Recommending Deployment Strategies for Collaborative Tasks

Dong Wei
NJIT, USA
dw277@njit.edu

Senjuti Basu Roy
NJIT, USA
senjutib@njit.edu

Sihem Amer-Yahia
CNRS, Univ. Grenoble Alpes, France
sihem.amer-yahia@cnrs.fr

ABSTRACT

Our work contributes to aiding requesters in deploying collaborative tasks in crowdsourcing. We initiate the study of recommending deployment strategies for collaborative tasks to requesters that are consistent with deployment parameters they desire: a lower-bound on the quality of the crowd contribution, an upper-bound on the latency of task completion, and an upper-bound on the cost incurred by paying workers. A deployment strategy is a choice of value for three dimensions: *Structure* (whether to solicit the workforce sequentially or simultaneously), *Organization* (to organize it collaboratively or independently), and *Style* (to rely solely on the crowd or to combine it with machine algorithms). We propose StratRec, an optimization-driven middle layer that recommends deployment strategies and alternative deployment parameters to requesters by accounting for worker availability. Our solutions are grounded in discrete optimization and computational geometry techniques that produce results with theoretical guarantees. We present extensive experiments on Amazon Mechanical Turk, and conduct synthetic experiments to validate the qualitative and scalability aspects of StratRec.

ACM Reference Format:
Dong Wei, Senjuti Basu Roy, and Sihem Amer-Yahia. 2020. Recommending Deployment Strategies for Collaborative Tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3318464.3389719

1 INTRODUCTION

Despite becoming a popular means of deploying tasks, crowdsourcing offers very little help to requesters. In particular, task deployment requires that requesters identify *appropriate deployment strategies*. A strategy involves the interplay of multiple dimensions: *Structure* (whether to solicit the workforce sequentially or simultaneously), *Organization* (to organize it collaboratively or independently), and *Style* (to rely on the crowd alone or on a combination of crowd and machine algorithms). A strategy needs to be commensurate to *deployment parameters* desired by a requester, namely, a lower-bound on quality, an upper-bound on latency, and an upper-bound on cost. For example, for a sentence translation task, a requester wants the translated sentences to be at least 80% as good as the work of a domain expert, in a span of at most 2 days, and at a maximum cost of $100. Till date, the burden is entirely on requesters to design deployment strategies that satisfy desired parameters. Our effort in this paper is to present a formalism and computationally efficient algorithms to recommend multiple strategies (namely k) to the requester that are commensurate to her deployment parameters, primarily for collaborative tasks.

A recent work [5] investigated the deployment of text creation tasks in Amazon Mechanical Turk (AMT) empirically. The authors validated the effectiveness of different strategies for different collaborative tasks, such as text summarization and text translation, and provided evidence for the need to guide requesters in choosing the right strategy. In this paper, we propose *to automate strategy recommendation.* This is particularly challenging because the estimation of the cost, quality, and latency of a strategy for a given deployment request must account for many factors.

To realize our contributions, we develop StratRec (refer to Figure 1), an optimization-driven middle layer that sits between requesters, workers, and platforms. StratRec has two main modules: *Aggregator* and *Alternative Parameter Recommendation* (ADPaR in short). *Aggregator* is responsible for recommending k strategies to a batch of incoming deployment requests, considering worker availability. If the platform does not have enough qualified workers to satisfy all requests, *Aggregator triages them by optimizing platform-centric goals, i.e., to maximize throughput or pay-off (Section 2.2).* Unsatisfied requests are sent to ADPaR, which recommends different deployment parameters for which k strategies are available.

In principle, *recommending deployment strategies involves modeling worker availability considering their skills for the*

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3389719

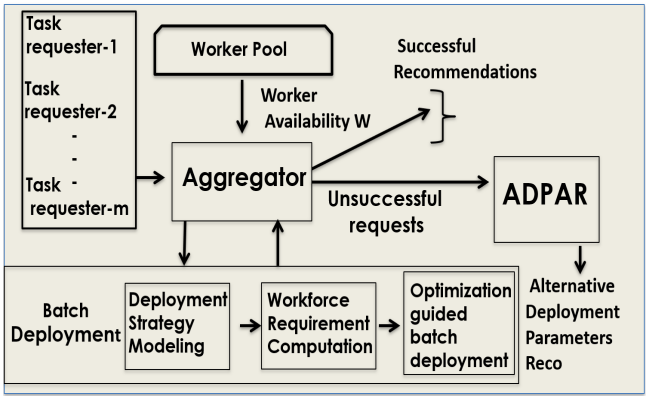


Figure 1: StratRec Framework

tasks that require deployment. This gives rise to a complex function that estimates parameters (quality, latency, and cost) of a strategy considering worker skills, task types, and worker availability. As the first ever principled investigation of strategy recommendation in crowdsourcing, we first make a binary match between workers' skills and task types and then estimate strategy parameters considering those workers' availability. Worker availability is captured as a probability distribution function (pdf) by leveraging historical data on a platform. For example, the pdf can capture that there is a 70% chance of having 7% of the workers and a 30% chance of having 2% of the workers available who are suitable to undertake a certain type of task. In expectation, this gives rise to 5.5% of available workers. If a platform has 4000 total workers available to undertake a certain type of task, that gives rise to a total of 220 available workers in an expected sense. StratRec works with such *expected values*.

Contribution 1. Modeling and Formalism: We present a general framework StratRec for modeling quality, cost, and latency of a set of collaborative tasks, when deployed based on a strategy considering worker availability (Section 3.1). The first problem we study is *Batch Deployment Recommendation* inside to deploy a batch of tasks to maximize two different platform-centric criteria: task throughput and pay-off. After that, unsatisfied requests are sent one by one to the *Alternative Parameter Recommendation* module (ADPaR). ADPaR solves an optimization problem that recommends alternative parameters for which k deployment strategies exist. For instance, if a request has a very small latency threshold that cannot be attained based on worker availability, ADPaR may recommend increasing the latency and cost thresholds to find k legitimate strategies. ADPaR does not arbitrarily choose the alternative deployment parameters. It recommends those alternative parameters that are *closest*, i.e., minimizing the ℓ_2 distance to the ones specified.

Contribution 2. Algorithms: In Section 3, we design BatchStrat, a unified algorithmic framework to solve the *Batch Deployment Recommendation* problem. BatchStrat is greedy in nature and provides exact results for the throughput maximization problem, and a 1/2-approximation factor for the pay-off maximization problem (which is NP-hard). In Section 4, we develop ADPaR-Exact to solve ADPaR that is geometric and exploits the fact that our objective function is monotone (Equation 3). Even though the original problem is defined in a continuous space, we present a discretized technique that is exact. ADPaR-Exact employs a sweep-line technique [9] that gradually relaxes quality, cost, and latency, and is guaranteed to produce the tightest alternative parameters for which k deployment strategies exist.

Contribution 3. Experiments: We conduct comprehensive real-world deployments for text editing applications with real workers and rigorous synthetic data experiments (Section 5). The former validates that worker availability *varies over time, and could be reasonably estimated through multiple real world deployments*. It also shows *with statistical significance that cost, quality, latency have a linear relationship with worker availability for text editing tasks*. Our real data experiments (Section 5.1.2) also validate that when tasks are deployed considering the recommendation of StratRec, with statistical significance, they achieve higher quality and lower latency, under the fixed cost threshold on an average, compared to the deployments that do not consult StratRec. These results validate the effectiveness of deployment recommendations of our proposed framework and its algorithms.

2 FRAMEWORK AND PROBLEM

2.1 Data Model

Crowdsourcing Tasks: A platform is designed to crowdsource tasks, deployed by a set of requesters and undertaken by crowd workers. We consider collaborative tasks such as sentence translation, text summarization, and puzzle solving [29, 30].

Deployment Strategies: A deployment strategy [17] instantiates three dimensions: *Structure* (sequential or simultaneous), *Organization* (collaborative or independent), and *Style* (crowd-only or crowd and algorithms). We rely on common deployment strategies [5, 17] and refer to them as $\mathcal{S}$. Figure 2 enlists some strategies that are suitable for text translation tasks (from English to French in this example). For instance, *SEQ-IND-CRO* in Figure 2(a) dictates that workers complete tasks sequentially (*SEQ*), independently (*IND*) and with no help from algorithms (*CRO*). In *SIM-COL-CRO* (Figure 2(b)), workers are solicited in parallel (*SIM*) to complete a task collaboratively (*COL*) and with no help from algorithms (*CRO*). The last strategy *SIM-IND-HYB* dictates a hybrid work style (*HYB*) where workers are combined with algorithms, for instance with Google Translate.

A platform could provide the ability to implement some strategies. For instance, communication between workers enables SEQ, while collaboration enables COL. Additionally, coordination between machines and humans may enable HYB. Therefore, strategies could be implemented inside or outside platforms. In the latter, a platform could be used solely for hiring workers who are then redirected to an environment where strategies are implemented. In all cases, we will assume a set of strategies $\mathcal{S}$ for a given platform.

