Hiring Under Uncertainty

Manish Raghavan [1] **Manish Purohit** [2] **Sreenivas Gollapudi** [2]

Abstract

In this paper we introduce the hiring under uncertainty problem to model the questions faced by hiring committees in large enterprises and universities alike. Given a set of n eligible candidates, the decision maker needs to choose the sequence of candidates to make offers so as to hire the k best candidates. However, candidates may choose to reject an offer (for instance, due to a competing offer) and the decision maker has a time limit by which all positions must be filled. Given an estimate of the probabilities of acceptance for each candidate, the hiring under uncertainty problem is to design a strategy of making offers so that the total expected value of all candidates hired by the time limit is maximized. We provide a 2-approximation algorithm for the setting where offers must be made in sequence, an 8-approximation when offers may be made in parallel, and a 10-approximation for the more general stochastic knapsack setting with finite probes.

1. Introduction

Hiring is a core activity of any enterprise where the timely fulfillment of staffing needs is critical to its functioning. In addition to estimating the quality and suitability of a candidate, the enterprise also needs to deal with uncertainty that arises as the result of good candidates rejecting the job offer. Balancing this trade-off between hiring good quality candidates while at the same time ensuring that all staffing needs are met by a deadline is one of the most challenging aspects of hiring in practice.

A number of algorithmic questions that are inspired by hiring settings have been well-studied in literature (see Section 1.2) including the popular *secretary problem* and its many variants. This line of work focuses on the online nature of the problem and the key question tackled is how to find a good set of candidates when the pool of future candidates is unknown. However, this line of research does not model the other source of uncertainty, i.e., the candidate itself may choose to reject the job offer (for instance, due to a better competing offer), which in turn raises the question of hiring enough candidates by the deadline.

During the hiring process, the "quality" (or value) of a candidate is often estimated by traditional hiring processes such as resume screening and formal interviews and even via algorithmic techniques (see Section 1.2). On the other hand, machine learning models can estimate the probability that a given candidate will accept a job offer based on various features such as the candidate's educational background, salary expectations, location preferences, and so on. Considering both the value as well as offer acceptance probability of each candidate leads to a rich collection of optimization problems. In this paper, we initiate the study of the hiring under uncertainty problem that aims to tackle the inherent trade-off at the heart of the hiring process - *how should we make job offers under uncertainty so that all staffing needs are met by a deadline, and yet hire the best available candidates*?

Formally, we consider the following model as the basis for all the variants we present in this paper. There is a set of n candidates, and we need to hire k of them. We do this by making offers to candidates, which we'll also refer to more abstractly as "probing" a candidate. Each candidate i has a known value v_i and probability p_i of accepting an offer, independent of all other candidates. We have a deadline of t time steps, after which we can't make any further offers. It takes one time step to make an offer and receive an answer from a candidate. Job offers are irrevocable, i.e., once a candidate accepts an offer, that position is "filled" and we cannot replace that candidate with a better candidate in the future. The total value of a chosen set of candidates is simply the sum of the individual candidate values. Our goal is to maximize the total expected value of the hired candidates. We also consider two natural generalizations of this model. First, we allow making parallel offers to multiple candidates in a given time step. Second, we consider the knapsack hiring problem where each candidate i has a size s_i and we have a budget B on the total size of hired candidates.

[1]Department of Computer Science, Cornell University [2]Google Research. Correspondence to: Manish Raghavan <manish@cs.cornell.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

The knapsack hiring problem models the scenario when the enterprise has a fixed budget and different candidates need to be offered different salaries.

We note that in all settings, we do not require v_i to be known precisely; all of our results hold if v_i is only known in expectation. However, our results are sensitive to errors in p_i and s_i. Making them robust to such errors is an interesting subject for future work.

1.1. Our Contributions

We summarize our contributions in this study.

- In Section 2, we offer a 2-approximation algorithm for hiring k candidates with a constraint of making at most t sequential offers.
- In Section 3, we consider the parallel offers model where we are allowed to make as many parallel offers each time step as the number of unfilled positions remaining and design a 8-approximation algorithm.
- In Section 4, we present a 10-approximation for the *knapsack hiring* problem where each candidate has a different size and the decision-maker is constrained by a total budget (as opposed to hiring k candidates).
- We offer a connection to other stochastic optimization problems such as stochastic matching and present a lower-bound for the stochastic matching problem.
- Finally, we show the efficacy of our algorithms using simulations on data drawn from different distributions.

1.2. Related Work

Theoretical questions inspired by hiring scenarios have long been studied in the online setting under the names of optimal stopping or "secretary" problems (Dynkin, 1963; Chow et al., 1964). A few extensions of this setting incorporate elements of our model. Kleinberg considers the case of hiring multiple candidates instead of the traditional single-hire case (Kleinberg, 2005). An older line of work considers a version of the secretary problem in which candidates may stochastically reject offers, although this is typically modeled as a fixed rejection probability (Smith, 1975; Tamaki, 1991; 2000; Ano & Ando, 2000).

In addition, more recent work on stochastic optimization considers a variety of related problems in the offline setting. This includes stochastic versions of submodular optimization (Asadpour et al., 2008; Gupta et al., 2017), knapsack (Dean et al., 2004; 2005; Bhalgat et al., 2011), bandits (Gupta et al., 2011; Ma, 2014), and matching (Bansal et al., 2010; Adamczyk et al., 2015; Baveja et al., 2018). Some special cases of our model (specifically, when one candidate is being hired) can be considered a special case of matching, and in fact, the results we derive here will provide lower bounds for stochastic matching. However, our model cannot in general be captured by any of these prior works.

