A General Framework for Sequential Batch-Testing

Rayen Tan* Alex Xu† Viswanath Nagarajan*

June 21, 2024

Abstract

We consider sequential testing problems that involve a system of n stochastic components, each of which is either working or faulty with independent probability. The overall state of the system is a function of the state of its individual components, and the goal is to determine the system state by testing its components at the minimum expected cost. In the classic setting, where each component is tested separately, sequential testing algorithms are known for several functions such as AND, k-of-n and score-classification. Moreover, many of these algorithms are "non adaptive", i.e., they test components in an *a priori* fixed order until the function is evaluated. We consider a batched setting that allows for testing multiple components simultaneously, at the expense of an extra setup cost. Our main result is a generic method that transforms a non-adaptive solution for the classic setting into a solution for the more general batched setting, while incurring only an additive $\frac{1}{\sqrt{2}}$ loss in the approximation ratio. Combined with previously-known approximation algorithms in the classic setting, we obtain batched algorithms for AND, k-of-n and score-classification functions with approximation ratios 1.707, 2.618 and 6.371 respectively. Our algorithm is also very efficient, running in $O(n^2)$ time for all the aforementioned functions. Finally, we evaluate the practical performance of our algorithm on random instances and observe that it performs very well in comparison to an exact (exponential time) dynamic programming method.

1 Introduction

Consider a complex system consisting of n stochastic components, where the state of the system is given by some function of the states of its individual components. The state (working or faulty) of each component is unknown, although the failure probabilities are known upfront. In order to determine the state of any component, one needs to perform a test which also incurs some cost. Often, the system state can be determined by testing just a subset of components. For example, if the system requires all components to be working, a single faulty component immediately determines the system status (irrespective of the status of other components). This motivates sequential testing problems, where the goal is to test components sequentially until the system state is determined. Such problems arise in a variety of applications in manufacturing, healthcare and artificial intelligence; see [Mor82] and [Ünl04] for surveys.

The classic setting in sequential testing involves performing tests one by one. Exact or approximation algorithms are known for several types of systems/functions in this setting. Some commonly studied cases are AND-functions or series-systems (determine whether all components are working),

*Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, USA. Research supported in part by NSF grants CMMI-1940766 and CCF-2006778.

†Computer Science and Engineering, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, USA.

k-of-n systems (determine if at least k components are working) and score-classification (determine a risk category based on the number of working components). We will discuss these systems in more detail later.

Solutions to sequential testing problems may generally be *adaptive*, i.e., the next test to perform may depend on the results of all previous tests. Nevertheless, for many problems there are good "non adaptive" solutions, that involve performing tests in an *a priori* fixed order until the function is evaluated. Although non-adaptive solutions are more restrictive, they are often preferable in practice because they are faster to implement. Moreover, for all the systems mentioned above, there are non-adaptive algorithms that achieve constant-factor approximations relative to optimal adaptive solutions.

In this paper, we study sequential testing problems in the more general *batched* setting, that was introduced by [Dal+17]. Here, several tests may be performed simultaneously while incurring an extra setup cost. This setting is motivated by applications in healthcare and manufacturing where one can exploit economies of scale in testing. For example, conducting medical diagnostic tests typically involve significant setup costs related to administering the tests and transporting samples to a laboratory; so it is preferable to aggregate tests in order to avoid paying these setup costs repeatedly. Our main result is an approximation algorithm for batched sequential testing (for any system) that leverages any non-adaptive approximation algorithm for the classical sequential testing problem. Our approximation ratio for the batched setting is only a small constant more than the classical (unbatched) setting. Combined with previously-known approximation algorithms in the classic setting, we obtain small constant-factor approximation algorithms for batched sequential testing for series-systems, k-of-n and score-classification.

1.1 Problem Definition

An instance of the sequential testing problem consists of a system with n stochastic components, indexed $[n] \coloneqq \{1, \ldots, n\}$. Each component $i \in [n]$ is "working" independently with probability p_i, and is "faulty" otherwise. The state of any component $i \in [n]$ is represented by a random variable (r.v.) $X_i \in \{0, 1\}$ where $X_i = 1$ if i is working and $X_i = 0$ if it is faulty. All the probabilities $\{p_i\}_{i=1}^n$ are known a priori. However, the status X_i of each component i can only be determined by testing it. The overall state of the system is given by a function $f : \{0, 1\}^n \to \mathbb{Z}^+$, and the goal is to evaluate $f(X_1, \ldots X_n)$. It is important to note that we may not need to test all components in order to determine the function value.

Unbatched sequential testing (`UST`). In the classical setting, denoted `UST`, components are tested individually. Testing any component $i \in [n]$ incurs cost c_i. The goal is to evaluate function f at the minimum expected cost.

Batched sequential testing (`BST`). In the batched setting that we consider, multiple components may be tested simultaneously. A *batch* refers to any subset of components that get tested together. Testing any batch incurs an additional setup cost of β (even if the batch consists of a single component). So, the cost to test any batch $S \subseteq [n]$ is $c(S) := \beta + \sum_{i \in S} c_i$. There are no restrictions on which components may be tested together in a batch. Again, the goal is to evaluate f at the minimum expected cost. Observe that setting $\beta = 0$ in `BST` recovers the classical setting (`UST`).

Policies for sequential testing. A solution for `BST` corresponds to a policy that at every step, determines the next batch to test based on the statuses of all previously tested components. Such policies are called *adaptive* because the sequence of tests performed depends on the random outcomes X_i. A simpler class of policies that we consider in this paper are *non-adaptive* policies: such a policy is described by a *fixed* sequence $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_m)$ of batches, where the $\mathcal{B}_j$'s partition

$[n]$. The non-adaptive policy $\mathcal{B}$ tests batches in the fixed order $\mathcal{B}_1, \mathcal{B}_2, \ldots$ until the function f is evaluated. If policy $\mathcal{B}$ terminates after testing K batches (which itself is a random variable) then its cost is $\sum_{j=1}^{K} c(\mathcal{B}_j)$. So, the expected cost of policy $\mathcal{B}$ is $C_{\exp}(\mathcal{B}) := \mathbb{E}_K\left[\sum_{j=1}^{K} c(\mathcal{B}_j)\right]$. In the special case of UST, any policy is just given by a sequence of singleton batches (i.e., each batch has a single component). We will also work with *randomized* non-adaptive policies. Such a policy is specified by a distribution $\mathcal{D}$ over batch-sequences, and involves first selecting a random sequence $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_m)$ according to $\mathcal{D}$ and then testing batches in this fixed order.