For the purpose of illustration, we will only use a few strategies in this paper. However, in principle, the number of possible strategies could be very large. The closest analogy is query plans in relational databases in which joins, selections, and projections could be combined any number of times and in different orders. Additionally, there exists multiple real world tools Turkomatic [19] or Soylent [4], that aid requesters in planning and solving collaborative tasks. In Turkomatic, while workers decompose and solve tasks, requesters can view the status of worker-designed workflows in real time; intervene to change tasks; and request new solutions. Such tools would certainly benefit from strategy recommendation.

Task Requests and Deployment Parameters: A requester intends to find one or more strategies (notationally k, a small integer) for a deployment d with parameters on quality, cost, and latency ($d.quality$, $d.cost$, $d.latency$) such that, when a task in d is deployed using strategy $s \in S$, it is estimated to achieve a crowd contribution quality $s.quality$, by spending at most $s.cost$, and the deployment will last at most $s.latency$.

	Quality	Cost	Latency
d_1	0.4	0.17	0.28
d_2	0.8	0.2	0.28
d_3	0.7	0.83	0.28
s_1	0.5	0.25	0.28
s_2	0.75	0.33	0.28
s_3	0.8	0.5	0.14
s_4	0.88	0.58	0.14

Table 1: Deployment Requests and Strategies

Example 1. *Assume there are* 3 *($m = 3$) task deployment requests for different types of collaborative sentence translation tasks. The first requester d_1 is interested in deploying sentence translation tasks for* 2 *days (out of* 7 *days), at a cost up to* \$100 *(out of* \$600 *max), and expects the quality of the translation to reach at least* 40% *of domain expert quality. Table 1 presents these after normalization between* $[0-1]$. *We set* $k = 3$.

A strategy s is suitable to be recommended to d, if $s.quality \geq d.quality\ AND\ s.cost \leq d.cost\ AND\ s.latency \leq d.latency$. Estimating the parameters $s.quality$, $s.cost$, $s.latency$ for each s and deployment d requires accounting for the worker pool and their skills who are available to undertake tasks in d. *A simple yet reasonable approach to that is to first match task types in a deployment request with workers' skills to select a pool of workers. Following that, we account for* worker availability *from this selected pool, since the deployed tasks are to be done by those workers. Thus, the (estimated) quality, cost and latency of a strategy for a task is a function of worker availability, considering a selected pool of workers who are suitable for the tasks.*

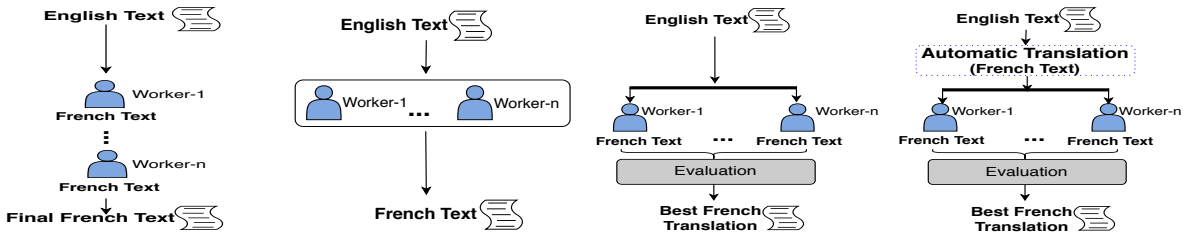
Worker Availability: Worker availability is a discrete random variable and is represented by its corresponding distribution function (pdf), which gives the probability of the proportion of workers who are suitable and available to undertake tasks of a certain type within a specified time $d.latency$ (refer to Example 1). This pdf is computed from historical data on workers' arrival and departure on a platform. StratRec computes the expected value of this pdf to represent the available workforce W, as a normalized value in $[0, 1]$. In the remainder of the paper, worker availability stands for worker availability in expectation, unless otherwise specified. How to accurately estimate worker availability is an interesting yet orthogonal problem and not our focus here.

2.2 Illustration of StratRec

StratRec is an optimization-driven middle layer that sits between requesters, workers, and platforms. At any time, a crowdsourcing platform has a batch of m deployment requests each with its own parameters as defined above, coming from different requesters. StratRec is composed of two main modules - *Aggregator* and *Alternative Parameter Recommendation* (or ADPaR).

For the purpose of illustration, continuing with Example 1, $\mathcal{S}$ consists of the set of 4 deployment strategies, as shown in Figure 2: *SIM-COL-CRO, SEQ-IND-CRO, SIM-IND-CRO, SIM-IND-HYB.* To ease understanding, we name them as s_1, s_2, s_3, s_4, respectively.

These requests, once received by StratRec, are sent to the *Aggregator*. First, it analyzes the Worker Pool to estimate worker availability. There is a 50% probability of having 700 workers and a 50% probability of having 900 workers out of 1000 suitable workers for sentence translation tasks available for the next 7 days. Thus, the expected worker availability W is 0.8. After that, it consults the *Deployment Strategy Modeling* in *Batch Deployment module* to estimate the quality, cost, and latency of a strategy (more in Section 3.1) for a deployment. Since all deployments are of the same type, Equation 4, could be used to estimate those Strategy parameters (also presented in Table 1). Then, it consults the *Workforce Requirement Computation* to estimate the workforce requirement of each strategy (more in Section 3.2 and Figure 3). Finally, the *Optimization Guided Batch Deployment* (refer to Section 3.3) is invoked to select a subset of requests that optimizes the underlying goal and recommends k strategies for each. Each



(a) SEQ-IND-CRO (b) SIM-COL-CRO (c) SIM-IND-CRO (d) SIM-IND-HYB

Figure 2: Deployment Strategies

unsatisfied request d_i is sent to ADPaR that recommends an alternative deployment d_i' to the requester for which there exist k deployment strategies.

Using Example 1, out of the three deployment requests, only d_3 could be fully served (considering either throughput or pay-off objective) and s_2, s_3, s_4 are recommended. d_1 and d_2 are then sent to ADPaR one by one.

2.3 Problem Definitions

Problem 1. **Batch Deployment Recommendation:** *Given an optimization goal F, a set $\mathcal{S}$ of strategies, a batch of m deployment requests from different requesters, where the i-th task deployment d_i is associated with parameters $d_i.quality$, $d_i.cost$ and $d_i.latency$, and worker availability W, distribute W among these requests by recommending k strategies for each request, such that F is optimized.*

The high level problem optimization problem could be formalized as:

$$\begin{aligned} &Maximize\ F = \sum f_i \\ &s.t. \sum \vec{w}_i \leq W\ AND \\ &d_i\ is\ successful \end{aligned} \tag{1}$$

where f_i is the optimization value of deployment d_i and $\vec{w}_i$ is the workforce required to successfully recommend k strategies it. A deployment request d_i is successful, if for each of the k strategies in the recommended set of strategies S_d^i, the following three criteria are met: $s.cost \leq d_i.cost$, $s.latency \leq d_i.latency$ and $s.quality \geq d_i.quality$.

Using Example 1, d_3 is successful, as it will return $S_d^3 = \{s_2, s_3, s_4\}$, such that $d_3.cost \geq s_4.cost \geq s_3.cost \geq s_2.cost$ & $d_3.latency \geq s_4.latency \geq s_3.latency \geq s_2.latency$ & $d_3.quality \leq s_4.quality \leq s_3.quality \leq s_2.quality$, and it could be deployed with the available workforce $W = 0.8$.

In this work, F is designed to maximize one of two different platform centric-goals: task throughput and pay-off.

Throughput maximizes the total number of successful strategy recommendations without exceeding W. Formally,

$$\begin{aligned} &\text{Maximize} \sum_{i=1}^{m} x_i \\ &\text{s.t.} \sum x_i \times \vec{w}_i \leq W \\ &x_i = \begin{cases} 1 & d_i.cost \leq s_j.cost \text{ AND} \\ & d_i.latency \leq s_j.latency \text{ AND} \\ & d_i.quality \geq s_j.quality \text{ AND} \\ & |S_d^i| = k, \forall i = 1, \ldots, m; j = 1, \ldots, |\mathcal{S}| \\ 0 & otherwise \end{cases} \end{aligned} \tag{2}$$

Pay-off maximizes $d_i.cost$, if d_i is a successful deployment request without exceeding W. The rest of the formulation is akin to Equation 2.

Problem 2. **Alternative Parameter Recommendation:** *Given a deployment d, worker availability W, a set of deployment strategies $\mathcal{S}$, and a cardinality constraint k, ADPaR recommends an alternative deployment d' and associated k strategies, such that, the Euclidean distance (ℓ_2) between d and d' is minimized.*

Formally, our problem could be stated as a constrained optimization problem:

$$\begin{aligned} &min \quad (d'.cost - d.cost)^2 + (d'.latency - d.latency)^2 \\ &\qquad + (d'.quality - d.quality)^2 \\ &s.t. \quad \sum_{j=1}^{|\mathcal{S}|} x_j = k \\ &x_j = \begin{cases} 1 & d'.cost \leq s_j.cost\ AND \\ & d'.latency \leq s_j.latency\ AND \\ & d'.quality \geq s_j.quality \\ 0 & otherwise \end{cases} \end{aligned} \tag{3}$$

Based on Example 1, if ADPaR takes the following input values $d_1 : (0.4, 0.17, 0.28)$ and $\mathcal{S}$. For d_1, the alternative recommendation should be $(0.4, 0.5, 0.28)$ with three strategies s_1, s_2, s_3.

