Nonmyopic Multifidelity Active Search

Quan Nguyen [1] **Arghavan Modiri** [2] **Roman Garnett** [1]

Abstract

Active search is a learning paradigm where we seek to identify as many members of a rare, valuable class as possible given a labeling budget. Previous work on active search has assumed access to a faithful (and expensive) oracle reporting experimental results. However, some settings offer access to cheaper surrogates such as computational simulation that may aid in the search. We propose a model of *multifidelity* active search, as well as a novel, computationally efficient policy for this setting that is motivated by state-of-the-art classical policies. Our policy is nonmyopic and budget aware, allowing for a dynamic tradeoff between exploration and exploitation. We evaluate the performance of our solution on real-world datasets and demonstrate significantly better performance than natural benchmarks.

1. Introduction

The goal of *active search* is to identify members of a rare and valuable class among a large pool of unlabeled points. This is a simple model of many real-world discovery problems, such as drug discovery and fraud detection. Active search proceeds by successively querying an oracle that returns a binary label indicating whether or not a chosen data point exhibits the desired properties. In many applications, this oracle is expensive, limiting the number of queries that could be made. The challenge is to design a policy to sequentially query the oracle in order to discover as many targets as possible, subject to a given labeling budget.

Active search has been extensively studied under various settings (Garnett et al., 2012; Jiang et al., 2017; 2018; 2019). Notably, Jiang et al. (2017) proved a hardness of approximation result, showing that no polynomial-time policy can approximate the performance of the Bayesian optimal policy within any constant factor. Thus active search is surprisingly hard. However, the authors also proposed an efficient approximation to the optimal policy that delivers impressive empirical performance. The key feature of their policy is its ability to account for the remaining budget and dynamically trade off exploration and exploitation.

Most previous work on active search has assumed access to only a single expensive oracle providing labels. In practice, however, we may have several methods of probing the search space, including cheap, low-fidelity surrogates. For example, a computational simulation may serve as a noisy approximation to an experiment done in a laboratory, and we may reasonably seek to use a cheaper computational search to help design the expensive experiments. This motivates the problem of *multifidelity active search,* where oracles of different fidelities may be simultaneously accessed to accelerate the search process. The central question in this task is how to effectively leverage these oracles in order to maximize the rate of discovery.

We present a model for multifidelity active search and study the problem under the framework of Bayesian decision theory. We propose a novel policy for this setting inspired by the state-of-the-art single-fidelity policy mentioned above. The result is an efficient approximation to the optimal multifidelity policy that is specifically tailored to take advantage of low-fidelity oracles. We also create aggressive branch-and-bound pruning strategies to increase the efficiency of our proposed algorithm, enabling scaling to large datasets. In a series of experiments, we investigate the performance of our policy on several real-world datasets for scientific discovery. Our solution outperforms various baselines from the literature by a large margin.

2. Problem Definition

We first introduce the general active search model and notations. Suppose we are given a finite set of points $\mathcal{X} \triangleq \{x_i\}$, which includes a rare, valuable subset $\mathcal{R} \subset \mathcal{X}$. The members of $\mathcal{R}$, which we call *targets* or *positives*, are not known *a priori*, but whether a given point $x \in \mathcal{X}$ is a target can be determined by making a query to an oracle that returns the binary label $y \triangleq \mathbb{1}\{x \in \mathcal{R}\}$. We assume that querying the oracle is expensive and that we only have a limited budget of t queries to do so. We denote a given dataset of queried points and their corresponding labels as $\mathcal{D} = \{(x_i, y_i)\}$. At

[1]Washington University in St. Louis, MO, USA [2]University of Toronto, Toronto, Canada. Correspondence to: Quan Nguyen <quan@wustl.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

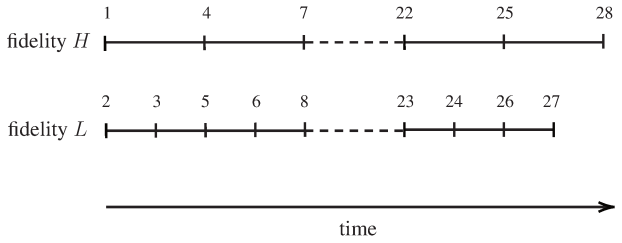

Figure 1. An illustration of our multifidelity active search model. Each short vertical line indicates when a query finishes and the next query is made. The numbers indicate the order in which queries are made across the two oracles. The low-fidelity oracle is $k = 2$ times faster than the high-fidelity oracle. The budget on H is $t = 10$; in total, $T = t + kt - k = 28$ queries are made.

times, we will use $\mathcal{D}_i$ to denote the dataset collected after i queries.

2.1. Multifidelity Active Search

In addition to the expensive oracle that returns the exact label of a query, we have access to other cheaper but more noisy oracles that are low-fidelity approximations to the exact oracle. We limit our setting to two levels of fidelity—one exact oracle and one noisy oracle, denoted as H and L, respectively—but note that our proposed algorithm can be extended to more than two fidelities with minor modifications, as we will discuss later.

For a given point $x \in \mathcal{X}$, we denote its exact label on H as y_H and its noisy label on L as y_L. Under this setting, a dataset $\mathcal{D}$ can be partitioned into $\mathcal{D}_L$, the observations made on L, and $\mathcal{D}_H$, those that are made on H. Recall that our objective is to query as many targets as possible; as such, to express our preference over different datasets, we use the natural utility function

$$u\left(\mathcal{D} = \mathcal{D}_L \cup \mathcal{D}_H\right) \triangleq \sum_{y_i \in \mathcal{D}_H} y_i,$$

the number of targets discovered on the H fidelity.

We assume that queries to different oracles are run *in parallel*, but a high-fidelity query takes longer to complete than a low-fidelity one. In particular, we will assume that each query on H is k times slower than a query on L.[1] That is, each time we make a query on H, we may make k sequential queries on L while waiting for the result. The process will then repeat until the budget is depleted. This model aims to emulate real-life scientific discovery procedures, the main motivation for active search, where multiple fidelities (e.g., computational and experimental campaigns) are often run in parallel but vary in response time.