Adaptivity Gap. While non-adaptive policies are much simpler to describe and implement than adaptive ones, they may result in a worse objective value. The *adaptivity gap*, introduced by [DGV08], quantifies this loss in objective. It is the worst-case (over all instances) of the ratio of the optimal non-adaptive cost to the optimal adaptive cost. In this paper, we will focus on non-adaptive policies. All our applications also have small constant-factor adaptivity gaps.

Specific functions studied. We apply our main result to the following common systems/functions.

- *Series system.* This is the simplest type of system, which corresponds to determining if all components are working. That is, $f(X_1, \cdots X_n) = 1$ when $X_i = 1$ for each $i \in [n]$.
- *k-of-n system.* Here, the function evaluates to one if at least k components are working. That is, $f(X_1, \cdots X_n) = 1$ when $\sum_{i=1}^{n} X_i \geq k$ and $f(X_1, \cdots X_n) = 0$ otherwise.
- *Score classification.* This is a more general setting, where the function returns a categorical output. There are $m+1$ thresholds $\alpha_0 = 0 < \alpha_1 < \cdots < \alpha_m = n$ and the function value is $j \in \{1, \cdots m\}$ if the number of working components $\sum_{i=1}^{n} X_i$ lies between α_{j-1} and α_j.

1.2 Results and Techniques

Our main result is the following "conversion" theorem that obtains an algorithm for batched sequential testing using any non-adaptive algorithm for the unbatched problem.

Theorem 1.1. *Suppose that there is a non-adaptive ρ-approximation algorithm for unbatched sequential testing (UST) for function f. Then, there is a randomized non-adaptive algorithm for batched sequential testing (BST) for the same function f with approximation ratio $\frac{1}{2}\left(1 + \rho + \sqrt{1+\rho^2}\right) \leq \rho + \frac{1}{\sqrt{2}}$.*

Our algorithm starts with a ρ-approximate solution $\mathcal{U}$ for the unbatched setting (UST). Using the fact that this is non-adaptive, the tests in $\mathcal{U}$ can be ordered *linearly* on a timeline representing cumulative testing cost. Then, we partition this timeline into batches of width depending on β (setup cost) and ρ (approximation ratio for UST), which results in our batched solution for BST. In order to optimize our approximation ratio, we choose the precise width carefully and also use a random offset in forming batches. The runtime of our (randomized) batched algorithm is essentially the same as the runtime of the underlying unbatched algorithm: we only incur additional $O(n)$ time to form the batches.

We also show that our unbatched-to-batched conversion method is nearly the best possible. When we start with an optimal unbatched solution (i.e., $\rho = 1$ above), our unbatched-to-batched conversion has approximation ratio 1.707. In this case, we show that *any* generic unbatched-to-batched conversion must incur a ratio of at least 1.697. In order to show this, we formulate a factor-revealing linear program (LP) that calculates the worst-case ratio (over all BST instances) of the best unbatched-to-batched conversion method. This LP has an exponential number of variables and we solve it numerically using column generation, to obtain the 1.697 lower boud. We note that

this limitation only applies to generic unbatched-to-batched conversions, and better approximation ratios for specific BST problems may still be possible.

Combining Theorem 1.1 with previously-known non-adaptive algorithms for UST we obtain the following batched sequential testing results.

- A 1.707-approximation algorithm for BST of series systems.
- A 2.618-approximation algorithm for BST of k-of-n systems.
- A 6.371-approximation algorithm for BST of score classification.

Our result for series-systems improves over the 6.829-approximation algorithm obtained in [Dal+16]. While [SS22] obtained an even better approximation ratio (a polynomial time approximation scheme) for BST of series systems, our algorithm is much more efficient, requiring only $O(n \log n)$ time. A (large) constant-factor approximation for batched testing of k-of-n systems and score classification follows from the work of [GGN23] on the (more general) weighted score classification problem. Our result provides algorithms with much smaller approximation ratios for BST of k-of-n systems and score classification.

We also obtain a deterministic algorithm achieving the above approximation ratios. This involves evaluating the expected cost of n different BST policies (corresponding to different offsets in the randomized algorithm). For the batched score-classification problem, which is our most general application, we obtain an $O(n^2)$ deterministic runtime using the fact that all n policies are derived from the *same* unbatched policy.

Finally, we computationally evaluate our algorithm for batched series-systems, k-of-n and SSClass. We use randomly-generated instances as in prior work of [SS22], which was only for batched series-systems. We compare our algorithm to the optimal adaptive policy, which we compute via dynamic programming. Our algorithm performs very well: its cost is typically within 25% of the optimum and its runtime is faster by several orders of magnitude.

1.3 Related Work

The sequential testing problem is crucial in engineering systems and has been well studied, with surveys by [Ünl04] and [Mor82]. There has been extensive work in the classical (unbatched) setting. For series systems, the natural greedy algorithm that performs tests in non-decreasing order of cost to failure-probability ratio is known to be optimal; see, [But72] and [Mit60]. The same algorithm also works for "parallel systems" where the system is said to work if any one component is working.

For (unbatched) k-of-n systems, [Ben81] obtained an optimal adaptive algorithm.[Gke+18] obtained a non-adaptive 2-approximation algorithm for k-of-n systems, relative to the optimal adaptive policy. Recently, [Gra+22] obtained an improved non-adaptive 1.5-approximation for *unit cost* k-of-n systems; they also showed that the adaptivity gap is exactly 1.5 in this case.

The (unbatched) score classification problem was introduced by [Gke+18]. This paper obtained two adaptive approximation algorithms with ratios $O(\log n)$ and $m-1$; the first algorithm was based on a connection to stochastic submodular cover (see [INZ16] and [DHK16]) and the second algorithm used the k-of-n algorithm from [Ben81]. Recently, [PS22] and [Liu22] obtained non-adaptive approximation algorithms with ratios 5.828 and 6 respectively; both these results are relative to the optimal adaptive policy (and hence bound the adaptivity gap). [Gra+22] considered the special case of score classification with *unit-cost* tests and obtained a non-adaptive 2-approximation algorithm (again, relative to the optimal adaptive policy). As observed in [Gke+22], algorithms for score classification can also be used to evaluate any *symmetric* function f where the value only depends on the number of components with $X_i = 1$.

[Gke+18] and [GGN23] studied a more general *weighted* score classification problem, where each component has a weight and the thresholds α_js are applied to the total weight of working components. [Gke+18] obtained an adaptive algorithm with a logarithmic approximation ratio. Subsequently, [GGN23] obtained a non-adaptive constant-factor approximation algorithm (and adaptivity gap). The weighted score classification problem with $m = 2$ classes is exactly the halfspace-evaluation problem, for which an adaptive 3-approximation algorithm was obtained previously by [DHK16]; a non-adaptive algorithm with a much larger constant factor also follows from the work of [Jia+20].

