A Fitness Function Elimination Theory For Blackbox Optimization And Problem Class Learning

2012

Gautham Anil
University of Central Florida

A FITNESS FUNCTION ELIMINATION THEORY FOR BLACKBOX OPTIMIZATION AND PROBLEM CLASS LEARNING

by

GAUTHAM ANIL
B.Tech. Computer Science, Calicut University, 2003
M.Tech. Computer Science, Indian Institute of Technology Powai, 2006

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2012

Major Professors:

Annie S. Wu
R. Paul Wiegand

ABSTRACT

The modern view of optimization is that optimization algorithms are *not* designed in a vacuum, but can make use of information regarding the broad *class* of objective functions from which a problem instance is drawn. Using this knowledge, we want to design optimization algorithms that execute quickly (*efficiency*), solve the objective function with minimal samples (*performance*), and are applicable over a wide range of problems (*abstraction*). However, we present a new theory for blackbox optimization from which, we conclude that of these three desired characteristics, only two can be maximized by any algorithm.

We put forward an alternate view of optimization where we use knowledge about the problem class and samples from the problem instance to identify which problem instances from the class are being solved. From this *Elimination of Fitness Functions* approach, an idealized optimization algorithm that minimizes sample counts over any problem class, given complete knowledge about the class, is designed. This theory allows us to learn more about the difficulty of various problems, and we are able to use it to develop problem complexity bounds.

We present general methods to model this algorithm over a particular problem class and gain efficiency at the cost of specifically targeting that class. This is demonstrated over the Generalized Leading-Ones problem and a generalization called LO^{**}, and efficient algorithms with optimal performance are derived and analyzed. We also

tighten existing bounds for LO^{***}. Additionally, we present a probabilistic framework based on our Elimination of Fitness Functions approach that clarifies how one can ideally learn about the problem class we face from the objective functions. This *problem learning* increases the performance of an optimization algorithm at the cost of abstraction.

In the context of this theory, we re-examine the blackbox framework as an algorithm design framework and suggest several improvements to existing methods, including incorporating problem learning, not being restricted to blackbox framework and building parametrized algorithms. We feel that this theory and our recommendations will help a practitioner make substantially better use of all that is available in typical practical optimization algorithm design scenarios.

To my parents Anil Kumar and Lalitha,

and to my wife, my love, Sarada.

ACKNOWLEDGMENTS

First and above all, I am deeply grateful to Dr. Paul Wiegand, my advisor, for his unconditional and unwavering support with my research goals. Right from the beginning he has helped me make sense of my research in what is correct, what is important, and more significantly, why. Without his advice and support, I am sure, this work would have been impossible.

I am also grateful for the encouragement, advice and patience from my co-advisor Dr. Annie Wu. My advisors' determination to demand high standards in my writing and my presentations despite my resistance has allowed me to put my thoughts down with a practical level of comprehensibility.

I am thankful to my committee members Dr. Kenneth O. Stanley, Dr. Thomas Jansen and Dr. Thomas Clarke. Dr. Stanley's advice and criticism helped me refocus my research in a more theoretically sound direction while not losing sight of its practicality. Dr. Jansen's encouragement and his advises grounded on theory has had a significant impact in this work. To Dr. Clarke, working with you in various research projects has been a pleasure and I thank you very much for finding time to advise me, even while in retirement.

Thanks are also due to Dr. Jeremiah T. Folsom-Kovarick, Siddharth Somvanshi, Nadeem Mohsin along with Ramya Pradhan and other members of the Evolutionary

Computation Lab @ UCF for providing an outsider's point of view while sitting through my brainstormings. Likewise, to Dr. Joel Lehman for presenting my GECCO'11 paper in my absence.

Special thanks to my parents for being very understanding and extremely supportive while I stayed half way around the world for years studying things that are hard to explain to anyone. In particular, I am thankful to my wife for being as supportive as she has been and motivating me through the last years of this Ph.D.

I am also grateful to Dr. David Kaup, Dr. Eduardo Salas and the EECS Dept. for support during my Ph.D.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

A traditional view of "blackbox optimization methods" typically emphasizes their lack of information [1, 2, 3]. After all, such methods appear to be knowledge-poor compared to alternative methods that might rely on deep insight regarding a specific problem instance, such as the formal, mathematical specification of the optimization problem, including (when available) its derivative. One turns to blackbox methods when one has no choice — when the objective function *cannot* be known to the optimizer *a-priori*. A more modern view, however, is that it is more realistic and constructive to recognize that such methods do *not* operate in a vacuum, but make use of information regarding the broad *class* of objective functions from which a problem instance is drawn [4, 5, 6, 7].

We claim that optimization algorithms should be arranged along three dimensions based on the following characteristics:

1. *Problem* ***abstraction***: The breadth of problem classes over which, this algorithm outperforms random search.

2. *Blackbox* ***performance***: A performance metric that minimizes the number of times the target problem instance must be sampled.

3. *Search **efficiency***: This is concerned with the time and space complexity of the algorithm excluding the evaluation of the target problem instance.

In this dissertation, we focus on these characteristics and present new algorithms that maximize them. We also discuss how this helps us understand existing algorithms and impacts the design of new algorithms.

Our view is that knowing the class from which an objective function hails can provide a significant amount of information. Such information constitutes *domain knowledge* of the larger, more general problem. This domain knowledge is critical for an algorithm to solve an objective function with good performance. But we find that in practice, this information is poorly utilized due to various practical restrictions resulting in poor performance. Moreover, conceptually one can turn the question on its head: Rather than blackbox optimization being about using domain knowledge to find a point that maximizes or minimizes some given objective function, it can be seen as searching through a given problem class to find the problem instance that matches the target objective function. Once the specific problem instance is known, the solution is given by the instance.

Existing blackbox optimization methods already efficiently encode some domain knowledge implicitly and procedurally via their various search operators. However, our conceptual framework allows us to step back and construct an algorithm that separates the notions of domain knowledge, how new samples are selected, and the mechanisms by which blackbox search is conducted. Our idealized algorithm, called *optimal elimina-*

tion of fitness functions (OEFF), works by iteratively removing all non-matching problem instances in the given problem class until the specific problem instance is uniquely identified. This allows us to consider hypothetical scenarios in which our algorithm is given free access to explicit and complete domain knowledge (e.g., a pair-wise table of all function and potential sample point mappings), which in turn allows us to develop blackbox performance and blackbox complexity bounds for the problem class. In this dissertation, we demonstrate how OEFF can be used to find blackbox complexity bounds by using it to bound a generalized ONEMAX problem class.

We recognize that no practical algorithm can maintain a complete table of all possible problem instances and sample point mappings; however, it turns out that there are special cases of problem classes where complete domain knowledge can be efficiently embedded into a search process in order to develop practical algorithms for that problem class that behave equivalently to our idealized algorithm. Such algorithms are therefore optimal from a blackbox performance point of view. They often have substantially improved efficiency over OEFF due to the embedding of the domain knowledge. In this dissertation, we demonstrate how this can be done by deriving an algorithm for a generalized LEADINGONES problem class that has optimal blackbox performance, while being very efficient. This gives us insight about the trade-offs between problem abstraction, efficiency, and performance.

We further apply this embedding approach to generalizations of generalized LEADINGONES called LO^{**} and LO^{***}. We derive an algorithm for the former that has optimal

blackbox performance and is very efficient. We tighten pre-existing bounds to $\Theta(n)$ for LO^{**} and $\Theta(n \log n)$ for LO^{***}. Next, we examine a generalization of the LONGPATH problem class and conclude that to OEFF, this problem is no harder than the underlying generalized ONEMAX problem class. So, we create a new problem class called STEEPPATH that is designed to be easier for an EA than LONGPATH while harder for OEFF. We show that OEFF and a traditional EA both achieve $\Theta(n^2)$ for this problem class.

Of course, when practitioners face a problem, they typically have only rough ideas regarding the problem class from which the instance is drawn — the precise class is unknown to them. In our view, this implies that the engineer has some kind of *partial domain knowledge*. Indeed, engineers often repeatedly face new problem instances from the same class and refine their algorithms based on past experience with those problems. In essence, they are performing a kind of manual *problem class learning*, gradually refining their partial knowledge of the problem class. Some types of *hyper heuristics* attempt to do this automatically [8], but with little to no theoretical guidance as to how to do this or what is possible. We use our framework to develop a theoretical grounding for problem class learning.

In this dissertation, we develop ideas for representing partial domain knowledge as probability distributions over problem classes and show how OEFF can be extended to make use of partial domain knowledge. We also use Bayesian update methods to ensure that the algorithm refines its understanding of the problem class over time. From a blackbox performance point of view, this algorithm is optimal in the sense that given

some initial partial domain knowledge and a series of problem instances drawn from some not-fully-known problem class, no algorithm could have a better blackbox performance on each of the problem instances. Recognizing our algorithm is not practical to implement, we provide high level advice for how to translate what we have learned into more practical methods to perform problem class learning. We also examine the transfer learning [9] scenario in the context of our theory and make suggestions for improvement.

At its heart, this dissertation is about how information affects or can affect optimization. We believe that all algorithm designers have to deal with trade-offs regarding information use while designing algorithm design. Our high level goal is to change the way engineers think about, design, and implement optimization algorithms such that they think first and foremost about what information is available and how it can be effectively used in the current context. We hope and believe this view will challenge conventional views of optimization.

1.1 Outline

In the next chapter, we discuss relevant literature on blackbox complexity and problem class learning. In chapter three, we lay out our framework, including terms and notation, a formal description of OEFF, and a high level discussion of our view of domain knowledge in OEFF. Chapter four demonstrates the use of OEFF to determine blackbox complexity bounds for a generalized ONEMAX problem class. Chapter five demonstrates

how we use the OEFF framework to construct a practical, optimal algorithm to solve problem instances drawn from a generalized LEADINGONES problem class. Chapter six lays out our new work on how the OEFF framework informs problem class learning. We show how partial domain knowledge can be represented as probability distributions over problem classes, describe how such knowledge can be optimally updated, extend OEFF to do this, and provide some high level advice for how to implement practical methods that perform problem class learning. The final chapter offers some general discussion about the merits of the OEFF perspective as well as a discussion of possible future directions of research.

1.2 Contributions

In this dissertation, we make the following contributions to optimization theory and design of optimization algorithms.

1. We present three characteristics of optimization algorithms that define dimensions along which, all blackbox optimization algorithms can be placed.

2. We present theory for an idealistic optimization algorithm that maximizes problem abstraction and blackbox performance. We demonstrate usage of this theory for performance bounding of problem classes using algorithm analysis. An equivalent result in information theory was first proven by Erdős and Renyi in [10]. However,

when we proved our result in 2009 we were not aware of this work and it was not commonly known in the community. Consequently, our work was accepted as original novel research at a conference (FOGA 2009).

3. We present and demonstrate a formal process that can trade-off the problem abstraction of said idealistic algorithm for algorithm efficiency. We use this method to construct an optimal algorithm for the Generalized LEADINGONES problem class and two of its generalizations. In addition, we apply this approach to analyze the LONGPATH problem class and contrast it against analysis of a problem class called STEEPPATH that we constructed.

4. We suggest problem learning as an effective approach to sacrifice efficiency for performance when facing multiple problem instances. We extend the idealistic algorithm to perform problem learning accurately.

5. In the context of the theory presented throughout this dissertation, we make seven suggestions that can be used to build a new, theoretically grounded framework for designing practical optimization algorithms. We provide general, high level suggestions for designing such methods, including advice to the wider transfer learning community.

CHAPTER 2
BACKGROUND & MOTIVATION

In mathematics, optimization is the process of identifying from a set of alternatives, the best element called a *solution* that has been defined by an objective function. An *objective function*, given an element, returns a real value indicating how good (or bad) it is. This value is called an *objective value* or, in evolutionary computation community, a *fitness value*. A solution is defined as the element with the highest (or lowest) fitness value. Then, optimization becomes a process of maximization (or minimization) of an objective function and the elements with the peak value become the function's solutions. A program that attempts to solve an objective function is called an optimization algorithm. One of the earliest examples of optimization algorithms are iterative approaches for identifying the maxima from Newton and Gauss. Such methods rely heavily on in-depth knowledge of a formalized problem instance, e.g., its derivative.

But for many practical optimization problems, this level of information is not available. Instead, a practitioner must turn to methods that rely on far less information about an individual instance of an optimization problem. Nevertheless, a practitioner of optimization does not write algorithms to solve objective functions in a vacuum. In practice,

when a customers needs help with some problem, they provide a practitioner with *some* information about the general problem. We identify three different levels of information and specificity in these customer requests.

1. *Informal description*: The customer provides an informal description of the problem. The practitioner is left to choose the search space, and define the solutions for various problem instances that the customer may want solved using objective functions. From the description, the practitioner also must glean the practical restrictions that limits the type of objective functions he can write and execute.

2. *Formal definition*: The customer specifies a formal problem description and the objective functions. But, the practitioner is not tied to this objective function. Thus, the customer may also provide associated restrictions that apply when designing a new objective function.

   However, in this scenario, the practitioner is free to write an algorithm that solves the objective function without directly relying on it. He may use the objective function and the formal description to write a solver that can directly construct a solution without needing to verify the solution using the objective function.

   Only if he chooses, must he utilize the *optimization framework*. Even then, he may write alternate objective functions that operate on the same solution space such that for any alternate objective function, its set of solutions is a subset of the corresponding objective function specified by the customer.

3. *Formal definition with fixed objective functions*: The customer specifies to the practitioner, a set of formal objective functions as a parameterized algorithm or a mathematical function where the parameter can be used to identify a specific objective function. The practitioner must construct a solver that uses these objective functions directly. Of course, the last situation appears to be a very unlikely request from a customer. However, this view of a problem scenario forms the basis for the blackbox framework. We discuss this issue further in subsection 2.1.

Let us examine a practical example involving a customer. Consider the situation where a salesman contacts an optimization expert (say, us) asking for a software to plan his routes for him so that he minimizes traveling time while visiting all his clients at least once. From the customer's description, we can know that this matches the traveling salesman problem, which becomes our formal problem description. The search space for us is the set of all paths through weighted graphs.

Next, we formulate TSP as an optimization problem by devising an objective function. We can reasonably expect to be asked to find the solution for an arbitrary arrangement of cities. Thus, for each potential arrangement represented as a weighted graph, we construct an objective function over the search space we chose such that its solution set is a subset of the solutions of the specific problem. For TSP, like most problems, this is an impractically large number of objective functions. Thus we must choose a parameterized objective function that will additionally take the weighed graph as input. The customers current path finding request can be converted to a town graph and the resulting objective

function is called the *problem instance*. The set of all possible problem instances form the *problem class*, which is encoded in this example by the parameterized objective function.

What is not obvious in this is why after having identified the formal description of the problem, the practitioner should choose to devise an optimization function and attempt to solve it, instead of attempting to write a direct solver. An example of such a direct solver is Dijkstra's algorithm for single source shortest path problem. Dijkstra's algorithm does not explicitly compute any objective function to verify the solution. Instead it directly constructs a solution from the samples. Based on this example, a generalized optimization framework seems unjustifiably restrictive as it forces the optimization algorithm to rely on the responses of the objective function alone.

However, some kind of general optimization framework confers the benefit of *problem interface unification*, which means that by necessarily being expressed as an objective function, all optimization problems present the same interface to the optimization algorithm. As a consequence, all optimization algorithms are compatible with all optimization problems though the resulting performance may vary. The practitioner can take advantage of this by utilizing a pre-existing optimization algorithm for a different problem and apply it to TSP.

The blackbox framework described below is a formalization of optimization. It has restrictions that match those for many optimization problems and helps their analysis.

2.1 Blackbox framework

To theoretically study algorithms we use restricted scenarios. A result derived under a restricted scenario may be more realistic if the restriction is chosen to match practical scenarios. A popular restricted scenario is the blackbox framework reviewed by Droste et al. [11]. It refers to situations in which additional information about the problem instance is restricted to only the fitness values it returns for sampled search points. In addition to often being aligned with practical scenarios, this restriction guarantees problem interface unification, an important benefit of the optimization framework. Algorithms commonly associated with this framework include evolutionary algorithms, particle swarm optimization etc. Analysis using this framework can help us understand the nature of the objective function using landscape analysis. We can also bound performance of various algorithms and determine the best among worst case performances over a problem class.

Importantly, the blackbox framework has no restriction on a-priori availability of information about the problem class. This information, which consists of all possible details about the objective functions that make up the problem class, is called (complete) *domain knowledge*. It is made available a-priori during the algorithm design phase. In contrast, all the information about a single instance is called *instance knowledge*. In section 3.2 we revisit these and present a representation for complete domain knowledge of any possible problem class.

The domain knowledge of a problem class, even though it can be very informative, by itself is never enough to solve a new instance unless all the instances in the problem class share a solution. This forces the algorithm to try to collect instance knowledge from the instance using the only allowed method — sampling. By default, it also assumes that algorithms have no restriction on the space needed or time spent computing the next sample. We use this scenario in this paper.

Note that many of the common methods we use today, such as evolutionary algorithms make use of substantially less information than even the traditional blackbox assumes. Thus, depending on the practical scenario, some additional restrictions may be warranted to make the analysis results more realistic. E.g., often, bounds are proven with polynomial limits on the time the optimization algorithm can spend computing the next sample. When analyzing performance, while lower bounds are more relevant under the basic blackbox restriction, upper bounds are often derived with the addition of the polynomial restriction. We can also further restrict information availability by limiting how many of the most recent samples can be used to do sample selection. The reasoning is that this is typical in practical optimization algorithms, making the bounds more relevant. As explained in [12], this restriction can influence the bounds and bring them closer to what is achievable in practice by such algorithms. But there still may be differences between algorithm performance and known problem complexity [13]. The extreme restriction of this type ([14] and [6]) is that of *unbiased* algorithms, where a new sample is a function of a single previous sample and this operation is invariant w.r.t. bit-wise exclusive-or

and the permutation of bit positions. This restriction is expected to mirror evolutionary algorithms with traditional bit-flip and uniform crossover.

When choosing a set of objective functions that map to problems, in practice, there often appears to be an intuitive objective function for the corresponding problem. However, the only requirement for an objective function of a problem is that its optimal points are solutions of the corresponding problem given to us by the customer. The fitness values of the rest of the sample space are up to the discretion of the scientist. If these values are all the same, then it is a needle-in-the-haystack (described in section 3.5) objective function, which are particularly hard to solve. Usually the objective function is chosen such that these fitness values help the optimization algorithm identify the solution of the objective function.

2.1.1 Performance, problem complexity and algorithmic time complexity

For this dissertation, we define *blackbox performance* of an algorithm over a problem class as the expected number of samples that the algorithm will take to solve a problem instance averaged over all instances from that problem class.

Once we are given a specific problem class, the blackbox restriction makes this performance metric natural as it meters the information flow from the instance to the algorithm. As these fitness values are typically costly to compute compared to other por-

tions of the optimization algorithm, this performance metric is quite relevant. However, as the fitness value is a real-value, there is no limit on the information it can encode. So, across all possible objective functions for an informal problem, any evaluation count based performance metric has limited relevance as they can be gamed by choosing objective functions whose fitness values are more informative. However, this is rarely an issue as the literature usually discusses performance over specific formalized problem classes, not informal descriptions of problems.

Another metric we use later is *blackbox problem complexity* (or just blackbox complexity), which is popular in literature. Unlike blackbox performance, it is a property of the problem class alone, not a property dependent on both the choice of algorithm and the problem class. It is defined as the minimal (over the possible randomized search heuristics) worst-case expected optimization time [5] where in this case only, time refers to the number of samples. Note that blackbox complexity bounds indicate what is possible (in principle) for algorithms that work under the blackbox assumptions, whereas blackbox performance gives bounds for an actual algorithm — so they are not the same. Still the derived performance bounds can sometimes match the blackbox complexity bounds.

Note that due to our blackbox scenario not bounding the time spent choosing the next sample, blackbox performance may be very different from time complexity. We prefer to focus on blackbox performance in this paper because, unlike it, time complexity is significantly impacted by the costs of the evaluation function, which is external to the optimization algorithm. On the other hand, blackbox performance ignores the computa-

tional costs of the optimization algorithm itself. Thus, we discuss these costs separately in Chapter 5.

To understand the impact of the blackbox restriction, consider the single source shortest path (SSSP) problem. The Dijkstra's algorithm runs in $O(|E| + |V|log|V|)$ where $|E|$ is the number of edges and $|V|$, of vertices. In contrast, [15] bounds the worst case performance over a specific blackbox variant of SSSP to $\Omega(|V|^2)$. Even ignoring the time complexity added by the optimization algorithm, this blackbox variant is never better and is worse if the number of edges is $O(|V|)$. This difference stems from the fact that Dijkstra's algorithm is *not* a blackbox algorithm and thus does not obey knowledge restrictions. In fact, given the availability of Dijkstra's algorithm, it seems unnecessarily extreme to treat SSSP with blackbox assumptions as the benefit of problem interface unification it confers is unnecessary for this well studied problem.

2.2 Domain knowledge in practical blackbox optimization algorithms

The blackbox framework viewpoint leads to a number of practical and commonly used optimization algorithms, metaheuristics [16] such as evolutionary algorithms (EAs) in particular [17]. Nearly all of these approaches work by using search operators to arrange a search space such that previously sampled points of the objective function can be used to try to infer where new, more optimal samples might be found. Ideally, one hopes to exploit as much information available in the problem class as possible by using domain

knowledge to bias how and where samples are taken and to understand what the fitness value tells us about the solution.

2.2.1 Domain knowledge in evolutionary algorithms

Evolutionary algorithms are a good example of domain knowledge use in a black-box algorithm. In a simple EA like the (1+1) EA, the mutation operator induces a topology over the search space. With an appropriately chosen mutation operator, the resulting topology can make it possible for the EA to produce incremental improvements to the current candidate solution. More complicated EAs like genetic algorithms maintain a population of samples and use operators that use multiple samples such as crossover etc. While such evolutionary algorithms often do more than simply climb a peak within the objective function, like most blackbox optimization methods, these algorithms clearly traverse a complex landscape whose structure is determined by search operators, and how they traverse it is determined by population mechanisms.