Algorithmic and data-driven approaches to hiring have become increasingly common with the rise of machine learning (Miller, 2015; Carmichael, 2015). In particular, there is a long line of work focused on predicting teacher quality from data (Kane & Staiger, 2008; Dobbie, 2011; Chalfin et al., 2016; Jacob et al., 2018). More broadly, Mullainathan & Spiess (2017) describe the integration of machine learning with traditional econometric techniques to better estimate quantities like employee performance. Furthermore, studying the gig economy, Kokkodis et al. (2015) use machine learning to estimate the likelihood that freelancers get hired.

2. Hiring Problem: How to fill k positions sequentially?

In this section, we consider the basic hiring problem where we want to hire k employees out of n potential candidates with a constraint of making at most t sequential offers.

2.1. Special case: Hiring a single employee ($k = 1$)

To develop some intuition about the problem as well as to illustrate some of the challenges posed, we begin with the case where $k = 1$, i.e. , we only want to hire one candidate. One might hope that a simple greedy algorithm is optimal in this special case. Unfortunately, as we will show, a number of seemingly natural greedy algorithms[1] do not yield optimal solutions.

However, we can still take advantage of structural properties of the solution. In particular, given a set of t candidates, the optimal order in which to make offers to them is in decreasing order of v_i. To see why, for any two candidates i and j, consider the four possible outcomes of making offers to them: both i and j accept, both reject, i accepts and j rejects, and vice versa. The only outcome in which the order of offers matters is when they both accept, since the position will go to the candidate receiving first offer, and the second offer will never be made. In this case, it is clearly better to make the first offer to the candidate with higher value.

Since the optimal algorithm must always make offers to candidates in decreasing order by value, we can write a dynamic program to compute the optimal subset of t candidates to potentially make offers to. Assume the candidates are sorted in non-increasing order of v_i, i.e., $v_1 \geq v_2 \geq \ldots \geq v_n$. Let $S(i, s)$ be the optimal expected value that can be achieved with s time steps remaining by only considering candidates i through n. Then, we have the recurrence

$$S(i, s) = \max\{p_i v_i + (1 - p_i)S(i+1, s-1), S(i+1, s)\}$$

where the two terms correspond to either making an offer to candidate i or not. Note that $S(1, t)$ then gives the value of

[1] For instance, sorting the candidates by decreasing p_i, v_i, or $p_i \cdot v_i$ and then making at most t offers until one accepts.

the optimal solution, and the offer strategy can be found by retracing the choices of the dynamic program.

2.2. General Problem: Hiring k employees ($k > 1$)

While the $k = 1$ case admits a clean solution, the general case where $k > 1$ is more complex. First, note that a simple k-approximation exists: with the dynamic program from Section 2.1, we can optimally fill a single slot. Doing so yields a candidate who is in expectation at least as good as any of the k candidates hired by the optimal strategy. In general, however, the optimal solution may display several non-monotonicities that make it difficult to extend the $k = 1$ solution.

Example 1. *Consider the following instance with $n = 4$, $t = 3$, and $k = 2$.*

$$(p_1, v_1) = (1, 1) \qquad (p_2, v_2) = (0.5, 1)$$
$$(p_3, v_3) = (0.5, 1) \qquad (p_4, v_4) = (0.1, 2)$$

We will show that in the optimal strategy, the offers made are not necessarily monotone in acceptance probability, value, or expected value. First, note that any deterministic strategy can be represented as a binary decision tree, where each node in the tree corresponds to a candidate to whom an offer is made. The two branches are the resulting strategies if the offer is accepted or rejected. Taking the convention that the right branch corresponds to acceptance, the optimal solution for the above instance is as shown in Figure 1.

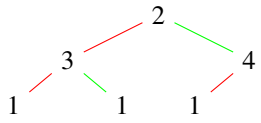


Figure 1. An optimal solution to Example 1

Note that there are several counter-intuitive effects at play here. First, despite having the lowest acceptance probability and expected value, candidate 4 still receives an offer with probability $1/2$. Second, the candidate with the highest expected value (candidate 1) receives an offer either last or not at all. Finally, despite the fact that candidates 1, 2, and 3 all have the same value, it is strictly optimal to make an offer to candidate 2 (or 3) before candidate 1, even though candidate 1 accepts with higher probability.

Thus, unlike in the $k = 1$ scenario, the optimal solution may not be value-ordered, so the dynamic programming approach discussed above cannot be optimal here. We conjecture that this problem is NP-hard for general k. In the remainder of this section, we present an approximation algorithm that runs in polynomial time and yields at least half of the optimal expected value. We first show that there exists a non-adaptive algorithm yielding a 2-approximation. Then, we show that a dynamic program similar to that in Section 2.1 gives an adaptive algorithm that is better than *any* non-adaptive algorithm, and hence is also a 2-approximation.

2.2.1. Establishing an adaptivity gap of 2

Gupta et al. (2017) study adaptivity gaps for stochastic probing problems where the goal is to maximize a given submodular function (or XOS function) over the set of active, probed elements. In this setting, each element e is active independently with probability p_e and the set of elements that are probed must satisfy given prefix-closed constraints. The Hiring with Uncertainty problem does not quite fit into their framework, since their framework allows one to choose the "best" set of active, probed elements, while in our setting we are forced to hire the *first* k candidates that are active. Nevertheless, we can leverage some insights from (Gupta et al., 2017) to show an adaptivity gap of 2 (as opposed to 3 obtained by them for stochastic probing).