Sequential testing under batched costs was introduced by [Dal+17], together with efficient heuristics for series systems (without performance guarantees). Then, [Dal+16] provided a $6.829+\varepsilon$ approximation algorithm for batched series systems. Subsequently, [SS22] obtained a PTAS for this problem. Note that adaptive and non-adaptive policies coincide for series systems: so the adaptivity gap is one in this case.

[GGN23] also studied the weighted score classification problem under batched costs, and obtained a constant-factor approximation algorithm; while they did not try to optimize their constant factor, the resulting ratio is very large (over 1000). We note that their result for batched costs relies on specific properties of their unbatched algorithm. So it does not provide any improved approximation ratio for the batched versions of special cases such as k-of-n systems and unweighted score-classification – even though much smaller approximation ratios are known in the unbatched setting (as discussed above). In contrast, we provide a generic reduction from batched to unbatched costs that works for *any* function f and unbatched algorithm: so we obtain small constant-factor ratios for these special cases as well. Moreover, the increase in our approximation factor (going from the unbatched to the batched setting) is at most 0.71, which is much smaller than what was obtained in [GGN23] for weighted score classification.

2 Generic Algorithm for Batched Testing

In this section, we prove Theorem 1.1. We first describe the algorithm.

Step 1: using the unbatched algorithm. We apply the unbatched ρ-approximation algorithm to the underlying UST instance, given by n components with probabilities $\{p_i\}_{i=1}^n$, testing costs $\{c_i\}_{i=1}^n$ and the same function f. Basically, this step ignores the setup cost. Let $\mathcal{U}$ denote the resulting *non-adaptive* unbatched policy; recall that $\mathcal{U}$ is just a sequence of components. By renumbering components, let $\mathcal{U} = \langle 1, 2, \cdots n\rangle$. We refer to the cumulative cost in policy $\mathcal{U}$ as time. For each component $i \in [n]$ we define its *completion time* as $\text{time}(i) := \sum_{j=1}^{i} c_j$, which is the total cost of testing components until i (under policy $\mathcal{U}$).

Step 2: forming batches. Next, we set the *batch-width* $\gamma := r\beta$, where r is a constant (to be optimized later) and β is the setup cost for the BST instance. We also choose a random offset ξ uniformly from the interval $[0, 1)$. For each $j = 1, 2, \cdots$, we define the j^{th} batch to consist of all components with completion time between $(j - 2 + \xi)\gamma$ and $(j - 1 + \xi)\gamma$, i.e.,

$$\mathcal{B}_j := \{i \in [n] : (j - 2 + \xi)\gamma < \text{time}(i) \le (j - 1 + \xi)\gamma\}, \quad \forall j \in \mathbb{Z}_+, j \ge 1.$$

Empty batches are skipped. We denote the (randomized) batched policy generated above as $\mathcal{B}$.

A small example is presented in Figure 1.

In the rest of this section, we prove that this algorithm achieves the approximation ratio stated in Theorem 1.1. Let $\mathsf{OPT_b}$ (resp. $\mathsf{OPT_u}$) denote the cost of the optimal batched (resp. unbatched) policy.

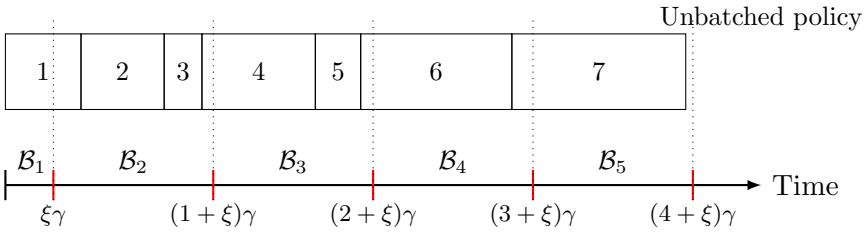

Figure 1: Example of the batching algorithm. The unbatched policy is placed onto a timeline and partitioned into batches of fixed width γ (except the first batch which has a random width $\xi\gamma$). In this example $\mathcal{B}_2 = \{1, 2, 3\}$, $\mathcal{B}_3 = \{4, 5\}$, $\mathcal{B}_3 = \{6\}$ and $\mathcal{B}_4 = \{7\}$. The empty interval $\mathcal{B}_1$ can be skipped without incurring any setup cost.

Lemma 2.1. *The optimal cost of the batched sequential testing instance is at least β more than the optimum of the unbatched instance, i.e., $\mathsf{OPT_b} \geq \beta + \mathsf{OPT_u}$.*

Proof of Lemma 2.1. Let $\mathcal{B}^*$ denote an optimal batched policy. We split the expected cost $\mathsf{OPT_b}$ of $\mathcal{B}^*$ into its expected setup cost (which is the number of batches times β) plus its expected testing cost (which accounts for the original costs c_i of the tested components). With probability one, $\mathcal{B}^*$ performs at least one batch of tests: so its expected setup cost is at least β. From $\mathcal{B}^*$, we construct an induced unbatched policy $\bar{\mathcal{U}}$ by testing components in the same order as $\mathcal{B}^*$, ordering tests within batches arbitrarily. This policy $\bar{\mathcal{U}}$ always terminates no later than the batched policy $\mathcal{B}^*$: note that $\bar{\mathcal{U}}$ may also terminate within a batch of $\mathcal{B}^*$. So, the expected cost of the unbatched policy $\bar{\mathcal{U}}$ is at most the expected testing cost of $\mathcal{B}^*$. Hence, the optimal unbatched cost $\mathsf{OPT_u}$ is also at most the expected testing cost of $\mathcal{B}^*$ and the result follows. □

Next, we upper bound the expected cost incurred by the randomized batched policy $\mathcal{B}$ in terms of the unbatched policy $\mathcal{U}$. In fact, we will prove such a bound for *every* realization.

Lemma 2.2. *Conditioned on any realization of the r.v.s $\{X_i\}_{i=1}^n$, we have*

$$\mathbb{E}_\xi[C_{exp}[\mathcal{B}]] \quad \leq \quad \left(1 + \frac{1}{r}\right) \cdot C_{exp}[\mathcal{U}] + \left(\frac{r}{2} + 1\right) \cdot \beta.$$

Hence, $\mathbb{E}_\xi[C_{exp}[\mathcal{B}]] \leq \max\left\{1 + \frac{1}{r}, \frac{r}{2} + 1\right\} \cdot \left(C_{exp}[\mathcal{U}] + \cdot\beta\right)$.