3 DEPLOYMENT RECOMMENDATION

We describe our proposed solution for *Batch Deployment Recommendation* (Problem 1). Given m requests and W, the *Aggregator* invokes BatchStrat, our unified solution to solve the batch deployment recommendation problem. There are three major steps involved. BatchStrat first obtains model parameters of a set of candidate strategies (Section 3.1), then computes workforce requirement to satisfy these requests (Section 3.2), and finally performs optimization to select a subset of m deployment requests, such that different platform-centric optimization goals could be achieved (Section 3.3).

We first provide an abstraction that serves the purpose of designing BatchStrat. Given m deployment requests and W workforce availability, we intend to compute a two dimensional matrix $\mathcal{W}$, where there are $|\mathcal{S}|$ columns that map to available deployment strategies and m rows of different deployment requests. Figure 3a shows the matrix built for Example 1. A cell w_{ij} in this matrix estimates the workforce required to deploy i-th request using j-th strategy. This matrix $\mathcal{W}$ is crucial to enable platform centric optimization for batch deployment.

3.1 Deployment Strategy Modeling

BatchStrat first performs deployment strategy modeling to estimate quality, cost, latency of a strategy s for a given deployment request d. As the first principled solution, it models these parameters as a linear function of worker availability, from the filtered pool of workers whose profiles match tasks in the deployment request [1]. Therefore, if d is deployed using strategy s, the quality parameter of this deployment is modeled as:

$$s_d.quality = \alpha_{qds}.(w_{qds}) + \beta_{qds} \tag{4}$$

Our experimental evaluation (Table 6) in Section 5.1, performed on AMT validates this linearity assumption with 90% statistical significance for two text editing tasks.

Model parameters α and β are obtained for every s, d, and parameter (quality, cost, latency) combination, by fitting historical data to this linear model. Once these parameters are known, BatchStrat uses Equation 4 again to estimate workforce requirement w_{qds} to satisfy quality threshold (cost and latency like-wise) for deployment d using strategy s. We repeat this exercise for each $s \in \mathcal{S}$, which comprises our set of candidate strategies for a deployment d.

[1] We note that StratRec could be adapted for tasks that do not exhibit such linear relationships.

3.2 Workforce Requirement Computation

The goal of the *Workforce Requirement Computation* is to estimate workforce requirement per (deployment, strategy) pair. It performs that in two sub-steps, as described below.

(1) Computing Matrix $\mathcal{W}$: The first step is to compute $\mathcal{W}$, where $w_{i,j}$ represents the workforce requirement of deploying d_i with strategy s_j. Recall that in Equation 4, as long as for a deployment d_i, the deployment parameters on quality, cost, and latency, i.e., $d_i.quality$, $d_i.cost$ and $d_i.latency$ are known, for a strategy, s_j, we can compute $w_{i,j}$, i.e., that is the minimum workforce needed to achieve those thresholds, by considering the equality condition, i.e., $s_j.quality = d_i.quality$ (similarly for cost and latency), and solving Equation 4 for w, with known (α, β) values. Using Example 1, the table in Figure 3a shows the rows and columns of matrix $\mathcal{W}$ and how a workforce requirement could be calculated for w_{11}. Basically, once we solve the workforce requirement of quality, cost, and latency(w_{qij}, w_{cij}, w_{lij}), the overall workforce requirement of deploying d_i using s_j is the maximum over these three requirements. Formally, they could be stated as follows:

$$w_{ij} = Max \begin{cases} d_i.quality = \alpha_{qij} w_{qij} + \beta_{qij} \\ d_i.cost = \alpha_{cij} w_{cij} + \beta_{cij} \\ d_i.latency = \alpha_{lij} w_{lij} + \beta_{lij} \end{cases}$$

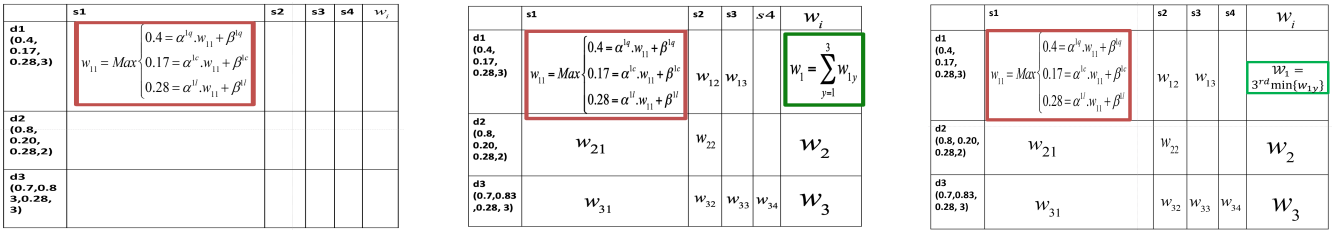
Using Example 1, w_{11} is the maximum over $\{w_{q11}, w_{c11}, w_{l11}\}$. Figure 3a shows how w_{11} needs to be computed for deployment d_1 and strategy s_1 for the running example.

Running Time: Running time of computing $\mathcal{W}$ is $O(m|\mathcal{S}|)$, since computing each cell w_{ij} takes constant time.

(2) Computing Workforce Requirement per Deployment: For a deployment request d_i to be successful, BatchStrat has to find k strategies, such that each satisfies the deployment parameters. In step (2), we investigate how to make compute workforce requirement for all k strategies, for each d_i. The output of this step produces a vector $\vec{W}$ of length m, where the i-th value represents the aggregated workforce requirement for request d_i. Computing $\vec{W}$ requires understanding of two cases:

- **Sum-case:** It is possible that the task designer intends to perform the deployment using all k strategies. Therefore, the minimum workforce (w_i) needed to satisfy cardinality constraint k_i is $\Sigma_{y=1}^{k} w_{iy}$ (where w_{iy} is the y-th smallest workforce value in row i of matrix $\mathcal{W}$.
- **Max-case:** The task designer intends to only deploy one of the k recommended strategies - in that case, $w_i = w_{iy}$, (where w_{iy} is the k-th smallest workforce value in row i of matrix $\mathcal{W}$).

Figures 3b and 3c represent how $\vec{W}$ is calculated considering sum-case and max-case, respectively.



(a) Requirement for (d_1, s_1)

(b) Aggregated requirement per request (Sum)

(c) Aggregated requirement per request (Max)

Figure 3: Computing Workforce Requirement

Running Time: The running time of computing the aggregated workforce requirement of the i-th deployment request is $O(|\mathcal{S}|klog|\mathcal{S}|)$, if we use min-heaps to retrieve the k smallest numbers. The overall running time is again $O(mk\ log|\mathcal{S}|)$.

3.3 Optimization-Guided Batch Deployment

Finally, we focus on the optimization step of BatchStrat, where, given $\vec{W}$, the objective is to distribute the available workforce W among m deployment requests such that it optimizes a platform-centric goal F. Since W can be limited, it may not be possible to successfully satisfy all deployment requests in a single batch. This requires distributing W judiciously among competing deployment requests and satisfying the ones that maximize platform-centric optimization goals, i.e., throughput or pay-off.

At this point, a keen reader may notice that the batch deployment problem bears a resemblance to a well-known discrete optimization problem that falls into the general category of assignment problems, specifically, Knapsack-type of problems [10]. The objective is to maximize a goal (in this case, throughput or pay-off), subject to the capacity constraint of worker availability W. In fact, depending on the nature of the problem, the optimization-guided batch deployment problem could become intractable.

Intuitively, when the objective is only to maximize throughput (i.e., the number of satisfied deployment requests), the problem is polynomial-time solvable. However, when there is an additional dimension, such as pay-off, the problem becomes NP-hard problem, as we shall prove next.

Theorem 1. *The Pay-Off maximization problem is NP-hard [33].*

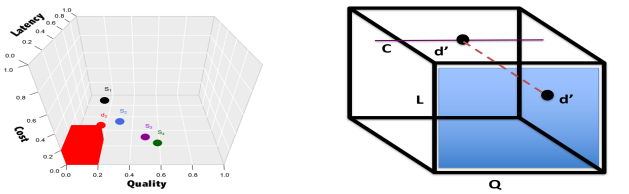
Our proposed solution bears similarity to the greedy approximation algorithm of the Knapsack problem [14]. The objective is to sort the deployment strategies in non-increasing order of $\frac{f_i}{\vec{w}_i}$. The algorithm greedily adds deployments based on this sorted order until it hits a deployment d_i that can no longer be satisfied by W, that is, $\Sigma_{i=1..x}\ d_i > W$. At that step, it chooses the better of $\{d_1, d_2, d_{i-1}\}$ and d_i and the process continues until no further deployment requests could be satisfied based on W. Lines 4 – 8 in Algorithm BatchStrat describe those steps.