In [18], we show that a (1+1) EA approximates the elimination process we discuss later. In a (1+1) EA with a mutation operator that flips each bit with some probability, let a be a candidate solution early in the optimization process. For any sample point b, we can create sequences of points in S from a to b such that each member in a sequence was mutated from the previous member. Each sequence has a probability that is the product of the probabilities of the individual mutations. Only sequences with monotonically

increasing fitness can be a sequence of candidate solutions in an (1+1) EA. The sum of the probabilities of all such sequences from a to b is the probability that b will become a candidate. Given an objective function, as the search proceeds, as fitness value of the current candidate solution a increases, this probability decreases for many points in the search space. Given the target objective function u, if $u(a) > u(b)$, due to the monotonicity requirement on the sequence, the probability of b becoming a candidate becomes 0. In this sense, (1+1) EA can be seen to perform an approximate elimination of potential candidate solutions. Similarly, even other EAs that fail to drive the probability to 0 can be considered to be eliminating candidates in a probabilistic sense.

When practitioners choose these operators for a particular application, the structure of the search landscape is changed to try to *bias* the algorithm for the current problem. For an operator to arrange the search space the way an EA needs, it needs to know a lot about the problem class. Thus, the choice of operators is a choice about the *kinds of problems* a practitioner believes he or she will face — a choice of the problem class to which it will or should be applied. In this sense, the practitioner is attempting to embed domain knowledge into the algorithm. When such bias is informative for the current instance, these algorithms can perform well on the problem instance relative to other methods.

Also, as explained in Section 2.1, because the scientist is free to choose to solve a different objective function set that shares solutions with the original problem class, he can develop a new problem class along with the optimization algorithm to simplify

incorporating the bias. However, this approach is attempted less often in the community. We revisit this option in a later chapter.

2.2.2 Having no domain knowledge

The original No Free Lunch (NFL) theorem [19] describes the other extreme: where there exists no bias to incorporate. It states that over the set of all possible instances over a search space, any two algorithms have the same performance. Here, the problem class contains objective functions with all permutations of sample points over the sample space.

The NFL theorem is also instructive about the case when some bias may exist, but the scientist is not aware of it during design phase. We introduce the concept of *subjective knowledge* as the information we have about the problem class. Contrast this with *objective knowledge*, which is complete domain knowledge about the true problem class. We discuss these concepts further in chapter 6.

When we have no subjective knowledge, we don't know which problem instance we may face. We must assume that the problem instances are being chosen from the set of all possible instances. As a consequence, NFL applies and we must conclude that no algorithm can be expected to perform better than random search. To be precise, random search under the blackbox framework samples uniformly at random, without repetition

from the sample space and terminates when the solution is sampled. This underlines the importance of having domain knowledge during algorithm design phase and incorporating it into the algorithm. However, it is important to note that removing functions from the set of all possible instances does not guarantee performance increase — NFL applies even when the problem class is closed under permutations of the search space. Nevertheless, in [20], the authors argue that it makes sense to examine a more restricted scenario for practical algorithms where Kolmogorov complexity of the problem class is limited, ensuring the incorporation of some biases in all practical optimization algorithms.

2.3 Non-blackbox optimization algorithms

To give context to the blackbox framework, we examine some algorithms that do not follow the restriction of the blackbox optimization algorithm. The characteristics of these algorithms that violate the blackbox framework's restrictions sheds light on the impact of the latter.

Multi-objective optimization (MOO) [21] is an approach that is technically not function optimization in the sense that the solution is not a point in the search space in general. Here, instead of optimizing a single objective function, the optimization algorithm attempts to maximize multiple objective functions. A consequence of this is that the solution to a multi-objective problem is a *pareto front* consisting of all the seen solution points that are non-dominated. A benefit of this approach is that in circumstances where the

problem description suggests maximizing multiple metrics, there is no need for the practitioner to judge their relative weight so that a single objective function can be devised.

It is important to note that MOO is not the same scenario as single objective optimization. However, it is possible to re-formulate a single objective optimization problem as a multi-objective optimization problem. This is called *multi-objectivization* [22]. One may ask whether this can make the problem harder to solve. There exists cases where MOO formulation is equivalent to the single objective problem. For example, given three objective functions $\{f_1, f_2, f_3\}$ s.t. $f_2(x) = 2 \cdot f_1(x)$ and $f_3(x) = 3 \cdot f_1(x)$. However, Brockoff et al. [23] suggests that in fact, in some cases, depending on the objectives, adding more objectives can make the problem easier to solve.

Sometimes, the resulting MOO formulation can provide more information about the problem instance per sample than the single objective function and this can improve performance. For example, Jensen [24] was able to use additional helper-objectives to guide evolutionary algorithms in high-dimensional spaces. Knowles et al. [22] also find that some single objective problems, when multi-objectivized, become easier to solve. Both of these attempts use additional objective functions that depend on the problem instance. This suggests that the limitation placed by the blackbox framework may be overly restrictive if such "helper-objectives" can be devised.

Novelty search [25] implements another approach to adding more functions that depend on the problem instance, and help solve it. In novelty search, additional information about the evaluation of the sample is made available to the algorithm. This is

considered behavior information and is tied to the sample. The behavior that generates this information can be varied, and includes for example, the path taken to solve a maze [25], behavior of artificial creatures in a virtual space [26], the behavior of a walking robot [27], the path of an ant in the santa-fe trail problem [28] etc. Novelty search uses the information tied to each sample to compute the Euclidean distance between them. Based on this distance function, the algorithm searches through the behavior space. It tries to search uniformly by maintaining archives of previously seen samples and avoiding exploring samples that are too close to samples present in the archive.

In some of these problems such as the maze problem, the novelty search approach, despite having a large degree of problem abstraction and reasonable efficiency, is able to achieve higher performance than conventional blackbox algorithms. We attribute this to its focus on the behavior space, using problem instance information beyond the objective function. Due to the blackbox restriction, searching through the space of behaviors during evaluation [29] is impossible under the blackbox framework. From these approaches we may conclude that, for many practical scenarios the blackbox framework is too restrictive.

2.4 Problem learning

Even if we did not have complete domain knowledge to incorporate into our algorithm during design time (*off-line*), we have the option of doing it *on-line* as the algorithm is being used. In some cases, the true problem class may be much narrower than the de-

signer believes at first. For example, one might approach the class of ONEMAX problems differently from the broader class of linear problems with a finite set of possible weights — assuming the designer knows this.

An algorithm can further miss out on domain knowledge when the designer could not incorporate all the domain knowledge he has into it. In these situations, as we are solving instances from the partially known problem class, we can infer more about the problem class from the instances. This knowledge can be added to existing domain knowledge and can aid solving future instances. We call this *problem learning*.

Most evolutionary algorithms perform optimization, but do no problem learning. As an example, consider Estimation of Distribution Algorithms (EDA) [30]. These are a variation on the classical genetic algorithm and they use a different operator for generating the new population. This operator uses all the individuals in the population along with their fitness values to generate a probabilistic model of the currently known concept of a "good" solution. This model is then sampled from enough times to generate the new population. An EDA uses only the information present in the current population to decide new samples and finally the solution. Thus, it does not do problem learning.

Of course, the degree to which an EA learns an instance can be more sophisticated. An example is NeuroEvolution of Augmenting Topologies (NEAT) [31]. Here, each individual is a representation of a neural network and the algorithm is a variation of a simple genetic algorithm. A neural network is evaluated based on its input-output map. However, an infinite number of neural networks can encode the same input-output map and

these are said to form a *neutral network* [32, 33, 34]. While members of a neutral network appear the same to an objective function, they are unique to the operator. Thus, the choice of the member of the neutral network influences the variability of its input-output map. Consequently, representations that contain neutral networks are capable of learning variability. This suggests that algorithms that use such representation are capable of learning more than just the solution [35]. Neutral networks also appear frequently in natural evolution [36, 37] and are tied to mutational robustness [38].

However, it is important to distinguish learning about the variability appropriate to a problem instance — which some call *evolvability* [39] — from problem learning. The latter requires that the domain knowledge, which can be in the form of variability, learned from one problem instance be used to solve another. In NEAT, it is reasonable to assume that to some degree, the algorithm chooses those neural networks that have shown not just the best fitness, but also the variability most suitable for the problem instance so far. This biases NEAT towards problem instances where the paths to the solution show consistent variability. However, in NEAT, this learned variability can be used only to aid solving the current instance, thus lacking problem learning.

In this section, we discuss occurrences of problem learning and we start with natural evolution. Then we present some algorithms such as structured demes in evolutionary computation and some hyper heuristics that attempt to do problem learning.

2.4.1 Problem learning in natural evolution

One of the most obvious uses of domain knowledge in nature is in mutation control and in particular, robustness [40, 41, 42, 43, 38, 44]. If certain systems in nature starting from the molecular level, are not robust, it would lead to the larger organism breaking down or producing too much variation in the offspring. This development of robustness against unwanted variation is a direct use of information acquired about what is "unwanted".

For example, MacDonaill [45] finds that there exists a type of parity checking in the chemical interactions between the base pairs on which the DNA is built upon. These provide optimal discrimination between complementary and non-complementary base pairs. Also, while this is not the only set of interactions that can be optimal, a mix of different sets is suboptimal. This leads to a type of lock-in that prevents gradual evolution of the base pairs from one set to another. This lock-in provides increased stability at higher levels.

Another example is related to the proteins in the body. Maintenance of a protein's tertiary structure is crucial to maintaining its function [40]. There exists many naturally occurring proteins that are resistant to changes in the tertiary structure and function despite changes in the amino acids. For example, 84% of amino acids in β-lactamase in *E. Coli* can suffer point mutation without affecting function.

Studies of *E. Coli* [41] also show robustness of the metabolic pathways to loss of enzymes. A study of knock down effect of individual enzymes in the metabolic pathways of *E. Coli* show that of the 48 enzymes, only loss of 7 were critical. Of the remaining 41, only loss of 9 produced growth rates below 95% of optimal. Clearly some trade-off has occurred and nature prefers such a redundant system over a potentially more efficient one.

Hsp90 is an important gene in the category of heat-shock genes, an essential set of genes. The role of Hsp90 is to bind to proteins in signal transduction pathways. When this gene is mutated [46], morphological deficiencies that can be traced back to changes in Hsp90 occurred at 1-5%. It can be concluded that Hsp90 acts as a cushion from genetic defects. It allows the organism to sustain higher mutation rates at the trait level while avoiding many of its deleterious effects. Clearly, the design of Hsp90 is influenced by what nature considers deleterious, which is information gathered about the problem domain.

But does these examples qualify as problem learning or are these improvements only the result of learning about the solution of a single fitness function? For it to be problem learning, there must be a change in the problem instance aka fitness function. In fact, in nature, the fitness function is changing by small amounts constantly and over large time lines, it may change substantially. In this sense, an organism may be attempting to optimize a fitness function significantly different from what its ancestors were optimizing, say 10,000 years ago. Thus, this is a form of problem learning on the part of the organism

because the heuristic, learned for a different fitness function by ancestors is still being used successfully for new fitness functions.

2.4.2 Problem learning in Evolutionary Algorithms

We find that problem learning can also be done in evolutionary algorithms. A direct albeit limited example of problem learning is seen in *meta-evolutionary algorithms* [47]. It involves automated tuning of algorithm parameters for a problem class. This is done over multiple instances from the problem class, and the resulting parameters are specific to the problem class, and thus, are a form of domain knowledge.

Altenberg [48] presents a different approach to gathering domain knowledge involving a type of transfer learning [9]. This is demonstrated on a problem where a particular mutation increases the fitness of the individual. However, once this mutation has spread through a population, the entire population dies off, making it a pathological mutation. In the representation for this study, there are sections that can suppress the effect of this mutation. He ran this problem in an island algorithm with each island having a separate population running separate EAs. There was periodic migration to an island in case of extinction, and the migrating sample repopulates the island. Altenberg noted that in time, the population learned to suppress the pathogenic mutation.

The separate populations are equivalent to separate instances. The good solution of one instance is being passed as the input to the next instance. This is an exchange of potential domain knowledge from one instance to another. After enough migrations, the information that has been retained can be expected to be domain knowledge. Even though the instances in the islands are actually the same, the same learning can happen with multiple instances which share the pathogenic property.

Another example for problem learning is the Anytime Learning approach [49], now referred to as Continuous and Embedded Learning [50]. The CEL algorithm maintains a model of the environment by monitoring it. A simulation based on this model is used to train a virtual robot. The learned behavior is then sent to a real world robot. This is a good example of a model building optimization algorithm.

CEL can also do problem learning [51] by remembering a library of best known robot behaviors for various environment models. When asked to perform in a new environment, CEL will model it, and then initialize with the best behavior for the closest available model from the library. This is a good example of explicit problem learning and is closer than most real world systems to the idealized problem learning approach presented in chapter 6, though there are significant differences, and it is unprincipled from a theoretical point of view.

Case Injected Genetic Algorithms (CIGAR) [52] is a similar approach that learns and applies information from the current instance to a future instance by using case-based memory and genetic algorithms.

2.4.3 Problem learning in Hyper Heuristics

Hyper heuristics, reviewed in [8] is an approach where high level heuristics called hyper-heuristics are created to apply low level heuristics, that in turn create the solution to the problem. The heuristic selection approach that involves high level heuristics selecting the appropriate low level heuristic has been applied to domains such as production scheduling, 1D packing etc. Burke et al. [53] present a heuristic generation approach for the bin-packing domain that outperforms manually designed heuristics. Heuristic generation approaches create new low level heuristics from components of existing low level heuristics.

The low level heuristics can be *constructive* heuristics, which build a solution incrementally. They can also be *perturbative* heuristics, which make small changes to a solution as in reinforcement learning approaches. See [8] for a comprehensive treatment.

If the hyper-heuristic training algorithm is capable of *online* learning, it can exploit task dependent local properties, to build hyper-heuristics. Based on our discussion so far, online learning is a bias as it assumes that the observed local properties will repeat.

But of interest to us is *offline* learning, which involves gathering information in the form of rules or programs from previously seen instances with the intention of using them to help solve a new instance. Terashima-Marin et al. [54] present a comparison of two offline learning hyper-heuristic systems, one using messy GAs and other us-

ing learning classifier systems for a variety of benchmark problems. Off-line learning in hyper-heuristics can be seen as a kind of transfer learning [9].

2.5 Making the best of Blackbox assumptions and questioning them

In the beginning of this chapter, we mentioned that we are often given an informal description of a problem to optimize. From this, we develop a formal description of the problem followed by a set of objective functions to optimize. When we build an optimization algorithm under the blackbox framework under this framework, we are allowed to incorporate domain knowledge about these objective functions. However, in many cases, we do not have complete domain knowledge or are unable to incorporate all that we have into the optimization algorithm, often both. To fill the gap, we sometimes employ algorithms that can learn domain knowledge on-line from the objective functions in a limited fashion.

Traditionally the blackbox optimization framework is considered to present a scenario that severely restricts information about what we are being asked to solve. In contrast, we feel that domain knowledge in blackbox optimization is a *lot* of information and the modern view of blackbox agrees with this. So, the relative dearth of domain knowledge in existing practical algorithms for various reasons has led to an opinion that blackbox restriction is perhaps too relaxed for practical scenarios. This is why blackbox bounds

are considered "too optimistic" and various recent papers suggest additional restrictions to bring the blackbox framework in line with practical algorithm design realities.

This dissertation looks at what is the best that could be achieved under the blackbox framework, which provides complete domain knowledge to algorithms. We also examine how can we achieve it and what that would cost us. To this end, we identify three key desirable characteristics of algorithms:

1. *Problem abstraction*: This indicates the breadth of problem classes that this algorithm targets. The easy way to increase problem abstraction is by targeting a bigger problem class. An algorithm outperforms random search averaged over all problem instances in its target problem class. A different algorithm may target a larger problem class, but often reduces the increase over random search.

   One way to increase problem abstraction is by introducing parameters to accept domain knowledge during execution time. The parameterized algorithm is effectively a consolidated algorithm that separately targets various problem classes. This maintains the performance over the applied problem class, but requires that the given domain knowledge accurately corresponds to that class.

2. *Blackbox performance*: This metric, defined before, is the expected number of samples that the algorithm will take to solve a problem instance averaged over all instances from the problem class it is applied to.

3. *Algorithmic efficiency*: This is concerned with the time and space complexity of the algorithm *per sample*, excluding the evaluation of the target problem instance.

The goal of an optimization practitioner is typically to find an algorithmic solution that is optimal in all three of these senses. As we will show, this is not possible and any algorithm, practical or otherwise, must make trade-offs that determine its practicality. The idealized algorithm we present in the next chapter shows how to maximize problem abstraction as well as blackbox performance. Then, we demonstrate a formal process that we can use to trade-off problem abstraction for the sake of blackbox performance. In chapter 6, we examine how we can ideally gather domain knowledge from previously seen instances to help us with future instances. We suggest problem learning as a way to gradually trade-off algorithmic efficiency for the sake of blackbox performance, if we were so inclined.

However, we disagree with the notion of the blackbox framework being overly relaxed in all aspects. We argue that it is too restrictive because we feel that in practical scenarios, there are more sources of information about the problem instance than just the objective function. We argue that these *instance knowledge sources* can be used to improve performance beyond what is possible under the blackbox framework.

CHAPTER 3
FITNESS FUNCTION ELIMINATION FRAMEWORK FOR OPTIMIZATION

As explained before, the blackbox framework allows access to complete domain knowledge. Algorithms make use of this information to various degrees. The question we want to answer is, what is the optimal way to use complete domain knowledge? Among existing approaches, evolutionary algorithms are explained as performing gradual improvement to a population of solutions. In section 2.2, we provide a different perspective. We suggest that in fact, evolutionary algorithms progressively zero-in on a solution by probabilistically eliminating potential solutions.

In this section we will discuss a new framework for optimization where we take this elimination idea and instead of focusing on solutions, we focus on the objective functions first. We use the samples from the unknown problem instance to eliminate objective functions from the problem class until the number of remaining fitness functions all share a solution. This framework consists of a representation for complete domain knowledge along with algorithms that select samples and use their fitness value for optimization. We justify this new framework by proving that it, while idealistic, is optimal w.r.t. the blackbox performance metric discussed earlier.

We begin with a brief explication of our notation.

3.1 Notational Preliminaries

This work considers optimization of functions that map from some discrete *sample space* S into real values. When maximizing, the goal is then to find some $\hat{s}$ such that $\hat{s} = \text{argmax}_{s \in S} f(s)$, where $f : S \mapsto \mathbb{R}$, and likewise for minimization we look for $\hat{s} = \text{argmin}_{s \in S} f(s)$. We refer to the situation in which we are asked to find the optimum of a specific, unknown target function as a *problem instance* and denote the unknown function as u. We use the term *problem class* to describe a known, finite set of functions $F := \{f_1, f_2, \ldots, f_m\}$, where $m = |F|$ describes the cardinality of the function set. We use the term $\mathcal{F}$ to denote the set of all possible functions mapping from some sample space S to some known set of fitness values. Similarly, $\mathcal{E} = (\wp(\mathcal{F}) - \phi)$ denotes the set of all possible problem classes.

We consider the *complete domain knowledge* of some class to be the mapping $\langle F, S \rangle : F \times S \mapsto \mathbb{R}$ where $\forall f \in F, \forall s \in S, \langle F, S \rangle (f, s) = f(s)$. This complete domain knowledge can be viewed as essentially a vast table with as many rows as there are functions in the function set, as many columns as there are samples in the sample space, and in which each cell represents the objective value for that function applied to that sample. Alternately, it is also sometimes useful to denote complete domain knowledge as a subset of $\mathcal{F}$ using an indicator function $I_F : \mathcal{F} \mapsto \{0, 1\}$, where $I_F(f)$ is 1 iff $f \in F$.

As the search process proceeds, using samples, some functions from F are identified as not being the same as the problem instance and are eliminated. We use the notation $\langle F_i, S_j \rangle$ where $S_j \subseteq S$ and $F_i \subseteq F$ to indicate such an intermediate stage where F_i is the set of un-eliminated functions. Sometimes shorthand such as F_i may be used for $\langle F_i, S_j \rangle$ for simplicity, where appropriate and when the context is clear.

Although the only restriction on S, the search space, assumed by most of the discussion in this paper is that it is finite, often we focus on a problem class in which the search domain is the set of binary strings of length n. In such cases when s is a sequence of bits, we use the notation $s[i \ldots j]$ to indicate a subsequence of s starting from $i > 0$ up to and including j. Further, we use the notation $f[i \ldots j]$ for functions also when all functions in the relevant problem class have unique solutions and this indicates a subsequence of the solution of f. Used alone, $[\ldots]$ indicates an ordered sequence whose contents can be accessed by index as described above for s.

Lastly, U indicates the subset of unsampled search points.

3.2 Representing complete domain knowledge

We can characterize an objective function as a unique set of relationships between the fitness values returned by points in a sample space. If we sample an objective function completely, we could know (in principle) all the possible relationships between the

sample points. Thus, all knowledge about an objective function can be represented as an ordered list of fitness values returned for a fixed ordering of the points in the sample space. This is *complete instance knowledge*.

For the next step, consider the axiom given below.

Axiom 1. *If x is a variable that can take values from a set V, and if the value of x is known to be fixed, then any knowledge about the value of x, can be described as a probability distribution over V.*

If x is the mapping of samples to fitness values that the unknown instance has, then the possible mappings that x can be in general form the set of all possible instances denoted by $\mathcal{F}$. As the problem instance is fixed during a single optimization process, using Axiom 1, we can describe any knowledge we can have about it as a probability distribution over $\mathcal{F}$.

However, it is important to note that this representation does not explicitly encode structural properties that may exist in the fitness space. Such properties can be mined from this explicit representation. Algorithms that use such properties to create heuristics for optimization require that these properties be mined by practitioners during algorithm design time. In contrast, the elimination approach described in section 3.3 is designed to use the explicit representation directly.

Note that by assessing performance in the blackbox framework using an average, one assumes that problem instances are drawn uniformly at random from the problem

class. Thus we can simplify this knowledge representation further into a boolean vector over members of $\mathcal{F}$. The $\mathcal{F}$ itself can be represented as a table where each instances is a row of fitness values and the points in the sample space form the columns. We can represent the information we have about the problem class by marking only the instances in the problem class as True. We can get an alternate representation for domain knowledge by removing the rows for functions not in F from the table representation of $\mathcal{F}$ to get a table for F. This knowledge about the problem class is *complete domain knowledge*.

This representation may appear to be too different from existing representation of domain knowledge, viz., optimization algorithms. The explanation is that optimization algorithms encode relationships between fitness values of sample points to aid in optimization, while this representation effectively enumerates the relationships.

In the next section, we present an algorithm that we prove, can use this representation ideally to solve an unknown instance.