Proof of lemma 2.2. We condition on a realization of the r.v.s $\{X_i\}_{i=1}^n$ throughout this proof. Suppose that the unbatched solution $\mathcal{U}$ terminates at time t and tests items $\{1, 2, \cdots i\}$. As the batched solution $\mathcal{B}$ tests the components in the same sequence as $\mathcal{U}$, it will test any component that $\mathcal{U}$ tests, plus any additional components in the same batch as component i.

Let K be the number of batches that policy $\mathcal{B}$ performs. Then, the total setup cost of policy $\mathcal{B}$ is $\beta \cdot K$. Moreover, the total cost of tested components is at most $(K - 1 + \xi) \cdot \gamma$ because the batch containing component i ends at this time. Note that K is a random variable even though we have conditioned on $\{X_i\}_{i=1}^n$: this is because K also depends on the random offset ξ. In order to bound the expected cost of $\mathcal{B}$, it is convenient to decompose $\mathcal{U}$'s termination time as $t = (k + q)\gamma$ where $k \in \mathbb{Z}_+$ and $q \in [0, 1)$. Then,

$$K = \begin{cases} k + 2 & \text{if } \xi < q \\ k + 1 & \text{if } \xi \geq q \end{cases}$$

As $\xi \in [0,1)$ uniformly, $\Pr[\xi < q] = q$ and $\mathbb{E}_\xi[K] = k + 1 + q$. Therefore,

$$\begin{aligned}\mathbb{E}_\xi[C_{\exp}[\mathcal{B}]] &= \mathbb{E}_\xi[\beta \cdot K + (K - 1 + \xi) \cdot \gamma] \\ &= (\beta + \gamma) \cdot \mathbb{E}_\xi[K] - \gamma/2 \\ &= (\beta + \gamma) \cdot (k + q) + \beta + \gamma/2 \\ &= \left(1 + \frac{1}{r}\right) \cdot t + \left(\frac{r}{2} + 1\right) \cdot \beta\end{aligned}$$

The last equality uses $\gamma = r \cdot \beta$ and $t = (k + q) \cdot \gamma$. This completes the proof of the lemma as $C_{\exp}[\mathcal{U}] = t$ (conditioned on the realization of X_is). □

Using Lemma 2.2 and taking expectation over the realizations $\{X_i\}_{i=1}^n$,

$$\begin{aligned}C_{\exp}[\mathcal{B}] &\leq \left(1 + \frac{1}{r}\right) \cdot C_{\exp}[\mathcal{U}] + \left(\frac{r}{2} + 1\right) \cdot \beta \\ &\leq \rho \left(1 + \frac{1}{r}\right) \cdot \mathsf{OPT}_{\mathsf{u}} + \left(\frac{r}{2} + 1\right) \cdot \beta \qquad (1) \\ &\leq \max\left\{\rho\left(1 + \frac{1}{r}\right), \left(\frac{r}{2} + 1\right)\right\} \cdot \mathsf{OPT}_{\mathsf{b}} \qquad (2) \\ &= \frac{1}{2}\left(1 + \rho + \sqrt{1 + \rho^2}\right) \cdot \mathsf{OPT}_{\mathsf{b}}. \qquad (3)\end{aligned}$$

Inequality (1) uses the ρ-approximation ratio for the unbatched instance and (2) is by Lemma 2.1. Finally, (3) is by choosing parameter $r = \rho - 1 + \sqrt{1 + \rho^2}$ so that $\rho\left(1 + \frac{1}{r}\right) = \frac{r}{2} + 1$. Also, note that the approximation ratio $\frac{1}{2}\left(1 + \rho + \sqrt{1 + \rho^2}\right) \leq \rho + \frac{1}{\sqrt{2}}$ for all $\rho \geq 1$. This completes the proof of Theorem 1.1.

3 Lower bound on unbatched to batched conversion

In Section 2, we obtained a generic method applicable to any sequential testing problem that converts a non-adaptive unbatched policy $\mathcal{U}$ into a batched policy $\mathcal{B}$. By creating batches of a fixed width $r\beta$ and using a uniformly random offset ξ, we showed in Lemma 2.2 that:

$$C_{\exp}[\mathcal{B}] \leq \max\left\{1 + \frac{1}{r}, \frac{r}{2} + 1\right\} \cdot (C_{\exp}[\mathcal{U}] + \beta). \qquad (4)$$

The best approximation ratio that our approach achieves for the unbatched-to-batched conversion is $1 + \frac{1}{\sqrt{2}} \approx 1.707$ (setting $r = \sqrt{2}$ above). A natural question that arises is whether a smaller ratio can be obtained using a different unbatched-to-batched conversion scheme, e.g., by choosing batches of variable widths or a different offset distribution. In order to address this question, we first formalize what it means to be an unbatched-to-batched conversion method.

Given a non-adaptive unbatched policy $\mathcal{U}$ (which is a permutation of the n components), a non-adaptive batched policy $\mathcal{B} = \langle \mathcal{B}_1, \cdots \mathcal{B}_m \rangle$ is said to be $\mathcal{U}$-consistent if, for every $j \in [m]$, the components $\bigcup_{\ell=1}^{j} \mathcal{B}_\ell$ in the first j batches form a *prefix* of $\mathcal{U}$. In other words, a $\mathcal{U}$-consistent batched policy respects the relative ordering of components imposed by $\mathcal{U}$. Any unbatched-to-batched conversion scheme (applied to $\mathcal{U}$) must return a $\mathcal{U}$-consistent batched policy. An unbatched-to-batched conversion scheme is said to have approximation ratio α if for any unbatched policy $\mathcal{U}$, it returns a $\mathcal{U}$-consistent batched policy $\mathcal{B}$ such that $C_{\exp}[\mathcal{B}] \leq \alpha \cdot (C_{\exp}[\mathcal{U}] + \beta)$. In this section,

we prove that any unbatched-to-batched conversion scheme has approximation ratio at least 1.697. This shows that our 1.707 approximation ratio is nearly the best possible. We note that this "lower bound" is only for general unbatched-to-batched conversions: better approximation ratios may still be achievable for specific batched sequential testing problems (using methods that do not use unbatched solutions in a black-box manner).