Running Time: The running time of this step is dominated by the sorting time of the deployment requests, which is $O(m\ logm)$.

Algorithm 1 Algorithm BatchStrat

1: **Input:** m deployment requests, $\mathcal{S}$, objective function F, available workforce W
2: **Output:** recommendations for a subset of deployment requests.
3: Estimate model parameters for each (strategy, deployment) pair.
4: Compute Workforce Requirement Matrix $\mathcal{W}$
5: Compute Workforce Requirement per Deployment Vector $\vec{W}$
6: Compute the objective function value f_i of each deployment request d_i
7: Sort the deployment requests in non-increasing order of $\frac{f_i}{\vec{w}_i}$
8: Greedily add deployments until we hit d_i, such that $\Sigma_{i=1..x}\ d_i > W$
9: Pick the better of $\{d_1, d_2, d_{i-1}\}$ and d_i

3.3.1 Maximizing Throughput. When task throughput is maximized, the objective function F is computed simply by counting the number of deployment requests that are satisfied by the `Aggregator`. Therefore, f_i, the objective function value of deployment d_i is the same for all the deployment requests and is 1. Our solution, BatchStrat-ThroughPut, sorts the deployment requests in increasing order of workforce requirement $\vec{w}_i$ to make $\frac{1}{\vec{w}_i}$ non-increasing. Other than that, the rest of the algorithm remains unchanged.



(a) Deployment parameters in 3-D space **(b) Projection of d' on (L, Q) plane**

Figure 4: ADPaR

Theorem 2. *Algorithm BatchStrat-ThroughPut gives an exact solution to the problem [33].*

3.3.2 Maximizing Pay-Off. Unlike throughput, when pay-off is maximized, there is an additional dimension involved that is different potentially for each deployment request. f_i for deployment request d_i is computed using $d^i.cost$, the amount of payment deployment d_i is willing to expend. Other than that, the rest of the algorithm remains unchanged.

Theorem 3. *Algorithm BatchStrat-PayOff has a 1/2-approximation factor [33].*

4 ADPAR

We discuss our solution to the ADPaR problem, that takes a deployment d and strategy set $\mathcal{S}$ as inputs, and is designed to recommend alternative deployment parameters d' to optimize the goal stated in Equation 3 (Section 2.3), such that d' satisfies the cardinality constraint of d.

Going back to Example 1 with the request d_2, StratRec there is no strategy that satisfies d_2 (refer to Figure 4a).

At a high level, ADPaR bears a resemblance to *Skyline and Skyband queries* [8, 16, 27] - but as we describe in Section 6, there are significant differences between these two problems - thus the former solutions do not adapt to solve ADPaR. Similarly, ADPaR is significantly different from existing works on query refinement [2, 11, 24, 25], that we further delineate in Section 6.

4.1 Algorithm ADPaR-Exact

Our treatment is geometric and exploits the monotonicity of our objective function (Equation 1 in Section 2.3). Even though the original problem is defined in a continuous space, we present a discretized technique that is exact. ADPaR-Exact, employs three sweep-lines [9], one for each parameter, quality, cost, and latency and gradually relaxes the parameters to produce the tightest alternative parameters that admit k strategies. By its unique design choice, ADPaR-Exact is empowered to select the parameter that is most suitable to optimize the objective function, and hence, produces exact solutions to ADPaR.

ADPaR-Exact has four main steps. Before getting into those details, we present a few simplifications to the problem for the purpose of elucidation. As we have described before, we normalize quality, cost, latency thresholds of a deployment or of a strategy in $[0, 1]$, and inverse quality to $(1 - quality)$. This step is just for unification, making our treatment for all three parameters uniform inside ADPaR, where smaller is better, and the deployment thresholds are considered as upper-bounds. With this, each strategy is a point in a 3-dimensional space and a deployment parameter (modulo its cardinality constraint) is an axis-parallel hyper-rectangle[9] in that space. Consider Figure 4a that shows the 4 strategies in Example 1 and d_2 as a hyper-rectangle.

Step-1 of ADPaR-Exact computes the relaxation (increment) that a deployment requires to satisfy a strategy among each deployment parameter. This is akin to computing $s_i.cost - d_2.cost$ (likewise for quality and latency) and when the strategy cost is smaller than the deployment threshold, it shows no relaxation is needed - hence we transform that to 0. The problem is studied for quality, cost, and latency (referred to as Q, C, L) (Table 3). It also initializes $d' = \{1, 1, 1\}$, the worst possible relaxation.

Step-2 of ADPaR-Exact involves sorting the strategies based on the computed relaxation values from step-1 in an increasing order across all parameters, as well as keeping track of the index of the strategies and the parameters of the relaxation values. The sorted relaxation scores are stored in list R, the corresponding I data structure provides the strategy index, and D provides the parameter value. In other words, $R[j]$ represents the j-th smallest relaxation value, where $I[j]$ represents the index of the strategy and $D[j]$ represents the parameter value corresponding to that. A cursor r is initialized to the first position in R (Table 4). Another data structure, a boolean matrix M of size $|\mathcal{S}| \times 3$ (Table 2) is used that keeps track of the number of strategies that are covered by the current movement of cursor r in list R. This matrix is initialized to 0 and the entries are updated to 1, as r advances.

Step-3 involves designing three sweep-lines along Q, C, and L (Table 5). A sweep line is an imaginary vertical line which is swept across the plane rightwards. The Q sweep-line sorts the $\mathcal{S}$ in C L plane in increasing order of Q (the other two works in a similar fashion). ADPaR-Exact sweeps the line as it encounters strategies, in order to discretize the sweep. At the beginning, each sweep-line points the k-th strategy along Q, C, L, respectively. d' is updated and contains the current Q, C, L value i..e, $d'.quality$ = Q, $d'.cost$ = C, and $d'.latency$ = L. Cursor r points to the smallest of these three values in R. Matrix M is updated to see what parameters of which strategies are covered so far.

At step-4, ADPaR-Exact checks if the current d' covers k strategies or not. This involves reading through I and checking if there exists k strategies such that for each strategy $s.quality \leq d'.quality$ and $s.cost \leq d'.cost$ and $s.latency \leq d'.latency$. If there are not k such strategies, it advances r to the next position and resets $d' = \{1, 1, 1\}$ again.

If there are more than k strategies, the new d', however, does not ensure that it is the tightest one to optimize Equation 3. Therefore, ADPaR-Exact cannot halt. ADPaR-Exact needs to check if there exists another d'' that still covers k strategies better than d'. This can indeed happen as we are dealing with a 3-dimensional problem and these three values in combination determine the objective function.

ADPaR-Exact takes a turn in considering the current values of each parameter based on d', and creates a projection on the corresponding 2-D plane, for the fixed value of the third parameter. Figure 4b shows an example in (Q, L) plane for a fixed cost. It then considers all strategies whose $s.cost \leq d'.cost$. After that, it finds the largest expansion among the two parameters such that this new d'' covers k strategies. This gives rise to three new deployment parameters, d''_C, d''_Q, d''_L. It chooses the best of these three and updates d'. At this point, it checks if M has k strategies covered. If it does, it stops processing and returns the new d' and the k strategies. If it does not, it advances the cursor r to the right.

Using Example 1, the alternative parameters are (0.75, 0.5, 0.28) for d_2 and s_1, s_2, s_3 are returned.

Lemma 1. *To cover k strategies, d′ needs to be initialized at least to the k^{th} smallest values on each paramete [33].*

Lemma 2. *Going by the relaxation value and parameter order of R and D, it ensures the tightest increase in the objective function in ADPaR-Exact [33].*

Theorem 4. *ADPaR-Exact produces an exact solution to the ADPaR problem [33].*

Running Time: Step-1 of Algorithm ADPaR-Exact takes $O(|\mathcal{S}|)$. Step-2 and 3 are dominated by sorting time, which takes $O(|\mathcal{S}|\ log|\mathcal{S}|)$. Step-4 is the most time-consuming and takes $O(|\mathcal{S}^3|)$. Therefore, the overall running time of the algorithm is cubic to the number of strategies.