Similar to the one shown in Figure 1, the optimal solution to any instance can be represented by a binary tree $\mathcal{T}$. Each node u of $\mathcal{T}$ corresponds to a candidate i (denoted by $cand(u)$) and has two outgoing edges leading to subtrees in case the candidate i is active (happens with probability p_i) or inactive (happens with probability $(1 - p_i)$). Any root to leaf path in this tree represents the sequence of offers made by the optimal algorithm in a particular realization. The tree $\mathcal{T}$ naturally defines a probability distribution $\pi_{\mathcal{T}}$ over root to leaf paths - start at the root and at each node u, follow the "yes" edge with probability p_i where $i = cand(u)$ and the "no" edge otherwise. Since the optimal strategy can make offers to at most t candidates, any such path must have at most t nodes. Further, since any strategy can only hire at most k candidates, any root to leaf path in $\mathcal{T}$ must have at most $k - 1$ "yes" edges. Thus any root to leaf path P can be decomposed into at most k "segments" where a segment is a maximal sub-path composed of only "no" edges as shown in Figure 2. Let $\mathsf{segments}(P) = \{S_1, S_2, \ldots, S_\ell\}$ denote the set of segments in P. For each segment $S \in \mathsf{segments}(P)$, let $\mathsf{last}(S)$ denote the *last* node on segment S.

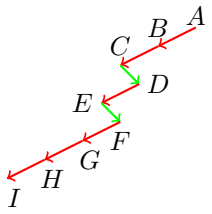


Figure 2. A path with 3 segments: ABC, DE, and $FGHI$.

Given the optimal tree $\mathcal{T}$, Procedure 1 samples a single path P according to the distribution $\pi_{\mathcal{T}}$ and then probes the candidates on each segment of P in descending order by value to hire at most one candidate from each segment. In the rest of this section, we show that Procedure 1 yields at least half of the total expected value of $\mathcal{T}$ in expectation.

Let $\mathsf{val}(\mathcal{T}) = \mathbb{E}_{P \sim \pi_{\mathcal{T}}} \left[\sum_{j=1}^{\ell} v_{\mathsf{last}(S_j)} \right]$ be the total ex-

Procedure 1 ApproxGivenTree($\mathcal{T}$)

$P \leftarrow$ a random path sampled from $\pi_{\mathcal{T}}$
$S_1, \ldots, S_\ell \leftarrow P$ divided into at most k segments {Each S_j is a list of candidates}
for $j \leftarrow 1, \ldots, \ell$ **do**
  $S'_j \leftarrow S_j$ sorted in decreasing order of value
  for each candidate i in S'_j **do**
    Make an offer to candidate i
    if i accepts **then**
      break
    end if
  end for
end for

pected value of the tree $\mathcal{T}$ (note that ℓ is a random variable). Similarly, let $\mathsf{proc}_1(\mathcal{T})$ be the expected value obtained by Procedure 1 on tree $\mathcal{T}$. For any segment S_j, we define $\mathsf{segval}(S_j)$ to be the expected value of the active candidate from S_j with largest value. Formally, if S_j consists of candidates $\{1, 2, \ldots, |S_j|\}$ sorted in non-increasing order of their values, then

$$\mathsf{segval}(S_j) \triangleq \sum_{i=1}^{|S_j|} p_i v_i \prod_{j<i} (1 - p_j).$$

We observe that $\mathsf{proc}_1(\mathcal{T}) = \mathbb{E}_{P\sim\pi_{\mathcal{T}}}\left[\sum_{j=1}^{\ell} \mathsf{segval}(S_j)\right]$, as Procedure 1 obtains the value of the active element with the largest value in each S_j. The following lemma shows that in expectation, this is a 2-approximation to $\mathsf{val}(\mathcal{T})$.

Lemma 1.

$$\mathbb{E}_{P\sim\pi_{\mathcal{T}}}\left[\sum_{j=1}^{\ell} \mathsf{segval}(S_j)\right] \geq \frac{1}{2}\mathbb{E}_{P\sim\pi_{\mathcal{T}}}\left[\sum_{j=1}^{\ell} v_{\mathsf{last}(S_j)}\right]$$

Proof. We proceed by induction over the segments. Let I be $\mathsf{last}(S_1)$, and let J be the random variable denoting the index of the first active candidate on S_1, so $J \leq I$. If no candidate on this segment is active, we'll say $J = 0$. In the base case, $\ell = 1$. Otherwise, by the inductive hypothesis,

$$\mathbb{E}_{P\sim\pi_{\mathcal{T}}}\left[\sum_{j=2}^{\ell} \mathsf{segval}(S_j)\right] \geq \frac{1}{2}\mathbb{E}_{P\sim\pi_{\mathcal{T}}}\left[\sum_{j=2}^{\ell} v_{\mathsf{last}(S_j)}\right].$$

In either case, it suffices to show that $\mathbb{E}[v_J] \geq \frac{1}{2}\mathbb{E}[v_I]$. This follows from Lemma 3.3 of Gupta et al. (2017) and the complete proof is deferred to the supplementary material. □

Removing adaptivity. Note that Procedure 1 is adaptive, since it probes within a segment until it finds an active element. However, we can use it to argue about a simpler non-adaptive algorithm: pick a random path down the optimal tree $\mathcal{T}^*$ and make offers to candidates on that path in decreasing order of value. This has value at least $\mathsf{proc}_1(\mathcal{T}^*)$ because for any realization of active elements, making offers in decreasing order of value is always beneficial. Thus, the adaptivity gap for this problem is at most 2.

2.2.2. A CONSTRUCTIVE 2-APPROXIMATION

We have shown that there exists a non-adaptive algorithm whose total expected value is at least half of the expected value of the optimal algorithm. However, this algorithm relied on the knowledge of the optimal decision tree $\mathcal{T}^*$ and is thus non-constructive. We now design a polynomial time algorithm (seqalg) whose expected value is at least the expected value of *any non-adaptive algorithm*, and hence is also at least half the expected value of the optimal algorithm.

