Task Deployment Recommendation with Worker Availability

Dong Wei
NJIT
New Jersey, USA
dw277@njit.edu

Senjuti Basu Roy
NJIT
New Jersey, USA
senjutib@njit.edu

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

***Abstract*—We study recommendation of deployment strategies to task requesters that are consistent with their deployment parameters: 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. We propose BatchStrat, an optimization-driven middle layer that recommends deployment strategies to a batch of requests by accounting for worker availability. We develop computationally efficient algorithms to recommend deployments that maximize task throughput and pay-off, and empirically validate its quality and scalability.**

I. Introduction

In crowdsourcing, task deployment is an important process that requesters undertake with very little help. Task deployment requires to specify not only the tasks, but also identify *appropriate deployment strategies*. A strategy involves the interplay of 3 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* that are typically provided as thresholds on quality (lower-bound), latency (upper-bound), and budget (upper-bound). For example, for a sentence translation task, a task designer 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 are consistent with desired deployment parameters. Our effort in this paper is to present a formalism and a computationally efficient algorithm to assist requesters by recommending k strategies that best achieve the desired deployment parameters.

A recent work [1] has empirically investigated the deployment of text creation tasks in Amazon Mechanical Turk (AMT). The authors validated the effectiveness of different strategies for different types of tasks such as text summarization and text translation, and provided empirical evidence for the need to guide requesters in choosing the right strategy. In this paper, we propose *to automate this strategy recommendation process*. This is particularly challenging because the estimation of the cost, quality and latency of a strategy must account for *worker availability* on the platform. Our contributions are: *we express deployment strategies as a function of worker availability, we formalize optimization problems to enable recommendation to a batch of requests, present principled algorithms, and validate them experimentally.*

We propose BatchStrat that consumes a batch of deployment requests, coming from different requesters and recommends strategies to them to optimize different goals. BatchStrat studies these incoming requests to obtain best deployment strategies given worker availability. If the platform does not have enough workers for all requests, it triages them by optimizing platform-centric goals, i.e., to maximize throughput or pay-off.

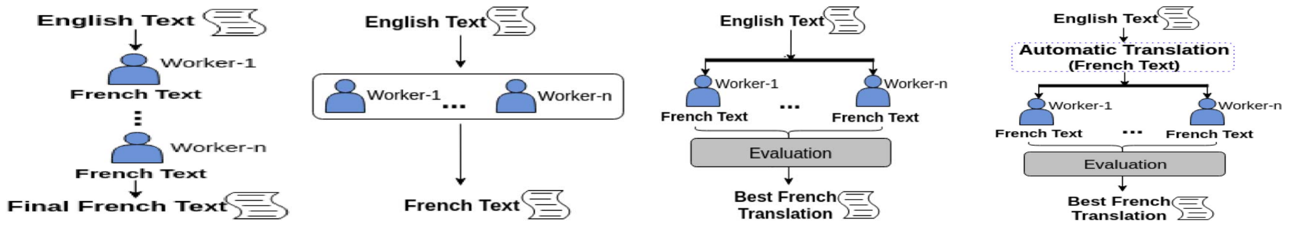
II. Data Model and Problem

Deployment Strategies: A deployment strategy [4] 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 [1], [4] and refer to them as $\mathcal{S}$. Figure 1 enlists some strategies that are suitable for text translation tasks (from English to French in this example). For instance, *SEQ-IND-CRO* in Figure 1(a) dictates that workers complete tasks sequentially (*SEQ*), independently (*IND*) and with no help from algorithms (*CRO*). In *SIM-COL-CRO* (Figure 1(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. We assume that for a given platform, the set of strategies is given and bounded by $|\mathcal{S}|$.

Deployment Parameters: A deployment request d has three parameters, $d.cost$, $d.quality$, and $d.latency$. Using Example 1, the minimum quality of request is 40%, the latency 2 days, and the budget $\$100$. These thresholds are referred to as deployment parameters. The goal is find one or more strategies (notationally k, a small integer) for each d, such that, for each strategy s, when d is deployed using s, it satisfies the deployment parameters.

Worker Availability: Worker availability/ available workforce W is a value in $[0, 1]$ which represents the proportion of workers who are available to undertake a task deployed by a requester within the specified time $d.latency$.