	Cost	Quality	Latency
s_1	0	0	1
s_2	0	0	1
s_3	0	0	0
s_4	0	0	0

Table 2: matrix M

	Cost	Quality	Latency
s_1	0.3	0.05	0
s_2	0.05	0.13	0
s_3	0	0.3	0
s_4	0	0.38	0

Table 3: Step 1

Relaxation R	0	0	0	0	0	0
Strategy Index I	1	2	3	4	3	4
Parameter D	L	L	L	L	C	C
Relaxation R	0.05	0.05	0.13	0.3	0.3	0.38
Strategy Index I	1	2	2	1	3	4
Parameter D	Q	C	Q	C	Q	Q

Table 4: Step 2

sweep-line(Q)	C, L plane	0.05	0.13	0.3	0.38
	$s.cost$	0.3	0.05	0	0
	$s.latency$	0	0	0	0
sweep-line(C)	Q, L plane	0	0	0.05	0.3
	$s.quality$	0.38	0.3	0.13	0.05
	$s.latency$	0	0	0	0
sweep-line(L)	C, Q plane	0	0	0	0
	$s.cost$	0.3	0.05	0	0
	$s.quality$	0.05	0.13	0.3	0.38

Table 5: Step 3

5 EXPERIMENTAL EVALUATION

In our real-world deployments, we estimate worker availability and demonstrate the need for optimization (Section 5.1). In synthetic data experiments (Section 5.2), we present results to validate the qualitative and scalability aspects of our algorithms.

5.1 Real Data Experiments

We perform two different real data experiments that involve workers from AMT focusing on text editing tasks. The first experiments (Section 5.1.1) empirically validate key assumptions in designing StratRec. the second experiments (Section 5.1.2) validate the effectiveness of StratRec when compared to the case where no recommendation is made.

5.1.1 Validating Key Assumptions. We consider two types of tasks: a) sentence translation (translating from English to Hindi) and text creation (writing 4 to 5 sentences on some topic) to validate the following questions:
1. Can worker availability be estimated and does it vary over time? We performed 3 different deployments for each task. The first deployment was done on the weekend (Friday 12am to Monday 12am), the second deployment was done at the beginning to the middle of the week (Monday to Thursday), the last one is from the middle of the week until the week-end (Thursday to Sunday). We design the HITs (Human Intelligence Tasks) in AMT such that each task needs to

Algorithm 2 Algorithm ADPaR-Exact for alternative deployment parameter recommendation

Require: $\mathcal{S}$, k, W, d, k.

1: Compute relaxation values $s.quality - d.quality$, $s.cost - d.cost$, $s.latency - d.latency$, $\forall s \in \mathcal{S}$.
2: Compute R by sorting $3|\mathcal{S}|$ numbers in increasing order.
3: Compute I and D accordingly.
4: Initialize M to all 0's and $d' = \{1, 1, 1\}$
5: Initialize Cursor $r = R[0]$
6: Sort (C L) , (Q L), and (Q C) planes based on the Q, C, L sweep-lines respectively.
7: x= k-th value in (C L), y= k-th value in (Q L), z= k-th value in (Q C) plane
8: Update d'= $\{x, y, z\}$
9: r = minimum $\{x, y, z\}$
10: Update matrix M
11: **if** d' covers $\geq k$ strategies **then**
12: Compute the best d'' better than d' that covers k strategies
13: **if** M covers k strategies **then**
14: $d' = d''$ and return
15: **if** M covers $< k$ strategies **then**
16: move r to the right
17: **if** d' covers $< k$ strategies **then**
18: Move r to the right
19: Update d''s one of the parameters by consulting R and D
20: go back to line 10

be undertaken by a maximum number of workers x. Worker availability is computed as the ratio of $\frac{x'}{x}$, where x' is the actual number of workers who undertook the task during the deployment time (although this does not fully conform to our formal worker availability definition, it is our sincere attempt to quantify worker availability using public platforms).

2. How does worker availability impact deployment parameters? We need to be able to calculate the quality, cost, and latency, along with worker availability. Latency and cost are easier to calculate, basically, it is the total amount of money that was paid to workers and the total amount of time the workers used to make edits in the document. Since text editing tasks are knowledge-intensive, to compute the quality of the crowd contributions, we ask a domain expert to judge the quality completed tasks as a percentage. Once worker availability, quality, cost, and latency are computed, we perform curve fitting that has the best fit to the series of data points.

3. How do deployment strategies impact different task types? We deployed both types of text editing tasks using two different deployment strategies *SEQ-IND-CRO* and *SIM-COL-CRO* that were shown to be effective with more than 70% of quality

Original Text	Mary had a little lamb, little lamb, little lamb, Mary had a little lamb, its fleece was white as snow. Everywhere that Mary went, Mary went, Mary went, Everywhere that Mary went, the lamb was sure to go.	*Lavender's blue, dilly dilly,* *Lavender's green* *When you are king, dilly dilly,* *I shall be queen*	Rock-a-bye, baby, in the treetop When the wind blows, the cradle will rock When the bough breaks, the cradle will fall And down will come baby, cradle and all
Sequential -independent- crowd	मेरी ने एक भेड पाली, भेड पाली, भेड पाली, मेरी ने एक भेड पाली सफेद बाली वाली. जहाँ भी मेरी जाती थी, जाती थी, जाती थी, जहाँ भी मेरी जाती थी, वो पीछे आती थी.	लैवेंडर की नीली, गहरी नीली, लैवेंडर का हरा जब आप राजा होते हैं, तो आप बहुत खुश होते हैं। मैं रानी बनूंगी	रॉक-ए-बाय, बेबी, ट्रीटोप में जब हवा चलेगी, तो खड़खड़ाहट उठेगी जब कड़ा फटेगा तो खटिया गिर जाएगी और नीचे आएगा बच्चा, पालना और सब
simultaneous - collaborative- crowd	मैरी के पास एक छोटा सा मेमना था, थोड़ा सा मेमना, थोड़ा सा भेड़ का बच्चा, मैरी के पास थोड़ा सा मेमना था, उसका ऊन बर्फ की तरह सफेद था। हर जगह मैरी चली गई, मैरी चली गई, मैरी चली गई, हर जगह मैरी चली गई, मेमने का जाना निश्चित था	लैवेंडर की नीली, गहरी नीली, लैवेंडर का हरा जब आप राजा होते हैं, तो आप बहुत खुश होते हैं। मैं रानी बनूंगी	रॉक-ए-बाय, बेबी, ट्रीटोप में जब हवा चलेगी, तो खड़खड़ाहट उठेगी जब कड़ा फटेगा तो खटिया गिर जाएगी और नीचे आएगा बच्चा, पालना और सब

Figure 5: Translation: Original Texts and Translation

score for short texts [5]. Since our effort here was to evaluate the effectiveness of these two strategies considering quality, cost, and latency, we did not set values for deployment parameters and we simply observed them through experimentation.

Tasks and Deployment Design: We chose three popular English nursery rhymes for sentence translation. Each rhyme consisted of 4-5 lines that were to be translated from English to Hindi (one such sample rhyme is shown in Figure 5). For text creation, we considered three popular topics, *Robert Mueller Report*, *Notre Dame Cathedral*, and *2019 Pulitzer prizes*. One sample text creation is shown in Figure 6.

We designed three deployment windows at different days of the week. Unlike micro-tasks in AMT, text editing tasks require significantly more time to complete (we allocated 2 hours per HIT). A HIT contains either 3 sentence translation tasks or three text creation tasks as opposed to micro-tasks, where a HIT may contain tens of tasks. For each task type, we validated 2 deployment strategies - in *SEQ-IND-CRO*, workers were to work in sequence and independently, whereas, in *SIM-COL-CRO*, workers were asked to work simultaneously and collaboratively. We created 2 different samples of the same study resulting in a total of 8 HITs deployed inside the same window. Each HIT was asked to be completed by 10 workers paid $2 each if the worker spent enough time (more than 10 minutes). This way, a total of 80 unique workers were hired for each deployment window, and a total of 240 workers were hired for all three deployments.

Worker Recruitment: For both task types, we recruited workers with a HIT approval rate greater than 90%. For

Strategy	TOPIC	TEXT
Sequential - independent-crowd	*Robert Mueller report*	*The Mueller Report, formally titled the Report on the Investigation into Russian Interference in the 2016 Presidential Election, is the official report documenting the findings of the Special Counsel investigation, led by Robert Mueller, into Russian efforts to interfere in the 2016 United States presidential election, allegations of conspiracy or coordination between Donald Trump's presidential campaign and Russia, and allegations of obstruction of justice. The report was submitted to Attorney General William Barr on March 22, 2019. This report addressed obstruction of justice, stating it "does not conclude that the President committed a crime, [and] it also does not exonerate him".*
simultaneous - collaborative-crowd	*Robert Mueller report*	*It was a report related to United States counterintelligence investigation of the Russian government's efforts to interfere in the 2016 presidential election. As of April 2019, thirty-four individuals were indicted by Special Counsel investigators. Eight have pled guilty to or been convicted of felonies, including at least five Trump associates and campaign officials. The report concluded that Russian interference in the 2016 presidential election did occur and "violated U.S. criminal law."*

Figure 6: Text Creation: Robert Mueller Report

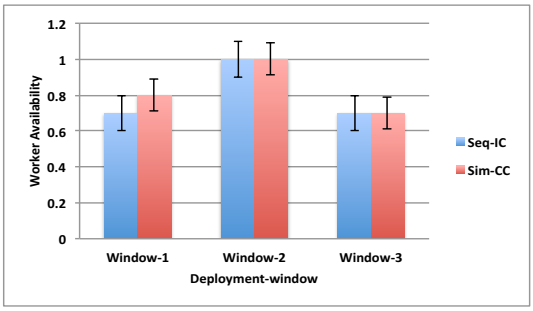


Figure 7: Worker Availability Estimation

sentence translation, we additionally filtered workers on geographic locations, either US or India. For text creation tasks, we recruited US-based workers with a Bachelor's degree.