[1]This assumption is for simplicity; *asynchronous* queries could be addressed with a slight modification of our proposed algorithm.

If we are given a budget of t queries on H, we can make $kt - k$ queries on L; in total, $T = t + kt - k$ queries are made. (Although a total of kt queries can be made on L, after the final query on H is made at iteration T, further queries on L do not affect the final utility. We thus terminate after T queries.) This query schedule illustrated in Figure 1 for $t = 10$ and $k = 2$.

2.2. The Bayesian Optimal Policy

We now derive the optimal (expected-case) policy using Bayesian decision theory. First, we assume access to a probabilistic classification model that computes the posterior probability that a point $x \in \mathcal{X}$ is a target given a dataset $\mathcal{D}$, $\Pr\left(y_H = 1 \mid x, \mathcal{D}\right)$, as well as the posterior probability that the same x is a positive on L, $\Pr\left(y_L = 1 \mid x, \mathcal{D}\right)$.

Suppose we are currently at iteration $i + 1 \leq T$, having observed the dataset $\mathcal{D}_i$, and now need to make the next query, requesting the label of an unlabeled point x_{i+1}. The Bayesian optimal policy selects the point that maximizes the expected utility of the terminal dataset $\mathcal{D}_T$, assuming future queries will too be made optimally:

$$x^*_{i+1} = \operatorname*{arg\,max}_{x_{i+1} \in \mathcal{X} \setminus \mathcal{D}_i} \mathbb{E}\left[u\left(\mathcal{D}_T \setminus \mathcal{D}_i\right) \mid x_{i+1}, \mathcal{D}_i\right].$$

If the query is on H, the expectation is taken over the posterior distribution of the label of the putative query x_{i+1}; if the query is on L, it is over the *joint* distribution of the L label of x_{i+1} and the label of the pending H query (recall that we sequentially query k points on L while waiting for an H query that runs in parallel to finish).

To compute this expected utility, we follow the backward induction procedure described in Bellman (1957). In the base case, we are at iteration T and need to make the last H query and

$$\begin{aligned}
&\mathbb{E}\left[u\left(\mathcal{D}_T \setminus \mathcal{D}_{T-1} \mid x_T, \mathcal{D}_{T-1}\right)\right] \\
&\quad = \sum_{y_H} u\left(\mathcal{D}_T \setminus \mathcal{D}_{T-1}\right) \Pr\left(y_H \mid x_T, \mathcal{D}_{T-1}\right) \\
&\quad = \Pr\left(y_H = 1 \mid x_T, \mathcal{D}_{T-1}\right).
\end{aligned}$$

The optimal decision at this final step is therefore to greedily query the point most likely to be a target, maximizing $\Pr\left(y_H = 1 \mid x_T, \mathcal{D}_{T-1}\right)$. For the second-to-last query, which is on L, the expected utility is

$$\begin{aligned}
&\mathbb{E}\left[u\left(\mathcal{D}_T \setminus \mathcal{D}_{T-2} \mid x_{T-1}, \mathcal{D}_{T-2}\right)\right] \\
&\qquad = \mathbb{E}\left[\max_{x_T} \Pr\left(y_H = 1 \mid x_T, \mathcal{D}_{T-1}\right)\right].
\end{aligned}$$

This is the expected future reward to be collected at the next and final step, which we have shown to be optimally the greedy query. Again, at this iteration, the second-to-last

H query is still pending, so the expectation is taken over the joint distribution of the label of that query and that of the putative L query. In general, we compute this expected utility at iteration $i+1 \leq T$ for an L query recursively as

$$\mathbb{E}\big[u\left(\mathcal{D}_T \setminus \mathcal{D}_i \mid x_{i+1}, \mathcal{D}_i\right)\big] = \mathbb{E}\left[\max_{x_{i+2}} \mathbb{E}\big[u\left(\mathcal{D}_T \setminus \mathcal{D}_{i+1}\right) \mid x_{i+2}, \mathcal{D}_{i+1}\big]\right]. \quad (1)$$

For an H query, this expected utility may still be recursively computed in the same manner but has a different expansion:

$$\mathbb{E}\big[u\left(\mathcal{D}_T \setminus \mathcal{D}_i \mid x_{i+1}, \mathcal{D}_i\right)\big] = \Pr\left(y_H = 1 \mid x_{i+1}, \mathcal{D}_i\right) + \mathbb{E}\left[\max_{x_{i+2}} \mathbb{E}\big[u\left(\mathcal{D}_T \setminus \mathcal{D}_{i+1}\right) \mid x_{i+2}, \mathcal{D}_{i+1}\big]\right]. \quad (2)$$

The new term in (2), $\Pr\left(y_H = 1 \mid x_{i+1}, \mathcal{D}_i\right)$, accounts for the possibility that our running reward increases if $x_{i+1} \in \mathcal{R}$, as we are querying on fidelity H. This is simply the probability that x_{i+1} is indeed a target. Also different from (1), the expectation is taken over the distribution of the label y_H of the putative point x_{i+1} only, as there is no pending query at this time. An intuitive interpretation of the sum in (2) is the balance between exploitation, the immediate reward $\Pr\left(y_H = 1 \mid x_{i+1}, \mathcal{D}_i\right)$, and exploration, the expected future reward $u\left(\mathcal{D}_T \setminus \mathcal{D}_{i+1}\right)$ conditioned on this putative query. Again, the exploitation term is not present in (1) when an L query is made, as the query cannot possibly increase our utility immediately.