We derive the lower bound by setting up a "factor revealing" problem that finds an instance of BST and an unbatched policy $\mathcal{U}$ where the ratio $\frac{C_{\exp}[\mathcal{B}]}{C_{\exp}[\mathcal{U}]+\beta}$ is high for *every* $\mathcal{U}$-consistent batched policy $\mathcal{B}$.

$$\max_{\substack{\texttt{BST instance} \\ \text{unbatched policy } \mathcal{U}}} \quad \min_{\substack{\mathcal{U}-\text{consistent} \\ \text{unbatched policy } \mathcal{B}}} \quad \frac{C_{\exp}[\mathcal{B}]}{C_{\exp}[\mathcal{U}]+\beta}. \tag{5}$$

Above, the inner minimization finds the best-possible batching of the given unbatched policy $\mathcal{U}$. The outer maximization finds the worst-case ratio over all BST instances (i.e., costs, probabilities and function f) and all unbatched policies.

We will reformulate the factor-revealing problem (5) as a linear program (LP) and solve it numerically. To this end, we first fix the setup cost β and a finite horizon T corresponding to an upper limit on the total cost of components (we also restrict to instances with integer costs). In order to find a larger lower-bound, the horizon T should be as large as practically possible. Next, we use p_t (for $t = 1, 2, \cdots T$) to denote the probability that policy $\mathcal{U}$ stops at time t; so $\sum_{t=1}^{T} p_t = 1$. Note that the p_ts depend on both the unbatched policy $\mathcal{U}$ and the function f to be evaluated. So, the expected cost of the unbatched policy is $C_{\exp}[\mathcal{U}] = \sum_{t=1}^{T} t \cdot p_t$ and $C_{\exp}[\mathcal{U}] + \beta = \sum_{t=1}^{T} (t+\beta) p_t$.

Representing batched policies. Viewed this way, a $\mathcal{U}$-consistent batched policy $\mathcal{B}$ is given by a sequence $\langle t_1, t_2, \cdots t_m \rangle$ of times, where the j^{th} batch contains all components that complete between time t_j and $t_{j+1} - 1$. For any such batched policy $\mathcal{B}$ and time t, let $C(\mathcal{B}, t)$ denote the cost incurred by $\mathcal{B}$ when the unbatched policy $\mathcal{U}$ stops at time t; i.e.,

$$C(\mathcal{B}, t) = \beta \cdot |\{j : t_j \leq t\}| + \min\{t_j - 1 : t_j > t\}.$$

The first term above is the setup cost of all batches so far (including the one containing t) and the second term is the total testing cost in these batches. Note that $C_{\exp}[\mathcal{B}] = \sum_{t=1}^{T} C(\mathcal{B}, t) \cdot p_t$. Let Σ denote the set of all $\mathcal{U}$-consistent batched policies.

LP reformulation. With the above definitions, problem (5) can be rewritten as:

$$\max_{\{p_t\}:\text{probabilities}} \quad \min_{\mathcal{B} \in \Sigma} \quad \frac{\sum_{t=1}^{T} C(\mathcal{B}, t) \cdot p_t}{\sum_{t=1}^{T} (\beta + t) \cdot p_t}.$$

We now replace the inner min operator with max, by using an auxiliary variable λ. Also, note that the above ratio objective is homogenous in p_t: so its value remains unchanged under any uniform (positive) scaling of p_t. So, we can linearize the ratio objective by re-scaling the p_t and adding a constraint for the denominator. Altogether, we obtain the following equivalent problem, which is an LP.

$$\begin{aligned}
\max \quad & \lambda & & \\
\text{s.t.} \quad & \lambda & \leq & \sum_{t=1}^{T} C(\mathcal{B}, t) \cdot p_t \qquad \forall \mathcal{B} \in \Sigma, \\
& \sum_{t=1}^{T} (\beta + t) \cdot p_t & \leq & 1, \\
& p & \geq & 0.
\end{aligned} \tag{P}$$

This LP has an exponential number of constraints, which makes it difficult to solve numerically. Instead, we work with its dual, to which we can apply column generation. The dual linear program is as follows.

$$\begin{array}{lrcll} \min & \mu & & & \\ \text{s.t.} & \sum_{\mathcal{B}\in\Sigma} y_{\mathcal{B}} & = & 1, & \\ & (\beta+t)\cdot\mu - \sum_{\mathcal{B}\in\Sigma} C(\mathcal{B},t)\cdot y_{\mathcal{B}} & \geq & 0 & \forall t\in[T], \\ & \mu, y & \geq & 0. & \end{array} \tag{D}$$

In order to solve (D) by column generation, we need an efficient algorithm to solve the associated "pricing problem", defined as follows. Given (non negative) values $\{p_t\}_{t=1}^T$, find $\mathcal{B}\in\Sigma$ that minimizes $\sum_{t=1}^T C(\mathcal{B},t)\cdot p_t$. Any algorithm to solve the pricing problem can be used within the column-generation method for solving (D). We provide a simple dynamic program to solve this pricing problem.

Solving the pricing problem. We first calculate the "non completion" probabilities of the unbatched solution $\mathcal{U}$ at all times, i.e.,

$$q_t = \sum_{t'\geq t} p_t, \quad \forall t\in[T].$$

For each time ℓ, we compute the value function $v(\ell)$ that minimizes the expected cost contribution from times $\{\ell,\ell+1,\cdots T\}$ over all batched solutions $\mathcal{B}\in\Sigma$. Note that $v(1)$ is the optimal value of the pricing problem. In order to calculate $v(\ell)$, the dynamic program tries all choices for the start time z of the next batch, i.e., the first batch is $\{\ell,\ell+1,\cdots z-1\}$. Value $v(\ell)$ is therefore the sum of contributions from the batch $[\ell,z-1]$ and all future batches (which is just $v(z)$), that is:

$$v(\ell) = \min_{\ell+1\leq z\leq T+1}\left[v(z) + \underbrace{\beta\cdot q_\ell}_{\text{batch setup cost}} + \underbrace{(z-\ell-1)\cdot q_\ell}_{\text{batch testing cost}}\right].$$

We also have $v(T+1)=0$ as the base case.

Numerical computation. We run the factor program with time horizon $T=500$ and $\beta=\{10,20,\ldots,500\}$. The runs were executed on an Apple M2 chip with 16 GB of memory. The LP took an average of 461 s to solve, and generated 1555 columns on average. The highest factor achieved was 1.697 with $\beta=70$, which took 1049 s and generated 2335 columns. The factor of 1.697 achieved is close to our 1.707 upper bound, suggesting that the general conversion algorithm from Section 2 is almost tight.