Enabling collaboration: After workers were recruited from AMT, they were directed to Google Docs where the tasks were described and the workers were given instructions. The docs were set up in editing mode, so edits could be monitored.

Experiment Design: An experiment is comprised of three steps. In Step-1, all initially recruited workers went through qualification tests. For text creation, a topic (Royal Wedding) was provided and the workers were asked to write 5 sentences related to that topic. For sentence translation, the qualification test comprised of 5 sample sentences to be translated from English to Hindi. Completed qualification tests were evaluated by domain experts and workers with more than 80% or more qualification scores were retained and invited to work on the actual HITs. In Step-2, actual HITs were deployed for 72 hours and the workers were allotted 2 hours for the tasks. In Step-3, after 72 hours of deployment, results were aggregated by domain experts to obtain a quality score. Cost and latency were easier to calculate directly from the raw data.

Summary of Results: Our first observation is that worker availability *can be estimated and does vary over time* (standard error bars added). We observed that for both task types, workers were more available during Window 2 (Monday-Thursday), compared to the other two windows. Detailed results are shown in Figure 7.

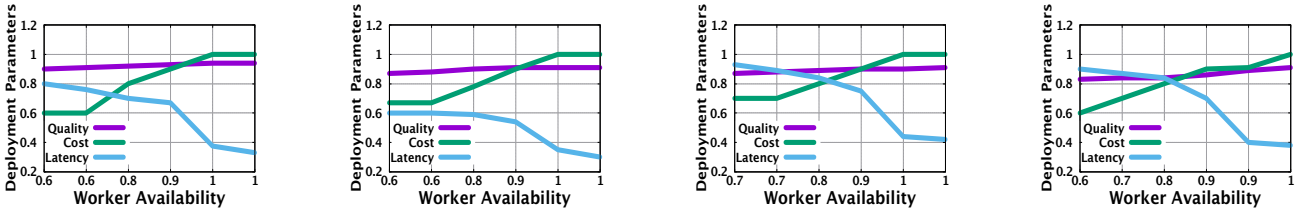
Our second observation is that each deployment parameter has a linear relationship with worker availability for text editing tasks. Quality and cost increase linearly with worker availability. Latency decreases with increasing worker availability. This linear relationship could be captured and the parameters (α, β) could be estimated. Table 6 presents these results and the estimated (α, β) always lie within 90% confidence interval of the fitted line.

Our final observation is that *SEQ-IND-CRO* performs better than *SIM-COL-CRO* for both task types. However, this difference is not statistically significant. On the other hand, *SEQ-IND-CRO* has higher latency. Upon further analysis, we observe that when workers are asked to collaborate and edit simultaneously, that gives rise to an edit war and an overall poor quality. Figure 8 presents these results.

Worker Availability and Deployment Parameters		
Task-Strategy	Parameters	α,β
Translation *SEQ-IND-CRO*	Quality	0.09, 0.85
	Cost	1.00, 0.00
	Latency	−0.98, 1.40
Translation *SIM-COL-CRO*	Quality	0.09, 0.82
	Cost	0.82, 0.17
	Latency	−0.63, 1.01
Creation *SEQ-IND-CRO*	Quality	0.10, 0.80
	Cost	1.00, 0.00
	Latency	−1.56, 2.04
Creation *SIM-COL-CRO*	Quality	0.19, 0.70
	Cost	1.00, −0.00
	Latency	−1.38, 1.81

Table 6: α, β Estimation

5.1.2 Validating the Effectiveness of StratRec. We are unable to ask specific user (task designer's) satisfaction questions in this experiment, simply because AMT does not allow to recruit additional task designers and only workers could be recruited. For this purpose, we deploy 10 additional sentence translation (translating nursery rhymes from English to Hindi) and 10 additional text creation tasks considering a set of 8 strategies.



(a) Translation *SEQ-IND-CRO* (b) Translation *SIM-COL-CRO* (c) Creation *SEQ-IND-CRO* (d) Creation *SIM-COL-CRO*

Figure 8: Relationship Between Deployment Parameters and Worker Availability

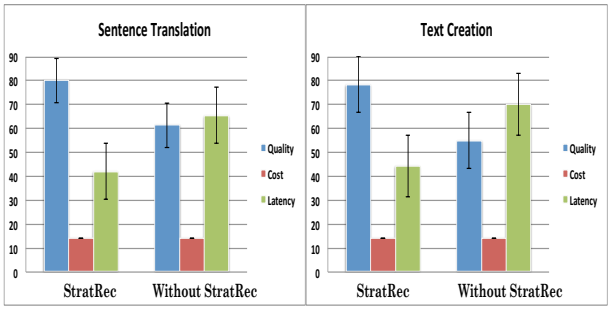


Figure 9: Average Quality, Cost, Latency Comparison of Deployments with and without StratRec

We create 2 mirror deployments for the same task (one according to StratRec recommendation and the other without) resulting in a total of 40 HITs deployed. For the latter scenario, the deployments were not recommended any structure, organization, or style and the workers were given the liberty to complete the task the way they preferred. Each HIT was asked to be completed by 7 workers paid $2 each if the worker spent enough time (more than 10 minutes). This way, a total of 280 unique workers are hired during this experiment. The quality, cost, and latency thresholds of each deployment are set to be 70%, $14, 72 hours.

The worker recruitment, and the rest of the experiment design, and result aggregation steps are akin to those steps that are described in Section 5.1.1. Figure 9 represents the average quality, cost, and latency results of these experiments with statistical significance.

Summary of Results: We have two primary observations from these experiments. **Our first observation is** that (Figure 9), when tasks are deployed considering recommendation of StratRec, with statistical significance, they achieve higher quality and lower latency, under the fixed cost threshold on an average compared to the deployments that do not consult StratRec. These results validate the effectiveness of deployment recommendations of our proposed framework and its algorithms.

Our second observation (upon further investigating the Google Docs where the workers undertook tasks), is that the deployments that do not consider StratRec recommendations have more edits, compared to that are deployed considering StratRec. In fact, on average, StratRec deployments have an average of 3.45 edits for sentence translation, compared to 6.25 edits on average for those deployed with no recommendations. Indeed, when workers were not guided, they repeatedly overrode each other's contributions, giving rise to an edit war.

5.2 Synthetic Experiments

We aim to evaluate the qualitative guarantees and the scalability. Algorithms are implemented in Python 3.6 on Ubuntu 18.10. Intel Core i9 3.6 GHz CPU, 16GB of memory.

5.2.1 Implemented Algorithms. We describe different algorithms that are implemented.

`Batch Deployment Algorithms. Brute Force:` An exhaustive algorithm which compares all possible combinations of deployment requests and returns the one that optimizes the objective function.
`BaselineG:` This algorithm sorts the deployment requests in decreasing order of $\frac{f_i}{\vec{w}_i}$ and greedily selects requests until worker availability W is exhausted.
BatchStrat: Our proposed solution described in Section 3.

ADPaR `Algorithms. ADPaRB:` This is a brute force algorithm that examines all sets of strategies of size k. It returns the one that has the smallest distance to the task designer's original deployment parameters. While it returns the exact answer, this algorithm takes exponential time to run.
`Baseline2:` This baseline algorithm is inspired by a related work [24]. The main difference though, the related work modifies the original deployment request by just one parameter at a time and is not optimization driven. In contrast, ADPaR-Exact returns an alternative deployment request, where multiple parameters may have to be modified.

Baseline3: This one is designed by modifying space partitioning data structure R-Tree [3]. We treat each strategy parameters as a point in a 3-D space and index them using an R-Tree. Then, it scans the tree to find if there is a minimum bounding box (MBB) that exactly contains k strategies. If so, it returns the top-right corner of that MBB as the alternative deployment parameters and corresponding k strategies. If such an MBB does not exist, it will return the top right corner of another MBB that has at least k strategies and will randomly return k strategies from there.
ADPaR-Exact: Our proposed solution in Section 4.

Summary of Results: Our simulation experiments highlight the following findings: **Observation 1:** Our solution BatchStrat returns exact answers for throughput optimization, and the approximation factor for pay-off maximization is always above 90%, significantly surpassing its theoretical approximation factor of 1/2. **Observation 2:** Our solution BatchStrat is highly scalable and takes less than a second to handle millions of strategies, and hundreds of deployment requests, and k. **Observation 3:** Our algorithm ADPaR-Exact returns exact solutions to the ADPaR problem, and significantly outperforms the two baseline solutions in objective function value. **Observation 4:** ADPaR-Exact is scalable and takes a few seconds to return alternative deployment parameters, even when the total number of strategies is large and k is sizable.