3.3 Optimal Elimination of Fitness Functions

The optimization algorithm we are presenting tries to solve an unknown problem instance u sampled from a problem class F by attempting to identify it in F. It does this by comparing $u(s)$ for all sampled points s with $f(s), \forall f \in F$. If $u(s)$ and $f(s)$ are not the same, f is eliminated as a potential match. This continues until there is only one un-

eliminated function. We may also stop earlier, when all the un-eliminated functions share a solution, which we can determine from the complete domain knowledge. This solution is returned as the solution of the unknown. This approach of eliminating mismatching fitness functions is the Fitness Function Elimination (FFE) framework and this algorithm is called Optimal Elimination of Fitness Functions (OEFF) and is shown as Algorithm 1. It is important to note that this algorithm is defined only if $\mathcal{F}$, and thus F, are finite.

Algorithm 1 OEFF(X,S,u)

while INTERSECT(FILTER(GETSOLUTIONS(),X)) $= \emptyset$ **do**

$x \leftarrow$ NEXTSAMPLE()

for $i = 1$ to $|\mathcal{F}|$ **do**

if $X[i] = True$ **then**

if $\mathcal{F}[i](x) \neq u(x)$ **then**

$X[i] \leftarrow False$

end if

end if

OEFFSTEP

end for

end while

return INTERSECT(FILTER(GETSOLUTIONS(),X))

OEFF takes as input, a boolean vector X of length $|\mathcal{F}|$. All the functions in $\mathcal{F}$ that are present in the problem class have the value $True$ in their position in X and $False$ otherwise. The second input S denotes an array storing contents of the search space, and u

denotes the unknown instance from the problem class that is to be optimized. $\mathcal{F}[i]$ returns the i^{th} objective function from the set of all possible functions. The helper functions used here are,

- GETSOLUTIONS: Returns a sequence of sets of solutions, one for each function in $\mathcal{F}$. This sequence is static and easy to compute or compress as $\mathcal{F}$ contains all possible functions and they can be ordered systematically.
- INTERSECT: Given a list of sets, returns their intersection.
- FILTER: Filters a list based on the boolean vector given as 2^{nd} parameter.
- NEXTSAMPLE: A placeholder for any function that selects samples for OEFF.

It is important to clarify some design choices for OEFF. An important difference between the blackbox framework and OEFF is that in the former, sampling the solution is necessary and sufficient for termination. OEFF shows us that for some problem classes, sampling the solution can be neither necessary nor sufficient to correctly identify a solution under all circumstances. Regarding necessity, OEFF can identify an objective function without sampling its solution, and once identified, OEFF can suggest the solution immediately, saving us a sample w.r.t to a valid algorithm under the blackbox framework. To disprove sufficiency, consider a problem class $F = \{f_1, \cdots, f_n\}$ such that,

$$f_i(x) = \begin{cases} x & x \leq i \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Let $u = f_m, m < n$. On evaluating $u(m)$ alone, even though OEFF has sampled the solution m, it cannot know that the solution has been sampled — along with other functions, f_n is still in the set of non-eliminated functions — and so, the solution may be n. Thus, OEFF only terminates when the the objective functions in the candidate pool share a solution. When we provide a new proof for NFL in section 4.2, we account for this difference.

3.3.1 Optimality of OEFF

Regarding the issue of optimality of OEFF, first, we prove that given a particular sample sequence, no algorithm can solve the unknown instance using fewer samples from it than OEFF. Next, we will present a strategy for sample selection for OEFF that chooses the sample that maximizes the blackbox performance of OEFF.

Theorem 1. OEFF *is optimal in the sense that given a particular sample sequence, it is impossible for any algorithm to be consistently correct in predicting the solution using fewer samples than what* OEFF *requires.*

PROOF. We will prove this by contradiction. Let there be an algorithm A that can be expected to accurately predict the solution with fewer samples than OEFF for an unknown u and for a specific sample sequence. From the fact that OEFF has not terminated, we can conclude that all the functions in the problem class that share fitness values for the

samples that A found adequate to solve u do not share a solution. If we look at the final solution prediction of A, then there must exist at least one function that u could be that does not have this prediction as the solution. If k is the size of the problem class, then by probability at least $\frac{1}{k}$, the prediction of A is wrong. Then A cannot be expected to be correct consistently. Thus, OEFF is optimal w.r.t. blackbox performance over any sample sequence, once it is specified. □

3.3.2 Example elimination process

We examine a simple optimal elimination process using random sampling. Consider the fitness function set $F = \{f_0, ..., f_9\}$ over a domain $S = \{x_0, \ldots, x_4\}$. Let the fitness value table be as given in Table 3.1. Given u as the unknown target function, let

$$\text{OEFFSTEP}(F, s, u) := \{f | u(s) = f(s), f \in F\}$$

Suppose the first random sample is x_3 and $u(x_3) = 2$. From the table, we see that only f_4 and f_8 satisfy this criterion, so $\{f_0, f_1, f_2, f_3, f_5, f_6, f_7, f_9\}$ are eliminated (8 functions) and $\text{OEFFSTEP}(F, x_3, u) = \{f_4, f_8\}$ remain (2 functions). Suppose the next sample is x_4 and $u(x_4) = 3$, which does not allow us to eliminate an additional function. Our third sample is x_1, $u(x_1) = 3$. As only f_8 gives x_1 a fitness value of 3, we conclude that u is f_8 whose solution we know to be x_0.

From this example, we can make two observations. One, the second sample contributed nothing to the search process, demonstrating that possibility. Secondly, the solution x_0 was not sampled, which has been explained as a possibility in section 3.3.

3.4 Sample selection for OEFF

For OEFF to be optimal, we need to complement it with a sample selection strategy that can choose the sample that minimizes the expected number of future samples required from then on. The simplest approach would be a brute force search through all sample sequences in some order, calculating the black box performance. Because we are exploring what is ideally possible, we follow this brute-force approach along with some optimizations to build a sample selection strategy. While some efficient strategy may be found under a more restricted case, we show that even though this greedy approach appears appropriate, it is not always optimal.

3.4.1 Greedy, entropy maximizing sample selection is not optimal

When devising a sample selection strategy for OEFF, it is useful to note that the more we eliminate, the closer we get to identifying the solution. The concept of maximizing elimination in a single step can be achieved by maximizing the entropy of the

resulting partitions. Here we demonstrate that such a greedy strategy is not optimal in general.

Let us revisit the example in Section 3.3.2 by applying this strategy to Table 3.1. To recap, we use $x_i \in S$ to refer to the points in the search space and F is the problem class $\{f_0, \ldots, f_9\}$. The sizes of various resulting sets generated by sampling S and their entropies are:

Partition sizes on x_0 : $[3, 3, 2, 2]$	$H(x_0) = 1.366$
Partition sizes on x_1 : $[3, 3, 4]$	$H(x_1) = 1.089$
Partition sizes on x_2 : $[6, 2, 2]$	$H(x_2) = 0.950$
Partition sizes on x_3 : $[6, 2, 2]$	$H(x_3) = 0.950$
Partition sizes on x_4 : $[4, 2, 2, 2]$	$H(x_4) = 1.332$

These equations clearly recommend x_0 as the best choice for the first sample. We need at least another sample in any case before we can terminate. Now, let $u(x_0) = 1$. Let OEFFSTEP$(F, x_0, u) = \{f_0, f_3, f_6\}$. Now, we see that no choice of a second sample will partition $\{f_0, f_3, f_6\}$ into sets of size 1. Hence, a third sample will be necessary.

Instead let x_1 be the first sample. The expected information gain is significantly less than the previous case. Let $u(x_1) = v$ and $F_1 =$ OEFFSTEP(F, x_1, u). Then, by choosing x_{1+v} as the next sample, (the example problem class is designed such that this method

of selecting the next sample point is correct) we find that $\forall v, |\textsc{OeffStep}(F_1, x_{1+v}, u)| = 1$. Thus a total of no more than 2 samples will be necessary, if we start sampling with x_1. Hence, the greedily maximizing information gain is an insufficient method to guarantee the best possible sample sequence for OEFF. However, this does not preclude it from being an effective strategy in practice from some problem classes.

3.4.2 Optimal sampling strategy for OEFF

For choosing ideal samples for OEFF, we approach the elimination process as a game with samples and their fitness values inducing branches at alternate levels in the game-tree. To solve this tree, from the position on the game-subtree we are in, we choose a sample and its branch that would lead to the lowest expected distance averaged over its leaf nodes. Unlike a typical game, our opponent chooses the problem instance at random and induces a fitness value and branch. The tree is shown in Figure 3.1 and the formal description follows.

Let $T_{\langle F_i, S_j \rangle}$ be a game-tree induced by some sets $F_i \subseteq \mathcal{F}$ and $S_j \subseteq S$ where $\mathcal{F}$ and S are both finite sets. This tree consists of two kinds of vertices called *f-set* vertices labeled $\langle F_i, S_j \rangle$ and *s-set* vertices labeled $\langle F_i, S_j, s_k \rangle$ where $s_k \in S_j$.

The root of $T_{\langle F_i, S_j \rangle}$ is the f-set vertex labeled $\langle F_i, S_j \rangle$. Its immediate children are s-set vertices $\forall s_k \in S_j, \langle F_i, S_j, s_k \rangle$. The children of s-set vertex are f-set vertices labeled $\forall u \in$

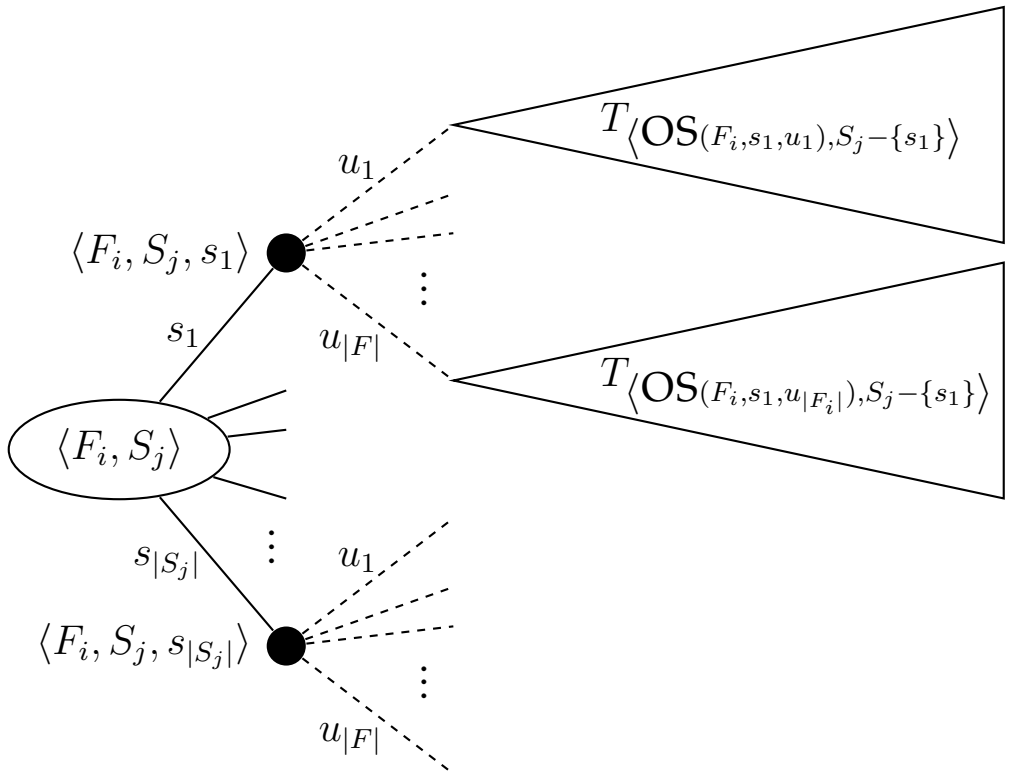

Figure 3.1: Illustration of game-tree conceptualization of OEFF and sample selection. Here OS refers to the function OEFFSTEP.

$F_i, \langle \textsc{OeffStep}(F_i, s_k, u), S_j - \{s_k\}\rangle$. The leaf nodes are the f-sets where the first member of the label is a singleton set.

In this game-tree, there exists an edge from an f-set for every possible sample OEFF can make, leading to an s-set. Similarly, from an s-set $\langle F_i, S_j, s_k\rangle$, there exists an edge for every fitness function in F_i, leading to an f-set. Figure 3.1 illustrates a portion of this tree.

Algorithm 2 NEXTOPTIMALSAMPLE(F,U)

if $|F| = 1$ **then**

return $[0, \text{false}]$

end if

$l_{\text{min}} = |S_{\text{u}}|$

for all s in S_{u} **do**

$l \leftarrow 0$

for all u in F **do**

$l = l + \frac{\textsc{NextOptimalSample}(\textsc{OeffStep}(F,s,u),S_{\text{u}}-\{s\})[0]+1}{|F|}$

end for

if $l < l_{\text{min}}$ **then**

$l_{\text{min}} \leftarrow l$

$s_{\text{min}} \leftarrow s$

end if

end for

return $[l_{\text{min}}, s_{\text{min}}]$

The optimal sample selection algorithm is given as Algorithm 2. This algorithm is given the current set of un-eliminated fitness functions F and the set of currently unsampled points U, and it returns a sample point that will minimize the expected number of future samples.

This algorithm explicitly searches $T_{\langle F,U\rangle}$ looking for the best sample. It is recursive, calculating average tree depth at the second-level s-sets and then the root f-set after calling itself recursively to calculate depth at the third level f-sets. The tree depth at a leaf node is 0. The tree depth at an s-set vertex is one more than the average of the child f-set vertices. The tree depth at a non-leaf f-set vertex is the minimum of the tree-depths of its child s-set vertices, and so on. This definition of tree depth aids us as it calculates the minimum expected number of samples OEFF will need before it terminates.

This sample selection algorithm is optimal in the sense that it calculates the expected tree depths exactly and suggests the sample that results in the lowest expected tree depth. This technique is exponential in both space and time over the sample space. Also, if there are m unique possible fitness values, then there are n^m fitness functions in $\mathcal{F}$. OEFF must search through as many fitness values as there are functions in the candidate pool to process each sample. Thus, optimal sampling algorithms for the general case that are polynomial on $\log n$ are unlikely since OEFF must process at least n^m values for each samples.

As we shall see later, we can use OEFF to bound the performance of a problem class. Currently, the most commonly used alternative is based on Yao's minimax theorem,

which allows us to bound worst case performance and cannot be used for average case performance. OEFF on the other hand, can also be used for optimizing the worst case by modifying its sample selection strategy to use the maximum rather than the average of the tree depths. Here, we explore maximizing the average case performance for OEFF, and not the worst case performance, as a tool to bound the latter already exists. We encourage future work for both variations of OEFF. In addition, for most of the problems examined in this dissertation, the average case performance is identical to worst case performance, making the distinction moot.

3.5 Implications and Uses of Fitness Function Elimination framework

An important contribution of the FFE framework is that by separating domain knowledge from the algorithm that uses it, the framework tells us that the identification of the unknown problem instance in the problem class is key to finding its solution with minimal samples. It stands to reason that we exploit every avenue that can help us with the identification.

This approach brings to light a disadvantage in the design of common evolutionary algorithms: outside of domain knowledge, only the fitness values from the unknown instance are used to identify it. We call this the *fitness value restriction*. There are exceptions of course, recall our mention of instance knowledge sources in section 2.5. If a practitioner chooses to use the multi-objectivization approach to supply an optimization

algorithm with additional helper functions, then this is different from providing domain knowledge and consequently is no longer a blackbox scenario for the original problem class. Such an algorithm has the potential to outperform OEFF, which uses only the objective values. These helper functions can be based on the evaluation of the objective function, ensuring that they do not incur any cost more than the evaluation of the objective function itself.

Note that this is actually consistent with OEFF being optimal. Because the objective value is from $\mathbb{R}$, it is possible to construct a new problem class where the helper function values are embedded in the objective value. OEFF can solve a problem instance from such a class at least as fast on average than any algorithm directly using the helper functions. Nevertheless, outperforming OEFF over the original problem class suggests the importance of designing optimization algorithms that use more information sources than just the objective function. Later, in section 6.4.1, we discuss the implication of using only the objective value for problem learning and come to a similar conclusion.

We can view the candidate pool maintained by OEFF as a model of the problem instance. OEFF refines this model by sampling until the model is adequate to identify a solution. An EA instead maintains a model of the solution by using a population of candidate solutions and improves its model incrementally. But a model of the solution can also be seen as a model of the unknown instance, favoring those objective functions whose solutions are present in the population over the rest. In this sense, by evolving the population, EAs perform an implicit, inexact variation of fitness function elimination.

As mentioned in the background, this point is discussed in an earlier paper [18]. Nevertheless, in practice it is evident that BBO algorithms often do not use the samples best for instance identification. We speculate that in part, this may be because the process of manually designing accurate operators from domain knowledge is difficult.

Because of their optimality w.r.t. blackbox performance, the OEFF algorithm along with the sample strategy, are useful for developing performance bounds. See Section 4 for some examples.

We consider the actual OEFF search method to be knowledge light instead of knowledge heavy as it receives all its domain knowledge through what are effectively run-time parameters. The parameterized nature of this information makes OEFF applicable over all finite problem classes, giving it two of the three key algorithm characteristics we discussed in section 2.5 (problem abstraction and blackbox performance). But it also increases the computational requirements of OEFF. In Section 5 we present a process by which OEFF and sample selection strategy can be modeled over a specific problem class. This embeds the domain knowledge of the problem class into OEFF to yield an algorithm that may have more practical computational resource requirements while maintaining optimality over that problem class, in essence, trading the problem abstraction characteristic for algorithm efficiency.

When compressing domain knowledge from a tabular form to algorithmic form, the Kolmogorov complexity of knowledge becomes relevant. The theorem below suggests that the relationship between the two is not monotonic.

Theorem 2 (Knowledge compressibility theorem)**.** *An increase in knowledge does not imply an increase in the Kolmogorov complexity of this knowledge and vice-versa.*

PROOF. First, we show a case where when knowledge increases, Kolmogorov complexity increases. Consider the problem class Needle-in-a-haystack [11]. This class consists of all $N_a, a \in S$ where $N_a(a) = 1$ and $N_a(x) = 0$, if $x \neq a$ and S is the n dimensional binary search space.

The Kolmogorov complexity of this problem class denoted by F_1 is $O(n)$ where n is the dimensionality of the search space. Let v be a binary string of length 2^n with Kolmogorov complexity more than twice the bounding function for the Kolmogorov complexity of F_1. Let F_2 be set to FILTER(F_1, v). As we can determine v given F_1 and F_2, the Kolmogorov complexity of F_2 is necessarily greater than that of F_1.

Next, we show that as knowledge continues to increase, Kolmogorov complexity can decrease. Consider F_3 which has only a single instance from F_1. Its Kolmogorov complexity is $O(n)$. Similar to the reasoning above, for some other v, the Kolmogorov complexity of F_2 is greater than the bounding function for the Kolmogorov complexity of F_3.

Thus, as knowledge increases from F_1 to F_2 and then to F_3, we see that the Kolmogorov complexity increases and then decreases. The same process in reverse proves the case for decreasing knowledge. □

The implication of this is that it may be possible to store complete domain knowledge for some problem classes with a relatively small space requirement. This sustains the possibility of practical algorithms that behave equivalent to OEFF for some problem class.

Another important benefit of separating domain and instance knowledge from logic is that we can easily reason about gathering domain knowledge. We take advantage of this in Section 6, to develop a theory for gathering domain knowledge from problem instances that were solved for use in future instances.

Table 3.1: A problem class with ten objective function over a search space with five sample points. Getting $u(x_3) = 2$ results in the middle table. The chosen sample is lightly shaded and the eliminated functions are darkly shaded. Then, getting $u(x_1) = 3$ results in the table on the right.

	x_0	x_1	x_2	x_3	x_4
f_0	1	1	1	1	1
f_1	2	1	2	1	2
f_2	3	1	3	1	4
f_3	1	2	1	1	1
f_4	2	2	1	2	3
f_5	3	3	1	1	1
f_6	1	2	1	3	1
f_7	2	3	1	1	2
f_8	4	3	2	2	3
f_9	4	3	3	3	4

	x_0	x_1	x_2	x_3	x_4
f_0	1	1	1	1	1
f_1	2	1	2	1	2
f_2	3	1	3	1	4
f_3	1	2	1	1	1
f_4	2	2	1	2	3
f_5	3	3	1	1	1
f_6	1	2	1	3	1
f_7	2	3	1	1	2
f_8	4	3	2	2	3
f_9	4	3	3	3	4

	x_0	x_1	x_2	x_3	x_4
f_0	1	1	1	1	1
f_1	2	1	2	1	2
f_2	3	1	3	1	4
f_3	1	2	1	1	1
f_4	2	2	1	2	3
f_5	3	3	1	1	1
f_6	1	2	1	3	1
f_7	2	3	1	1	2
f_8	4	3	2	2	3
f_9	4	3	3	3	4

CHAPTER 4
DEVELOPING PERFORMANCE BOUNDS USING OEFF

A potential use of OEFF is in proving blackbox complexity bounds. The performance upper bound for OEFF with a particular sample selection strategy over a problem class acts as an upper bound for the problem class. Admittedly, performance of any algorithm is a valid upper bound. Nevertheless, as OEFF is optimal over the performance metric we are bounding, the bounds we can prove for OEFF is likely to be closer to the real bound than for any other algorithm. Thus if we are attempting to prove upper bounds by algorithm analysis, it makes sense to analyze the performance of OEFF rather than any other algorithm. Unlike upper bounds, only performance lower bounds proven using the optimal sample selection strategy or over all sample sequences can be used as the lower bounds for the blackbox performance of the problem class.

While runtime performance of the optimal blackbox optimization algorithm is not equivalent to its blackbox complexity, often they are the same. They will differ when some instances in the problem class are substantially more difficult than other instances. We are able to use OEFF for blackbox complexity of generalized ONEMAX because its instances

are similar. Recall that we may modify the sample selection for OEFF to optimize blackbox complexity instead of blackbox performance.

We will now use the former property to improve the upper bound for the black box complexity of the generalized ONEMAX problem class. We revise our proof to clarify the inconsistency raised by Doerr et al. [12] in an earlier version [18] of the proof. The sample selection strategy we choose is random sampling uniformly over the search space.