The work of Dong Wei and Senjuti Basu Roy are supported by the National ScienceFoundation under Grant No.: 1814595 and Office of Naval Research under Grant No.: N000141812838



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

Figure 1: Deployment Strategies

	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 I: Deployment Requests and Strategies

Modeling Strategy Parameters: The deployment recommendation process must be capable to estimate the parameters of a strategy s (cost, quality, latency if s is used to deploy d) to check if it is suitable for a deployment d. Since the deployed tasks are to be done by workers who are available and qualified to undertake the deployed tasks, the estimated strategy parameters, i.e., (estimated) quality, cost and latency of a strategy is a function of worker availability and task type.

Example 1: Assume there are 3 ($m = 3$) task deployment requests for different types of text editing tasks. Requester-1 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 I presents all 3 deployment requests after normalization between $[0-1]$. For the purpose of this example, we assume worker availability W to be 0.8 for the next 7 days and set $k = 3$.

For the purpose of illustration, $\mathcal{S}$ consists of the set of 4 deployment strategies, as shown in Figure 1: *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. s_1 costs \$150 and takes 2 days (out of 7 days) and ensures at least a 50% quality. s_2, s_3, s_4 have corresponding parameters. Strategy parameters are normalized and presented in the last column of Table I.

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 as follows:

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

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 for $k = 3$, as it will return $S_d^3 = \{s_2, s_3, s_4\}$, such that $d_3.cost \geq s^4.cost \geq s^3.cost \geq s^2.cost$ and $d_3.latency \geq s^4.latency \geq s^3.latency \geq s^2.latency$ and $d_3.quality \leq s^4.quality \leq s^3.quality \leq s^2.quality$, because 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:* It maximizes the total number of successful strategy recommendations without exceeding W. Formally speaking,

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

- *Pay-off:* It 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.

III. Batch Deployment Recommendation

Before getting into the details, we provide an abstraction which serves the purpose of designing BatchStrat, our proposed solution: given m deployment requests and W workforce availability, Problem 1 could be modeled in the form of 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.

A particular cell $w_{ij} \in \mathcal{W}$ corresponds to how much workforce is needed to deploy request i with strategy j. The challenge, however, is that each d has three different requirements of quality, cost, and latency. Therefore, it has to estimate workforce requirement per (deployment, strategy) pair first. That constitutes step-1 of BatchStrat. Once for every (i, j), a (deployment, strategy) workforce requirement $w_{i,j}$ is computed, step-2 estimates the aggregated workforce requirement per deployment, since it has to recommend k different strategies to each deployment. To do that, it aggregates over each deployment request and produces a vector $\vec{W}$ of length m that estimates the workforce requirement for each strategy to be deployed successfully, meeting the quality, cost, latency thresholds as well as k. Finally, step-3 invokes an optimizer that determines how to allocate the available workforce W among competing deployment requests to optimize different platform-centric goals. The pseudo-code of BatchStrat is presented in Algorithm 1. We now present the details.

A. Computing Workforce Requirement per Deployment

Given $\mathcal{W}$, for each deployment request d_i and strategy s^j, the workforce requirement is the maximum of workforce requirement to satisfy the quality, cost, and latency threshold. However, to find k strategies for a deployment d_i, we turn our attention back to matrix $\mathcal{W}$ again and investigate how to compute workforce requirement for all k strategies for d_i.

The objective here is to produce a vector $\vec{W}$ of length m, where the i-th value represents the aggregated workforce requirement for request d_i. To understand how to compute $\vec{W}$, we need to consider the **sum-case:** where the task designer intends to perform the deployment using all k strategies, in which case, 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 $\mathcal{W}$; and the **max-case:** where the designer intends to only deploy one of the k strategies, in which case, $w_i = w_{iy}$, (where w_{iy} is the k-th smallest workforce value in row i of matrix $\mathcal{W}$).

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

B. Optimization-Guided Batch Deployment

Since W is 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 resemblance to a well-known discrete optimization problem that falls into the general category of assignment problems, specifically, Knapsack-type of problems [2]. 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 decision version of the Pay-Off maximization problem is NP-Complete.

Proof 1: (sketch): An instance of the famous $0/1$ Knapsack problem could be reduced to the decision version of the Pay-off Maximization problem.

Our solution bears similarity to the greedy algorithm of the Knapsack problem [3]. 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, $\Sigma_{x=1..i} d_i > W$. At that step, it chooses the best of $\{d_1, d_2, d_{i-1}\}$ and d_i and the process continues until no further deployment requests could be satisfied for W (lines 4-8 in Algorithm BatchStrat).