By definition, any non-adaptive algorithm must choose a fixed sequence of t potential candidates and make offers to them in order until k of them accept. Further, as discussed in Section 2.1, the optimal such algorithm must probe the candidates in non-increasing order by value. However, using a dynamic programming strategy similar to that in Section 2.1, we can find the *optimal* algorithm (not necessarily non-adaptive) that probes candidates in non-increasing order by value. This must be better than the optimal non-adaptive algorithm and hence is also a 2-approximation.

Dynamic Program (seqalg)**.** Assume that candidates are sorted in non-increasing order of their values v_i. Let $S(i, \ell, s)$ be the optimal expected value achieveable by hiring at most ℓ candidates in s time steps by only considering candidates i through n. We obtain the following recurrence:

$$\begin{aligned} S(i,\ell,s) = \max\{&p_i(v_i + S(i+1, \ell-1, s-1)) \\ &+ (1-p_i)S(i+1,\ell,s-1), S(i+1,\ell,s)\}. \end{aligned} \tag{1}$$

where the two terms correspond to either making an offer to candidate i or not. Let seqalg denote the dynamic program constructed using the above recurrence. We abuse notation slightly and let $\mathsf{seqalg}_{k,t} = S(1, k, t)$ be the expected value obtained by this algorithm.

Let $\mathsf{seqopt}_{k,t}$ be the value of the optimal adaptive strategy. Lemma 1 shows that the optimal non-adaptive strategy is a 2-approximation to $\mathsf{seqopt}_{k,t}$. Because seqalg is at least as good as any non-adaptive strategy, we have that for a set of candidates $\mathcal{C}$, $\mathsf{seqalg}_{k,t}(\mathcal{C}) \geq \frac{1}{2}\mathsf{seqopt}_{k,t}(\mathcal{C})$.

A lower bound. It is an open question as to whether the above analysis is tight. However, by modifying the probabilities and values in Example 1, we show in the supplementary material that no algorithm that provides a value-sorted solution (including seqalg) can get more than 0.927 of the optimal algorithm in general.

3. Filling k Positions in Parallel

In the previous section, we considered the problem of hiring with sequential offers. However, if we have k positions to fill, we could in principle make k offers per timestep. This is clearly more powerful than the sequential offer model, since any sequence of sequential offers is valid in the parallel model. We'll treat the constraint of filling k positions as hard, meaning that if at a particular timestep there are $\ell < k$ remaining unfilled positions, we can only make ℓ offers at that time, though it would be an interesting future direction to consider a relaxed version in which we hire at most k candidates with high probability.

Intuitively, the more slots remain available, the more offers can be made, which is beneficial when there are many high-value low-probability candidates. This means an optimal strategy must somehow balance the tension between two conflicting objectives: filling slots and maximizing the number of offers that can be made to risky candidates. The following example demonstrates this tension.

Example 2. *Consider the example with $n = 2t - 1$ candidates and $k = 2$. $2t - 2$ of the candidates have $p_i = 1/(2t-2)$ and $v_i = 1$, and the last candidate has $p_n = 1$ and $v_n = 1$.*

Even though candidate n will surely accept an offer, the optimal strategy here is to make offers to all of the low-probability candidates (2 at a time) until one of them accepts, and then to make an offer to candidate n, who will definitely accept. As t gets large, this yields value approximately $\frac{2e-1}{e} \approx 1.63$ in expectation, Making an offer to candidate n first can only get value approximately $\frac{2\sqrt{e}-1}{\sqrt{e}} \approx 1.39$, since we can only make one offer per timestep after we fill the first slot. Thus, the order in which offers are made significantly impacts the overall value.

An 8-approximation algorithm (paralg). We now design paralg, a constructive 8-approximation algorithm, drawing on the results in Section 2. The basic idea is to relax the parallel offer instance with t timesteps to a sequential offer instance with $k \cdot t$ timesteps, solve this using $\mathsf{seqalg}_{k,kt}$, and use the result to construct a solution to the original instance.

Given a set of candidates $\mathcal{C}$, let $\mathsf{paropt}_{k,t}(\mathcal{C})$ be the expected value of the optimal solution with parallel offers, filling k slots in t timesteps. Then, $\mathsf{paropt}_{k,t}(\mathcal{C}) \le \mathsf{seqopt}_{k,kt}(\mathcal{C})$, since any sequence of parallel offers can be done in sequence over kt timesteps.

We can apply the dynamic programming algorithm seqalg from Section 2 to $\mathcal{C}$ to get a sequential-offer strategy over kt timesteps yielding expected value at least $\frac{1}{2}\mathsf{seqopt}_{k,kt}(\mathcal{C})$. Let $\mathcal{T}$ be the resulting decision tree. We'll show how to convert this sequential-offer decision tree over kt timesteps into a parallel-offer strategy over t timesteps.

Procedure 2 ParallelFromSequential($\mathcal{T}$)

1: $P \leftarrow$ a random path sampled from $\pi_{\mathcal{T}}$
2: $S_1, \dots, S_\ell \leftarrow$ the segments of P
3: $S'_1, \dots, S'_m \leftarrow$ segments split such that each has length at most t.
4: Sort each segment S'_j in decreasing order of v_i.
5: Let U be the indices of the k segments with highest $\mathsf{segval}(\cdot)$
6: **for** $s \leftarrow 1, \dots, t$ **do**
7: **for** $j \in U$ **do**
8: Make an offer to candidate i, the s^{th} candidate from S'_j, at this timestep s.
9: **if** i accepts **then**
10: Remove j from U
11: **end if**
12: **end for**
13: **end for**

Let $\mathsf{proc}_2(\mathcal{T}^*)$ be the expected value of the parallel-offer strategy produced by Procedure 2, where $\mathcal{T}^*$ is the output of $\mathsf{seqalg}_{k,kt}(\mathcal{C})$. Then, we have the following.