Perhaps unsurprisingly, computing the optimal policy is a daunting task: the time complexity of computing the expectation term in (1) and (2) is $\mathcal{O}\big((4n)^\ell\big)$, where $\ell = T - i$ is the number of remaining queries and n is the number of unlabeled points. This optimal policy is computationally intractable, and suboptimal approximations are needed in practice. One natural solution is to shorten the lookahead horizon, pretending there are only $\ell' < T - i$ iterations remaining. This idea constitutes *myopic* policies, the most straightforward of which is the greedy strategy that queries the point with the highest probability of being a target in the single-fidelity setting, setting $\ell' = 1$. However, the design of a greedy policy for an L query is not obvious, as no immediate reward can be obtained by making the query.

2.3. Hardness of Approximation

In addition to demonstrating the intractability of the analogous Bayesian optimal policy in the classical single-fidelity setting, Jiang et al. (2017) proved that *no* polynomial-time policy can approximate the optimal policy (in terms of the expected terminal utility) by *any* constant factor. This was done by constructing an adversarial family of active search problems featuring "hidden treasures" that are provably difficult to uncover without exponential work. By modifying details of this construction, the performance of any polynomial-time policy can be made arbitrarily worse in expectation than that of the optimal policy.

This hardness result naturally extends to our multifidelity model, as even access to a *perfectly faithful* low-fidelity oracle cannot aid a polynomial-time active search policy in one of these adversarial examples. If we assume positives on L also count towards our utility, one such active search problem with a faithful low-fidelity oracle reduces to a single-fidelity problem with a budget on H increased by a factor of $(k+1)$. Unless k is *exponential* in the initial budget, any polynomial-time policy remains incapable of approximating the optimal policy. Under our model, only positives on H count towards our utility, so the performance of any policy is further reduced. We therefore obtain the same hardness of approximation result. However, we can still reasonably aim to design efficient approximations to the optimal policy that perform well in practice.

3. Efficient Nonmyopic Approximation

Our proposed algorithm is inspired by the ENS policy, introduced by Jiang et al. (2017) for the single-fidelity active search setting. ENS offers an efficient and nonmyopic approximation to the optimal policy by assuming that after the putative query, all remaining budget will be spent *simultaneously in one batch*. Under this heuristic, the optimal decision following the putative query is to greedily construct the batch with points having the highest probabilities. The expected utility of this batch is simply the sum of these highest probabilities due to linearity of expectation and therefore can be computed efficiently. An interesting interpretation by Jiang et al. (2017) about ENS is that it matches the optimal policy given that after the putative query, the labels of the remaining unlabeled points are conditionally independent.

3.1. One-stage Approximation

As a stepping stone to our proposed policy, consider the following adoption of ENS. We reapply the heuristic by assuming that after the putative query, which can be on either L or H, all remaining H queries will be made simultaneously in one batch. Again, the optimal choice for this batch is the set of most promising points, which we will refer to as the greedy H batch. Using the summation-prime symbol $\sum'_s$ to denote the sum of the top s terms, we approximate the maximum expected utility of the remaining portion of the search in (1) and (2) as

$$\max_{x_{i+2}} \mathbb{E}\big[u\left(\mathcal{D}_T \setminus \mathcal{D}_{i+1}\right) \mid x_{i+2}, \mathcal{D}_{i+1}\big] \approx \sum\nolimits'_{\ell_H} \Pr\left(y_H = 1 \mid x_{i+2}, \mathcal{D}_{i+1}\right), \quad (3)$$

where $\ell_H = \lfloor (T-i-1)/(k+1) \rfloor$ is the number of remaining H queries. The resulting policy queries the point that maximizes the expected utility in (1) and (2), using (3) as an approximation. This policy is cognizant of the remaining budget on H and performs well in our experiments (see appendix). However, by assuming that the greedy H batch will be queried immediately following the putative point, the strategy fails to consider future queries that could be made to the low-fidelity oracle in its lookahead; this motivates the design of our proposed policy described below.

3.2. Two-stage Approximation

Our main contribution is a policy we call MF–ENS, an efficient approximation to the optimal policy that additionally accounts for future L queries. As above, we assume in our lookahead that after the putative query, our H budget will be spent simultaneously. However, prior to committing to that final batch, we assume we may make $\overline{k} \leq k$ additional L queries (also simultaneously) with the goal of improving the expected utility of the final H batch. To faithfully emulate our search model, we set $\overline{k}$ to be the number of L queries remaining before the next H query is made.[2] We denote this batch of exploratory L queries as $X_L \subset \mathcal{X} \setminus \mathcal{D}_L$ and the corresponding labels as $Y_L \in \{0,1\}^{\overline{k}}$. In the language of *approximate dynamic programming* (Bertsekas, 1995), the policy described in the previous subsection is a *one-stage rollout* policy where the base policy selects the optimal batch on H following the putative query. MF–ENS on the other hand is a *two-stage rollout* policy whose base policy first queries the L batch X_L, and upon observing Y_L, adaptively queries the updated optimal H batch.

How should we construct X_L to best improve the greedy H batch at the second stage? One option would be to appeal to nonmyopic policies for *batch active search* (Jiang et al., 2018), but the best-promising batch policies become prohibitively expensive in this context as we would need to construct a new batch for *each* putative query and label. In general, the conditional dependence among the labels in the set Y_L poses a computational challenge in approximating the expected utility gained from querying a given X_L batch.

Recall that in the single-fidelity setting, the ENS policy is optimal if labels become conditionally independent after the chosen point. Let us make a similar assumption to ease computation: we assume that labels become conditionally independent after the putative query, *except* for the pair of labels (on H and L) corresponding to each point. That is, revealing the label y_L of a point x is allowed to affect our belief about the corresponding label y_H, but not the belief about any other label of any other point. This structure allows for efficiently sharing information between the fidelities and enables our multistage lookahead approach.[3]

We now aim to quantify the value of querying an unlabeled point on L in improving the final H batch. Our solution is motivated by a heuristic search for alternatives to the members of the greedy H batch that is assembled under the one-stage approximation. Given a putative query, we still construct that same greedy H batch. Then, for each candidate x not in the greedy H batch, we consider the expected marginal gain in utility of querying it on L and modifying the membership of the H batch in light of its newly revealed label y_L.