Running Time: This step is dominated by the sorting time of the deployment requests, which is $O(m\ log\ m)$.

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: Compute Workforce Requirement Matrix $\mathcal{W}$
4: Compute Workforce Requirement per Deployment Vector $\vec{W}$
5: Compute the objective function value $f_i$ of each deployment request $d_i$
6: Sort the deployment strategies in non-increasing order of $\frac{f_i}{\vec{w_i}}$
7: Greedily add deployments until we hit $d_i$, such that $\Sigma_{x=1..i} d_i > W$
8: Pick the better of $\{d_1, d_2, d_{i-1}\}$ and $d_i$
```

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. 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.

Theorem 2: Algorithm BatchStrat-ThroughPut gives an exact solution to the problem.

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.

Proof 2: (sketch): The proof directly follows from [5].

IV. Experiments

A. Batch Deployment Recommendation

We compare differnet algorithms. All algorithms are implemented in Python 3.6 on Ubuntu 18.10. Intel Core i9 3.6 GHz CPU, 16GB of memory.

`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 $\frac{f_i}{w_i}$ and greedily selects requests until worker availability W is exhausted.

BatchStrat: Our proposed solution described in Section III.

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.

B. Quality

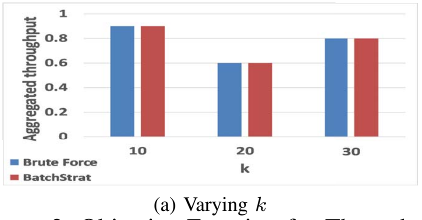
Goal: We aim to validate*dow does* BatchStrat *fare to optimize different platform-centric goals?*

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 assume there is a linear relationship between parameters and Worker Availability. We generate the *slope* uniformly from an interval $[0.5, 1]$. Then, we set $intercept = 1 - slope$ 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 2 shows the results of throughput of BatchStrat by varying k compared with the two baselines (the same could be observed when varying m and $|\mathcal{S}|$). Figure 3 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.

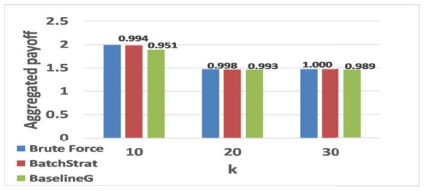


(a) Varying k

Figure 2: Objective Function for Throughput

C. Scalability

Since the `BaselineG` has the same running time as BatchStrat, we only compare `Brute Force` and BatchStrat. The default setting for $|\mathcal{S}|$, k and W are 30, 10 and 0.75, respectively.



(a) Varying k

Figure 3: Objective Function and Approximation Factor for Payoff

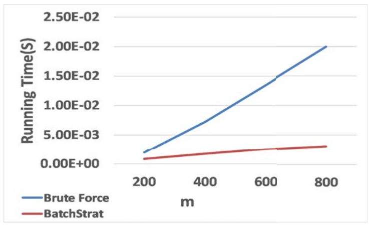


Figure 4: Running time for Batch Deployment Varying m

Figure 4 shows that `Brute Force` takes exponential time with increasing m, whereas BatchStrat scales linearly. 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 $\mathcal{S}$.

V. Future work

Our preliminary work opens up more than one research directions. First and foremost, how to estimate worker availability for different types of tasks is a challenging problem that requires deep investigation in its own merit. Then, an interesting open problem is to come up with principled yet practical models to establish relationship between deployment parameters and strategy parameters. Throughout this paper, we have assumed that the estimated quality, cost, and latency of a set of tasks deployed using a strategy is a function of the task type and worker availability. However, how to realistically model such functions or learn them from historical data for different types of tasks remains to be a part of our ongoing investigations. Finally, an interesting extension is to explore the recommendation of alternative deployment parameters if a request cannot be satisfied as formulated. This would open the possibility of recommending different deployment parameters for which k strategies are available, thereby guiding requesters further in task deployment.

References

[1] R. M. Borromeo et al. Deployment strategies for crowdsourcing text creation. *Information Systems*, 2017.
[2] M. R. Garey and D. S. Johnson. *Computers and intractability*. wh freeman New York, 2002.
[3] O. H. Ibarra et al. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM (JACM)*, 1975.
[4] O. A. E. Kadi. Exploring crowdsourcing deployment strategies through recommendation and iterative refinement. *MS Research Report*.
[5] E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 1979.