4 Applications

In this section, we describe several applications of our result. In each of these cases, we obtain (small) constant-factor approximation algorithms for the batched problem (`BST`) using Theorem 1.1 and previously-known non-adaptive algorithms for the unbatched problem (`UST`). We note that all the approximation ratios are relative to optimal adaptive policies: so we also bound the adaptivity gaps of all these batched testing problems. Figure 2 shows the relationships between the different problems. Table 1 summarizes these results: it provides the unbatched and batched approximation ratios as well as the running times. We note that the runtime of the (randomized) batched algorithm

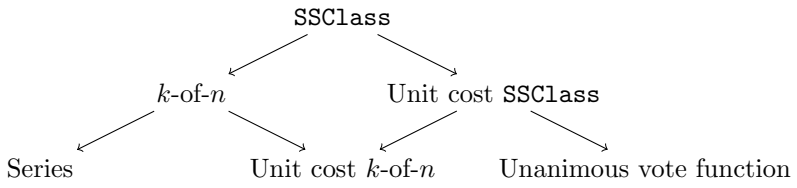

Figure 2: Systems studied in this paper in decreasing generality

Table 1: Summary of results for specific systems

Problem	**Unbatched ratio ρ**	**Runtime**	**Batched ratio**
Series system (AND)	1	$O(n \log n)$	1.707
Unit cost k-of-n	1.5	$O(n^2)$	2.151
k-of-n	2	$O(n \log n)$	2.618
Unanimous vote	1.618	$O(n^2)$	2.260
Unit cost `SSClass`	2	$O(n \log n)$	2.618
`SSClass`	5.828	$O(n \log n)$	6.371

in Theorem 1.1 is just the runtime of the unbatched algorithm: the unbatched-to-batched conversion only takes linear time.

Series systems: The simplest systems are *series systems*, where f is the AND-function, i.e. we need all components of the system to be working. Here, if any component fails then the state of the system is 0, and testing stops. Otherwise, we continue to test until it is determined that all components are working (in which case the system state is 1). It is well known that testing components in increasing order of the ratio $c_i/(1-p_i)$ is optimal for (unbatched) series systems, see e.g., [But72]. This is clearly a non-adaptive algorithm: so Theorem 1.1 applies with $\rho = 1$. While our approximation ratio of 1.707 is more that the $1+\epsilon$ ratio obtained in [SS22], our algorithm is much simpler and its $O(n \log n)$ runtime is much better than the $n^{O(1/\epsilon)}$ time in the previous paper.

k-of-n system: A k-of-n system works as long as the number of working components is at least some threshold k. Setting $k = n$ recovers the *series* system. For k-of-n systems, we can stop testing once we have found k working components or $n-k+1$ failures. In the *unbatched* setting, an optimal adaptive algorithm is known for k-of-n systems, but our result cannot be applied to this. However, there is also a non-adaptive 2-approximation algorithm ([Gke+18]), to which Theorem 1.1 can be applied.

In the special case of k-of-n with unit-cost tests (i.e., $c_i = 1$ for all i), [Gra+22] obtained a better non-adaptive algorithm with approximation ratio 1.5. They also showed that the adaptivity gap is exactly 1.5 for unit-cost k-of-n.

Score classification (`SSClass`): This is a more general problem, introduced by [Gke+18], that allows for categorical output. This models applications in healthcare for example, where patients are often diagnosed as having {*low*, *moderate*, *high* etc.} susceptibility to a disease based on the number of positive/negative test results. Similarly, in manufacturing, one may want to sort products into different quality brackets based on the number of tests they pass. Formally, there are $m+1$ thresholds $\alpha_0 = 0 < \alpha_1 < \cdots < \alpha_m = n$, where the function evaluates to j if the number of working components $\sum_{i=1}^{n} X_i$ is between α_{j-1} and α_j. When $m = 2$, the `SSClass` problem reduces to k-of-n. Recently, [PS22] and [Liu22] obtained non-adaptive approximation algorithms for (unbatched) `SSClass` with approximation ratios 5.828 and 6 respectively. So, Theorem 1.1 can be applied to this setting.

The special case of SSClass with unit-cost tests was also considered separately in [Gra+22], where they obtained a better 2-approximation algorithm, again via a non-adaptive policy. So, using Theorem 1.1 with this improved algorithm gives a better approximation ratio for batched SSClass in the unit-cost case. [Gra+22] also considered a special case of unit-cost SSClass, namely *unanimous vote* function. This problem has $m = 3$ classes where the categories correspond to $\sum_{i=1}^{n} X_i = 0$ (all-negative), $1 \le \sum_{i=1}^{n} X_i \le n-1$ (uncertain) and $\sum_{i=1}^{n} X_i = n$ (all-positive). [Gra+22] gave a non-adaptive $\frac{1+\sqrt{5}}{2} \approx 1.618$ approximation algorithm for (unbatched) unanimous vote function. Combined with Theorem 1.1, we again obtain an (improved) approximation algorithm for the batched version.

We note that [GGN23] obtained a constant-factor non-adaptive algorithm for the more general *weighted* SSClass problem, where each component i has a weight w_i and the thresholds are applied to the total weight $\sum_{i=1}^{n} w_i X_i$ of working components. In fact, they also extended their result to the batched setting. However, their approximation ratios for both unbatched and batched settings, are large constants (over 1000). Theorem 1.1 can indeed be applied to the unbatched algorithm from [GGN23] to get an improved approximation ratio for batched weighted SSClass: but the resulting approximation ratio is still a large constant.

5 Deterministic Algorithm

In this section, we discuss how the (randomized) algorithm in Theorem 1.1 can be made deterministic.

Recall that the only randomized step in this algorithm is in choosing the random offset ξ. For any fixed offset o, the resulting batched policy $\mathcal{B}(o)$ is deterministic. Moreover, the expected cost of the randomized batch policy is $\mathbb{E}_\xi[C_{\exp}(\mathcal{B}(\xi))]$. In order to obtain a deterministic policy, we can compute the batched policies $\mathcal{B}(o)$ for all offsets o, and return the one with the minimum objective. The approximation ratio remains the same because

$$\min_o C_{\exp}(\mathcal{B}(o)) \le \mathbb{E}_\xi[C_{\exp}(\mathcal{B}(\xi))].$$

Although there are a large number of choices for the offset o, the number of distinct policies $\mathcal{B}(o)$ is at most $n+1$. So, we need to evaluate the objective of these $O(n)$ policies in order to determine the best one. A naive implementation requires n times the runtime of a single policy evaluation. Below, we show that a better overall runtime can be obtained using the fact that these $n+1$ batched policies are all derived from the same unbatched policy. For concreteness, we focus on SSClass, which is our most general application.