5.2.2 *Quality Experiment.*

Batch Deployment Recommendation. Goal: We validate the following two aspects: (i) *how many deployment requests* BatchStrat *can satisfy without invoking* ADPaR? (ii) *How does* BatchStrat *fare to optimize different platform-centric goals?* We compare BatchStrat with the other two baselines, as appropriate.

Strategy Generation: The dimension values of a strategy are generated considering uniform and normal distributions. For the normal distribution, the mean and standard deviation are set to 0.75 and 0.1, respectively. We randomly pick the value from 0.5 to 1 for the uniform distribution.

Worker Availability: For a strategy, we generate α uniformly from an interval $[0.5, 1]$. Then, we set $\beta = 1 - \alpha$ to make sure that the estimated worker availability W is within $[0, 1]$. These numbers are generated in consistence with our real data experiments.

Deployment Parameters: Once W is estimated, the quality, latency, and cost - i.e., the deployment parameters, are generated in the interval $[0.625, 1]$. For each experiment, 10 deployment parameters are generated, and an average of 10 runs is presented in the results.

Figure 10 shows the percentage of satisfied requests by BatchStrat with varying k, m, $|\mathcal{S}|$, W. In general, normal distribution performs better than uniform. Upon further analysis, we realize that normal distribution has a very small standard deviation, and is thereby able to satisfy more requests. As shown in Figure 10(a), the percentage of satisfied requests decreases with increasing k, which is expected. Contrarily, the effect of increasing batch size m is less pronounced. This is because all requests use the same underlying distribution, allowing BatchStrat to handle more of them. With more strategies $|\mathcal{S}|$, as Figure 10(c) illustrates, BatchStrat satisfies more requests, which is natural, because with increasing $|\mathcal{S}|$, it simply has more choices. Finally, in Figure 10(d), with higher worker availability BatchStrat satisfies more requests. By default, we set $|\mathcal{S}| = 10000, m = 10, k = 10, W = 0.5$.

Figure 11 shows the results of throughput of BatchStrat by varying k, m,$|\mathcal{S}|$, compare with the two baselines. Figure 12 shows the approximation factor of BatchStrat and BaselineG. BatchStrat achieves an approximation factor of 0.9 most of the time. For both experiments, the default values are $k = 10, m = 5, |\mathcal{S}| = 30, W = 0.5$ because brute force does not scale beyond that.

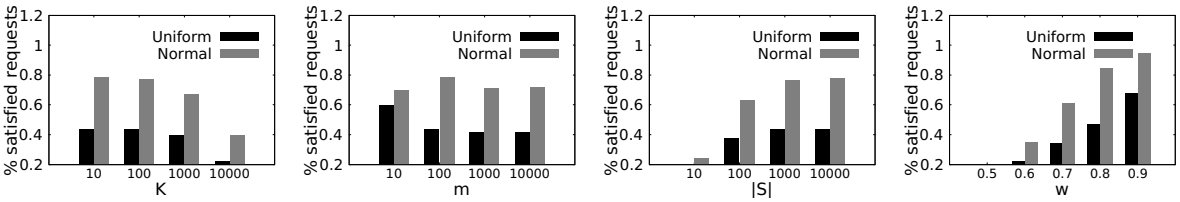
Alternative Deployment Recommendation ADPaR. The goal here is to measure the objective function. Since ADPaRB takes exponential time, to be able to compare with this, we set $|\mathcal{S}| = 20, k = 5, W = 0.5$ for all the quality experiments that has to compare with the brute force. Otherwise, the default values are $|\mathcal{S}| = 200, k = 5$.

In Figure 13, we vary $|\mathcal{S}|$ and k and plot the Euclidean distance between d and d' (smaller is better). Indeed, ADPaR-Exact returns exact solution always. The other two baselines perform significantly worse, while Baseline 3 is the worst. That is indeed expected, because these two baselines are not optimization guided, and does not satisfy our goal. Naturally, the objective function decreases with increasing $|\mathcal{S}|$, because more strategies mean smaller change in d', making the distance between d and d' smaller. As the results depict, optimal Euclidean distance between d and d' increases with increasing k, which is also intuitive, because, with higher k value, the alternative deployment parameters are likely to have more distance from the original ones.

5.2.3 *Scalability Experiments.*

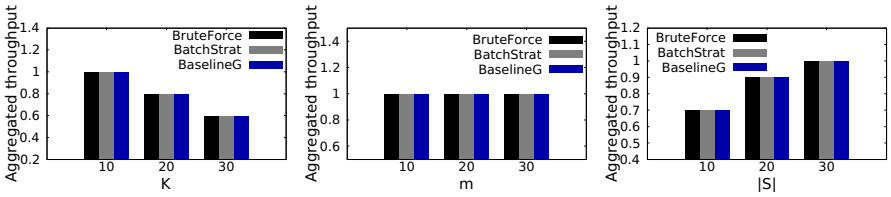
Our goal is to evaluate the running time of our proposed solutions. Running time is measured in seconds. We present a subset of results that are representative.

Batch Deployment Recommendation. Since the BaselineG has the same running time as that of BatchStrat (although qualitatively inferior), we only compare the running time between Brute Force and BatchStrat. The default setting for $|\mathcal{S}|$, k and W are 30, 10 and 0.75, respectively.



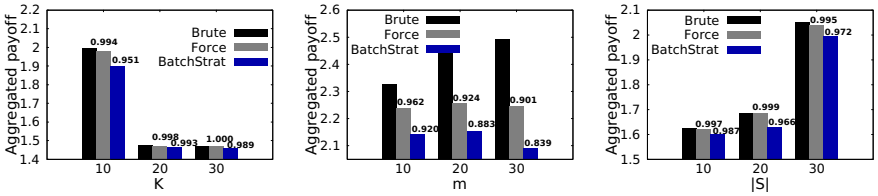
(a) Varying k (b) Varying m (c) Varying $\mathcal{S}$ (d) Varying W

Figure 10: Percentage of satisfied requests before invoking ADPaR



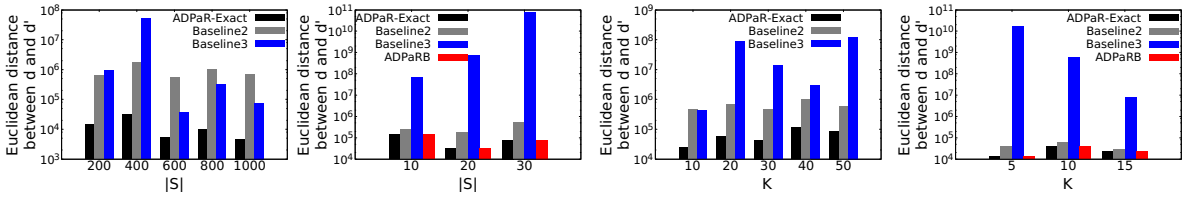
(a) Varying k (b) Varying m (c) Varying $\mathcal{S}$

Figure 11: Objective Function for Throughput



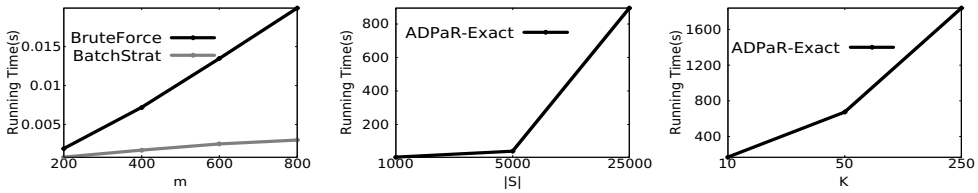
(a) Varying k (b) Varying m (c) Varying $\mathcal{S}$

Figure 12: Objective Function and Approximation Factor for Payoff



(a) without Brute Force (b) with Brute Force (c) without Brute Force (d) with Brute Force

Figure 13: Quality Experiments for ADPaR



(a) Batch Deployment Varying m (b) ADPaR Varying $|\mathcal{S}|$ (c) ADPaR Varying k

Figure 14: Scalability Experiments

The first observation we make is, clearly BatchStrat can handle millions of strategies, several hundreds of batches, and very large k and still takes only a few fractions of seconds to run. It is easy to notice that the running time of this problem only relies on the size of the batch m (or the number of deployment requests), and not on k or $\mathcal{S}$. As we can see in Figure 14a, `Brute Force` takes exponential time with increasing m, whereas BatchStrat scales linearly.

Alternative Deployment Recommendation. We vary k and $|\mathcal{S}|$ with defaults set to 5 and 10000 respectively, and evaluate the running time of ADPaR-Exact. W is set to 0.5. As Figures 14b and 14c attest, albeit non-linear, ADPaR-Exact scales well with k and $|\mathcal{S}|$. We do not present the baselines as they are significantly inferior in quality.