Recall that observing y_L only changes the distribution of y_H of the same x under our assumption. Let p_* be the lowest success probability among the current H batch and consider two cases. If $\Pr(y_H = 1 \mid x, \mathcal{D})$ *exceeds* p_* as y_L is revealed, we swap out the corresponding least-promising member of the batch with x and thus increase the final batch's expected utility. Otherwise, we do not modify the current batch. The value of querying x on L and observing y_L, denoted as $v(x \mid \mathcal{D})$, is then

$$v(x \mid \mathcal{D}) \triangleq \mathbb{E}_{y_L}\big[\max(\Pr(y_H = 1 \mid x, y_L, \mathcal{D}) - p_*, 0)\big].$$

This score can be rapidly computed for most models under our conditional independence assumption. With this value function in hand, we then greedily construct X_L with the candidates having the highest values. Another computational benefit of label independence is that $v(x \mid \mathcal{D})$ automatically vanishes for points already labeled on H, as querying its L label does not affect the belief of our model and thus cannot improve the H batch. This reduces our search space in computing the exploratory batch X_L.

We now proceed to the last step of the rollout procedure in MF–ENS: marginalizing over Y_L, the labels of X_L. As previously described, for each possible value of Y_L, we approximate the optimal sequence of queries following the putative one and X_L with the updated greedy H batch given the newly revealed labels Y_L:

$$\max_{x_{i+1}} \mathbb{E}\big[u(\mathcal{D}_T \setminus \mathcal{D}_i) \mid x_{i+1}, \mathcal{D}_i\big] \approx \mathbb{E}_{Y_L}\left[\sum\nolimits'_{\ell_H} \Pr(y_H = 1 \mid x, X_L, Y_L, \mathcal{D}_i)\right]. \quad (4)$$

At each iteration, MF–ENS queries the candidate maximizing the expected utility in (1) and (2), as approximated by (4). As an extension of ENS, our approach is nonmyopic and aware of the remaining budget; we will demonstrate the impact of this reasoning in our experiments. The policy also factors in future L queries, actively taking advantage of the ability to query the low-fidelity oracle. We give the pseudocode for the policy in the appendix.

[2] When making an H query, $\overline{k} = k$; when making an L query, we subtract the number of L queries since the pending H query.

[3] This is only used in policy construction and not in inference!

3.3. Extension to Multiple Low Fidelities

So far, we have assumed our model only consists of one exact oracle H and one noisy oracle L. To extend MF–ENS to settings where there are multiple noisy oracles $\{L_i\}$ that approximate H, potentially at different levels of fidelity, we still aim to design each query to maximize the expected utility on H, marginalizing future experiments on the $\{L_i\}$ oracles. Once again limiting the conditional dependence among labels to those of the same point, now between *each* lower-fidelity L_i and H, we identify a batch of appropriate size for each L_i, marginalize their labels, update the probabilities on H, and compute the approximate expected utility. When there is only one low fidelity, this strategy reduces to the base version of MF–ENS we have presented here.

3.4. Implementation and Pruning

Active search requires a classification model computing a given point's success probability with an oracle. We extend the k-nearest neighbor introduced by Garnett et al. (2012) to our multifidelity setting by allowing information observed on L to propagate to H. Specifically, when calculating the probability that an unlabeled point $x \in \mathcal{X}$ is a positive on H, $\Pr(y_H = 1 \mid x, \mathcal{D})$, we take into account the revealed labels of its nearest neighbors on both fidelities, as well as its own L label, y_L. Effectively, we treat each given point $x \in \mathcal{X}$ as having two separate copies: one corresponding to its H label, denoted as x_H, and one corresponding to its L label, denoted as x_L. Compared to the single-fidelity k-nearest neighbor, the set of nearest neighbors of x_H is now doubled to include both copies of its original neighbors and its own copy on L, x_L. Formally, denote $\text{NN}_{\text{single}}(x)$ as the original nearest neighbor set of x. The nearest neighbor set of x_H under our multifidelity predictive model is

$$\begin{aligned}\text{NN}(x_H) \triangleq \{x_L\} &\cup \left\{x'_H : x' \in \text{NN}_{\text{single}}(x)\right\} \\ &\cup \left\{x'_L : x' \in \text{NN}_{\text{single}}(x)\right\}.\end{aligned}$$

To account for the unknown accuracy of the low-fidelity oracle, we apply a damping factor $q \in (0, 1)$ to the weights of the neighbors on L; q is dynamically set at each iteration via maximum likelihood estimation.

This model performs well in practice, is nonparametric, and can be efficiently updated in light of new data. The last feature is essential in allowing for fast lookahead, the central component of our method. The time complexity of a naive implementation of MF–ENS is $\mathcal{O}\left(2^k\, n^2 \log n\right)$,[4] where k is the number of L queries made for each H query and n is the size of the candidate pool. The corresponding time complexity of the single-fidelity ENS is $\mathcal{O}\left(n^2 \log n\right)$ (Jiang et al., 2017), to which MF–ENS adds a factor of 2^k from the exhaustive marginalization of the exploratory L labels. We may also take advantage of the implementation trick developed by Jiang et al. (2017), which reduces the time complexity to $\mathcal{O}\left(2^k\, n\,(n + m \log m)\right)$, where $m \ll n$ is the maximum number of unlabeled points whose probabilities are affected by a newly revealed label. The interested reader may refer to §3.2 of Jiang et al. (2017) for more detail.