4.1 Upper bounding blackbox complexity of OneMax

Let us consider applying the OEFF with a uniformly random sample selection strategy on a common generalization of the ONEMAX:

Definition 1. *We refer to the* generalized ONEMAX *problem class as the class of pseudo-Boolean problems,* $f : \{0,1\}^n \mapsto \mathbb{R}$*, where given some target string* $\hat{x} \in \{0,1\}^n$*:*

$$\textsc{OneMax}^{(\hat{x})}(x) = n - \sum_{i=1}^{n} |x_i - \hat{x}_i|$$

There are several things to note about this problem class. First, the class is not small — there are 2^n possible candidate functions in $\mathcal{F}$, and each function can be uniquely identified by its solution string. Second, the running time complexity of a standard (1+1) EA on such a class is $\Theta(n \lg n)$[55]. Finally, the theoretical lower bound for any blackbox algorithm is known to be $\Omega\left(n/\lg n\right)$ [4], though the authors of this proof do not believe this to be tight.

Since we can identify a function by its solution string, we can also compute the hamming distance of each solution string to the solution string of the target function. We use this fact to construct hamming distance sets K_d, the partition of $\mathcal{F}$ generated by u based on hamming distances:

$$K_d = \{f | f \in \mathcal{F},\ u(f) = d\}$$

Let's consider the question: What is the probability that a particular function will be eliminated from the hamming level, K_d, given some random sample x. First, we attempt to find the condition under which $u(x) \neq f(x)$ for $f \in K_d$ by examining the following example. Let $n = 8$, $u = 11111111$ and $f = 11110000$. Thus, $f \in K_4$. For $u(x)$ and $f(x)$ to be the same, the number of bits in x that are the same as u must be equal to the ones same as f. In the bit positions where u and f are identical (the first four bit positions), this is inevitable. Among the bits that are different, we need half of them to be same as u and the rest f. E.g., if the last four bits are 0001 or 1110 etc., then u and f can be differentiated and f will be eliminated. But if it is 0011 or 1010 etc., it is not possible to differentiate them. This example makes it clear why any K_d where d is odd are automatically eliminated with any sample: if f has an odd number of bits equivalent in u, then x cannot be equidistant from both u and f.

In general, when d is even, there are $\binom{d}{\frac{d}{2}}$ ways to choose half the bits where u and f are different. And there are 2^{n-d} ways to choose the bits where they are the same. Thus the probability that a sample x will *not* eliminate $f \in K_d$ where $2 \leq d \leq n$ and d is even (d

is assumed to be even from now on) is given as follows

$$\begin{aligned} P_e(d) &= \frac{\binom{d}{\frac{d}{2}}.2^{n-d}}{2^n} \\ &= \frac{\binom{d}{\frac{d}{2}}}{2^d} \end{aligned}$$

Next, we shall prove the Lemma below to simplify the composition of the fitness functions remaining after $\frac{cn}{\lg n}$ samples are made.

Lemma 1. *For a constant c, after $\frac{cn}{\lg n}$ samples the expected number of functions remaining in K_2 is higher than any $K_d, 2 < d \leq n$*

PROOF. The expected number of samples in K_d after $\frac{cn}{\ln n}$ samples is

$$\left(\frac{\binom{d}{d/2}}{2^d}\right)^{\frac{cn}{\ln n}} \binom{n}{d}$$

We have to prove, that for all $2 < d \leq n$,

$$\left(\frac{\binom{2}{2/2}}{2^2}\right)^{\frac{cn}{\ln n}} \binom{n}{2} > \left(\frac{\binom{d}{d/2}}{2^d}\right)^{\frac{cn}{\ln n}} \binom{n}{d}$$

$$\frac{1}{2}\left(\frac{1}{2}\right)^{\frac{cn}{\ln n}} > \left(\frac{\binom{d}{d/2}}{2^d}\right)^{\frac{cn}{\ln n}} \frac{(n-2)...(n-d+1)}{d!}$$

Let us consider the base of the first term on the R.H.S. Applying Sterling's approximation for factorials, we get

$$\begin{aligned}\frac{\binom{d}{d/2}}{2^d} &= \frac{d!}{\frac{d}{2}!\frac{d}{2}!2^d} \\ &\approx \frac{e^{d\ln d - d + \frac{\ln d}{2} + \frac{\ln 2\pi}{2}}}{e^{d\ln\frac{d}{2} - d + \ln\frac{d}{2} + \ln 2\pi + d\ln 2}} \\ &= e^{\frac{\ln d - 2\ln\frac{d}{2} - \ln 2\pi}{2}} \\ &= e^{\frac{\ln\frac{4d}{d^2 2\pi}}{2}} \\ &= \sqrt{\frac{2}{\pi d}}\end{aligned}$$

Using this result, we need to prove

$$\frac{1}{2}\left(\sqrt{\frac{\frac{1}{4}}{\frac{2}{\pi d}}}\right)^{\frac{cn}{\ln n}} = \left(\frac{\pi d}{8}\right)^{\frac{cn}{2\ln n}} > \frac{(n-2)\ldots(n-d+1)}{d!}$$

As R.H.S is smaller than $\left(\frac{n}{d}\right)^d$, ignoring constant term, we need to prove

$$\left(\frac{\pi d}{8}\right)^{\frac{cn}{2\ln n}} > \left(\frac{n}{d}\right)^d \tag{4.1}$$

If d is assumed to be a constant value independent of n, the R.H.S. of equation 4.1 is polynomial while the L.H.S. is super-polynomial and the inequality is valid.

When d is thought of as $kn, k \leq 1$, in equation 4.1, we find the R.H.S. is $O(2^n)$. For the L.H.S., consider the function $\left(\frac{n}{q}\right)^{\frac{1}{2\ln n}}$. For $q > 1$, the function approaches $\sqrt{e}$ asymptotically. In equation 4.1, $q = \frac{8}{k\pi} \geq \frac{8}{\pi}$. Picking an arbitrary value $1 < b < \sqrt{e}$, for $c > \frac{\ln 2}{\ln b}$, we have

$$\left(\left(\frac{n}{q}\right)^{\frac{1}{2\ln n}}\right)^{cn} > b^{cn} = \Omega(2^n)$$

Thus for $2 < d \leq n$, as the L.H.S. of equation 4.1 is $\Omega(2^n)$, and the R.H.S. is $O(2^n)$, the lemma is proven. $\square$

Lemma 2. *After some samples say S_1, $P(E_n, E_{n-2}, \cdots, E_2) > P(E_2)^{n/2}$, where E_r is the event that after some samples, all the functions in K_r have been eliminated.*

PROOF: First, we show that $\forall r > 2, P(E_r|E_{r-2}, \cdots, E_2) > P(E_r)$. W.l.o.g, let 1^n be the solution. W.l.o.g, let $k_r \in K_r$ in general be the string $1^{n-r}0^r$. If k_2 is eliminated by a sample x, then x ends in 00 or 11. Let E_{k_r} be the event that k_r is eliminated by a random sample. k_r will not be eliminated if and only if the sample x has exactly $r/2$ 1s in the last r bits. So,

$$P(E_{k_r}) = \frac{2^r - \binom{r}{r/2}}{2^r}$$

Recall that if k_2 got eliminated, then x ends in 00 or 11. If it ends in 00, k_r will survive if the $r-2$ bits before the last two bits has exactly $r/2$ ones. If ending in 11, it needs exactly $r/2$ zeros. Thus,

$$P(E_{k_r}|E_{k_2}) = 2\left(\frac{2^{r-2} - \binom{r-2}{r/2}}{2^{r-2}}\right)$$

Taking the inequality $P(E_{k_r}|E_{k_2}) > P(E_{k_r})$ and solving for r, we note that for $r \geq 4$, $P(E_{k_r}|E_{k_2}) > P(E_{k_r})$. In general, the probability of a function in K_r being eliminated after

samples in S_1 is, $1-(1-P(E_{k_r}))^{|S_1|}$. But, if K_2 is empty, this means that,

$$\forall k_2 \in K_2, \exists x_{k_2} \in S_1 \text{ s.t.}, k_2(x_{k_2}) \neq u(x_{k_2}) \tag{4.2}$$

$$\forall k_r \in K_r, \exists k_2 \in K_2 \text{ s.t.}, k_r(k_2) = (r-2) \wedge P(k_r(x_{k_2}) \neq u(x_{k_2})) > P(E_{k_2}) \tag{4.3}$$

Thus, given E_2, probability of this function being eliminated because of samples in S_1 is higher because we know x_{k_2} exists. As all members of K_r have a higher probability of being eliminated, $P(E_r|E_2) > P(E_r)$. Similarly, as proposition 4.2, 4.3 holds just as well given $(E_{r-2}, \cdots, E_2)$, we find that,

$$\forall r > 2, P(E_r|E_{r-2}, \cdots, E_2) > P(E_r) \tag{4.4}$$

By chain rule, we have,

$$\begin{aligned} P(E_n, E_{n-2}, \cdots, E_2) =& P(E_n|E_{n-2}, \cdots, E_2) \cdot P(E_{n-2}|E_{n-4}, \cdots, E_2) \cdots P(E_2) \\ >& P(E_n) \cdot P(E_{n-2}) \cdots P(E_2) && \text{(from equation 4.4)} \\ >& P(E_2)^{n/2} && \text{(from lemma 1)} \quad \square \end{aligned}$$

Theorem 3. *For the domain of generalized* ONEMAX *of size n, the expected number of samples of the target function u required by OEFF to eliminate all incompatible functions is $O(n/\lg n)$.*

PROOF. We will determine that asymptotically almost surely there are no functions left in K_2 after $\frac{cn}{\lg n}$ samples. First, note that the probability of a random sample eliminating a function in K_2 is $\frac{\binom{2}{1}}{2^2} = \frac{1}{2}$, and the probability that it will not be eliminated is also $\frac{1}{2}$. The probability that it will be eliminated after $\frac{cn}{\lg n}$ samples is $1 - \left(\frac{1}{2}\right)^{\frac{cn}{\lg n}}$.

There are $\binom{n}{2}$ functions in K_2 before elimination. The probability that all these functions will be eliminated after $\frac{cn}{\lg n}$ is as follows:

$$\left(1 - \frac{1}{2^{\frac{cn}{\lg n}}}\right)^{\frac{n(n-1)}{2}}$$

Thus, from lemma 2 the probability of there being no functions other than the solution is at least:

$$\begin{aligned} P_E &= \left(\left(1 - \frac{1}{2^{\frac{cn}{\lg n}}}\right)^{\frac{n(n-1)}{2}}\right)^{\frac{n}{2}} \\ &= \left(1 - \frac{1}{2^{\frac{cn}{\lg n}}}\right)^{\frac{n^2(n-1)}{4}} \end{aligned}$$

Given that $n^4 < 2^{\frac{cn}{\lg n}}$ and $n^3 > \frac{n^2(n-1)}{4}$ for sufficiently large n, we know:

$$\begin{aligned} P_E &> \left(1 - \frac{1}{n^4}\right)^{n^3} \\ &= \left(\left(1 - \frac{1}{n^4}\right)^{n^4}\right)^{\frac{1}{n}} \\ &= \left(\left(1 - \frac{1}{n^4}\right)^{n^4-1}\left(1 - \frac{1}{n^4}\right)\right)^{\frac{1}{n}} \\ &\geq \left(\frac{1}{e}\left(1 - \frac{1}{n^4}\right)\right)^{\frac{1}{n}} \end{aligned}$$

Since this asymptotically approaches 1, so does P_E. □

Recall from the introduction that a similar bound can be concluded from an earlier paper in the field of information theory by Erdős and Renyi [10]. However, that work was unknown within the community when the proof above was developed. Consequently, the proof was accepted as original novel research at a conference [18].

4.2 Lower bounding blackbox complexity

We can also lower-bound the blackbox complexity of a problem class by proving the lowest sample-complexity that OEFF can achieve over that problem class given any sample sequence. These lower bounds are relevant because they reflect the absolute minimum samples required by an algorithm to solve the worst-case instance. An example of such a bound is the important No Free Lunch theorem [19]. Using Yao's min-max theorem, it was originally proved over $\mathcal{F}$ (the set of all possible instances) and later sharpened [56] to hold over problem classes that are closed under permutation. We present below an alternate proof for the sharpened NFL that demonstrate the capability of our approach.

Lemma 3. *If F is a problem class not closed under permutation, $\exists g \in \mathcal{F}-F, f \in F$ s.t., $\exists x, y \in S$ where,*

1. $g(x) = f(y) \cap g(y) = f(x)$.
2. *Either x or y is a solution of g.*

PROOF. By definition, if F is not closed under permutation, $\exists g \in \mathcal{F}$ such that, $\exists z, y \in S$ where swapping the fitness values of z and y in g will give a function in F, say f. If z or y is a solution of g, the lemma is proved. Otherwise, let x (different from y or z) be a solution of g. We can swap fitness values of z and y using x as an intermediary in three steps such that each of these steps involve the fitness value of the solution. Additionally,

as one of these swaps has to start with a function not in F and result in a function in F, the lemma is proven. □

Theorem 4. *A No Free Lunch result holds over the set of functions F if and only if F is closed under permutation. [56]*

PROOF. To prove sufficiency, we need to show that if the set F is closed under permutation, then OEFF, given any sample sequence, cannot be expected to have better performance than random search, as well as that no algorithm can have worse performance than OEFF. First, we show that any two sample sequences have identical performance in OEFF.

Let S_m be a search space of size m and V_m be an arbitrary set of fitness values (with repetition) of size m. Then let PERM(S_m, V_m) be the set of fitness functions that map from S to V in all possible ways. With $|S_1| = 1$, we find that any sample sequence has the same expected performance over PERM(S_1, V_1) for all $|V_1| \geq 1$. Let us assume this property holds over PERM(S_n, V_n) for all $|V_n| \geq n$ where $|S_n| = n$.

W.l.o.g, let V_{n+1} be an arbitrary set of fitness values of size $n + 1$. W.l.o.g, consider $x_1, x_2 \in S_{n+1}$. After we have sampled x_1 and eliminated functions, we can effectively remove x_1 from the sample space as we will not sample it again. So, performance over our candidate pool is same as performance over PERM$(S_{n+1} - x_1, V_{n+1} - u(x_1))$. Similarly, after sampling x_2, performance is same as over PERM$(S_{n+1} - x_2, V_{n+1} - u(x_2))$. Because of

our inductive assumption, over both of these problem classes, any two sample sequences have the same expected performance.

If the highest value in V_n is repeated j times, then on sampling x_1 or x_2, the probability of sampling the solution is $j/|V_n|$. Thus, any sample sequence starting with x_1 is expected to need the same number of samples as if it had started with x_2. As our choice of x_1 and x_2 are arbitrary, NFL applies and the inductive step is proven. Thus, by induction, over a problem class closed under permutation, any two sample sequences have identical expected performance. Thus, OEFF cannot do better than random search and as it cannot do worse than random search, it has performance identical to random search.

Similarly, no algorithm can do worse then OEFF because if it could, then there must exist at least two sample sequences with different expected performance such that OEFF chooses the better one and this algorithm chooses the other. As all sample sequences are shown to have the same performance, this is impossible. No algorithm can do better than OEFF either. Thus, NFL holds.

To prove necessity, let us assume F is not closed under permutation. Then by lemma 3, $\exists g \in \mathcal{F} - F, f \in F$ s.t. swapping fitness values of two samples say, x and y produces f where one of these samples, say x, is a solution of g. With random sampling, there is a non zero probability of all possible sample sequences. Consider the sample sequence consisting of all samples other than x, y and the solutions of g. After eliminating every function except f, OEFF will terminate now or might have terminated earlier, suggesting at least y as the solution. However, there is a finite probability that random sampling will

sample x first and will need to make one more sample. This is more than OEFF, even accounting for the difference of one sample mentioned at the end of section 3.3. Thus, if F is not closed under permutation, OEFF can be faster than random sampling and NFL does not apply. □

4.3 Discussion

The FFE framework provides a unified approach to blackbox complexity bounding by simplifying the choice of algorithm for analysis to OEFF. We can prove performance lower bounds using FFE by proving lower bounds for OEFF given optimal sample selection or all sample sequences. Similarly, upper bound for OEFF given any sample selection strategy is also the upper bound for the blackbox complexity of the problem class.

A key disadvantage of using FFE for bounds is that it covers only the unrestricted case with unlimited space and time available to select the next sample. Next, we examine how we can use OEFF and optimal sampling to create potentially practical algorithms for specific problem classes.

An important observation from the FFE framework is that while domain knowledge is required to be part of algorithms, it can be present as algorithm input/parameters instead of the more popular notion of embedding it in algorithm code. This separates the algorithm from the domain knowledge, and we can see that the basic optimization algo-

rithm is quite simple. Recall that we represent domain knowledge in OEFF as a boolean vector over $\mathcal{F}$ denoting which functions are present in the target problem class. We can interpret this boolean vector as a model of the unknown objective function. Thus, OEFF is in fact developing a model of the problem instance and identifying its solution by using the model as soon as it is accurate enough.

A disadvantage of OEFF arises due to it having a very verbose representation for domain knowledge, which is a consequence of its design as an optimal algorithm for any problem class. Because of this verbosity, for each sample selection, substantial overhead is incurred due to searching though the domain knowledge. In the next section, we examine some techniques for incorporating specific a-priori domain knowledge into OEFF to reduce this overhead at the cost of problem abstraction.

CHAPTER 5
MODELING OEFF OVER SPECIFIC PROBLEM CLASSES

Recall that OEFF is by design optimal with respect to blackbox performance over any problem class given the right domain knowledge. Thus, the code of OEFF is knowledge light because it assumes nothing about the problem class. However, it is heavily parameterized in the sense that as discussed before, its parameters can represent all possible domain knowledge. We also discussed how these characteristics lead to massive overhead per sample as a result of storing and processing these parameters.

One approach we suggest to alleviate this concern is to derive an algorithm from OEFF that trades off problem abstraction for reduced overhead by duplicating or modeling the behavior of OEFF when it is given some specific domain knowledge as parameters. This will not change the blackbox performance over the specific problem class. But for many problem classes, this can result in reduced computational overhead and improved time complexity compared to OEFF.

Our approach to modeling OEFF uses the game-tree shown in Figure 3.1, which is a representation of the behavior of OEFF over an arbitrary problem class. However, once we have a-priori domain knowledge, the game-tree for OEFF will no longer be arbitrary —

its nodes will have specific sets of fitness functions and edges will have specific samples and objective functions. This new game-tree reflects the specific behavior of OEFF over this problem class. By simplifying this game-tree in a way that does not affect the final result, we will be deriving new behavior that can produce results identical to OEFF. We can encode this new behavior into an algorithm, which will be equivalent to OEFF for this problem class and likely will execute with far lower overhead. That is, we trade-off problem abstraction for efficiency.

In later sections, we present some game-tree simplification techniques and demonstrate the complete modeling process by developing an OEFF equivalent algorithm for the Generalized Leading Ones problem class defined in Section 5.2.1 as well as a further generalization. We then build on this result to tighten the bound for another generalization of Leading Ones. We also examine the LONGPATH problem class and conclude that as this problem class is a singleton set, it is trivial for OEFF. So, we develop the STEEPPATH problem class that has the opposite property — over it, a (1+1)EA is no worse than OEFF. We then present the OEFF modeling process over STEEPPATH problem class and derive performance bounds.

5.1 Techniques for modeling OEFF

In this section, we present two techniques that can simplify the game-tree for OEFF by exploiting repetitive patterns in it. Note that future research may uncover more techniques.

5.1.1 Game tree simplification by exploiting State Duplication

The first simplification technique exploits **state duplication** in the game-tree. Note how in Figure 3.1, the number of edges leading to child nodes from an s-set vertex are numbered up to $|F|$. However, different edges may lead to the same f-set, which are then duplicated in the tree. One reason for this is that given a particular sample, different unknown fitness functions can produce the same un-eliminated set. In fact,

Observation 1. *If for* $u \in F$, $F_u = \textsc{OeffStep}(F, s, u)$, *then* $\forall v \in F_u, F_u = \textsc{OeffStep}(F, s, v)$.

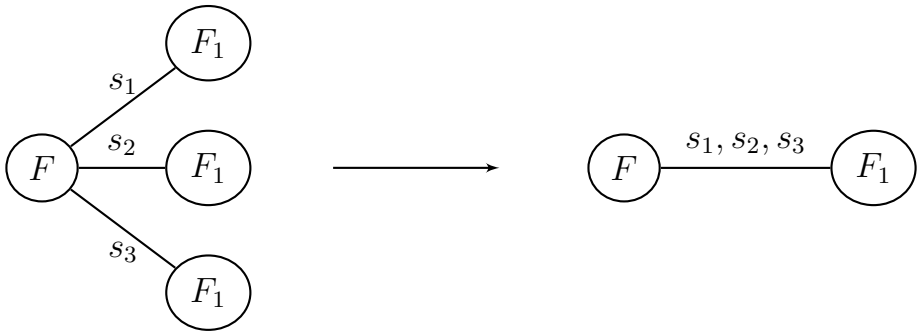

Figure 5.1: Illustration of exploiting state duplication in a game-tree.

As shown in figure 5.1, we can simplify the game-tree by collapsing the duplicated states. Thus, for many function classes, once we have access to the table, there is opportunity for eliminating a large number of potential nodes from the game-tree. It is easy to see that the reduction in states is increased when more fitness functions share fitness values. For example, in the domain of Generalized Leading Ones (see Section 5.2.1), half of all fitness values are 0, half of all the rest are 1 and so on.

Given below is another useful property where U is the set of unsampled points.

Theorem 5. *The tree depth from $\langle F_i, U\rangle$ is independent of U.*

PROOF. Let us assume the contrary. Let $U, U' \subseteq S$ such that the tree depth of $\langle F_i, U\rangle$ and $\langle F_i, S'_u\rangle$ are different. This means there exists a $u \in F_i$ such that the path taken by OEFF from $\langle F_i, U\rangle$ to $\langle \{u\}, S_1\rangle$ is w.l.o.g shorter than the one from $\langle F_i, U'\rangle$ to $\langle \{u\}, S_2\rangle$ for some $S_1, S_2 \subset U$. Let X be the set of samples from the shorter path. For the other path, OEFF would not have chosen a longer path if X could have eliminated $F_i - \{u\}$. We must conclude that the ability of X to eliminate $F_i - \{u\}$ depends on more than u, F_i and X itself. This is not true as $f \in F - \{u\}$ will be eliminated if $\exists s \in X$ s.t. $f(s) \neq u(s)$. Thus OEFF always takes the same number of samples from $\langle F_i, U\rangle$ to $\langle \{u\}, S_3\rangle$ for any $S_3 \subset U \subseteq S$. □

Sample selection searches through the game-tree described before and thus is concerned with unsampled points. But OEFF only uses tree depth during execution. Thus while modeling OEFF, we ignore prior samples and thus reduce the number of states.