6 RELATED WORK

Crowdsourcing Deployment: Till date, the burden is entirely on the task requester to design appropriate deployment strategies that are consistent with the cost, latency, and quality parameters of task deployment. A very few related works [1, 35] have started to study the importance of appropriate deployment strategies but these works do not propose an algorithmic solution and are limited to empirical studies. A recent work [13] presents the results of a 10-month deployment of a crowd-powered system that uses a hybrid approach to fast recruitment of workers, called *Ignition*. These results suggest a number of opportunities to deploy work in the online job market.

Crowdsourcing Applications: A number of interactive crowd-powered systems have been developed to solve difficult problems and develop applications [4, 7, 12, 18–20, 23, 28, 31]. For instance, Soylent uses the crowd to edit and proofread text [4]; Chorus recruits a group of workers to hold sophisticated conversations [22]; and Legion allows a crowd to interact with a UI-control task [21]. A primary challenge for such interactive systems is to decrease latency without having to compromise with the quality. A comprehensive survey on different crowdsourcing applications could be found at [34]. All crowd-powered systems share these challenges and are likely to benefit from StratRec.

Query planning and Refinement: The closest analogy of deployment strategy recommendation is recommending the best query plan in relational databases, in which joins, selections and projections could be combined any number of times. Typical parametric query optimization problems, like [15], only focus on one objective to optimize. Afterward, multi-objective problems have been studied, with a focus on optimizing multiple objectives at the same time [32]. Our work borrows inspiration from that and studies the problem in the deployment context, making the challenges unique and different from traditional query planning.

Query reformulation has been widely studied in Information Retrieval [11]. In [24], authors take users' preference into account and propose an interactive method for seeking an alternative query which satisfies cardinality constraints. This is different from ADPaR since it only relaxes one dimension at a time. Aris et al. [2] proposed a graph modification method to recommend queries that maximize an overall utility. Mottin et al. [25] develop an optimization framework where solutions can only handle Boolean/categorical data.

Skyline and Skyband Queries: Skyline queries play an essential role in computing favored answers from a database [6, 8]. Based on the concepts of skylines, other classes of queries arise, especially top-k queries and k-skyband problems which aim to bring more useful information than original skylines. Mouratidis et al. [26, 27] study several related problems. In [26], sliding windows are used to track the records in dynamic stream rates. In [27], a geometry arrangement method is proposed for top-k queries with uncertain scoring functions. Because our problem seeks the optimal group of k strategies, it is similar to the top-k queries problem. However, unlike Skyband or any other related work, ADPaR recommends alternative deployment parameters. Thus, these solutions do not extend to solve ADPaR.

7 CONCLUSION

We propose an optimization-driven middle layer to recommend deployment strategies. Our work addresses multi-faceted modeling challenges through the generic design of modules in StratRec that could be instantiated to optimize different types of goals by accounting for worker availability. We develop computationally-efficient algorithms and validate our work with extensive real data and synthetic experiments.

This work opens up several important ongoing and future research directions. As an ongoing investigation, we are deploying additional types of tasks using StratRec to evaluate its effectiveness. Our future investigation involves adapting batch deployment to optimize additional criteria, such as worker-centric goals, or to combine multiple goals inside the same optimization function. Understanding the computational challenges of such an interactive system remains to be explored. Finally, how to design StratRec for a fully dynamic stream-like setting of incoming deployment requests, where the deployment requests could be revoked, remains to be an important open problem.

ACKNOWLEDGMENTS

The work of Dong Wei and Senjuti Basu Roy are supported by the National Science Foundation, CAREER Award #1942913, IIS #1814595, and by the Office of Naval Research Grant No: N000141812838.

REFERENCES

[1] BJ Allen et al. 2018. Design Crowdsourcing: The Impact on New Product Performance of Sourcing Design Solutions from the Crowd. *Journal of Marketing* (2018).

[2] Aris Anagnostopoulos et al. 2010. An optimization framework for query recommendation. (2010).

[3] Norbert Beckmann et al. 1990. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD*. Acm.

[4] Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. 2010. Soylent: A Word Processor with a Crowd Inside. In *IN PROC UIST'10*.

[5] Ria Mae Borromeo et al. 2017. Deployment strategies for crowdsourcing text creation. *Information Systems* (2017).

[6] Stephan Borzsony et al. 2001. The skyline operator. In *ICDE*. IEEE.

[7] Lydia B Chilton, Greg Little, Darren Edge, Daniel S Weld, and James A Landay. 2013. Cascade: Crowdsourcing taxonomy creation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1999–2008.

[8] Jan Chomicki et al. 2013. Skyline queries, front and back. *SIGMOD* (2013).

[9] Mark De Berg et al. 1997. Computational geometry. In *Computational geometry*. Springer.

[10] Michael R Garey and David S Johnson. 2002. *Computers and intractability*. wh freeman New York.

[11] Susan Gauch et al. 1991. *Search improvement via automatic query reformulation.* Technical Report. UNC Chapel Hill, Computer Science.

[12] Benjamin M Good and Andrew I Su. 2013. Crowdsourcing for bioinformatics. *Bioinformatics* 29, 16 (2013), 1925–1933.

[13] Ting-Hao Kenneth Huang and Jeffrey P Bigham. 2017. A 10-month-long deployment study of on-demand recruiting for low-latency crowdsourcing. In *Fifth AAAI Conference on Human Computation and Crowdsourcing*.

[14] Oscar H Ibarra et al. 1975. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM (JACM)* (1975).

[15] Yannis E Ioannidis, Raymond T Ng, Kyuseok Shim, and Timos K Sellis. 1992. Parametric query optimization. In *VLDB*, Vol. 92. Citeseer, 103–114.

[16] Wen Jin et al. 2007. The multi-relational skyline operator. In *ICDE*. IEEE.

[17] Ouiame Ait El Kadi. [n.d.]. Exploring Crowdsourcing Deployment Strategies through Recommendation and Iterative Refinement. *MS Research Report* ([n. d.]).

[18] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E Kraut. 2011. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 43–52.

[19] Anand Kulkarni, Matthew Can, and Björn Hartmann. 2012. Collaboratively crowdsourcing workflows with turkomatic. In *Proceedings of the acm 2012 conference on computer supported cooperative work*. ACM, 1003–1012.

[20] Anand P Kulkarni, Matthew Can, and Bjoern Hartmann. 2011. Turkomatic: automatic recursive task and workflow design for mechanical turk. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2053–2058.

[21] Walter S Lasecki, Raja Kushalnagar, and Jeffrey P Bigham. 2014. Legion scribe: real-time captioning by non-experts. In *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility*. ACM, 303–304.

[22] Walter S Lasecki, Rachel Wesley, Jeffrey Nichols, Anand Kulkarni, James F Allen, and Jeffrey P Bigham. 2013. Chorus: a crowd-powered conversational assistant. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 151–162.

[23] Christopher H Lin, Mausam Daniel, and S Weld. 2012. Dynamically switching between synergistic workflows for crowdsourcing. In *In Proceedings of the 26th AAAI Conference on Artificial Intelligence, AAAI'12*. Citeseer.

[24] Chaitanya Mishra et al. 2009. Interactive query refinement. In *EDBT*. ACM.

[25] Davide Mottin et al. 2013. A probabilistic optimization framework for the empty-answer problem. *VLDB* (2013).

[26] Kyriakos Mouratidis et al. 2006. Continuous monitoring of top-k queries over sliding windows. In *SIGMOD*. ACM.

[27] Kyriakos Mouratidis and Bo Tang. 2018. Exact Processing of Uncertain Top-k Queries in Multi-criteria Settings. *PVLDB* (2018).

[28] Barzan Mozafari, Purna Sarkar, Michael Franklin, Michael Jordan, and Samuel Madden. 2014. Scaling up crowd-sourcing to very large datasets: a case for active learning. *Proceedings of the VLDB Endowment* 8, 2 (2014), 125–136.

[29] Julien Pilourdault et al. 2017. Motivation-aware task assignment in crowdsourcing. In *EDBT*.

[30] Habibur Rahman et al. 2018. Optimized group formation for solving collaborative tasks. *The VLDB Journal* (2018), 1–23.

[31] Klaas-Jan Stol and Brian Fitzgerald. 2014. Two's company, three's a crowd: a case study of crowdsourcing software development. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 187–198.

[32] Immanuel Trummer and Christoph Koch. 2016. Multi-objective parametric query optimization. *ACM SIGMOD Record* 45, 1 (2016), 24–31.

[33] Dong Wei, Senjuti Basu Roy, and Sihem Amer-Yahia. 2020. Recommending Deployment Strategies for Collaborative Tasks. *arXiv preprint arXiv:2003.06875* (2020).

[34] Man-Ching Yuen, Irwin King, and Kwong-Sak Leung. 2011. A survey of crowdsourcing systems. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*. IEEE, 766–773.

[35] Haichao Zheng et al. 2011. Task design, motivation, and participation in crowdsourcing contests. *International Journal of Electronic Commerce* (2011).