We also extend existing branch-and-bound pruning strategies to further reduce the computation time of our policy at each search iteration. First, following previous work (Garnett et al., 2012; Jiang et al., 2017), we establish an upper bound of the score function that is the approximate expected utility defined by (4). This allows us to eliminate candidates whose score upper bounds are lower than the current best score we have found, as their actual scores cannot possibly be the final best score. These upper bounds are computationally cheap to evaluate, so applying this pruning check only adds a trivial overhead to each search iteration. Further, we develop an extension of this pruning strategy by making use of the fact that computing the score of a point involves marginalizing over its unknown label. We thus apply similar pruning checks at every step during this marginalization, which allows us to identify and prune suboptimal candidates "on the fly," avoiding any unnecessary computation.

Concretely, denote by $f(x)$ the score of an unlabeled point $x \in \mathcal{X}$, defined by (4) for MF–ENS. Suppose x has a posterior probability of $\pi = \Pr(y = 1 \mid x, \mathcal{D})$, where y is the label to be returned by the oracle we are currently querying. As previously described, we compute $f(x)$ by calculating its *partial values* while marginalizing over y:

$$f(x) = \pi\, f(x \mid y = 1) + (1 - \pi)\, f(x \mid y = 0),$$

where $f(x \mid y)$ is the partial score of x according to (4) when conditioned on a value of y. Suppose before computing either $f(x \mid y = 0)$ or $f(x \mid y = 1)$, we know these partial scores are upper bounded by $\overline{u}(x)$ given a new positive label and $\underline{u}(x)$ given a new negative label:

$$f(x \mid y = 1) < \overline{u}(x); \quad f(x \mid y = 0) < \underline{u}(x).$$

As such, $f(x)$ need not be evaluated if

$$\pi\, \overline{u}(x) + (1 - \pi)\, \underline{u}(x) < f^*, \tag{5}$$

where f^* is the current best score we have found. This pruning strategy has been found to offer a significant speedup in previous work (Garnett et al., 2012; Jiang et al., 2017; 2018).

We extend this strategy by considering the case in which a given candidate x is not pruned because (5) is not satisfied, and we proceed with the calculation of $f(x)$. Now, suppose we have computed only $f(x \mid y = 1)$ and not yet $f(x \mid y = 0)$ and observe that

$$\pi\, f(x \mid y = 1) + (1 - \pi)\, \underline{u}(x) < f^*,$$

[4] Note that we are assuming k is small enough that 2^k is a small constant; if this is not the case, we may approximate (4) by sampling instead, replacing 2^k by s, the number of samples used.

then we may also safely conclude that $f(x)$ cannot possibly exceed f^* without needing to go further and compute $f(x \mid y = 0)$. If this condition is met, we simply abort the computation of the current score $f(x)$ and move on to the next unpruned candidate. For each H query, we apply this partial pruning check once (either after conditioning on $y_H = 1$ and before on $y_H = 0$ or vice versa) for each candidate that is not eliminated by the full pruning check (5). For an L query, we may do this at most three times for each candidate, as the marginalization over the joint distribution of the putative label and the pending H label when computing $f(x)$ involves four different possible label combinations. We quantify the effectiveness of these pruning strategies and show that they can significantly reduce the computation time of our methods in the appendix.

At an iteration where the current best score f^* does not exceed the majority of the score upper bounds, many candidates may be left unpruned. In order to help our policy avoid having to calculate the scores of a large number of candidates and consequently spending too much time on a single iteration, we place an upper limit on how many candidates are to be considered before we terminate the search. Our approach is to first follow the lazy-evaluation strategy introduced by Jiang et al. (2018) and sort the candidates by their score upper bounds. With this sorting, candidates with higher upper bounds will be considered first, and a candidate will never be considered if it will be pruned later on. Now, at each iteration, if after having considered u candidates and noticing that there are still unpruned points remaining, we simply consider a randomly selected subset of size at most s of the unpruned set, before terminating the search and returning the current best candidate. In our experiments, we set $u = s = 500$ for MF–ENS.

We note that this strategy is only used when pruning fails to reduce our search space to be below $u + s$, and to allow us to collect results over a long horizon over many repeated experiments. When applied in a real-life planning setting, MF–ENS can still cover the entirety of a large pool of candidates if desired, even with a significant portion of the pool unpruned. In our experiments, MF–ENS takes approximately 30 seconds to reach its quota of 1000 candidates when pruning is unsuccessful; the time for it to fully cover a pool of 100 000 points (about the size of the real-world datasets used in our experiments) is thus well under one hour. In short, our policy remains tractable in real-life settings, even without successful pruning.

4. Related Work

This work is an extension of the larger active search paradigm, first introduced by Garnett et al. (2012). Active search is a variant of *active learning* (Settles, 2009) where the goal is not to learn an accurate model but to find and query positive labels. Previous work has studied active search under different settings such as finding a given number of targets as fast/cheaply as possible (Warmuth et al., 2002; 2003; Jiang et al., 2019), making queries in batches (Jiang et al., 2018), or when points have real-valued utility (Vanchinathan et al., 2015). Our work generalizes ENS, the policy proposed by Jiang et al. (2017) from the single-fidelity setting. The authors of that work demonstrated that their policy is nonmyopic and aware of its exact budget, allowing it to automatically balance between exploration and exploitation during search and outperform various baselines by a large margin. We will make the same observations about our policy.

Multifidelity active search was first examined by Klyuchnikov et al. (2019), who specifically considered the search problem of a recommender system: identifying items that users of a given application are interested in. They modeled predictions made by a trained preference model as output of a low-fidelity oracle and proposed a co-kriging predictive model (Álvarez et al., 2012) to perform inference on the users' true preferences. Under their setting, queries to the oracles are made sequentially, one after another. Our multifidelity setting is different, modeling situations where oracles of different fidelities are available to run in parallel and vary in their response times, common in scientific experiments and testing. Regardless, our proposed policy could be naturally adopted to their sequential model; the only difference in the computation is that the marginalization over a pending H label is no longer necessary. Further, as we will show in later experiments, our algorithm outperforms the adoption of their upper confidence bound (UCB) policy for various datasets. To our knowledge, our work is the first to tackle multifidelity active search using Bayesian decision theory.