Lemma 2.

$$\mathsf{proc}_2(\mathcal{T}^*) \ge \frac{1}{8}\mathsf{paropt}_{k,t}(\mathcal{C})$$

Proof. By Lemma 1,

$$\mathsf{seqalg}_{k,kt}(\mathcal{C}) \ge \frac{1}{2}\mathsf{seqopt}_{k,kt}(\mathcal{C}) \ge \frac{1}{2}\mathsf{paropt}_{k,t}(\mathcal{C}).$$

To complete the proof, we must show that $\mathsf{proc}_2(\mathcal{T}^*) \ge \frac{1}{4}\mathsf{seqalg}_{k,kt}(\mathcal{C})$. Note that Procedure 2 yields an offer strategy in the parallel model, while $\mathcal{T}^*$ represents a strategy in the sequential model with kt timesteps.

First, observe that by applying Lemma 1 again, if we could make offers along each segment of a random path down $\mathcal{T}^*$ in decreasing order of value, we'd get a 2-approximation to $\mathsf{seqalg}_{k,kt}(\mathcal{C})$, since we'd get the maximum active element on each segment. Since there are at most k segments, we could make an offer to the highest valued candidate from each segment in the first time step and proceed down each segment in parallel, discarding a segment once a candidate accepts an offer. However, since some segments may have length more than t, we may not have enough offers to go all the way down each segment. Consequently, in step 3 of Procedure 2, we partition the segments further so that each new segment contains at most t candidates. More formally, if a segment $S_j \in \mathsf{segments}(P)$ has length $at + b$ for some integers $a, b \ge 0$ and $b < t$, arbitrarily split the candidates in S_j into $a + 1$ new segments such that a of those segments have exactly t candidates.

Let $|S_j|$ be the size of the jth segment. When we split it up into new segments of length at most t, S_j will be turned

into $\lceil \frac{|S_j|}{t} \rceil$ segments. Thus, after splitting, the number of new segments is at most

$$\sum_{j=1}^{\ell} \left\lceil \frac{|S_j|}{t} \right\rceil \leq \sum_{j=1}^{\ell} \frac{|S_j|}{t} + 1 \leq k + \frac{\sum_{j=1}^{\ell} |S_j|}{t} \leq 2k.$$

Thus, if we were to pick k segments uniformly at random, each segment would have probability at least $1/2$ of being selected randomly into U, meaning that

$$\begin{aligned}\sum_{j=1}^{m} \mathbb{E}\left[\mathsf{segval}(S'_j) \Pr[j \in U]\right] &\geq \frac{1}{2} \sum_{j=1}^{m} \mathbb{E}\left[\mathsf{segval}(S'_j)\right] \\ &\geq \frac{1}{2} \sum_{j=1}^{\ell} \mathbb{E}\left[\mathsf{segval}(S_j)\right] \\ &\geq \frac{1}{4} \mathbb{E}\left[\mathsf{seqalg}_{k,kt}(\mathcal{C})\right],\end{aligned}$$

where the last inequality follows from Lemma 1. We can derandomize this by choosing the k segments with highest expected value, which must be at least as good as k random segments. As a result, we have

$$\mathsf{proc}_2(\mathcal{T}^*) \geq \frac{1}{4} \mathbb{E}\left[\mathsf{seqalg}_{k,kt}(\mathcal{C})\right] \geq \frac{1}{8} \mathsf{paropt}_{k,t}(\mathcal{C}).$$

□

Thus, our final algorithm (which we call $\mathsf{paralg}_{k,t}$, or just paralg when k and t are clear) is to first apply $\mathsf{seqalg}_{k,kt}$ to $\mathcal{C}$, producing a tree $\mathcal{T}^*$, and build a parallel-offer strategy from $\mathcal{T}^*$ with Procedure 2.

$$\mathsf{paralg}_{k,t}(\mathcal{C}) \triangleq \text{ParallelFromSequential}(\mathsf{seqalg}_{k,kt}(\mathcal{C})) \tag{2}$$

As noted above, our final approximation factor will be $\min(k, 8)$, since simply filling one position optimally yields a k-approximation.

4. Knapsack Hiring Problem

We now consider the knapsack hiring problem that directly generalizes the vanilla hiring problem studied in Section 2. In this case, in addition to a value v_i and probability p_i, each candidate i also has a size s_i. Instead of a number of slots k, we have a budget B on the total size of the hired candidates. As earlier, we have a deadline of t time steps and can make only one offer per time step.

The knapsack hiring problem is closely related to the well-studied stochastic knapsack problem (Dean et al., 2004; 2005), which is as follows: we are given n items, each with a known value and "size distribution". When an item is added to the knapsack, its size is drawn from this distribution. Once an item exceeds the capacity, this item must be discarded and no further items can be added to the knapsack. In the multidimensional version, both the capacity of the knapsack and the size of an item are vectors, and the process ends once any component of the vector capacity is exceeded. We observe that the two models differ slightly since in the knapsack hiring problem, both the value and size of an item (candidate) is stochastic.

We first give a reduction from the knapsack hiring problem to the multidimensional stochastic knapsack problem. For simplicity, we assume that the budget $B = 1$ without loss of generality. We construct an instance of 2-dimensional stochastic knapsack as follows - the knapsack capacity is $[1\ t]^\top$ where the first dimension represents the budget constraint and the second dimension represents the number of allowed probes. The size of item i is represented by the vector $[s_i\ 1]^\top$ if the item exists when it is probed (happens with probability p_i) and $[0\ 1]^\top$ otherwise. The value v'_i of item i is set to the expected value obtained from candidate i, i.e., $v'_i = p_i v_i$. With this reduction, the optimal solution to the knapsack hiring problem remains unchanged if items deterministically contribute value $v'_i = p_i v_i$, as we show in the supplementary material.