Theorem 5.1. *Suppose that there is a non-adaptive ρ-approximation algorithm for unbatched score classification (SSClass) with runtime $T(n)$. Then, there is a deterministic $\frac{1}{2}\left(1+\rho+\sqrt{1+\rho^2}\right)$-approximation algorithm for batched SSClass with runtime $\mathcal{O}(T(n)+n^2)$.*

The first step in the algorithm just runs the unbatched approximation algorithm, which takes $T(n)$ time. We now analyze the runtime for the remaining steps. Recall that the non-adaptive unbatched policy is denoted $\mathcal{U} = \langle 1, 2, \cdots n\rangle$. We first tabulate values $\{P_{s,i} : 0 \le s, i \le n\}$ where $P_{s,i} = \Pr[\sum_{\ell=1}^{i} X_\ell = s]$ is the probability of observing exactly s working components among $\{1, 2, \cdots i\}$. This can be done in $O(n^2)$ time using the following recurrence:

$$P_{s,i} \;=\; p_i \cdot P_{s-1,i-1} + (1-p_{i-1}) \cdot P_{s,i-1}, \quad \forall 1 \le i \le n,\ 0 \le s \le n.$$

Using these values and the stopping criterion for `SSClass`, for each $i \in [n]$, we can compute the probability $Q(i)$ that the unbatched policy $\mathcal{U}$ stops after testing the first i components. This can also be implemented in $O(n^2)$ time.

We are now ready to evaluate the objective of any batched policy $\mathcal{B} = \langle \mathcal{B}_1, \cdots, \mathcal{B}_m \rangle$ derived from $\mathcal{U}$. Note that each batch $\mathcal{B}_j$ consists of a set of consecutive components; let e_j denote the last component in batch j. Then, the cost of batch j is $c(\mathcal{B}_j) = \beta + \sum_{i=e_{j-1}+1}^{e_j} c_i$. So, the objective value of policy $\mathcal{B}$ is

$$C_{\exp}(\mathcal{B}) = \sum_{j=1}^{m} Q(e_{j-1}) \cdot c(\mathcal{B}_j),$$

which can be computed in linear time. Finally, we only need to evaluate the objective of $n+1$ such batched policies, which requires $O(n^2)$ time in total. This completes the proof of Theorem 5.1.

6 Computational Experiments

We carry out computational experiments to evaluate the performance of our deterministic batched testing algorithm (Theorem 5.1). We implement this algorithm for three problem settings: series systems, k-of-n, and `SSClass`. The code is implemented in Python 3.12.4 and is run on an Apple M2 computer with 16 GB of memory.

6.1 Instances

For series systems (which is the simplest problem), we use a previously-used instance generation method from [SS22]. The costs are as follows:

- *Testing cost* c: Cost of testing any component i is set as $c_i \sim \text{Uniform}(1, 10)$.
- *Batch costs* β: We consider three batch costs $\beta = \{n/4, n/2, n\}$. These batch costs are calibrated so that the resulting batches are non-trivial.

There are two different scenarios for probabilities, which correspond to higher/lower likelihood of failure. Scenario 1 has each $p_i \sim \text{Uniform}(0.5, 1)$ and Scenario 2 has $p_i \sim \text{Uniform}(0.9, 1)$. We do not consider small p_i values as such instances tend to be easy for series systems: note that it suffices to observe just a single failure.

For the more complex settings of k-of-n and `SSClass`, we use the same testing/batch costs as above. However, for the probabilities, we just set each p_i uniformly from the range $[0, 1]$. For k-of-n systems, we consider three choices for $k = \{\lfloor n/4 \rfloor, \lfloor n/2 \rfloor, \lceil 3n/4 \rceil\}$. For `SSClass`, we consider $m = \{3, 4, 5\}$ classes and choose the thresholds α_j uniformly at random.

We consider instances with number of components n between 5 and 15. For each "configuration" (choice of n, β, scenario, k etc.), we generate 10 instances.

6.2 Optimal adaptive policy

We compare the performance of our approximate policy to the optimal adaptive policy. The optimal adaptive policy can be computed by dynamic programming (DP) in exponential time.

The state of the dynamic program is a tuple (T, w) where T is the set of tested components and w is the number of working components in T. We note that this state-space encapsulates all information required to evaluate function f for `SSClass`(and hence k-of-n and series systems). In particular, at state (T, w) we know that the number of working components is between w and

$w + n - |T|$: if this range is contained in some score interval $[\alpha_{j-1}, \alpha_j)$, then f returns j and the policy stops. The DP value function $v(T, w)$ equals the minimum (additional) expected cost to evaluate function f, conditioned on being at state (T, w). We can compute this value function by iterating over all possible batches of tests. Formally, the recurrence is:

$$v(T, w) = \min_{B \subseteq [n] \setminus T} \left[c(B) + \sum_{u=0}^{|B|} \Pr\left(\sum_{i \in B} X_i = u \right) \cdot v(T \cup B, w + u) \right].$$

The first term above is the cost of the batch B tested at state (T, w), which is incurred with probability one. The second term above is the expected cost of the optimal policy starting from the next state (which depends on the outcomes of batch B).

In order to obtain an efficient implementation, we first precompute the probabilities $\Pr\left(\sum_{i \in B} X_i = u\right)$ for all $B \subseteq [n]$ and $0 \leq u \leq n$, which takes time $\mathcal{O}(n \cdot 2^n)$. Then, evaluating value $v(T, w)$ for any state requires $\mathcal{O}(n \cdot 2^n)$ time, which corresponds to trying all choices for batch B. As the number of states is $2^n \times n$, the overall time is $\mathcal{O}(n^2 \cdot 4^n)$. In fact, a more careful calculation shows that the runtime is $\mathcal{O}(n^2 \cdot 3^n)$.

Our exact algorithm differs from that in [SS22] for series systems. The method in [SS22] iterates through all $n!$ permutations of unbatched policies σ, and finds the optimal batching for each σ. We note that their approach only yields an optimal *non-adaptive* policy, whereas our approach finds an optimal adaptive policy. While there is no benefit of adaptivity for batched series systems, for the more complex settings (k-of-n and `SSClass`) adaptive policies may be strictly better than non-adaptive: so it is important to compare to the adaptive optimum. Moreover, even for series systems, our implementation is faster, since it does not require enumerating through $n!$ permutations.

6.3 Results

We run our algorithm and the optimal adaptive algorithm with number of components $n = \{5, \ldots, 15\}$. The maximum size is capped at 15 to ensure that the optimal adaptive algorithm can terminate in a reasonable amount of time. For each of the problem settings, we tabulate the average and maximum empirical approximation ratio – the ratio of the cost achieved by our algorithm to the optimal adaptive cost.