Active search is equivalent to *Bayesian optimization* (BO) (Brochu et al., 2010; Snoek et al., 2012) with binary observations and cumulative reward. Multifidelity BO itself has been studied considerably, and policies corresponding to common acquisition functions have been adopted to multifidelity settings, including expected improvement (Huang et al., 2006; Picheny et al., 2013), knowledge gradient (Poloczek et al., 2017; Wu et al., 2020), and UCB (Kandasamy et al., 2017). However, most of these policies are derived from or motivated by greedy approximations to the optimal policy under different utility functions, and to our knowledge, no *nonmyopic* multifidelity BO policies have been proposed.

Active search is related to the *multi-armed bandit* (MAB) problem (Lai & Robbins, 1985). In particular, querying the label of a point can be viewed as "pulling an arm" in MAB; in active search, an arm cannot be pulled twice but is correlated to its neighbors. Kandasamy et al. (2016) studied a formulation of multifidelity MAB in which each arm may be pulled on different fidelities at different costs, and proposed

Table 1. Experiment results with an H budget of $t = 300$, averaged across all repeated experiments for each setting. θ is the simulated false positive rate of the low-fidelity oracle; k is the number of L queries made between two H queries (i.e., the speed ratio between the two fidelities). Each entry denotes the average number of targets found across the repeated experiments and the corresponding standard error in parentheses. The best performance in each column is highlighted bold.

	ECFP4				GpiDAPH3				BMG			
	$\theta = 0.1$		$\theta = 0.3$		$\theta = 0.1$		$\theta = 0.3$		$\theta = 0.1$		$\theta = 0.3$	
	$k = 2$	$k = 5$	$k = 2$	$k = 5$	$k = 2$	$k = 5$	$k = 2$	$k = 5$	$k = 2$	$k = 5$	$k = 2$	$k = 5$
ENS	218 (4.5)	218 (4.5)	213 (4.2)	213 (4.2)	197 (4.7)	197 (4.7)	186 (5.3)	186 (5.3)	279 (1.6)	279 (1.6)	278 (1.7)	278 (1.7)
MF–UCB	227 (4.8)	238 (4.4)	200 (5.7)	206 (5.8)	200 (5.7)	212 (5.7)	187 (5.5)	200 (5.3)	284 (1.0)	289 (1.0)	274 (2.0)	277 (1.7)
UG	228 (4.9)	237 (4.4)	207 (5.8)	209 (5.8)	204 (5.6)	211 (5.7)	191 (5.6)	202 (5.6)	283 (1.2)	287 (1.1)	278 (2.0)	280 (1.6)
MF–ENS	**244 (3.5)**	**250 (3.2)**	**226 (4.5)**	**230 (4.4)**	**230 (3.8)**	**243 (3.4)**	**208 (5.0)**	**221 (4.3)**	**290 (0.7)**	**294 (0.6)**	**286 (1.1)**	**286 (1.1)**

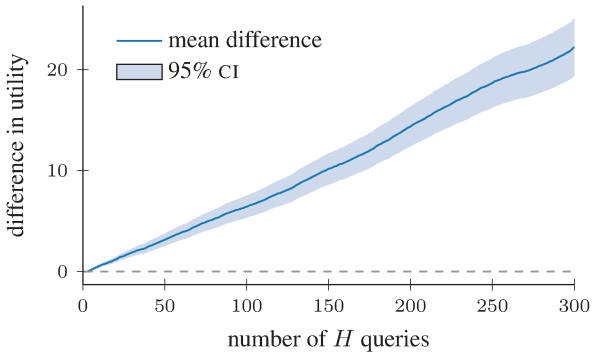

Figure 2. The difference in cumulative targets found between MF–ENS and ENS, averaged across all experiments and datasets.

a policy that is a variant of UCB. By assuming the accuracy of the lower fidelities is known, the authors derived strong theoretical guarantees for their proposed policy.

5. Experiments

We now compare the empirical performance of MF–ENS against several benchmarks.[5] As previously described, ENS is a state-of-the-art, nonmyopic active search policy in the single-fidelity setting. In our experiments, this policy simply ignores the low-fidelity oracle, serving as a single-fidelity baseline to illustrate the benefit of having access to low-fidelity queries. We also test against the MF–UCB policy for multifidelity active search, recently proposed by Klyuchnikov et al. (2019). Under this policy, each candidate x has a UCB–style score of $\alpha(x, \mathcal{D}) = \pi + \beta\sqrt{\pi(1-\pi)}$, where π is the probability of x having a positive label on the fidelity being queried and β is the exploration/exploitation tradeoff parameter.[6] We set $\beta = 0.01$ for L queries and $\beta = 0.001$ for H queries, as suggested in the same work. For another benchmark, we consider a simple but natural heuristic for multifidelity optimization, that low-fidelity queries should serve to narrow down the most-promising search regions (exploration), so that more informed queries could be made on the higher-fidelity oracle (exploitation). Inspired by this heuristic, we design a policy we call UG (for *uncertainty* and *greedy* sampling) that always queries the most uncertain points on L and the points most likely to be positive on H. Uncertainty sampling is chosen for the role of exploration due to its popularity as an active learning technique (Lewis & Gale, 1994). UG may also be viewed as the limit of UCB when β approaches infinity on L for maximum exploration and 0 on H for maximum exploitation.