Dean et al. (2005) give a general $1 + 6d$-approximation to multidimensional stochastic knapsack, where d is the number of knapsack constraints. Directly applying this, we would get a 13-approximation in our 2-dimensional case. However, by leveraging the structure of the finite-probe problem, we can tighten this to a 10-approximation.

Without loss of generality, we assume that $s_i \leq 1$ for all i (otherwise the item would never fit in the knapsack). We also normalize the number of probes to 1, so each item uses $1/t$ probes. Let $\mu(i)$ denote the vector of expected size of item i, meaning $\mu(i) = [p_i s_i \ \ 1/t]^\top$. Let $\mu(S) = \sum_{i \in S} \mu(i)$. We use the notation $\mu_1(S)$ and $\mu_2(S)$ to denote the first and second components of $\mu(S)$ respectively. Further, let $\mathsf{size}(i)$ denote the vector of the realized size of item i and let $\mathsf{size}(S) = \sum_{i \in S} \mathsf{size}(i)$.

Procedure 3 KnapsackFiniteProbes(p, v, s)

$m_1 \leftarrow \max_i v'_i = \max_i p_i v_i$
$\mathcal{L} \leftarrow$ the sequence of all items with $\|\mu(i)\|_1 \leq 1/3$, sorted in non-increasing order of $\dfrac{v'_i}{\|\mu(i)\|_1}$
$m_{\mathcal{L}} \leftarrow \sum_{i=1}^{\ell} v'_i(1 - \sum_{j \leq i} \mu_1(j))$, where ℓ is the smallest integer such that $\sum_{i=1}^{\ell} \|\mu(i)\|_1 < 1$
if $m_1 \geq m_{\mathcal{L}}$ **then**
    Probe the item with highest expected value v'_i
else
    Probe the items in $\mathcal{L}$ until the knapsack is full
end if

Our algorithm (Procedure 3) takes the better of two strate-

gies: probing the item with highest expected value and probing a sequence of "small" items. Exactly evaluating $m^*_{\mathcal{L}}$, the expected value of the second of these strategies, may be difficult; however, we can show that $m_{\mathcal{L}} = \sum_{i=1}^{\ell} v'_i(1 - \sum_{j \le i} \mu_1(j)) \le m^*_{\mathcal{L}}$. We obtain v'_i for item i if and only if the first i items in $\mathcal{L}$ all fit inside the knapsack. Thus, $m^*_{\mathcal{L}} = \sum_{i=1}^{\ell} v'_i \Pr[\|\mathsf{size}(\mathcal{L}_i)\|_\infty \le 1]$, where $\mathcal{L}_i$ denotes the set of first i items in $\mathcal{L}$. By Claim 3, $m^*_{\mathcal{L}} \ge m_{\mathcal{L}}$. Note that Claim 3 applies since the constraint $\sum_{i=1}^{\ell} \|\mu(i)\|_1 < 1$ implies $\ell < t$.

Claim 3. *For any set A of at most t items,* $\Pr[\|\mathsf{size}(A)\|_\infty \le 1] \ge 1 - \mu_1(A)$.

See the supplementary material for a proof.

Let $\mathsf{greedy} = \max\{m_1, m_{\mathcal{L}}\}$ be a lower bound on the expected value of Procedure 3. Let A be the random set of items that are probed by the optimal adaptive algorithm, and let opt be the expected value of the optimal algorithm.

Lemma 4 ((Dean et al., 2005), Lemma 4.2 and Lemma 4.3)**.** $\mathsf{opt} \le (1 + 3\mathbb{E}[\|\mu(A)\|_1])\mathsf{greedy}$.

For any adaptive algorithm, we can bound the expected size of the set of items probed using Lemma 2 from Dean et al. (2004). In particular, we can bound the expected size of the first component as $\mathbb{E}[\mu_1(A)] \le 2$. On the other hand, since the optimal adaptive algorithm can never probe more than t items, $\mathbb{E}[\mu_2(A)] \le 1$. Substituting these bounds into Lemma 4 gives us the desired 10-approximation:

$$\begin{aligned}\mathsf{opt} &\le (1 + 3\mathbb{E}[\|\mu(A)\|_1])\mathsf{greedy} \\ &\le (1 + 3(\mathbb{E}[\mu_1(A) + \mu_2(A)]))\mathsf{greedy} \\ &\le (1 + 3 \cdot 3)\mathsf{greedy} = 10 \cdot \mathsf{greedy}.\end{aligned}$$

5. A Lower Bound for Stochastic Matching

The hiring with uncertainty problem with $k = 1$ can be viewed as a special case of the stochastic matching problem, which is as follows: given a graph $G = (V, E)$, probabilities p_e and values v_e for all $e \in E$, and patience parameters t_v for all $v \in V$, the goal is to obtain a matching with maximum expected weight. As in our hiring problem, edges can be probed sequentially. If an edge e is found to exist, it must be added to the matching, contributing value v_e. Each probe decreases the patience parameters of the incident vertices by 1, and when a vertex runs out of patience, it cannot be matched.