The results for series-systems, k-of-n and `SSClass` are tabulated in Tables 2, 3 and 4 respectively. For series systems, for each n, we report the average and maximum ratios separately for the two scenarios. Here, each reported value is an aggregate of 30 instances, comprising 3 different batch-costs with 10 replications each. For k-of-n and `SSClass`, each reported value is an aggregate of 90 instances, comprising 3 different batch-costs with 3 values of k or number of intervals (and 10 replications each).

Our algorithm's performance was very good in each of the three systems: the empirical approximation ratios for series-systems, k-of-n and `SSClass`were at most 1.336, 1.821 and 1.827 respectively. These bounds are much better than the theoretical bounds of 1.707, 2.618 and 6.371 for these problems. The running time of our algorithm is also several orders of magnitude faster than the exact approach, taking just 1.45 milliseconds on average (for $n = 15$) compared to 183 seconds for the exact method.

7 Conclusion

We considered the batched version of sequential testing, which allows for multiple tests to be performed simultaneously. We obtained a generic batching algorithm that transforms any (non-

Table 2: Approximation ratio for series systems

		Number of components n										
		5	6	7	8	9	10	11	12	13	14	15
Scenario 1	**mean**	1.031	1.029	1.021	1.018	1.020	1.018	1.023	1.015	1.019	1.015	1.015
	max	1.116	1.133	1.082	1.074	1.077	1.060	1.084	1.037	1.075	1.047	1.045
Scenario 2	**mean**	1.146	1.159	1.182	1.191	1.199	1.201	1.188	1.201	1.206	1.211	1.195
	max	1.304	1.285	1.283	1.294	1.279	1.297	1.280	1.336	1.279	1.312	1.295

Table 3: Approximation ratio for k-of-n

	Number of components n										
	5	6	7	8	9	10	11	12	13	14	15
mean	1.182	1.193	1.209	1.231	1.298	1.295	1.297	1.303	1.301	1.327	1.319
max	1.712	1.749	1.668	1.626	1.777	1.731	1.703	1.648	1.821	1.741	1.622

Table 4: Approximation ratio for `SSClass`

	Number of components n										
	5	6	7	8	9	10	11	12	13	14	15
mean	1.038	1.057	1.065	1.100	1.090	1.106	1.118	1.104	1.137	1.146	1.154
max	1.226	1.481	1.414	1.509	1.469	1.827	1.603	1.713	1.535	1.645	1.523

adaptive) unbatched solution into a batched solution, while increasing the approximation ratio by a small amount. We also showed that our approximation ratio is nearly the best possible for any unbatched-to-batched conversion. Combined with existing results for the unbatched setting, our result provides a unified approximation algorithm for several batched testing problems, such as series systems, k-of-n systems and score classification.

References

[Ben81] Yosi Ben-Dov. "Optimal Testing Procedures for Special Structures of Coherent Systems". In: *Management Science* 27.12 (1981), pp. 1410–1420.

[But72] Richard Butterworth. "Some Reliability Fault-Testing Models". In: *Operations Research* 20.2 (1972), pp. 335–343.

[Dal+16] Rebi Daldal et al. "Approximation algorithms for sequential batch-testing of series systems". In: *Naval Research Logistics* 63.4 (2016), pp. 275–286.

[Dal+17] Rebi Daldal et al. "Sequential testing in batches". In: *Annals of Operations Research* 253.1 (2017), pp. 97–116.

[DGV08] Brian C. Dean, Michel X. Goemans, and Jan Vondrák. "Approximating the Stochastic Knapsack Problem: The Benefit of Adaptivity". In: *Mathematics of Operations Research* 33.4 (2008), pp. 945–964.

[DHK16] Amol Deshpande, Lisa Hellerstein, and Devorah Kletenik. "Approximation Algorithms for Stochastic Submodular Set Cover with Applications to Boolean Function Evaluation and Min-Knapsack". In: *ACM Transactions on Algorithms* 12.3 (2016), 42:1–42:28.

[GGN23] Rohan Ghuge, Anupam Gupta, and Viswanath Nagarajan. "Non-Adaptive Stochastic Score Classification and Explainable Halfspace Evaluation". In: *Operations Research (to appear)* (2023).

[Gke+18] Dimitrios Gkenosis et al. "The Stochastic Score Classification Problem". In: *26th Annual European Symposium on Algorithms*. Ed. by Yossi Azar, Hannah Bast, and Grzegorz Herman. Vol. 112. LIPIcs. 2018, 36:1–36:14.

[Gke+22] Dimitrios Gkenosis et al. "The Stochastic Boolean Function Evaluation Problem for Symmetric Boolean Functions". In: *Discrete Applied Mathematics* 309 (2022), pp. 269–277.

[Gra+22] Nathaniel Grammel et al. "Algorithms for the Unit-Cost Stochastic Score Classification Problem". In: *Algorithmica* 84.10 (2022), pp. 3054–3074.

[INZ16] Sungjin Im, Viswanath Nagarajan, and Ruben Van Der Zwaan. "Minimum Latency Submodular Cover". In: *ACM Transactions on Algorithms* 13.1 (2016), 13:1–13:28.

[Jia+20] Haotian Jiang et al. "Algorithms and Adaptivity Gaps for Stochastic k-TSP". In: *11th Innovations in Theoretical Computer Science Conference*. Vol. 151. 2020, 45:1–45:25.

[Liu22] Naifeng Liu. *Two 6-approximation Algorithms for the Stochastic Score Classification Problem*. 2022. URL: `http://arxiv.org/abs/2212.02370`.

[Mit60] L. Mitten. "An analytic solution to the least cost testing sequence problem". In: *Journal of Industrial Engineering* 11.1 (1960), p. 17.

[Mor82] Bernard M. E. Moret. "Decision Trees and Diagrams". In: *ACM Computing Surveys* 14.4 (1982), pp. 593–623.

[PS22] Benedikt M. Plank and Kevin Schewior. *Simple Algorithms for Stochastic Score Classification with Small Approximation Ratios*. 2022. URL: `http://arxiv.org/abs/2211.14082`.

[SS22] Danny Segev and Yaron Shaposhnik. "A Polynomial-Time Approximation Scheme for Sequential Batch Testing of Series Systems". In: *Operations Research* 70.2 (2022), pp. 1153–1165.

[Ünl04] Tonguç Ünlüyurt. "Sequential testing of complex systems: a review". In: *Discrete Applied Mathematics* 142.1-3 (2004), pp. 189–205.