Datasets for multifidelity active search are not readily available due to the relative novelty of the problem setting, at least not in our motivating area of scientific discovery. However, numerous high-quality datasets are available in the single-fidelity setting, which we will adopt and use to simulate multifidelity search. Namely, for each dataset, we simulate the noisy labels returned by the low-fidelity oracle using the following procedure. We first create a duplicate of the true labels. Given $\theta \in (0, 1)$, we randomly select a fraction θ of the positives from this duplicate set and "flip" their labels to negative. We also randomly select the same number of negatives and flip their values to positive. This perturbed set of labels is then used as the L labels. This construction yields simulated L labels with a false positive rate of θ and a false negative rate of $\theta\, r/(1-r)$, where $r \in (0, 1)$ is the prevalence rate of the positive set $\mathcal{R}$ in $\mathcal{X}$. In a typical active search problem, $\mathcal{R}$ is rare and $r \ll 1$, making the false negative rate much lower than the false positive rate, a common characteristic of many real-world scientific discovery and testing procedures.

We set $\theta \in \{0.1, 0.3\}$. We set k, the number of L queries that are made for each H query, to be either 2 or 5, and set the budget on H to be 300. In each experiment, a policy starts with an initial training dataset of one randomly selected target whose L label is also positive.

[5] Matlab implementations of our policies are available at: https://github.com/KrisNguyen135/multifidelity-active-search.

[6] The authors also considered an odd scenario in which the correlation between the two oracles is *negative*. We assume that this correlation is always positive, and it is constructed to be so.

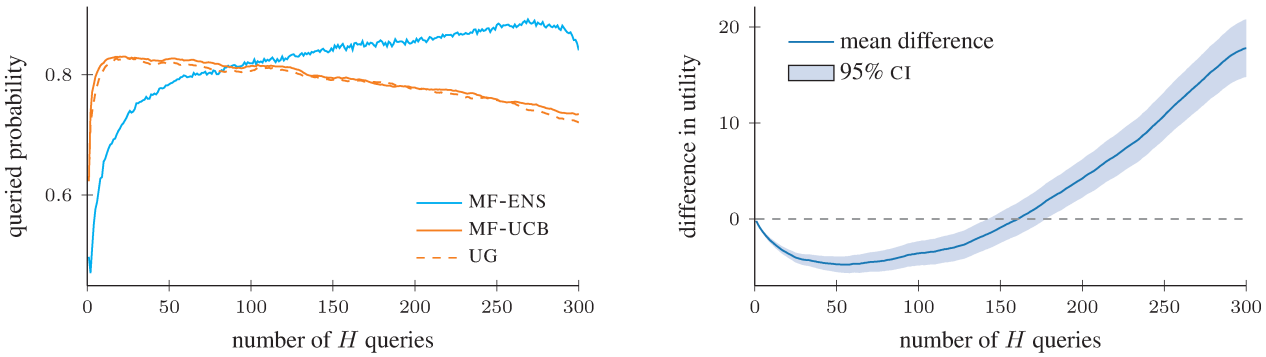

Figure 3. An illustration of budget-awareness exhibited by MF–ENS. Left: The average progressive probabilities of points queried on H by different active search policies. Right: The difference in cumulative targets found between MF–ENS and MF–UCB. The results are averaged across all experiments and datasets.

5.1. Datasets

We conducted experiments on three real-world scientific discovery datasets used in previous studies (Jiang et al., 2017; 2018; 2019). The first two come from drug discovery, where the goal is to discover chemical compounds exhibiting binding activity with a given protein. Each protein defines the target for an active search problem. Here we used the first 50 proteins from the BindingDB database (Liu et al., 2007) described by Jiang et al. (2017). A set of 100 000 compounds sampled from the ZINC database (Sterling & Irwin, 2015) served as a shared negative set. Features for the compounds are binary vectors encoding chemical characteristics, also known as a chemoinformatic fingerprint. We considered two fingerprints delivering good performance in previous studies: ECFP4 and GpiDAPH3. The size of the positive set for these 50 targets ranged from 205 to 1488 (mean 538), having an average prevalence rate of $r \approx 0.5\%$. For each of these two datasets and 50 targets, we repeat each experiment five times, for a total of 500 search simulations.

The other dataset is related to a materials science application. The targets in this case are alloys that can form bulk metallic glasses (BMGs), which have higher toughness and better wear resistance than crystalline alloys. This dataset comprises 106 810 alloys from the materials literature (Kawazoe et al., 1997; Ward et al., 2016), 4275 of which exhibit glass-forming ability ($r \approx 4\%$). We repeated each experiment 50 times for this dataset.

We report the average number of targets found by each policy across the experiments with standard errors in parentheses in Table 1; each column corresponds to a specific setting of θ and k under a dataset. We observe that our policy MF–ENS outperforms all baselines by a large margin. In each column, a two-sided paired t–test rejects the hypothesis that the average difference in the number of targets found between MF–ENS and any baseline is zero with overwhelming confidence, returning a p–value of at most 4×10^{-5}. We report these p–values in the appendix. Finally, looking across the columns, we notice the expected trends: performance of all algorithms except for ENS, which does not utilize fidelity L, improves with higher k (when L is cheaper) or with lower θ (when L is more accurate).

5.2. Performance Gain from Multifidelity Search

To further examine the benefit of having access to more than just the exact oracle, we visualize the difference in the cumulative number of targets found between MF–ENS and the single-fidelity policy ENS, averaged across all experiments, in Figure 2. We observe that MF–ENS completely dominates ENS, finding roughly linearly more targets throughout the search. This large difference in empirical performance illustrates the usefulness of the simulated low-fidelity oracle in our experiments, and suggests that our approach is likely to benefit search with any budget.

5.3. Nonmyopic Behavior

We have claimed that MF–ENS is nonmyopic and aware of its remaining budget at any given time during a search. We demonstrate this nonmyopia by first comparing the progressive probabilities of the points queried on H (at the time of the queries being made) by the policy against the myopic baselines MF–UCB and UG, averaged across all experiments, in the left panel of Figure 3. Initially, MF–ENS chooses points with lower probabilities, exploring the space. As the search progresses, MF–ENS queries more promising points, smoothly transitioning to exploitation. The opposite trend can be observed for UCB and UG, whose probabilities decrease over time due to greedy behavior. This difference translates to distinct patterns in the cumulative reward achieved by these policies. The right panel of Figure 3 shows the difference in the cumulative number of targets found between MF–ENS and MF–UCB, also averaged across all experiments. During the first half of the search, MF–ENS

appears to perform *worse* than MF–UCB, but quickly recovers and outperforms the latter in the end. The corresponding plot comparing MF–ENS and UG shows a similar trend and is deferred to the appendix.