5.1.2 Game tree simplification by exploiting Sample Symmetry

The second technique we present for modeling OEFF and optimal sample selection, exploits **symmetry of samples** in the game-subtrees they induce.

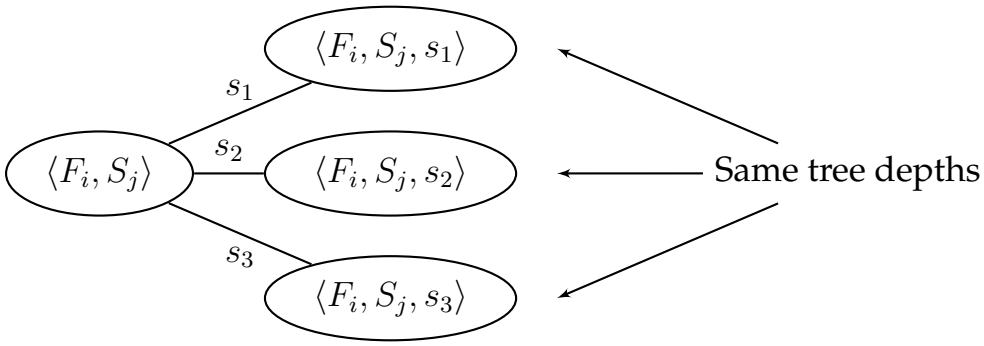

Figure 5.2: Illustration of exploiting symmetry of samples in a game-tree. In this case, for the next sample, we can uniformly choose from $\{s_1, s_2, s_3\}$

As seen before, in the general case, the sample selection algorithm needs to have access to the tree depth from each sample to determine the best sample. However, in some cases, it does not matter what the actual depth of the subtree is because the depth of all possible subtrees is the same (see figure 5.2). In some cases, we can at least identify a set of samples with the same lowest tree depth. This eliminates the need for actually calculating the tree depth and the optimal sample selection strategy would be to choose any sample from this set.

However, though any sample is optimal on the metric we are optimizing — average case performance, for a specific sample, there can be a pathological objective function. Instead, if we choose a sample uniformly at random from this set of optimal samples, in

some cases, we can improve our worst case performance. This is formalized below in Theorem 6.

Definition 2. *Two sets of fitness functions F_1 and F_2 over a sample space S are said to be* permutable *with each other if a one-to-one mapping ψ from $S_1 \subseteq S$ to $S_2 \subseteq S$ exist such that,*

$$\forall f \in F_1, \exists g \in F_2 \text{ s.t. } \forall s \in S_1, f(s) = g(\psi(s))$$

We will denote this as $F_1 \longleftrightarrow_{S_1,S_2} F_2$. We can also say f permutes to g to express the one-to-one mapping between F_1 and F_2.

Lemma 4. *If $F_1 \longleftrightarrow_{S_1,S_2} F_2$, and if S_1 and S_2 consists of all samples that may be selected for sampling for F_1 and F_2 respectively, then F_1 and F_2 has the same tree depth.*

PROOF. We will prove the two trees isomorphic by induction. Consider $T_{\langle F_1,S_1\rangle}$ and $T_{\langle F_2,S_2\rangle}$. We know that for the two root f-set vertices $\langle F_1, S_1\rangle$ and $\langle F_2, S_2\rangle$, $\forall s \in S_1, \psi(s) \in S_2$ and vice-versa and $F_1 \longleftrightarrow_{S_1,S_2} F_2$.

Let $\langle F_i, S_i\rangle$ be a vertex in $T_{\langle F_1,S_1\rangle}$ and $\langle F'_i, S'_j\rangle$ be a vertex in $T_{\langle F_2,S_2\rangle}$ such that $F_i \longleftrightarrow_{S_i,S'_i} F'_i$. Consider the child f-set vertices $\langle \text{OEFFSTEP}(F_i, s_k, u), S_i - \{s_k\}\rangle$ and $\langle \text{OEFFSTEP}(F'_i, \psi(s_k), u'), S'_j - \{\psi(s_k)\}\rangle$ where $u \in F_i$ and u permutes to u'. For f in F_i and g in F'_i such that $f \longleftrightarrow_{S_i,S'_i} g$, $f(s_k) = u(s_k)$ iff $g(\psi(s_k)) = u'(\psi(s_k))$. Thus, $\text{OEFFSTEP}(F_i, s_k, u) \longleftrightarrow_{S_i-\{s_k\},S'_i-\{\psi(s_k)\}} \text{OEFFSTEP}(F'_i, \psi(s_k), u')$. Additionally, $\forall s \in S_i - \{s_k\}, \psi(s) \in S'_i - \{\psi(s_k)\}$.

Thus by induction, the two trees are isomorphic. Consequently, their tree depths are the same. □

Theorem 6. *Two subtrees $\langle F_i, U, s_1\rangle$ and $\langle F_i, U, s_2\rangle$ have the same tree depth if there exists a one-one mapping of $\longleftrightarrow_{U-\{s_1\},U-\{s_2\}}$ from every set present in the multi-set $\{\text{OEFFSTEP}(F_i, s_1, u)|u \in F\}$ to another set in the multi-set $\{\text{OEFFSTEP}(F_i, s_2, u)|u \in F\}$.*

PROOF. Tree depth of subtree from a s-set vertex is calculated by averaging the depth of child f-set vertices (see Figure 3.1) and adding 1. Because of the mapping, for every subtree depth that we average for s_1, there is a subtree from s_2 with the same depth because of Lemma 4. Thus the averaged values are identical. Consequently, the tree depth from branches selected by s_1 and s_2 are the same. □

Now we can use these two techniques on the game-tree induced Generalized Leading Ones in an attempt to simplify it and derive an algorithm based on it.

5.2 Constructing an optimal algorithm for Generalized Leading Ones

In this section, we demonstrate the complete modeling process by applying the two techniques described in the previous section to develop an OEFF equivalent algorithm for the Generalized Leading Ones problem class defined below.

5.2.1 Generalized Leading Ones

We consider a particular generalization of the pseudo-Boolean LEADINGONES problem class (abbreviated here as GLO). For a space of binary sequences of length n, GLO can be defined as follows.

$$\text{GLO}_{\hat{x}}(x) = \sum_{i=1}^{n} \prod_{j=1}^{i} (x_i \equiv \hat{x}_i) \tag{5.1}$$

where $\hat{x}$ is a sequence of length n hidden in the fitness function. As there are 2^n unique strings in the search space, with each of them as a unique hidden sequence, there are 2^n unique fitness functions in GLO with that sequence as the solution. A fitness function in GLO returns the length of the longest prefix common between the sample point and the hidden sequence. These fitness functions are characterized by large plateaus that occur because changes outside the shared prefix do not influence the fitness value. The plateaus make it hard for a randomized search algorithm to optimize locally.

Even though simple EAs can only achieve $\Theta(n^2)$ over GLO, there exists an obvious algorithm that can achieve exactly n: simply flip bits one at a time. It is given as Algorithm 3, where f is the fitness function and on termination, the solution lies in x. In addition, the blackbox complexity for GLO is actually well known [11] as $n/2 + o(n)$ which is consistent with this algorithm. Indeed, part of the motivation for choosing this problem class is that we have existing results to compare with our derived algorithm.

Algorithm 3 Solving GLO one bit at a time

binary[N] $x \leftarrow [0, \ldots, 0]$

for $i = 0$ to $N - 1$ **do**

  if $f(x) = i$ **then**

    $x[i] \leftarrow 1$

  end if

end for

5.2.2 Modeling over Generalized Leading Ones

Let $F' = \text{OEFFSTEP}(F, s_k, u)$. For exploiting state duplication, using Theorem 1 and the defintion of GLO, we make this observation about OEFF on GLO.

Observation 2. *$\forall s_i \in S$ such that $u(s_i) \leq u(s_k)$,* $\text{OEFFSTEP}(F', s_i, u) = F'$

One can come to the same conclusion from observing the fitness table of GLO. From the definition of GLO, we know that this is because the fitness function in GLO returns the length of the longest shared prefix common to both the sample point and the solution of the fitness function. As the fitness functions OEFF has not eliminated must also share this prefix, a sample with a shorter prefix cannot eliminate them.

In GLO, as there are exactly $n + 1$ unique fitness values being returned by every function, from Observation 2 we can conclude that for a given unknown, a maximum of $n+1$ different elimination sets can be formed by OEFF. But different unknowns may form different elimination sets.

Consider a set F_i^s containing fitness functions sharing a prefix of length i where s is the sample that produced it. As explained later in Equation 5.3, $u(s) = i - 1$. Let us define F_i^s as follows.

$$F_i^s = \left\{ f \;\middle|\; \begin{array}{l} f \in F, \\ \\ f[1 \ldots i] = u[1 \ldots i] = s[1 \ldots i-1]\bar{s}[i] \end{array} \right\} \tag{5.2}$$

where $u[i \ldots j]$ is the subsequence of the solution of u (as u has a unique solution), from index $i > 0$ up to and including j. The set F_i^s for $0 \leq i \leq n$ and $s \in \{0,1\}^n$ form all the possible sets of un-eliminated fitness functions.

There is no transition from any $F_i^{s_i}$ to any $F_j^{s_j}$ for $j < i$ as $F_j^{s_j} \supset F_i^{s_i}$ if $s_i[1 \ldots j-1] = s_j[1 \ldots j-1]$ and $F_j^{s_j} \cap F_i^{s_i} = \emptyset$ otherwise. There is a transition for $j > i$ iff $u(s_j) = j - 1$. This can be written as follows,

$$\textsc{OeffStep}(F_i^b, s, u) = \begin{cases} F_i^b & u(s) < i \\ \\ F_{u(s)+1}^s & u(s) \geq i \end{cases} \tag{5.3}$$

This reduction in states results in a new tree given in Figure 5.3. Let us examine how optimal sample selection behaves in this tree. For an arbitrary state $F_i^{b_i}$, we will attempt to identify the set of optimal samples which we will prove symmetrical. Consider a sample s such that $s[0 \ldots i] \neq b_i[0 \ldots i-1]\bar{b_i}[i]$. As $u(s) < i$, the state will remain F_i. Let,

$$S_i^{b_i} = \left\{ s \;\middle|\; \begin{array}{l} s \in \{0,1\}^n, \\ \\ s[0 \ldots i] = b_i[0 \ldots i-1]\bar{b_i}[i] \end{array} \right\} \tag{5.4}$$

Consider any two samples $s_1, s_2 \in S_i^{b_i}$. For some $u_1 \in F_i^{b_i}$, let $d_1 = u_1(s_1) + 1$.

$$\textsc{OeffStep}(F_i^{b_i}, s_1, u_1) = F_{d_1}^{s_1} \tag{5.5}$$

We know there exists $u'_1 \in F_i^{b_i}$ such that $u'_1(s_2) = u_1(s_1)$. In fact,

Observation 3. $\forall u_i, \exists! u'_i \; s.t. \; u_i(s_1) = u'_i(s_2)$

Additionally,

$$\textsc{OeffStep}(F_i, s_2, u'_1) = F_{d_1}^{s_2} \tag{5.6}$$

Theorem 7. $F_{d_1}^{s_1} \longleftrightarrow_{S_{d_1}^{s_1}, S_{d_1}^{s_2}} F_{d_1}^{s_2}$.

PROOF. Consider the map M where $\forall y \in \{0,1\}^{n-d_1}$ we map $s_1[1 \dots d_1 - 1]\bar{s_1}[d_1]y \in S_{d_1}^{s_1}$ to $s_2[1 \dots d_1 - 1]\bar{s_2}[d_1]y \in S_{d_1}^{s_2}$.

Let $f \in F_{d_1}^{s_1}$. Choose $f' \in F_{d_1}^{s_2}$ such that $f[d_1 \dots n] = f'[d_1 \dots n]$. Let $s_3 \in S_{d_1}^{s_1}$. It is clear that $f(s_3) = f'(M(s_3))$. In other words, f permutes to f'. The theorem follows from Definition 2. □

Additionally, from $F_{d_1}^{s_1}$, we will not choose a sample point outside $S_{d_1}^{s_1}$. Similarly for $F_{d_2}^{s_2}$. We also know from Observation 3 that the number of $F_{d_1}^{s_1}$ formed by u_i are the same as the number of $F_{d_1}^{s_2}$ formed by u'_i. Thus, from Theorem 6, it follows that all the samples in S_i^b are symmetrical and have the same tree depth.

Thus the best sample selection strategy from F_i would be to randomly pick a sample from $S_i^{b_i}$. We cannot benefit from Theorem 5 to reduce states further for OEFF since, due to the property of GLO, we already ignore all prior samples.

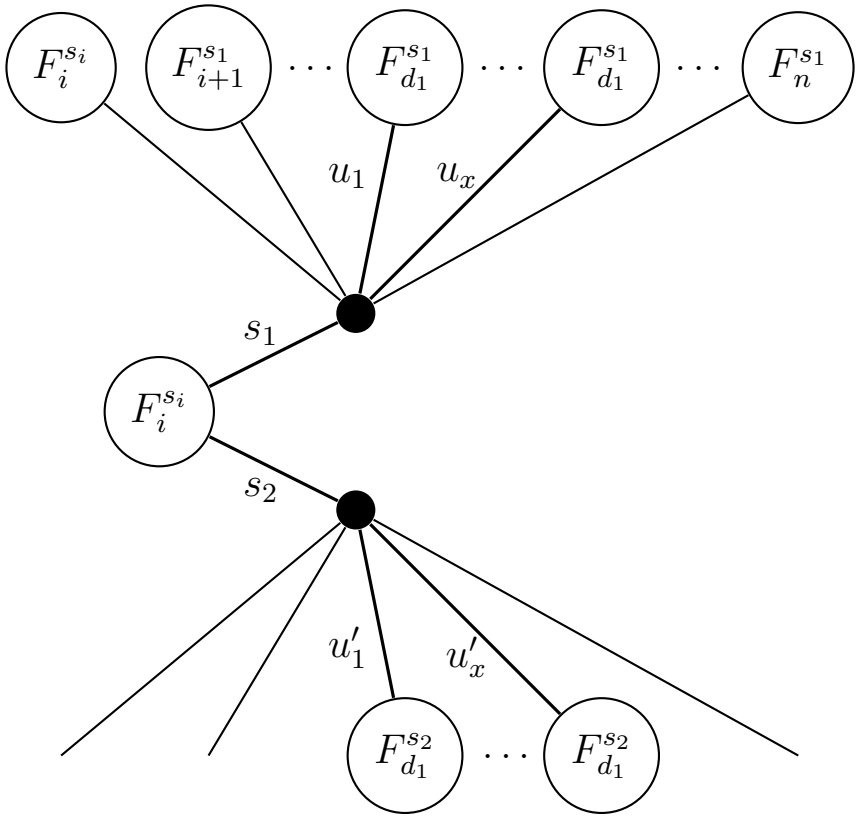

Figure 5.3: Game tree of OEFF for GLO after exploiting state-duplication. Two levels from an arbitrary node $F_i^{S_i}$ is shown.

5.2.3 Deriving the algorithm and its blackbox performance

From an arbitrary state $F_i^{b_i}$, we randomly pick a sample from $S_i^{b_i}$. This sample shares a prefix of length i with the solution. The probability of the $i+j^{\text{th}}$ bit being the first wrong bit is $\frac{1}{2^j}$ for $j \geq 1$. Thus the probability of getting a new fitness $i+j-1$ is also $\frac{1}{2^j}$. Equation 5.3 tells us that this is also the probability of transitioning to state F_{i+j}. Using these probabilities to calculate the expectation of j given m higher states,

$$E_m(j) = \sum_{i=1}^{m} \frac{i}{2^i} = 2 - \frac{m+2}{2^m}$$

For large n, m can also be expected to be large.

$$\lim_{m \to \infty} E_m(j) = 2$$

Thus for large n, the expected number of samples OEFF with optimal sampling will take to solve Generalized Leading Ones is $\frac{n}{2}$. As the number of samples is a sum of as many independent random variables with finite variance, by law of large numbers, the variance on the blackbox complexity approaches 0 for large n. We conclude that this is the exact blackbox complexity of Generalized Leading Ones.

An algorithm modeling the behavior of OEFF with optimal sampling can achieve this bound with linear space and time complexity. This is given as Algorithm 4. The function RANDOM(k) returns a random binary sequence of length k.

Our bound for the expected first-hitting time for Algorithm 4 is consistent with the general blackbox complexity shown in [11]. In fact, the sampling strategy they con-

Algorithm 4 OPTIMALGLO(u)

binary[n] $s \leftarrow$ RANDOM(n)

$f_{\text{best}} \leftarrow u(s)$

while $f_{\text{best}} < n$ **do**

$s[f_{\text{best}} + 1] \leftarrow \bar{s}[f_{\text{best}} + 1]$

if $f_{\text{best}} = n - 1$ **then**

break

end if

$s[f_{\text{best}} + 2 \ldots n] \leftarrow$ RANDOM($n - f_{\text{best}} - 1$)

$f_{\text{best}} \leftarrow u(s)$

end while

return s

sider for establishing these bounds is equivalent to Algorithm 4, and it's clear that when one considers success probabilities for our algorithm, the bounds from [11] also hold — it is lower bounded by $n/2 - o(n)$ and upper bounded by $n/2 + o(n)$. It is important to note that the algorithm that is derived by OEFF is identical to the optimal algorithm suggested in [11]. Moreover, because we have shown that this is equivalent to OEFF using an optimal sampling strategy, we know that this simple algorithm embeds *all* relevant domain knowledge for this problem class. Additionally, it is worth pointing out that were the blackbox complexity bounds *not* known for GLO, runtime analysis of OPTIMALGLO would establish these bounds, per our discussion in the previous chapter.

5.3 Constructing an optimal algorithm for LO^{**}

We can further generalize the GENERALIZEDLEADINGONES problem class by assigning a non-zero weight to each bit. This generalization is called LO^{**} and was previously defined in [4]. The objective function is given below.

$$LO^{**}_{\hat{x},w}(x) = \sum_{i=1}^{n} \left(w_i \cdot \prod_{j=1}^{i} (x_i \equiv \hat{x}_i) \right), \tag{5.7}$$

where $\hat{x}$ is a binary sequence of length n that is the solution of the fitness function and $w \in \mathbb{R}^n_+$. A fitness function in LO^{**} returns the total weight of the bits in the longest prefix common between the sample point and the solution. Key differences between LO^{**} and

GLO are that the plateaus can occur at any fitness value, and that the fitness function space is infinite due to the fact that w exists in a continuous, real valued space.

LO^{**} makes a good case to consider for our discussion for several reasons. First, though OEFF itself is restricted to finite problem classes, it is important to note that the FFE framework allows us to say something about some infinite problem classes analytically, and it is possible to derive efficient and black-box optimal algorithms using the ideas of OEFF, even if OEFF itself cannot optimize such classes.

5.3.1 Outline and introduction

In LO^{**}, each objective function is characterized by both $\hat{x}$, the solution and the weight vector w. As functions with all combinations of weights and solution can occur, it is important to note that for every weight vector, all possible solutions can occur.

The goal of an algorithm for LO^{**} is to identify the solution — $\hat{x}$ of the unknown objective function u. Identifying the weight vector of u is not always required to suggest the correct $\hat{x}$. The weight vector of u is relevant if and only if it helps improve the performance of the algorithm. We consider a subset of optimization algorithms for LO^{**} that do not examine the magnitude of the fitness values of samples, only their relative ordering. As a consequence, these *weight blind* algorithms do not learn about the weight vector of u.

To derive an optimal algorithm for LO^{**}, first we show that no algorithm for LO^{**} can outperform the best weight blind algorithm for LO^{**}. We do this by showing that for any function $u \in LO^{**}$, we can select a finite subset of objective functions $LO^{**}_u \subset LO^{**}$ "around" u such that even over this smaller finite subset, it is impossible to use what we can learn about weights of u to solve it using fewer samples than an optimal weight blind algorithm.

Next, in sections 5.3.4 and 5.3.5 and we derive the ideal weight blind algorithm derived from OEFF over some subset of LO^{**} with a finite set of weights. As optimal weight blind algorithm for a finite set of weights remain optimal even for infinite set of weights, we conclude that the derived algorithm is the optimal algorithm for LO^{**}.

Before we begin, let's clarify some terminology. First, lets consider things from the perspective of sampling. Note that all of the LEADINGONES problems we have discussed so far have the property that the objective value is 0 unless at least some of the prefix matches the correct solution. Moreover, it is also the case that if $u(\hat{s}_i) > u(\hat{s}_j)$ then, $\hat{s}_i$ must match more of the prefix of the true solution of the target function than $\hat{s}_j$. Observing this, we see that if $\hat{s}$ is the sample with the highest fitness seen so far, we know that this sample shares at least as long a prefix with the solution as any other sample we have seen. Given a sample $\hat{s}$, we say that the sample has *"solved"* all the bits in the shared prefix between the sample and the target function's solution. We refer to the first bit (from left to right) that differs from the target function's solution as the "blocking bit" of $\hat{s}$. Bits after a blocking

bit are effectively "disabled" and do not affect the fitness value. Like GLO , the position of the blocking bit is often unknown to the algorithm.

Now let's consider things from the perspective of elimination of functions. As OEFF removes functions as candidates for the target function, when all remaining functions have the same bit value for a particular bit position of the potential solution — when $\hat{x}_i$ is the same value for all remaining functions — we say that OEFF *"knows"* that bit. Likewise, when OEFF removes all functions save for those with a particular weight at the same position, we say that OEFF *knows* that weight for the target function.

5.3.2 Proving the optimality of weight blind algorithms

A defining property of LO^{**} is that unlike GLO, the fitness value may not be able to directly tell us how many bits have been solved. This is because each bit can be weighted differently and the observed increase in fitness can suggest multiple numbers of solved bits. We want to show that in LO^{**}, this makes keeping track of weights inconsequential to identifying the solution. We demonstrate this by showing that given any $u \in LO^{**}$, we can find a subset of LO^{**} with a finite set of weights such that weight vectors cannot be eliminated fast enough to eliminate solutions faster than a weight blind version of OEFF.