The state-of-the-art approach for this problem is to form a probing strategy by solving the linear program relaxation:

$$\max_{x \in [0,1]^{|E|}} \sum_{e \in |E|} p_e v_e x_e \qquad \text{s.t. } \forall v \sum_{e \in \delta(v)} x_e \le t_v \qquad (3)$$
$$\forall v \sum_{e \in \delta(v)} p_e x_e \le 1$$

This LP relaxation has been the primary approach for stochastic matching since Bansal et al. (2010), yielding a 2.845-approximation for bipartite graphs (Adamczyk et al., 2015) and a 3.224-approximation for general graphs (Baveja et al., 2018). However, little is known about the tightness of upper bound produced by the LP. Not only is there an integrality gap, but there is also a *probing* gap – the LP does not fully account for the random realizations of probes.

With $k = 1$, the hiring problem is a special case of stochastic matching, since it can be expressed as matching on a star-shaped graph. Thus, we can use it provide a lower bound on the worst-case slack created by the LP. In the supplementary material we provide an example showing that the gap between the LP value and the expected value of the optimal probing strategy must be at least $1 - 1/e$, meaning no probing strategy can approximate the optimal LP value to a factor better than $\frac{e}{e-1} \approx 1.581$.

6. Experiments

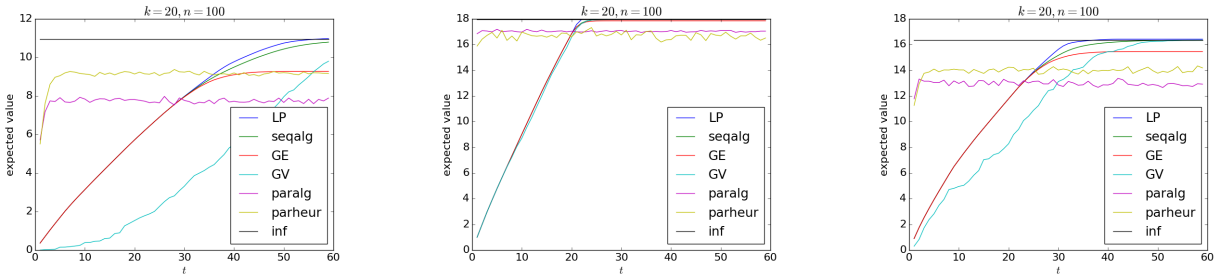
We test the performance of our algorithms for the HIRING WITH UNCERTAINTY problem in both the sequential and parallel offers setting via simulations. We generate simulated data sets as follows. The values for $n = 100$ candidates are chosen uniformly at random from $[0, 1]$. We consider three models to generate the probabilities:

- **Negative correlation:** Higher-value candidates are less likely to accept offers. We sample p_i's according to a Beta distribution, with $p_i \sim \text{Beta}(10(1 - v_i), 10v_i)$.
- **Positive correlation:** Higher-value candidates are more likely to accept: $p_i \sim \text{Beta}(10v_i, 10(1 - v_i))$.
- **No correlation:** $p_i \sim \text{Uniform}[0, 1]$.

On each of these data sets, we consider the performance of our three algorithms each with $k = 20$, namely

- **seqalg**: The dynamic programming algorithm from Section 2 to make t sequential offers.
- **paralg**: The parallel approximation algorithm from Section 3. We take the best solution over 100 random samples (of paths).
- **parheur**: We consider the following heuristic strategy to make offers in the parallel model. Note that the parallel approximation algorithm effectively partitions the set of candidates into up to $2k$ sets, selects the best k of them, and makes offers to candidates in those sets in decreasing order of value. Our heuristic, then, is to randomly partition the set of candidates into k disjoint sets and use the optimal single-slot solution from Section 2.1 on each set independently to decide which of them to make offers to. These offers can be made in parallel since the sets are disjoint.

For comparison, we include two natural greedy baselines.



(a) Negative Correlation (b) Positive Correlation (c) No Correlation

Figure 3. Comparison of different algorithms on three simulated data sets.

We probe candidates in decreasing order of expected value $p_i \cdot v_i$ (**GE**) and value v_i (**GV**). We also plot two upper bounds on the value obtained by an optimal algorithm: **LP**, the value obtained by a natural LP relaxation (similar to (3)), and **inf**, the optimal algorithm with $t = \infty$ (sort the candidates by decreasing value and make offers until k candidates accept).

Figures 3(a), 3(b), and 3(c) demonstrate the performance of our algorithms on the three data sets with negative correlation, positive correlation, and no correlation respectively. Beyond the theoretical guarantees, seqalg performs well empirically and dominates the greedy baselines, especially in the more natural setting where values and probabilities are negatively correlated. seqalg is in general quite close to the LP upper bound – much closer than the theoretical guarantee of 2. Thus, the LP is a fairly tight upper bound on the maximum value achievable by probing.

Even for moderately small values of t, seqalg outperforms paralg, despite the fact that it makes 1 offer per time step when paralg makes multiple offers at a time. Moreover, parheur almost always outperforms paralg. The relatively poor performance of paralg is to be expected. Recall that paralg takes the solution tree to seqalg with kt offers and probes candidates on the segments of a random path down this tree. By construction, candidates on this path are sorted by value, so high-value candidates are concentrated in a small number of segments. paralg can only select at most one candidate per segment, so it must ignore some high-value candidates. In contrast, parheur partitions the candidates randomly, making it more likely that each set in the partition contains high-value candidates.

6.1. Knapsack setting

We also provide simulated results for a slightly modified version of the 10-approximation algorithm (**approx**) in the knapsack setting, where instead of calculating the lower bound $m_{\mathcal{L}}$ on the greedy strategy as in Procedure 3, we estimate the true expected value $m^*_{\mathcal{L}}$ by simulating runs of the greedy branch of the algorithm. We compare the LP relaxation (**LP**) to our algorithm. In addition, we compare to a natural greedy baseline (**greedy**), which probes items in decreasing order of $p_i v_i / s_i$. We sample s_i from a truncated Pareto distribution on $[0, 1]$, and sample $v_i \in [0, 1]$ from a Beta distribution positively correlated with $\sqrt{s_i}$. We use $\sqrt{s_i}$ so that expected v_i's exhibit diminishing returns in s_i. We choose $p_i \sim \text{Uniform}[0, 1]$ and set a budget of $B = 1$.