Overall, this phenomenon perfectly highlights the automatic tradeoff between exploration and exploitation exhibited by MF–ENS: the policy makes its initial queries to explore the search space, often requesting labels that are not the most likely to be positive and failing to collect substantial immediate reward; however, as the budget decreases, its queries grow more exploitative and are ultimately more successful than those from myopic policies by leveraging what it has learned.

6. Conclusion

We have proposed a multifidelity active search model in which an exact oracle and a cheaper surrogate are queried in parallel. We presented a novel nonmyopic policy (based on two-stage rollout) for this setting that reasons about the remaining queries on both fidelities seeking to maximize the total number of discoveries. Our policy is aware of the remaining budget and dynamically balances exploration and exploitation. Experiments on real-world data demonstrate that the policy significantly outperforms myopic benchmarks.

Acknowledgements

We would like to thank the anonymous reviewers for their feedback. QN and RG were supported by the National Science Foundation (NSF) under award numbers OAC–1940224 and IIS–1845434.

References

Álvarez, M. A., Rosasco, L., and Lawrence, N. D. Kernels for Vector-Valued Functions: A Review . *Foundations and Trends in Machine Learning*, 4(3):195–266, 2012.

Bellman, R. *Dynamic Programming*. Princeton University Press, 1957.

Bertsekas, D. P. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific Belmont, MA, 1995.

Brochu, E., Cora, V. M., and De Freitas, N. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. 2010. arXiv preprint arXiv:1012.2599 [cs.LG].

Garnett, R., Krishnamurthy, Y., Xiong, X., Schneider, J., and Mann, R. Bayesian Optimal Active Search and Surveying. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

Huang, D., Allen, T. T., Notz, W. I., and Miller, R. A. Sequential kriging optimization using multiple-fidelity evaluations. *Structural and Multidisciplinary Optimization*, 32(5):369–382, 2006.

Jiang, S., Malkomes, G., Converse, G., Shofner, A., Moseley, B., and Garnett, R. Efficient Nonmyopic Active Search. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1714–1723, 2017.

Jiang, S., Malkomes, G., Abbott, M., Moseley, B., and Garnett, R. Efficient nonmyopic batch active search. In *Advances in Neural Information Processing Systems 31*, pp. 1099–1109, 2018.

Jiang, S., Garnett, R., and Moseley, B. Cost effective active search. In *Advances in Neural Information Processing Systems 32*, pp. 4880–4889, 2019.

Kandasamy, K., Dasarathy, G., Schneider, J., and Póczos, B. The Multi-fidelity Multi-armed Bandit. In *Advances in Neural Information Processing Systems 29*, pp. 1785–1793, 2016.

Kandasamy, K., Dasarathy, G., Schneider, J., and Póczos, B. Multi-fidelity Bayesian Optimisation with Continuous Approximations. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 2861–2878, 2017.

Kawazoe, Y., Yu, J.-Z., Tsai, A.-P., and Masumoto, T. (eds.). *Nonequilibrium Phase Diagrams of Ternary Amorphous Alloys*, volume 37A of *Condensed Matters*. Springer-Verlag, 1997.

Klyuchnikov, N., Mottin, D., Koutrika, G., Müller, E., and Karras, P. Figuring out the User in a Few Steps: Bayesian Multifidelity Active Search with Cokriging. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 686–695, 2019.

Lai, T. L. and Robbins, H. Asymptotically Efficient Adaptive Allocation Rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

Lewis, D. D. and Gale, W. A. A Sequential Algorithm for Training Text Classifiers. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 3–12, 1994.

Liu, T., Lin, Y., Wen, X., Jorissen, R. N., and Gilson, M. K. BindingDB: A web-accessible database of experimentally determined protein–ligand binding affinities. *Nucleic acids research*, 35(suppl_1):D198–D201, 2007.

Picheny, V., Ginsbourger, D., Richet, Y., and Caplin, G. Quantile-Based Optimization of Noisy Computer Experiments with Tunable Precision. *Technometrics*, 55(1): 2–13, 2013.

Poloczek, M., Wang, J., and Frazier, P. Multi-Information Source Optimization. In *Advances in Neural Information Processing Systems 30*, pp. 4288–4298, 2017.

Settles, B. Active Learning Literature Survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems 25*, pp. 2951–2959, 2012.

Sterling, T. and Irwin, J. J. Zinc 15–Ligand Discovery for Everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337, 2015.

Vanchinathan, H. P., Marfurt, A., Robelin, C.-A., Kossmann, D., and Krause, A. Discovering Valuable Items from Massive Data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1195–1204, 2015.

Ward, L., Agrawal, A., Choudhary, A., and Wolverton, C. A general-purpose machine learning framework for predicting properties of inorganic materials. *npj Computational Materials*, 2(1):1–7, 2016.

Warmuth, M. K., Rätsch, G., Mathieson, M., Liao, J., and Lemmen, C. Active learning in the drug discovery process. In *Advances in Neural Information Processing Systems 15*, pp. 1449–1456, 2002.

Warmuth, M. K., Liao, J., Rätsch, G., Mathieson, M., Putta, S., and Lemmen, C. Active Learning with Support Vector Machines in the Drug Discovery Process. *Journal of Chemical Information and Computer Sciences*, 43(2): 667–673, 2003.

Wu, J., Toscano-Palmerin, S., Frazier, P. I., and Wilson, A. G. Practical Multi-Fidelity Bayesian Optimization for Hyperparameter Tuning. In *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence*, pp. 788–798, 2020.