For any unknown function $u \in LO^{**}$ with weight vector $\hat{w}$, we develop a weight set W_u as follows.

$$W_u = \left(\bigcup_{j=1}^{n} \bigcup_{i=1}^{n} \left\{ \frac{\hat{w}_j}{i} \right\} \right) \bigcup \left(\bigcup_{j=1}^{n} \bigcup_{i=j}^{n} \left\{ \sum_{k=j}^{i} \hat{w}_k \right\} \right) \tag{5.8}$$

Let LO_u^{**} be the problem class of all functions from LO^{**} whose weights consist entirely of values from W_u. Several things are worth noting about LO_u^{**}. First, obviously, $LO_u^{**} \subset LO^{**}$. Second, since LO_u^{**}, like LO^{**}, is parameterized by a solution vector $\hat{x}$, all solutions are represented in LO_u^{**} and with equal proportion. Since there are a finite number of positions in the bit string, a finite number of possible weight values, and a finite number of potential solution strings, there are a finite number of functions in LO_u^{**} and OEFF can be applied to it.

Finally, note that for every potential target function f in LO^{**}, there exists some finite subset LO_f^{**} though they are obviously not all the same.

A weight blind version of OEFF only uses information contained in the relative ordering of fitness values of its samples. For example, if the latest sample x returned fitness v, the highest fitness so far, OEFF would try to use the absolute value of v to eliminate functions that return a different fitness value for x. However, a weight blind version of OEFF would only use the fact that v gave the highest fitness so far, and eliminate function for which that would not be true. As all fitness values returned by u are a sum of consecutive weights starting from w_1, including 0, there are exactly $n + 1$ unique fitness values. As $w_i > 0$ for $1 \leq i \leq n$, the weight blind OEFF exploits the fact that for $x, y \in S$, x shares a longer prefix with the solution of u than y if and only if $u(x) > u(y)$. Thus this

algorithm depends only on the fact that the weights are positive, which holds true for every function in LO^{**}. From this, we can make the following observation.

Observation 4. *The weight blind version of* OEFF *for* LO^{**}_u *and* $LO^{**}_{u'}$ *for any* $u, u' \in LO^{**}$ *are the same algorithm.*

Next, consider OEFF and a weight blind algorithm applied to u given $u \in LO^{**}_u$.

Lemma 5. OEFF *can never eliminate more potential solutions sampling* u *given* $u \in LO^{**}_u$ *than the optimal weight blind algorithm, also given* $u \in LO^{**}_u$.

PROOF: As u is the unknown, the fitness increases are always a sum of a sequence of consecutive weights of u. Consider an example u with weight vector $< a, b, c, d >$. For the example,

$$\bar{W}_u = \left\{ \frac{a}{2}, \cdots, \frac{d}{2}, \frac{a}{3}, \cdots, \frac{d}{3}\frac{a}{4}, \cdots, \frac{d}{4} \right\} \bigcup \{a, b, c, d, a+b, b+c, c+d, a+b+c, b+c+d, a+b+c+d\}$$

As a consequence of making a new sample x , compared to some previous sample y three outcomes are possible for a weight blind OEFF.

1. *Fitness goes up*: This can only happen if the first bit different between x and y was the blocking bit for y. Thus, weight blind OEFF can eliminate functions that do not share all the bits up to and including that blocking bit.

Though OEFF is able to know the sum of the weights of newly solved bits, it does not help. For any increase in fitness $w_i + \cdots + w_{i+k}$ resulting from having solved k new bits in the target functions, there exist functions in the candidate pool that can result in the same increase solving any number of remaining bits. Since this is always true for any k, while many functions will get eliminated based on their weights, no potential solutions can be eliminated in addition to what a weight blind OEFF can do.

In the example, lets say the first sample produced fitness v. Thus, all functions that share no bits are eliminated by a weight blind OEFF, solving one bit. OEFF cannot do any better because, If $v = a + b + c + d$, that could be just one bit with weight $a + b + c + d$, two bits with weights $(a + b + c), d$ and so on up to four bits. Similarly, if $v = a$, it could be just one bit with weight a or two bits with weight $a/2$ each or three bits with weight $a/3$ each or four with weight $a/4$ each.

2. *Fitness does not change*: This can only happen if the first bit (say bit i) different between x and y occurs after the blocking bit of y. A weight blind algorithm now knows that bit i and later bits were not solved in y, resulting in elimination of functions for whom blocking bit would occur on or after position i.

   In this case, OEFF cannot eliminate any additional solutions in any case because the lack of difference in fitness provides no additional weight related information to eliminate solution. Recall that this hold true even for GLO.

3. *Fitness goes down*: As the functions eliminated by neither OEFF nor its weight blind version is affected by the order of the samples, the assertion for this case follows from the case where fitness goes up.

Thus, we conclude that the weight blind OEFF algorithm eliminates exactly the same potential solutions as OEFF. □

To summarize, from observation 4, we know that the weight blind version of OEFF for LO^{**}_u and $LO^{**}_{u'}$ for any $u, u' \in LO^{**}$ are the same algorithm. From lemma 5, it follows that this single weight blind algorithm can solve $u \in LO^{**}$ with no more samples than any other algorithm that is given $u \in LO^{**}_u$.

For any algorithm to solve LO^{**}, it must identify the solution, not a particular function (many functions have the same solution). From an elimination perspective, this means that all functions with the wrong solution must be ruled out as a possibility. Since LO^{**}_u contains functions with all possible solutions, to solve the problem, any algorithm for LO^{**} must be able to rule out all $f \in LO^{**}_u$ for which the solution is wrong, no matter what is in $LO^{**} - LO^{**}_u$. As we have shown that this task cannot be performed with fewer samples than the weight blind OEFF, and as we have shown that this algorithm can solve all $u \in LO^{**}$, we conclude that it is the optimal algorithm for LO^{**}.

5.3.3 Simplifying the game tree

To derive the optimal weight blind algorithm, we first simplify the state space for OEFF over a finite subset of LO^{**} with an arbitrary but finite set of weights by merging all the states whose functions have the same set of possible solutions, but different weights. This will not affect the distribution of solutions because, at any point of time, any weight vector in the candidate pool has associated with it, all the solutions in the candidate pool. In this reduced state space, the states are differentiated only by the un-eliminated solutions that are represented. We develop an algorithm based on this game-tree.

Given a sample, we can separate all solutions into sets (called *position* sets) based on the position of the first bit that is different between the solution and the highest sample so far. In one such set with first different position i, there are 2^{n-i} functions because, the bits that follow bit i can take any value. In any set of un-eliminated solutions, either all or none of the members of this set will be present in that state depending upon how many bits we know of the solution. Thus, any set of un-elimination functions, and thus any state, can be characterized using the best sample so far and which of its position sets are represented in the set of un-eliminated solutions. It can be equally characterized by the probability for each position set with which the unknown solution can be its member. This results in the following state representation.

$$F^{\hat{s},u(\hat{s})}_{<p_0,p_1,\cdots,p_n>} \tag{5.9}$$

where u is the unknown, $\hat{s}$ is the sample with the highest fitness seen so far, and p_j is the probability of the $(j+1)^{\text{th}}$ bit of $\hat{s}$ being the first bit different from the solution, which as reader will recall is called the *blocking bit* of $\hat{s}$. p_n is the probability of having no blocking bits. Please pay attention to the fact that the probability for the first bit is p_0.

Now, let us consider sample selection. Let the original sample s_1 be chosen uniformly at random. Let i be the index of the first bit that changes between sample s_j and the next sample $s_{j'}$ such that $u(s_{j'})$ is the highest fitness observed so far and $u(s_j)$ is the next highest. In comparing $u(s_{j'})$ to $u(s_j)$, three outcomes are possible:

1. $u(s_{j'}) > u(s_j)$, *fitness goes up*: Bit i *was* the blocking bit. OEFF now knows at least i bits. The new blocking bit occurs somewhere after i and we present the probability of those positions being the blocking bit later. The bits in $s_{j'}$ after i up to the next blocking bit are being used by u to calculate the difference in fitness between s_j and $s_{j'}$.

2. $u(s_{j'}) < u(s_j)$, *fitness goes down*: The blocking bit occurs after bit i. The values of bits in $s_{j'}$ after position i do not affect $u(s_{j'})$ because flipping i has essentially disabled them.

3. $u(s_{j'}) = u(s_j)$, *fitness remains the same*: The blocking bit occurs before bit i. So, the bits after $i-1$ were definitely disabled during fitness evaluation for both v_j and $v_{j'}$.

Based on the second and third case, we need only flip one bit, as the value of the rest of the bits do not matter. So, their values can be chosen based on the first case.

In this case, the bits after i in any sample cannot be distinguished by u. So, as they were generated at random in s_1, we can keep them same as in s_1. Thus, flipping just one bit, say j, and sampling is equivalent to flipping bit j and flipping bits to its right at random and sampling. Among these, for simplicity, we follow a sampling strategy involving flipping only one bit at a time.

To solve more bits, we must flip the blocking bit. So, based on p_j values, we may be aware that certain bits cannot be the blocking bit, and we can avoid flipping the bits with $p_j = 0$. To denote the new sample we get after flipping a bit, we introduce the function $v(s, d) = s[0 : d-1]\bar{s}[d-1]s[d : n]$, where s is a sample point and $d \leq n$.

Now that sample selection is substantially simplified to choosing a bit, we notice that our state representation has an unlimited number of states due to the probability distribution. However, only a subset of them can actually occur. So, let us reason about the states that can occur as a result of sampling after flipping a bit and choose a more concise representation. Again, we consider the following possible outcomes of the action of flipping bit y:

1. *Fitness goes up*: If the fitness went up, then y necessarily was the blocking bit and we just flipped it. OEFF now knows bits 0 to y and the new blocking bit is now positioned after y. Thus, $p_j = 0, (j \leq y)$. In the candidate function set, there are 2^{n-y-1} different solutions, that share bits 0 to y and differ on the rest. As $2^{(n-y-1)-1}$ of these have a different bit in $\hat{x}_{y+1}$ than the last sample, with probability $1/2$, the

blocking bit is at $y+1$. Similarly, with probability $1/4$, the blocking bit is at $y+2$. Finally, with probability $1/2^{n-y-1}$, the blocking bit is at $n-1$ and that is also the probability that all the bits have been solved (p_n). Let i denote the right-most blocking bit that we flipped in all the samples made so far.

2. *Fitness goes down*: This means that we flipped a bit we had already solved. Thus, OEFF knows that we have solved at least bit up to and including bit y resulting in $p_j = 0, (j \leq y)$. If $y \leq i$, we cannot eliminate any function. Otherwise, the resulting candidate function set and p_j are identical to the previous case. Let g be the right-most bit flip that resulted in a reduction or increase in fitness ($g \geq i$).

3. *Fitness does not change*: This means we flipped a bit that does not influence the fitness because the blocking bit is to its left. Thus, $p_j = 0, (j \geq y)$. In this case, OEFF can eliminate functions from the candidate pool whose blocking bits fall on or after y. There are 2^{n-y} of these. That leaves $2^{n-g-1} - 2^{n-y}$ in the candidate pool.

   We are interested in the smallest y that resulted in no change in fitness after the most recent increase in fitness due to flipping the blocking bit. This is because every time a blocking bit is flipped, the new blocking bit can fall anywhere and we cannot assume anything about its right side limits. Let r denote this smallest y. By default, after flipping the blocking bit, immediately after a fitness increase, r takes the value $n+1$ to reflect the fact that the new blocking bit could be at n.

Having examined the events that change the state, we note that all probability distributions have contiguous non-zero probability values — no zero probability position i has non-zero probabilities at both $i-1$ and $i+1$. In other words, all positions, whose position sets are present in the set of un-eliminated solution, are contiguous.

Considering the present/not-present nature of position sets and relationship between their sizes, we can represent all possible F-states from equation 5.9 that can occur using $K_{i,g,r}^{\hat{s},u(\hat{s})}$. For convenience, the variables are,

- $\hat{s}$: The sample with the highest fitness so far.
- u: The unknown function being solved.
- i: The right-most blocking bit that we flipped in all the samples made so far.
- g: Index of the right-most bit whose flip resulted in any change in fitness.
- r: Index of the left-most bit whose flip resulted in a decrease in fitness since the most recent increase in fitness. If no decrease has occurred, it is $n+1$.

The underlying probability distribution over blocking bit positions denoted by j can be given as follows:

$$K^{\hat{s},u(\hat{s})}_{i,g,r}(j) = \begin{cases} \frac{1}{2^{n-g-1}} & \text{if } j = r-1 \wedge r = n+1 \\ \frac{2^{n-j-1}}{2^{n-g-1}} = \frac{1}{2^{j-g}} & \text{if } g < j < r-1 \wedge r = n+1 \\ \frac{2^{n-j-1}}{2^{n-g-1}-2^{n-r}} & \text{if } g < j < r \wedge r < n+1 \\ 0 & \text{otherwise } (j \geq r \vee j \leq g) \end{cases} \tag{5.10}$$

So far we have reduced the state space of a weight blind OEFF substantially and we have a concise representation that reflects the information contained in the candidates sets of OEFF. Next, we further examine the sample selection strategy over this new state space.

5.3.4 Deriving an ideal sample selection strategy

We are concerned with the optimal bit to flip for any state. Once the blocking bit is flipped, the resulting state is independent of the samples taken before the latest sample. Thus, the optimal bit flip strategy minimizes the number of samples to flipping the blocking bit. Theorem 8 below proves the sample selection strategy that achieves this.

We observe that there are two types of probability distributions: where $r = n + 1$ and where $r < n$ in $K_{i,g,r}$. In the former, the last non-zero probability is same as the one before it and in the latter, the last one is half the one before it. We note that for both distributions, ignoring the position of the first non-zero probability, the values are defined entirely by the number of non-zero probability values. Let us denote the former by $P(x)$

where there are $x+1$ non-zero probabilities and the latter by $P'(x)$ where there are x non-zero probabilities.

Theorem 8. *To eliminate the blocking bit from positions $j < n$ that have non-zero probabilities p_j in $P(x)$ and $P'(x)$, the optimal strategy is to sample after flipping bits one at a time starting from bit $g+1$ until and including bit $g+x$ and the minimum number of samples that is required to do this, denoted by $E(P(x)), E(P'(x))$, is x.*

PROOF: We will prove this by induction on x. Consider the base case $P(1)$. Here, $g = n-2$ and our ideal option is to flip $n-1$. Once this is done, we would have $p_n = 1$ and all else 0. Next, consider $P'(1)$ when $g = n-3$. Our ideal sample is $n-2$ because it is the only possible blocking bit. As a result, p_{n-2} would become 0. Now, consider $P'(1)$ for an arbitrary g. It is clear that the optimal sample still flips $g+1$ and results in the blocking bit being eliminated from bit $g+1$. Thus the base case is proven.

Assume that the this property holds true for $P(m)$ and $P'(m)$. Thus, $\forall x \leq m, E(P(x)) = E(P'(x)) = x$. Now, consider $P(m+1)$, which always is identical to $K_{i,n-m-2,n+1}$. Let us compare flipping bit $g+1$ with flipping bit $g+y, y \geq 2$. If we flip $g+1$, with probability $1/2$, we will gain in fitness and otherwise, we lose fitness. With $g+y$ we can also see no change. For our hypothesis to be true, the samples we need if we start by flipping $g+1$

should be no greater than if we start by flipping $g + y$. This can be written as follows.

$$\begin{aligned}
&\frac{1}{2} \cdot (1 + E(P(m))) + \frac{1}{2} \cdot (1 + E(P(m))) \\
&\qquad \leq K_{i,n-m-2,n+1}(n - m - 2 + y) \cdot (1 + E(P(m - y + 1))) \qquad \boxed{\text{fitness increase}} \\
&\qquad + \left(\sum_{j=y+1}^{n} K_{i,n-m-2,n+1}(n - m - 2 + j) \right) \cdot (1 + E(P(m - y + 1))) \quad \boxed{\text{decrease}} \\
&\qquad + \left(\sum_{j=1}^{y-1} K_{i,n-m-2,n+1}(n - m - 2 + j) \right) \cdot (1 + E(P'(y - 1)) + E(P(m - y + 2)))
\end{aligned}$$

In the last line we add $E(P(m - y + 2))$ because, having pushed the blocking bit out of bits $(g + 1) \cdots (g + y - 1)$ using $E(P'(y - 1))$ samples, we must still push the blocking bit out of the rest of the bits.

$$\begin{aligned}
\frac{1}{2} \cdot (1 + m) + \frac{1}{2} \cdot (1 + m) \;&\leq\; \frac{1}{2^y} \cdot (1 + (m - y + 1)) \\
&\quad + \frac{1}{2^y} \cdot (1 + (m - y + 1)) \\
&\quad + \left(1 - \frac{1}{2^{y-1}}\right) \cdot (1 + (y - 1) + (m - y + 2))) \\
(1 + m) \;&\leq\; \frac{1}{2^{y-1}} \cdot (m - y + 2) + \frac{2^{y-1} - 1}{2^{y-1}} \cdot (m + 2) \\
0 \;&\leq\; 2^{y-1} - y
\end{aligned}$$

This holds true for all $y \geq 2$. Thus our optimal sample flips $g + 1$ and so, $E(P(m + 1)) = 1 + E(P(m)) = 1 + m$. Next, consider $P'(m + 1)$ from $K_{i,g,g+m+2}$. We have $K_{i,g,g+m+2}(g + y) = \frac{1}{1-2^{-m}} \cdot \frac{1}{2^y}$.

$$\frac{1}{1-2^{-m}}\cdot\frac{1}{2}\cdot(1+E(P'(m)))+\left(1-\frac{1}{1-2^{-m}}\cdot\frac{1}{2}\right)\cdot(1+E(P'(m)))$$

$$\leq K_{i,n-m-2,n+1}(n-m-2+y)\cdot(1+E(P(m-y+1))) \qquad \boxed{\text{fitness increase}}$$
$$+\left(\sum_{j=y+1}^{n}K_{i,n-m-2,n+1}(n-m-2+j)\right)\cdot(1+E(P(m-y+1))) \quad \boxed{\text{decrease}}$$
$$+\left(\sum_{j=1}^{y-1}K_{i,n-m-2,n+1}(n-m-2+j)\right)\cdot(1+E(P'(y-1))+E(P(m-y+2)))$$

$$\begin{aligned}(1+m) \;\leq\; & \frac{1}{1-2^{-m}}\cdot\frac{1}{2^y}(1+m-y+1)\\ & +\left(\sum_{j=y+1}^{m+1}\frac{1}{1-2^{-m}}\cdot\frac{1}{2^j}\right)\cdot(1+m-y+1)\\ & +\left(\sum_{j=1}^{y-1}\frac{1}{1-2^{-m}}\cdot\frac{1}{2^j}\right)\cdot(m+2)\end{aligned}$$

$$\begin{aligned}(1+m) \;\leq\; & \frac{1}{1-2^{-m}}\cdot\frac{1}{2^y}(m-y+2)\\ & +\frac{1}{1-2^{-m}}\left(\frac{1}{2^y}-\frac{1}{2^{m+1}}\right)\cdot(m-y+2)\\ & +\frac{1}{1-2^{-m}}\left(1-\frac{1}{2^{y-1}}\right)\cdot(m+2)\\ \leq\; & \frac{1}{1-2^{-m}}\left(\left(\frac{1}{2^{y-1}}-\frac{1}{2^{m+1}}\right)(m-y+2)-\frac{m+2}{2^{y-1}}+m+2\right)\\ \leq\; & \frac{1}{1-2^{-m}}\left(m+2-\frac{y}{2^{y-1}}-\frac{m-y+2}{2^{m+1}}\right)\end{aligned} \tag{5.11}$$

For $y=2$, we have

$$m+1\leq\frac{1}{1-2^{-m}}\left(m+1-\frac{m}{2^{m+1}}\right),$$

which holds true for $m \geq y - 1$. As we increase y, for the same m, the two negative terms in equation 5.11 decrease and so, the inequality must hold. Thus our ideal sample flips bit $g + 1$ and thus, $E(P'(m + 1)) = 1 + E(P'(m)) = m + 1$. □

5.3.5 The optimal algorithm for LO^{**}

Based on this analysis, we can write the optimal algorithm for LO^{**} as given in Algorithm 5. The initial sample can tell us the first bit with $1/2$ probability. Otherwise, it will take an extra sample for the first bit. So solving the first bit is expected to take 1.5 samples. After the first bit is solved, we will be at $K^s_{0,0,n+1}$. Then, as we are flipping one bit at a time, and as we cannot converge until we have flipped the last bit to ensure that $p_n = 1$, we will flip an additional $n - 1$ bits. So, this algorithm will take exactly $n + 1/2$ samples.

The work [4] determines the blackbox complexity upper-bound of this class as $n + 1$ by considering enumerative search of each bit position as the worst case. Their lower bound, $n/2 - O(n)$ is taken optimistically from the GLO lower bound. Here we show that this optimism is not warranted: Blackbox algorithms cannot hope to do better than enumeratively searching through the bit positions.

Algorithm 5 OPTIMAL$LO^{**}(u)$

binary[n] $s \leftarrow$ RANDOM(n), $g \leftarrow 0$, $f_{last} \leftarrow 0$

while $g < n$ **do**

$f_{new} \leftarrow u(s)$

if $f_{new} > f_{last}$ **then**

$s[g+1] \leftarrow \bar{s}[g+1]$ {Flip bit $g+1$}

$f_{last} \leftarrow f_{new}$

$g \leftarrow g+1$

else if $f_{new} < f_{last}$ **then**

$s[g] \leftarrow \bar{s}[g]$ {Undo previous flip}

$s[g+1] \leftarrow \bar{s}[g+1]$ {Try again}

$g \leftarrow g+1$

else

$s[g] \leftarrow \bar{s}[g]$ {Initial sample has fitness 0. Flip the first bit.}

$f_{last} \leftarrow f_{new}$

end if

end while

return s

5.4 Analyzing OEFF on a generalization of LONGPATH

In this section, we examine behavior of OEFF over our generalization of the LONGPATH objective function which we call f_{LP}. The function is similar to a generalized ONEMAX with solution $\{0\}^n$. However, starting at $\{0\}^n$, there is a specific sequence of sample points that differ from adjacent samples by a single bit such that the fitness of each sample in this sequence is one higher than its predecessor.

This path is valid only for odd dimensions. P_l denotes the path for l dimensions. $P_1 = \{0, 1\}$. To construct this path P_{l+2}, create the path for P_l recursively. Then, create subpath S_{00} by prepending "00" to each point in P_l and S_{11} by prepending "11" to each point in P_l in the reverse order. The bridge point is obtained by prepending "01" to the last point in P_l. We can get P_{l+2} by joining S_{00}, the bridge point and S_{11}. The length of this sequence is $3 \cdot 2^{(n-1)/2} - 1 = O(2^n)$ for odd n. In addition, due to the design of this sequence, we can know that it ends at $11 \cdot 0^{n-2}$.

