 1, 17] can be decomposed into simple voting as human tasks: filtering [5, 14, 27, 26] into two-option voting (Yes or No), image tagging [39], join [25, 38, 37], entity resolution [37, 16, 11, 36], and schema matching[40, 41] into two-option or multiple voting (connecting the same entities), ranking and top-k [8, 18, 35, 31] to Borda voting(numbering by order). Meanwhile, in order to break the close world assumption in traditional database, human are enrolled to provide extraneous information to answer certain queries: item enumeration [9, 13, 33], content composing [3, 20], counting [23], taxonomy creation[6], and so on.

Moreover, several recent works have also been developed to optimize the performance of crowdsourcing platforms from different aspects[14, 21, 42, 22]. In particular, J. Gao et al. propose a difficulty control framework for the tasks based on majority voting aggregation rules[14]; A. Kitter et al. propose CrowdForge as a prototype to decompose complex tasks such as article writing and science journalism to small HITs[21]; the work by H. Zhang et al. studies how to decompose the itinerary planning task with global constraints such as sequence and coverage [42]; and a novel work by K. Anand et al. proposes to recruit crowds to improve the process of task transformation and control[22]. Note that most of the aforementioned work focus on higher-level query transformation from a specific type of task into the form of HITs 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 the crowd-powered databases.

10. CONCLUSION

In this paper, we propose *SLADE*, which builds a bridge between the requirements of large-scale tasks from customers and the crowd workers effectively. In this work, the core problem is formulated as the *Effective Decomposition (ED) Problem*, which is proven to be NP-hard. To solve the ED Problem, we study this problem 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 have verified our proposed algorithms through extensive empirical studies over representative crowdsourcing platforms.

11. REFERENCES

[1] Y. Amsterdamer, S. B. Davidson, T. Milo, S. Novgorodov, and A. Somech. Oassis: query driven crowd mining. In *SIGMOD*, 2014.

[2] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart. Crowd mining. In *SIGMOD*, 2013.

[3] Y. Baba and H. Kashima. Statistical quality estimation for general crowdsourcing tasks. In *SIGKDD*, 2013.

[4] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 1998.

[5] C. C. Cao, J. She, Y. Tong, and L. Chen. Whom to ask? jury selection for decision making tasks on micro-blog services. *PVLDB*, 2012.

[6] L. B. Chilton, G. Little, D. Edge, D. S. Weld, and J. A. Landay. Cascade: crowdsourcing taxonomy creation. In *CHI*, 2013.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3rd Version)*. MIT Press, 2009.

[8] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*, 2013.

[9] D. Deutch and T. Milo. Mob data sourcing. In *SIGMOD*, pages 581–584, 2012.

[10] G. Dobson. Worst-case analysis of greedy heuristics for integer programming with nonnegative data. *Math. Oper. Res.*, 1982.

[11] J. Fan, M. Lu, B. C. Ooi, W.-C. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *ICDE*.

[12] S. Faradani, B. Hartmann, and P. G. Ipeirotis. What's the right price? pricing tasks for finishing on time. In *Human Computation*, 2011.

[13] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, 2011.

[14] J. Gao, X. Liu, B. C. Ooi, H. Wang, and G. Chen. An online cost sensitive decision-making method in crowdsourcing systems. In *SIGMOD*, 2013.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[16] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, 2014.

[17] R. Gomes, P. Welinder, A. Krause, and P. Perona. Crowdclustering. In *NIPS*, 2011.

[18] S. Guo, A. G. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD*, 2012.

[19] J. Howe. *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*. Crown Business, 2009.

[20] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *GIS*, 2012.

[21] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut. Crowdforge: Crowdsourcing complex work. In *UIST*, 2011.

[22] A. Kulkarni, M. Can, and B. Hartmann. Collaboratively crowdsourcing workflows with turkomatic. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, 2012.

[23] A. Marcus, D. R. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. *PVLDB*, 2012.

[24] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Demonstration of qurk: a query processor for humanoperators. In *SIGMOD*, 2011.

[25] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Human-powered sorts and joins. *PVLDB*, 2011.