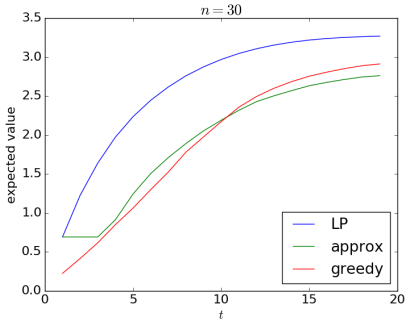


Figure 4. Experimental results for knapsack setting

As the results in Figure 4 show, our algorithm performs roughly as well as **greedy**, but it does better for very small values of t.

7. Conclusions

With the increased use of data-driven techniques in hiring, predictions for employment outcomes are becoming increasingly accurate. Leveraging these predictions can be non-trivial, leading to the family of stochastic optimization problems we have considered here. As we have shown, imposing a finite number of offers can lead to highly complex solutions; however, by imposing an intuitive structure on the solution space, we are able to derive approximation algorithms that perform well, both theoretically and in practice.

References

Adamczyk, M., Grandoni, F., and Mukherjee, J. Improved approximation algorithms for stochastic matching. In *Algorithms-ESA 2015*, pp. 1–12. Springer, 2015.

Ano, K. and Ando, M. A note on Bruss' stopping problem with random availability. *Lecture Notes-Monograph Series*, pp. 71–82, 2000.

Asadpour, A., Nazerzadeh, H., and Saberi, A. Stochastic submodular maximization. In *International Workshop on Internet and Network Economics*, pp. 477–489. Springer, 2008.

Bansal, N., Gupta, A., Li, J., Mestre, J., Nagarajan, V., and Rudra, A. When lp is the cure for your matching woes: Improved bounds for stochastic matchings. In *European Symposium on Algorithms*, pp. 218–229. Springer, 2010.

Baveja, A., Chavan, A., Nikiforov, A., Srinivasan, A., and Xu, P. Improved bounds in stochastic matching and optimization. *Algorithmica*, 80(11):3225–3252, 2018.

Bhalgat, A., Goel, A., and Khanna, S. Improved approximation results for stochastic knapsack problems. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pp. 1647–1665. SIAM, 2011.

Carmichael, S. G. Hiring c-suite executives by algorithm. *Harvard Business Review*, 2015.

Chalfin, A., Danieli, O., Hillis, A., Jelveh, Z., Luca, M., Ludwig, J., and Mullainathan, S. Productivity and selection of human capital with machine learning. *American Economic Review*, 106(5):124–27, 2016.

Chow, Y., Moriguti, S., Robbins, H., and Samuels, S. Optimal selection based on relative rank (the secretary problem). *Israel Journal of mathematics*, 2(2):81–90, 1964.

Dean, B. C., Goemans, M. X., and Vondrdák, J. Approximating the stochastic knapsack problem: The benefit of adaptivity. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pp. 208–217. IEEE, 2004.

Dean, B. C., Goemans, M. X., and Vondrák, J. Adaptivity and approximation for stochastic packing problems. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 395–404. Society for Industrial and Applied Mathematics, 2005.

Dobbie, W. Teacher characteristics and student achievement: Evidence from teach for america. *Unpublished manuscript, Harvard University*, 2011.

Dynkin, E. B. The optimum choice of the instant for stopping a markov process. *Soviet Mathematics*, 4:627–629, 1963.

Gupta, A., Krishnaswamy, R., Molinaro, M., and Ravi, R. Approximation algorithms for correlated knapsacks and non-martingale bandits. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pp. 827–836. IEEE, 2011.

Gupta, A., Nagarajan, V., and Singla, S. Adaptivity gaps for stochastic probing: Submodular and XOS functions. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1688–1702. SIAM, 2017.

Jacob, B. A., Rockoff, J. E., Taylor, E. S., Lindy, B., and Rosen, R. Teacher applicant hiring and teacher performance: Evidence from dc public schools. *Journal of Public Economics*, 166:81–97, 2018.

Kane, T. J. and Staiger, D. O. Estimating teacher impacts on student achievement: An experimental evaluation. Technical report, National Bureau of Economic Research, 2008.

Kleinberg, R. A multiple-choice secretary algorithm with applications to online auctions. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 630–631. Society for Industrial and Applied Mathematics, 2005.

Kokkodis, M., Papadimitriou, P., and Ipeirotis, P. G. Hiring behavior models for online labor markets. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pp. 223–232. ACM, 2015.

Ma, W. Improvements and generalizations of stochastic knapsack and multi-armed bandit approximation algorithms. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1154–1163. Society for Industrial and Applied Mathematics, 2014.

Miller, C. C. Can an algorithm hire better than a human. *The New York Times*, 25, 2015.

Mullainathan, S. and Spiess, J. Machine learning: an applied econometric approach. *Journal of Economic Perspectives*, 31(2):87–106, 2017.

Smith, M. A secretary problem with uncertain employment. *Journal of applied probability*, 12(3):620–624, 1975.

Tamaki, M. A secretary problem with uncertain employment and best choice of available candidates. *Operations Research*, 39(2):274–284, 1991.

Tamaki, M. Minimal expected ranks for the secretary problems with uncertain selection. *Lecture Notes-Monograph Series*, pp. 127–139, 2000.