The problem was constructed by Horn et. al. [57] as a means to show that EAs need crossover — supposing that a mutation based EA would have to slowly traverse an exponentially long path. However, as Rudolph [58] showed, due to the structure of this path, a simple mutation based EA can make leaps along this path using multiple bit mutations. Thus, a simple (1+1) EA can find the solution in $O(n^3)$ time. No one has yet posited a blackbox complexity bound for common generalizations for LONGPATH. However for the generalization given below, it is easy to see that it is $O(n/\log n)$.

This function f_{LP} has the interesting property that the length of the path as well as ratio of the sample space to the length grows exponential to the size of the search space. To study this problem, a generalization makes the problem relevant to our blackbox discussion, since knowledge of a single instance class is tantamount to knowing the solution. Let our problem class, generalized LONGPATH be,

$$F^*_{\text{LP}} = \{f_y | y \in \{0,1\}^n \wedge f_y(x) = f_{\text{LP}}(x \oplus y)\},$$

where $\oplus$ is the bitwise exclusive or operator.

To solve an objective function from F^*_{LP}, a (1+1) EA would first solve the generalized ONEMAX in $O(n \log n)$. Then, the (1+1) EA can solve the path in $O(n^3)$.

Despite the generalization, due to the specific and monotonic increase in fitness on the path, on sampling a single sample on the path, OEFF can immediately identify the problem instance and solve the unknown. However, due to the exponentially small chance of hitting the path by random, OEFF cannot expect to identify the instance quickly based on this. It can solve the underlying generalized ONEMAX problem class in $O(n/\log n)$. As the solution of the generalized ONEMAX problem is a point on the path, and the path is unique in the class, OEFF need not take any more samples, all functions except the target function will have been eliminated.

Thus, in this problem, despite the generalization, there is a a large difference in the performance of EAs and OEFF. Still, it is worth mentioning that one could take the

optimal algorithm for ONEMAX from [10], run it to the start of the path, then flip the first two bits to solve the problem.

5.5 STEEPPATH: Easier for EAs, tougher for OEFF

The generalized LONGPATH that we devised above is shown to be no harder for OEFF than the underlying generalized ONEMAX, primarily due to the predictable nature of the path. Thus, we introduce an interesting variation called STEEPPATH that has the opposite property. It is easy to get on the path due to fixed starting point, but it is harder to identify where it ends due to randomization. Additionally, the path is no longer than the number of dimensions. This results in this problem class being easier for EAs (specifically the (1+1) EA) and OEFF is able to do no better than the (1+1) EA. The STEEPPATH problem class is given below.

$$f_{<v_1,v_2,\cdots,v_k>}(x) = \begin{cases} 2^n & x = \hat{x}, \\ \max_{i=1}^{k} \prod_{j=1}^{i} (x_{v_j} + 1)(x_{v_j} \equiv \hat{x}_{v_j}) & \text{otherwise}, \end{cases} \tag{5.12}$$

where $0 \leq k \leq n/2$, and $\forall i, j \leq k$ s.t. $i \neq j, v_i \neq v_j$. Also, $\hat{x}$ is the solution, which has 1s only at v_i.

In this problem class, any objective function returns 0 except for a specific shortest path between $\{0\}^n$ and its solution, which could be any string with no more than $n/2$ 1s. In this path, $\{0\}^n$ is evaluated to 0 (unless it is the solution). From then on, for a specific

sequence of non-repeating single-bit flips, the resulting samples start with a fitness of 2 and then doubles for every flip. The solution is always given a fitness of 2^n. This problem class can be also seen as a generalization of LEADINGONES.

5.5.1 Simplifying the states

As this problem class is already discrete, there is no need for discretization. By default, OEFF will maintain all surviving objective functions in the problem class. However, after each sample, we can say this about the un-eliminated functions:

Lemma 6. *After sampling s that has d 1's, and getting a fitness of 2^d, the un-eliminated functions include only those functions whose $v_{d+1}, \cdots, v_k$ is a subset of positions of 0's in s. We call this set $F^s_{(d)}$.*

PROOF: In the STEEPPATH problem class, consider functions with the solution 11001000. The v_i values for any these functions will be a permutation of $\{1, 2, 5\}$ which are the positions of 1s in the solution. Generalizing from this, it is clear that initially, all permutations of every set in $\wp(\{1, \cdots, n\})$ of size at most $n/2$ are present as the sequence of v_i for an objective function. After sampling s, all the functions with $k \leq d$ get eliminated. All functions whose $v_1, \cdots, v_d$ matches the 1's in s survive. For the surviving functions, the remaining 1s in the solution whose positions do not exist in $v_1, \cdots, v_d$, must be in po-

sitions that have 0s in s. Thus the position of these remaining 1s is a subset of the position of 0s in s. □

Let us initially consider a sampling strategy of first sampling the all zero string. If the fitness is 2^n, we terminate. Otherwise, we are at $F^s_{(0)}$ where s is the all zero string. If we flip a random 0-bit and sample, there is $\frac{1}{n}$ chance that that bit position was v_1 and by lemma 6, we reduce the problem class. If we fail to increase the fitness, we can remove those functions whose v_1 was that bit position. Let u_j be the bits that were flipped for the j^{th} failed sample after the most recent fitness increase. This results in the following state,

$$F^s_{(d),<u_1,\cdots,u_{k'}>}$$

From a given state, as we remember the failed flips, we need not repeat them as they will produce no change in state. Now, if we only flip one bit of s at a time, then u_i can be just a number indicating the position of the flipped bit. Otherwise, it is a set of positions. We show below that we need only flip one bit at a time.

Lemma 7. *From any given state $F^s_{(d),<u_1,\cdots,u_{k'}>}$ consisting of only single 0-bit flips with $0 \leq k' < (n-d)$, flipping only one 0-bit at a time is better than more than one.*

PROOF: At state $F^s_{(d),<u_1,\cdots,u_{k'}>}$, as k' can be 0, this includes the state where we have made no failed samples at all after the most recent fitness increase. Thus, if we show single-bit flips as optimal, we will never flip more than 1 bit at a time.

To reach state $F^{s'}_{(d+b)}$, we have to flip a specific set of b bits in s viz., $\{v_{d+1}, \cdots, v_{d+b}\}$. We consider two approaches: flipping bits one at a time, solving b bit separately and

flipping b bits together, solving them in one go. We start with the latter and show that it is slower than the former.

In the candidate pool for $F^s_{(d),<u_1,\cdots,u_{k'}>}$, v_{d+1} can take $(n-(d+k'))$ values. $v_{d+i}, ((d+1) < (d+i) \leq n/2)$ can take $(n-(d+i-1))$ values. Thus, there are,

$$\frac{(n-(d+k'))\cdot(n-(d+1))!}{(n-j)!}$$

functions with their $k=j$. Furthermore, there are,

$$S(b) = \frac{(n-(d+k'))\cdot(n-(d+1))!}{b!\cdot(n-(d+b))!}$$

ways to choose b bits, of which, at most one is correct. As we can remember our failed samples in u, we can expect to take $S(b)/2$ samples before we can make a transition, as long as $k \geq d+b$. Functions with $d < k < d+b$ will force OEFF to sample $S(b)$ samples. However, we remain optimistic and stick with $S(b)/2$. We find that for $4 \leq b \leq n/2, S(b) = \Omega((n-d)^3)$.

If $b=1$, from $F^s_{(d),<u_1,\cdots,u_{k'}>}$, we will reach $F^{s'}_{(d+1)}$ after $\frac{n-(d+k')}{2}$. Note that this is not an optimistic estimate as we will always make the transition. So, to reach $F^{s'}_{(d+b)}$, we can expect to take $(n-(d+k')) + \sum_{i=d+1}^{d+b-1} \frac{n-i}{2}$ samples.

We find that $S(1) = O((n-d)^2) < \Omega((n-d)^3)$. Additionally, for $b=2,3$, $S(1)$ remains linear while $S(2)$ and $S(3)$ are quadratic and cubic respectively on $n-d$. From this, we conclude that our best strategy is single-bit flip. □

Having chosen single bit flip, we can expect to solve a function with k 1's in $\sum_{i=0}^{k-1} \frac{n-i}{2}$ samples. $p(k)$, the probability of this function occurring is,

$$p(k) = \frac{k!}{\sum_{j=0}^{n} j!}$$

Thus, the expected number of samples from $F^s_{(0)}$ is,

$$E(n) = \sum_{k=0}^{n} \left(p(k) \cdot \sum_{i=0}^{k-1} \frac{n-i}{2} \right)$$

Now, from the initial state, if we do not sample $\{0\}^n$, with probability $\frac{1}{\sum_{j=0}^{n} j!}$, we will take n samples. However, with the remaining probability, we will gain 1 sample. Thus, it is superior to not sample $\{0\}^n$ first and directly sampling its single bit flips. So the expected number of samples is,

$$E(n) = \frac{n}{\sum_{j=0}^{n} j!} + \sum_{k=0}^{n} \left(\frac{k!}{\sum_{j=0}^{n} j!} \cdot \sum_{i=0}^{k-1} \frac{n-i}{2} \right) = \Theta(n^2)$$

5.5.2 Comparison to EA

An (1+1) EA with single-bit flip can be initialized for this problem class with the all zero string as the initial population. To an EA, this is no different than LEADINGONES. A bit-flip based EA must flip the leading bit, which at a mutation probability of $1/n$ takes roughly an expected n mutations – and it must do this n times (at most, depending on k). Runtime analysis for the (1+1)-EA for GLO has been done, $\Theta(n^2)$. Thus, we see that for

the STEEPPATH problem, the optimal algorithm for it needs the same order of samples as a simple EA.

We can add a generalized ONEMAX to the plateau of STEEPPATH and generalize it similar to how we did LONGPATH. The resulting class can be a super-class of generalized ONEMAX as well as a variant of a subset of LO^{***} (discussed in the next section). However, this change will just add an $O(n/\log n)$ phase to the OEFF algorithm and will not fundamentally affect its behavior. This generalization with also not affect the EA behavior. Consequently, we have identified a fairly broad class of problems for which a simple (1+1) EA is an optimal blackbox algorithm.

5.6 Deriving a lower bound for LO^{***}

The LO^{***} is a generalization of LO^{**} where the instance and the solution must have some fixed permutation applied to it before evaluation. Otherwise, the evaluation is like LO^{**}.

$$LO^{***}_{\hat{x},w,v}(x) = \sum_{i=1}^{n} \left(w_{v_i} \cdot \prod_{j=1}^{i} (x_{v_i} \equiv \hat{x}_{v_i}) \right), \tag{5.13}$$

where $\hat{x}$ is a binary sequence of length n that is the solution of the fitness function and $w \in \mathbb{R}^n_+$ such that $\sum_{j=1}^{n} w_j = 1$. A fitness function in LO^{***} returns the total weight of the

bits in the longest prefix based on some order v, common between the sample point and the solution.

We will repeat the definition of *know* and *solve* here. Value of a bit i is considered solved by s if, $\forall 1 \leq j \leq i, s_{v_j} = \hat{x}_{v_j}$. An $\hat{x}_i$, w_i or v_i is said to be *known* by OEFF if the candidate pool reflected in the current state consists of functions that share the same $\hat{x}_i$, w_i or v_i respectively.

Let us briefly consider similarities between this problem class and LO^{**}. Our goal is still to match the sample to the solutions. We cannot do any better than flipping the current *blocking bit*, the first mismatching bit, to make progress towards this goal. In addition, lemma 5 still holds true. We can also make the following observation.

Observation 5. OEFF *cannot know the weight or value of bits that follow the current blocking bit. As for the position, it can know that they are not positioned among known bit positions.*

However, in LO^{**}, it was impossible for OEFF to know the value of a bit without knowing the value and weights of all previous bits. This simplification is no longer possible. When a blocking bit is flipped, the next blocking bit v_i can be an arbitrary number of bits ahead in v and at an arbitrary position. And the position of the blocking bit can be found without finding out the position or values of the intermediate bits. Also, it is possible to terminate without knowing these values due to $\sum_{i=1}^{n} w_{v_i} = 1$, ensuring that finding these values have to be properly justified as the ideal strategy for OEFF.

For these reasons, we will avoid deriving an exact optimal algorithm and simply derive a lower bound based on OEFF. If we can lower bound the number of samples OEFF will take, then we have effectively developed a lower bound for the LO^{***} problem class. Because we are deriving a lower bound, rather than an actual algorithm, we can optimistically ignore some samples that we would take during search, which simplifies things.

We examine a process where after a blocking bit has been solved, it is learned by OEFF. Additionally, OEFF also learns the bits that come before the most recently learned blocking bit in the order imposed by v. Thus, once OEFF has learned the current blocking bit v_i, before it attempts to learn the next blocking bit, it always knows the weights, values and positions of that bit and all previous bits $v_j, w_{v_j}, \hat{x}_{v_j}, (j \leq i)$. Here, we will only count samples that OEFF will take to learn the position and value of the blocking bits and ignore the samples required to learn other values.

Having learned blocking bit at v_i, OEFF cannot know the index of the next blocking bit other than that it is one of the indexes that are in $\{1, \cdots, n\} - \{v_1, \cdots, v_i\}$. Let us examine all possible samples and their contribution to identifying the position of the next blocking bit. Let s be the next sample while s' the previous sample that has the highest fitness value seen so far.

Observation 6. $u(s) < \sum_{j=1}^{i} w_{v_j}$ *iff* s *and* s' *differ in bit positions* $v_1, \cdots, v_i$.

Thus, if the fitness value of s is lower, as we are assuming OEFF knows all the previous bit values, weights and positions, this sample will not help eliminate any functions. So, we consider samples that change only positions $v_{i+1}, \cdots, v_n$. Let the positions that we changed in s be $z_1, \cdots, z_k$. If the fitness value goes down, that means $\exists z_j, j' \leq i$ s.t. $z_j = v_{j'}$. It it remains the same, then all z_j are after the blocking bit. Finally, if it increases, then at least one z_j was the blocking bit and the blocking bit has moved.

So, we can identify the blocking bit by splitting $x = \{1, \cdots, n\} - \{v_1, \cdots, v_i\}$ randomly into two halves and flipping the bits in the first half. If we gain in fitness, we know that this half has the blocking bit $v_{i'}$ and none of the bits before it after v_i. We can now do a binary search in this half so that we can converge to the blocking bit in $\log |x|$. If we gain no fitness, we can try flipping bits in the other half and follow the same procedure.

However, if the set that contains the blocking bit also contains a bit before it, we would be unable to gain fitness by flipping either halves. Then, we have to repeat the process with a smaller number of bits per flip in the hope that we get the blocking bit $v_{i'}$ alone, without any bits $v_j, j < i'$ in the same set. We consider the previous attempt iteration 1 and begin iteration 2. In this iteration, we split the x into $1/4$ sets of equal sizes at random and sample after flipping the corresponding bits in each set looking for a fitness a fitness increase. At iteration m, we will divide the bits in x into $1/2^m$ sets. As soon as an increase in fitness is seen, we proceed with a binary search for the blocking bit.

If we do all the iterations down to where the sets are single bits, it would take us $|x|\log(|x|)$ samples. For the lower-bound analysis, we optimistically assume we always get a fitness increase in the first iteration. Thus, identifying the blocking bit takes $\log(|x|)$ samples. As flipping each blocking bit gains us only 2 bits, we conclude that we take at least $\sum_{j=0}^{n/2}\log(2\cdot j))$ samples. Thus, the number of samples required to solve LO^{***} is $\Omega(n\log n)$.

The alternative to learning about all the bits up to the blocking bit is not learning about them. So, we examine the alternate strategy that involves not learning the position of the blocking bit or any of the bits that have been solved. This would allow us to avoid the binary search after the iteration stage and allow us to solve early blocking bits in $O(1)$ each.

However, assume we have solved $n/2$ bits. We need to find a set of bits such that it includes the blocking bit and does not include any of the first $n/2$ bits in v. We can expect to need $\log(n)-1$ iterations for each blocking bit as only if the bit positions were divided into sets of size 2 can we expect to find such a set. As each iteration is $\Omega(n)$, each blocking bit after $n/2$ samples will take $\Omega(n\log(n))$. Even if we do only the final iteration, that is still $\Omega(n)$, resulting in $\Omega(n^2)$ for the algorithm. Thus the resulting algorithm will be clearly slower than the bound proven above. It is also clear that learning some bits while not learning others is not going to improve the performance.

Having examined both the option of learning v and not learning it, we conclude that the best strategy is to learn it and that this algorithm is $\Omega(n\log n)$. Given this, we are

able to relax our assumptions about the weights adding up to 1 as this strategy does not depend on it. Similar to LO^{**}, the upper bound $n \log n + O(n)$ for the blackbox complexity of LO^{***} is derived in [4], while the lower bound is based on GLO. We were able to show that, like LO^{**}, the upper bound is asymptotically tight.

5.7 Implications and limitations of algorithm design by formal modeling

Formal modeling cannot work with a partial understanding of the problem class and requires complete domain knowledge. Additionally, this knowledge must be in a form that can be used to build a game-tree for the problem class. This can limit the applicability of this approach.

Also, in many cases, the formal process we have described may be a very complicated task to implement. In addition, it can only be used for deriving an algorithm that behaves exactly like OEFF and such an algorithm may not be practical. In such cases, it makes sense to approximate OEFF instead of modeling it exactly. An approximation process has more flexibility than modeling due to the relaxation of the requirement to be equivalent to OEFF. Our attempt to model OEFF teaches us about some trade-offs that we face while approximating OEFF. We observe three extremes for algorithms:

1. An algorithm can be like OEFF and acquire large amounts of domain knowledge through parameters. This provides OEFF with adequate domain knowledge to be optimal w.r.t. blackbox performance. Due to the parameterized nature of domain knowledge, it also achieves a high degree problem abstraction. On the other hand, it loses performance during sample computation (referred to as *efficiency* from now on) as it needs to mine through the large parameterized domain knowledge.

2. Instead, it can be like the final algorithm presented in section 5.2.3. By giving up problem abstraction, it has domain knowledge available a-priori. This domain knowledge is mined and incorporated into the algorithm. Consequently, it is optimal over the target problem class *and* may have substantially increased efficiency.

3. Lastly an algorithm can be like the random search strategy. Random sampling eschews domain knowledge in any form. Consequently, it achieves a limited yet predictable performance of $(|S| + 1)/2$ (for problems classes with a single solution per function). However, this lack of domain knowledge confers it efficiency as well as total problem abstraction. Admittedly, random search is $\Theta(|S|)$ on space if the list of samples needed to avoid duplicate samples is maintained as a boolean array. It is $\Theta(\log |S|)$ on time as it needs to generate as many random bits and only constant time to check for duplication using the boolean array. Enumerative search is $\Theta(\log |S|)$ on space to store the latest sample and can avoid generating a random number per sample. Nevertheless, we consider only random search because,

enumerative search can have pathological cases where sampling $|S|$ point may be required.

From this, we argue the following corollary.

Corollary 1. *Given domain knowledge in an explicit representation, of the three key characteristics of algorithms viz. problem abstraction, blackbox performance and algorithm efficiency, an algorithm can maximize only two.*

DISCUSSION: In practice, we have efficient algorithms with high blackbox performance for specific problem classes that search through a large space for a solution. These algorithms employ search heuristics that take advantage of shared structures in the objective functions of the problem class. However, such targeted algorithms exhibit low blackbox performance for problem class that do not have the same structure.

Given domain knowledge in this explicit representation, we have two choices: use it as it is at run-time or manually mine it for structure during design time to derive efficient heuristics. If we do the latter, the algorithm that results can be efficient and can have good performance, but only for that specific problem class. On the other hand, if we use the domain knowledge as it is during run-time, we have the OEFF algorithm.

For OEFF, which depends on such a representation, eliminating objective functions based on each sample requires looking up upto $|F_i|$ values from the table. In the sample selection strategy, as we have to examine all samples, simply to create child nodes for $< F_i, U >$, we must examine all $|U| \cdot |F_i|$ values. Even though this value decreases over

time due to a reduction in the candidate pool, we need to look up at least $2 \cdot |U|$ values as there are always at least 2 functions in the candidate pool. So, depending upon the problem class, sample selection from the initial state, $< F, S >$, can take any where from $2 \cdot |S|$ table lookups to $\Omega(|S|! \cdot |F|)$ lookups. As we cannot make any assumptions about these values in the *general case*, each sample requires examining as many values as the size of the search space, which is inefficient. Thus, it is impossible to maximize all three characteristics.

If domain knowledge for any problem class can be provided in any form, then an algorithm can require that domain knowledge be made available in the form of optimal heuristics for that problem class. The algorithm can then employ them for searching for the solution efficiently. Such an algorithm maximizes all three characteristics. However, having any domain knowledge available as relevant structural information is an unrealistic scenario.

We already show that OEFF, algorithm modeling OEFF and random search are three algorithms that maximizes the three pairwise combinations of characteristics, proving that maximizing any two is possible. □

Figure 5.4 shows this trade-off between key characteristics of algorithms. Thus, there cannot be a "perfect" algorithm under the blackbox framework that maximizes all three characteristics. Thus, all algorithms must make compromises to some degree and the ideal mix is highly context dependent.

In the next section, we make some more observations about the blackbox framework and issues concerning designing of blackbox optimization algorithms.

5.8 Observations on practical algorithm design

We feel that those who are willing to invest a lot of time to incorporate domain knowledge into their algorithm will find some kind of generalized optimization framework an unnecessary complication. Thus, we assume that the practitioners above all else favor problem abstraction, which facilitates algorithm reuse and makes it easy to solve their problem using existing work. We are also of the opinion that when compared to blackbox performance, maximizing efficiency is not a very important requirement for users of the blackbox optimization framework.

Given this point of view, we note that a vast majority of optimization algorithms used in practice choose to treat an objective function as a virtual blackbox in the sense that they examine only the fitness value returned by the function and do not examine how it was computed. Note that we call this design choice the *fitness value restriction*. Typical examples are Genetic Algorithm, NEAT, Particle Swarm Optimization etc. There are exceptions, viz. novelty search and multi-objectivization approaches. While the fitness value restriction improves interface compatibility, we note some disadvantages:

1. *Fitness value restriction leads to complicated algorithms.* The NFL theorem underlines the necessity of domain knowledge for performance better than random search. Thus, a practitioner is forced to write the algorithm and objective functions such that the assumptions of the optimization algorithm matches the objective functions closely. Using domain knowledge to manually build such algorithms can be complicated and we feel that it makes it difficult to achieve ideal performance.