[26] A. G. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom. Optimal crowd-powered rating and filtering algorithms. *PVLDB*, 2014.

[27] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD*, 2012.

[28] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: A system for declarative crowdsourcing. *PVLDB*, 2012.

[29] H. Park and J. Widom. Query optimization over crowdsourced data. *PVLDB*, 2013.

[30] T. Pfister, X. Li, G. Zhao, and M. Pietikainen. Recognising spontaneous facial micro-expressions. In *ICCV*, 2011.

[31] V. Polychronopoulos, L. de Alfaro, J. Davis, H. Garcia-Molina, and N. Polyzotis. Human-powered top-k lists. In *WebDB 2013*, 2013.

[32] A. Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM J. Comput.*, 1999.

[33] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Crowdsourced enumeration queries. In *ICDE*, 2013.

[34] V. V. Vazirani. *Approximation algorithms*. Springer, 2001.

[35] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis. Max algorithms in crowdsourcing environments. In *WWW*,

2012.

[36] N. Vesdapunt, K. Bellare, and N. N. Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 2014.

[37] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 2012.

[38] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, 2013.

[39] X. S. Yang, R. Cheng, L. Mo, B. Kao, and D. W. Cheung. On incentive-based tagging. In *ICDE*, 2013.

[40] C. J. Zhang, L. Chen, H. V. Jagadish, and C. C. Cao. Reducing uncertainty of schema matching via crowdsourcing. *PVLDB*, 2013.

[41] C. J. Zhang, Z. Zhao, L. Chen, H. V. Jagadish, and C. C. Cao. Crowdmatcher: crowd-assisted schema matching. In *SIGMOD*, 2014.

[42] H. Zhang, E. Law, R. Miller, K. Gajos, D. Parkes, and E. Horvitz. Human computation tasks with global constraints. In *CHI*, 2012.

APPENDIX

Detailed Description of Baseline Algorithm

As discussed in Section 4.3, we present the baseline algorithm, which adopts the randomized rounding algorithm of the *covering integer programming*[34], to which the integer programming form of the ED problem is equivalent. The main idea of the baseline algorithm consists of the following two steps. First, we utilize the column generation technique to solve the linear programming (LP) relaxation of the integer programming form of ED and obtain an LP optimal solution. Then, in the second step, we perform the randomized rounding algorithm based on the LP optimal solution to obtain the decomposition plan and cost. This algorithm can obtain $\ln n$-approximation ratio with the $1 - \delta$ probability, where δ is a probabilistic error, and its time complexity depends on the heuristic strategy of the column generation.

The pseudo code of the baseline algorithm is shown in Algorithm 5, which intends to find an approximation solution satisfying the reliability thresholds via our proposed hybrid randomized rounding approach. Line 2 first uses the column generation technique to get the LP optimal solution, denoted as LP^*. Then, lines 4-11 perform the randomized rounding algorithm for covering integer programming[4].

Algorithm 5: A Baseline Algorithm

```
Input: A large-scale task $T = \{a_1, \dots, a_{|T|}\}$, a set of task
       bins $B = \{b_1, \dots, b_{|B|}\}$, a set of reliability thresholds
       $\{t_1, \dots, t_{|T|}\}$
Output: The decomposition plan $DP_T$ and its cost $Cost_T$
1  $n \leftarrow |T|$;
2  $LP^* \leftarrow CG(T, B, t)$;
3  $R \leftarrow 1 + \frac{4ln(1-r_l)ln(2n)}{ln(1-t)}$;
4  for $y_j \in LP^*$ do
5      $p_j \leftarrow R \cdot y_j - \lfloor R \cdot y_j \rfloor$;
6      if Rounding "success" with the probability $p_j$ then
7          $\hat{y_j} \leftarrow \lceil R \cdot y_j \rceil$;
8      else
9          $\hat{y_j} \leftarrow \lfloor R \cdot y_j \rfloor$;
10     $DP_T \leftarrow DP_T \cup \hat{y_j}$;
11     $Cost_T \leftarrow Cost_T + \hat{y_j} \cdot c_j$;
12 return $DP_T$ and $Cost_T$;
```