2. *Common objective function design choices along with fitness value restriction leads to artificial upper-bounds on performance.* The fitness value restriction does not limit the information content in a fitness value. Thus, even with fitness value restriction, depending on the objective function, the blackbox complexity can be arbitrarily small. However, in practice, in addition to the fitness value restriction, objective functions are often limited in what they inform through the fitness value. The fitness value usually is a grade for points in the search space and is used as such by the optimization algorithm. If we assume that in practice, it is difficult to incorporate maximum possible information into each fitness value and still use it effectively in an optimization algorithm, then the fitness value restriction limits how informative each evaluation can be about the solution, which affects performance negatively. algorithm design guideline is that it limits domain knowledge access to instance information. In practice, we feel this is often an artificial restriction that should be ignored if it does not help the practitioner design algorithms of adequate performance.

3. *Fitness value restriction is not proven to be necessary for problem interface unification.* One advantage of having exactly one instance knowledge source (the objective function) is that the algorithm written for one objective function is compatible with any objective function, performance aside. This important advantage attracts many users who want to take advantage of the large library of existing optimization algorithms. However, we argue that it has not been shown that this restriction is required for a unified problem interface. We recommend exploration of frameworks that allow additional instance knowledge sources while maintaining an equivalent level of problem interface unification, allowing reuse of algorithms.

Another important observation that is relevant to practical approximations is that OEFF builds a model of the problem class using samples. Model building during optimization can be seen in various existing algorithms [59]. However, most of these approaches involve modeling the underlying patterns in the set of candidate solutions to determine a new solution, a relatively domain knowledge light approach. In contrast, OEFF stresses the importance of modeling the problem instance because it defines the relationship between the samples and the solution of the instance.

This suggests the use of approximations of OEFF where simpler models of the problem instance can be learned from samples and a learned model can be used to partially identify the solution with no additional samples. Creating such an algorithm would require the practitioner to define a model for the problem instance as well as define the relationship between a model and the solution.

In Chapter 7, we introduce theory that describes how we can gather domain knowledge from problem instances. We also make recommendations for designing an optimization algorithm that reflect the observations above.

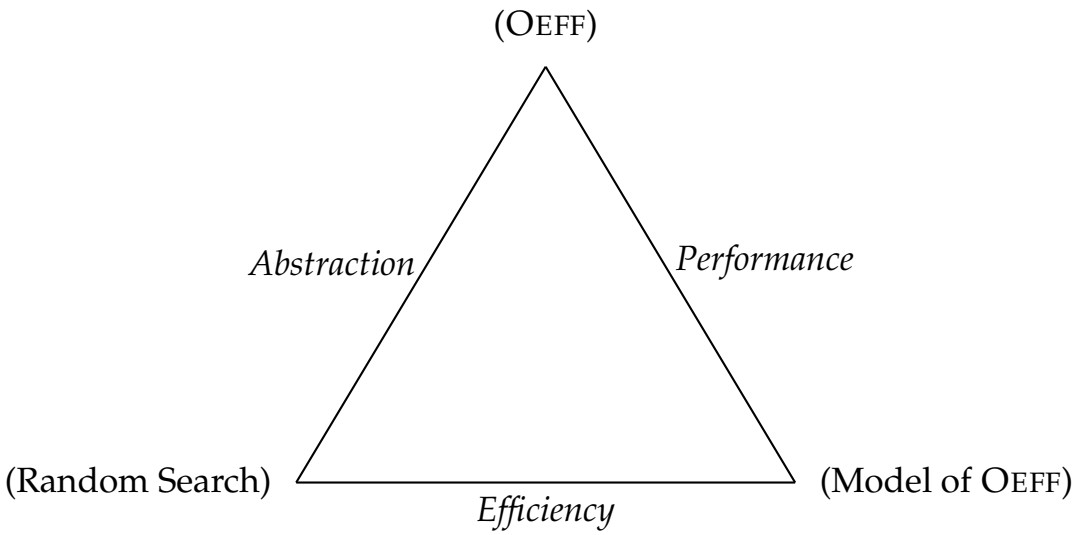

Figure 5.4: The triangular trade-off between problem abstraction, blackbox performance and algorithm efficiency.

CHAPTER 6
PARTIAL DOMAIN KNOWLEDGE REPRESENTATION AND USE

So far we have discussed the situation where we have complete domain knowledge as specified by the blackbox framework. But in practice, this scenario is rare and our domain knowledge is usually incomplete. We call our knowledge the *subjective* knowledge. It is subjective because initially it is our belief about the problem class and is not something we can query. In contrast, we call the correct complete domain knowledge about the actual problem class as *objective* knowledge, something that is independent of our opinion. Importantly, objective truth impacts reality while subjective knowledge does not.

To devise a search strategy for a problem class, a scientist will try to improve his subjective knowledge as much as possible and then incorporate it into the algorithm. In our view, almost all algorithms for realistic problems are built using incomplete subjective domain knowledge rather than objective knowledge from an accurate formal description of the problem. Given this lack of domain knowledge, it makes sense to try to add to our subjective knowledge from the instances we are able to solve from the problem class. We call this *problem learning*.

Recall our discussion of existing problem learning approaches in section 2.4. From that discussion, some obvious questions arose:

1. Under what scenario can we update knowledge of the problem class?

2. What is optimal from an informational point of view?

3. What are the limitation of *any* approach to doing this?

In machine learning, there are well known Bayesian analysis results that show what is possible to learn in, for example, classification tasks. Likewise, learning-to-learn (their analogue to problem learning) has also been examined this way [60] . We review this in the next section.

Building on this, we describe an ideal representation for domain knowledge that may be incomplete. We then describe the scenario for problem learning and integrate this representation into OEFF and the sample selection strategy, enabling it to use partial domain knowledge ideally. In section 6.3, we derive Bayesian formulas for updating this partial domain knowledge using information about seen problem instances. Finally, we discuss the benefits and limitations of this — both from an idealistic point of view as well as a practical point of view.

6.1 A model for learning-to-learn in classification tasks

To help us understand problem learning in optimization, let us first examine the analogous learning-to-learn in classification tasks. We base this section on the classic theoretical treatment by Baxter [60]. In that paper, Baxter presented both empirical as well as Bayesian approaches to learning-to-learn. Differences in these two approaches arise primarily due to the latter's belief in an objective truth (analogous to objective problem class). As this is consistent with our belief also, we focus only on the Bayesian approach. First, let us review the Bayesian view of classification, for context.

The goal in classification is to use a set of labeled points to infer a general target function that can be used to label future unseen points. We start with some partial knowledge about this function and update our knowledge from labeled points. Formalizing this, the components of Bayes approach to modeling single-task learning are:

1. An input space X and an output space Y.

2. A set of probability distributions P_θ on $X \times Y$, parametrized by $\theta \in \Theta$.

3. A prior distribution $p(\theta)$ on Θ.

A single probability distribution from line 2 represents a classification problem. For example, if we are doing character recognition, X would be a set of images (typically a subset of R^d representing pixel intensities) and Y would be the set of characters and digits. The set in line 2 represents a set of all possible classification problems, with any

mapping between X and Y. The problem that is currently given to learn is denoted as θ^*, the exact value of which is unknown.

From line 3, we see that a prior probability distribution $p(\theta)$ is available, along with $\{P_\theta : \theta \in \Theta\}$, represents the a-priori knowledge or bias of the learner. Thus, this represents the incomplete knowledge the algorithm has regarding the *current* classification task. Next, we see how this knowledge can be used for classification.

6.1.1 Learning: Using and updating knowledge of the current task

We can formally represent the training set consisting of labeled points from X as $z = \{(x_1, y_1), \ldots, (x_m, y_m)\}$. Contents of z are sampled according to P_{θ^*} where θ^*, the objective true classification, is unknown. Recall that the goal of classification is to use z to update the prior distribution $p(\theta)$. It is possible to compute a posterior distribution $p(\theta|z)$ as follows,

$$p(\theta|z) = \frac{p(z|\theta)p(\theta)}{p(z)} \tag{6.1}$$

$$= \frac{\prod_{i=1}^{m} p(x_i, y_i|\theta)p(\theta)}{\int_\Theta p(z|\theta)p(\theta)d(\theta)} \tag{6.2}$$

The probability distribution of possible outputs for an input x^* can be determined using $p(\theta|z)$ and averaging over Θ.

$$p(y|x^*; z) = \int_\Theta p(y|x^*; \theta)p(\theta|z)d(\theta)$$

So far, we have seen a theoretical model that demonstrates how we can best use our incomplete knowledge and labeled points to improve our understanding of the classification task. Our claim of "best" is based on our use of the Bayes theorem, which specifies formula for accurate update of knowledge from evidence. However, our interest extends beyond single task learning into what Baxter calls *learning-to-learn*. This is where we try to figure out the appropriate initial $p(\theta)$ for the type of classification tasks that we continue to face.

6.1.2 Learning-to-learn: Learning about the initial task knowledge

Consider the scenario where we are facing multiple classification tasks one after the other. Associated with each classification task is its unique objective distribution P_{θ^*}. These tasks are being sampled from $\{P_\theta : \theta \in \Theta\}$ by some probability distribution Q over it. Thus, Q is an *objective* distribution because it decides the nature of the classification tasks we will face.

The goal of learning-to-learning is learning Q. To learn Q, we must focus on the set to which Q belongs — the set of prior probability distributions $\{P_\pi : \pi \in \Pi\}$. Like Q, each P_π is a probability distribution on Θ. Let $Q = P_{\pi^*}$ for some $\pi^* \in \Pi$.

Thus, the components of Bayes approach to modeling learning-to-learn are:

1. An input space X and an output space Y.

2. A set of probability distributions P_θ on $X \times Y$, parametrized by $\theta \in \Theta$.

3. A set of prior distributions P_π on Θ, parametrized by π.

4. An *objective* prior distribution P_{π^*} where $\pi^* \in \Pi$.

5. A subjective *hyper-prior* distribution P_Π on Π.

Due to the two-tiered structure, this model is an example of *hierarchical Bayesian model*.

To use this model first we need to sample. First, n values of θ are sampled from Θ based on P_{π^*}, the objective prior distribution. Then for each θ_i, (x, y) are sampled m times from $X \times Y$ based on P_{θ_i}. Here, the learner receives $n \times m$ samples. Let $x_{i,j}$ correspond to j^{th} sample based on θ_i. For convenience, let the whole set of samples be $z^{(n,m)}$.

First, the learner generates the posterior distribution $p(\theta^n|z^{(n,m-1)})$ on the set of all n tasks, Θ^n.

$$p(\theta^n|z^{(n,m-1)}) = \frac{p(z^{(n,m-1)})p(\theta^n)}{p(z^{(n,m-1)})} \tag{6.3}$$

$$= \frac{p(\theta^n)\Pi_{i=1}^n \Pi_{j=1}^{m-1} p(z_{ij}|\theta_i)}{p(z^{(n,m-1)})} \tag{6.4}$$

where $p(z^{(n,m-1)}) = \int_{\Theta^n} p(\theta^n)\Pi_{i=1}^n \Pi_{j=1}^{m-1} p(z_{ij}|\theta_i) d\theta^n$. Then it uses the posterior distribution to generate a predictive distribution on Z^n.

$$p(z^n|z^{(n,m-1)}) = \int_{\Theta^n} p(z^n|\theta^n)p(\theta^n|z^{(n,m-1)})d\theta^n$$

Of particular interest to us is how this model for solving classification tasks translates to the problem of finding the optimum of fitness functions. In the next section, we develop an idealized formal representation for any domain knowledge incomplete or otherwise based on our existing representation for complete domain knowledge.

6.2 Idealized problem class learning in optimization

Our ideas for problem class learning in Blackbox optimization are based on the ideas reviewed in the previous section. To develop the representation for incomplete domain knowledge, we take the model for learning-to-learn in classification tasks and map to the field of optimization. This leads to almost analogous components.

1. An input space X (a finite sample space) and an output space Y (a finite set of real valued fitness values),

2. The set F of all possible fitness functions mapping from X to Y (set F is necessarily finite),

3. The power set $\wp(F)$ of the previous set referring to the set of all possible fitness function sets,

4. A particular set from $\wp(F)$ which represents the problem, the *objective* problem class,

However, this lacks a representation for incomplete or uncertain domain knowledge. As we assume an objective problem class that can be repeatedly sampled, the complete domain knowledge regarding sampled instances is fixed. As this problem class is a member of $\wp(F)$, from axiom 1, we can conclude that a probability distribution over $\wp(F)$ sufficiently constitutes a representation for any domain knowledge. Thus, to the above list, we add:

5. A *subjective prior* distribution Θ on $\wp(F)$ that denotes any prior knowledge about the problem class.

An algorithm that can use incomplete domain knowledge must be able to directly or indirectly use Θ. Our approach is to first collapse Θ to a probability distribution over objective functions called β. Then we modify OEFF and sample selection to use β instead of the boolean vector we described before. This is justified because of Axiom 1 combined with the fact that for OEFF, the unknown function is fixed. Thus Θ is too detailed a representation for OEFF and β is adequate. The β function can be computed as follows:

$$\forall f \in F, \beta(f) = \sum_{G \in \wp(F)} \Theta(G) \cdot \frac{1_G(f)}{|G|} \tag{6.5}$$

where 1_G is an indicator function for membership in G.

When developing OEFF, we assumed that the unknown instance is chosen uniformly at random from the problem class. Thus, we were able to represent domain knowledge using only a boolean representation instead of a full probability distribution.

However, now that a probability distribution is being given to us that may have unequal non-zero probabilities, the boolean representation for OEFF is rendered inadequate. To compensate for this, we do not provide domain knowledge directly to OEFF. Consequently, it starts with a boolean vector with all functions marked TRUE. We move the domain knowledge usage and the termination condition to the sample selection strategy.

We adjust the sample selection strategy to weigh the branches to the *f-set* vertices based on β after adjusting for eliminated functions. We terminate when sample selection returns $[0, \text{false}]$ or even earlier when we find that all functions with non-zero probability in β share a solution. The modified sample selection algorithm is given as Algorithm 6.

In the next section, we shall see how information gathered during a optimization can be used to update Θ.

6.3 Updating partial domain knowledge

Before we start examining how we can update incomplete domain knowledge, let us review the context under which we perform this update. We are trying to devise an algorithm that is trying to learn about optimizing some incompletely known problem class by updating the algorithm's belief about that class. The steps involved in this process can be described as given below:

1. Start with some subjective prior knowledge about the problem class

Algorithm 6 NEXTOPTIMALSAMPLE(F,U)

if functions in $\{f|f \in F \cap \beta(f) \neq 0\}$ share a solution **then**

return $[0, \text{false}]$

end if

$l_{\text{min}} = |S_{\text{u}}|$

for all s in S_{u} **do**

$l \leftarrow 0$

for all u' in F **do**

$F_{\text{temp}} \leftarrow \text{OEFFSTEP}(F, s, u')$

$l = l + \frac{\beta(u')}{\sum_{f \in F} \beta(f)} \cdot (\text{NEXTOPTIMALSAMPLE}(F_{\text{temp}}, S_{\text{u}} - \{s\})[0] + 1)$

end for

if $l < l_{\text{min}}$ **then**

$l_{\text{min}} \leftarrow l$

$s_{\text{min}} \leftarrow s$

end if

end for

return $[l_{\text{min}}, s_{\text{min}}]$

2. Get a random problem instance from the objective set.

3. Use the subjective prior to solve the instance.

4. Use the knowledge gathered from the instance while solving it to update the subjective prior.

5. Go to 2.

Introducing the variables, let us say the algorithm has Θ_0 as subjective prior. Θ_0 is converted to β_0. This β_0 is used to solve the first instance given using the OEFF and the sample selection algorithm.

At the end of the run, after we have solved the instance, while we may not be sure which instance exactly was solved, we can tell that the functions in the eliminated set H could not be the one that was solved. We can use this piece of information "The solved instance is not present in H" to update Θ_0 and produce Θ_1. This process repeats and we generate β_1 using Θ_1 and solve the next instance using β_1 and so on.

As the incomplete domain knowledge consists of a probability distribution Θ_i over $\mathcal{E}$, we must figure out how to update this probability distribution. We can do this by deriving the equation for updating the probabilities of individual functions in $\mathcal{E}$.

Consider the following events.

- C_f: the event that f is the current unknown objective function.

- E_H: The event that by the time the instance was solved, exactly the functions in H were eliminated. $H' = F - H$.

- E_G: The event that G is the actual problem class.

Theorem 9. *Given* $P(E_G) = \Theta(G)$,

$$P(E_G|E_H) = \frac{\Theta(G)|G \cap H'|}{|G| \sum_{g \in H'} \beta(g)} \tag{6.6}$$

PROOF. The probability of selecting a particular set of k samples is $\frac{1}{C_k^{|S|}}$. Given C_f, the set eliminated from F by sampling is unique to the samples selected. Thus,

$$P(E_H|C_f) = \begin{cases} \frac{1}{C_k^{|S|}} & \text{if } f \in H', \\ 0 & \text{otherwise} \end{cases} \tag{6.7}$$

Also, by definition, $P(C_f) = \beta(f)$. From these two we get,

$$\begin{aligned} P(E_H) &= \sum_{f \in F} P(E_H|C_f).P(C_f) \\ &= \sum_{f \in H'} \frac{1}{C_k^{|S|}} \beta(f) \\ &= \frac{\sum_{f \in H'} \beta(f)}{C_k^{|S|}} \end{aligned}$$

Thus we can compute $P(C_f|E_H)$ as follows:

$$\begin{aligned} P(C_f|E_H) &= \frac{P(E_H|C_f) \cdot P(C_f)}{P(E_H)} \\ &= \begin{cases} \frac{\beta(f)}{\sum_{g \in H'} \beta(g)} & \text{if } f \in H', \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

This tells us how to update our β based on the functions eliminated so far. The next step is to figure out how to update the Θ based on H after the solution is found. We have,

$$P(C_f|E_G) = \begin{cases} \frac{1}{|G|} & \text{if } f \in G, \\ 0 & \text{otherwise} \end{cases} \tag{6.8}$$

since the unknown is chosen uniformly from the problem class. Then,

$$\begin{aligned} P(E_G|C_f) &= \frac{P(C_f|E_G) \cdot P(E_G)}{P(C_f)} \\ &= \begin{cases} \frac{\Theta(g)}{\beta(f)\cdot|G|} & \text{if } f \in G, \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

$$\begin{aligned} P(E_G|E_H) &= \frac{P(E_G \cap E_H)}{P(E_H)} \\ &= \frac{1}{P(E_H)} \sum_{f \in F} P(E_G \cap E_H \cap C_f) \\ &= \frac{1}{P(E_H)} \sum_{f \in F} P(E_G|C_f \cap E_H) P(E_H \cap C_f) \end{aligned}$$

When $P(C_f \cap E_H) \neq 0$, it follows that $C_f \cap E_H \equiv C_f$. Thus we replace $C_f \cap E_H$ in the first term with C_f.

$$\begin{aligned}
P(E_G|E_H) &= \frac{1}{P(E_H)} \sum_{f \in F} P(E_G|C_f) P(E_H \cap C_f) \\
&= \sum_{f \in F} (P(E_G|C_f) \cdot P(C_f|E_H)) \\
&= \sum_{f \in G} \frac{\Theta(g)}{\beta(f) \cdot |G|} \cdot \begin{cases} \frac{\beta(f)}{\sum_{g \in H'} \beta(g)} & \text{if } f \in H', \\ 0 & \text{otherwise} \end{cases} \\
&= \sum_{f \in G, H'} \frac{\Theta(G)}{|G| \sum_{g \in H'} \beta(g)} \\
P(E_G|E_H) &= \frac{\Theta(G)|G \cap H'|}{|G| \sum_{g \in H'} \beta(g)} \qquad (6.9)
\end{aligned}$$

To confirm, the new β and the new Θ should sum to 1. It is obvious that the former is true. For the latter, when expanding β by definition, we find,

$$\sum_{f \in H'} \beta(f) = \sum_{G \in \mathcal{E}} \frac{\Theta(G)|G \cap H'|}{|G|}$$

Using this in equation 6.9 and summing $P(E_G|E_H)$ over all $G \in \mathcal{E}$, we get 1. □

6.4 Implications and discussion

Problem learning involves gathering domain knowledge from problem instances seen so far to help solve a future unknown instance. We presented a theory based on the Elimination of Fitness Functions approach that describes how this may be done ideally.

Based on this approach, we see that in EFF, problem learning involves a probability distribution over complete domain knowledge and rules to update this probability distribution based on seen samples from a solved problem instance.

The theory we have presented for problem learning suggests that in practice, given an optimization algorithm, problem learning can be achieved by approximating the probability distribution over the domain knowledge representation and approximating the update rules. Unlike OEFF, as most practical optimization algorithms encode domain knowledge, a requirement for complete problem learning is that this probability distribution must span a space of optimization algorithms. In fact, our choice of space over which we approximate the probability distribution that we learn reflects the knowledge that can be gathered by problem learning. E.g., in meta-evolutionary algorithms described in section 2.4.2, we approximate a probability distribution over algorithm parameters by maintaining a population of these parameters, limiting our learning ability to such a space.

The space of code for optimization algorithms is harder to search through than the space of parameters of optimization algorithms. This holds true even if the parameters are pieces of code such as in hyper-heuristics. This stresses the importance of having optimization algorithms that accept a significant portion of their final domain knowledge through algorithm parameters. These can be more easily modified than traditional optimization algorithms to become powerful problem learning optimizers.

6.4.1 Can we determine the benefits of problem learning?

An important question on the issue of domain knowledge gathering is whether or not it can be shown to improve performance. Another is whether we can predict how much performance improvement will result.

First, we examine what happens when we attempt problem learning given no prior domain knowledge. To understand more about having no domain knowledge, consider the lemma given below.

Lemma 8. *If all possible problem classes are equi-probable, then so are all the objective functions and vice versa.*

PROOF: Let 1_G be an indicator function for membership in G. We can compute the probability distribution over individual objective functions denoted by β as follows.

$$\begin{aligned} \forall f \in \mathcal{F}, \beta(f) &= \sum_{G \in \wp(\mathcal{F})} \Theta(G) \cdot \frac{1_G(f)}{|G|} \\ &= \frac{1}{\wp(\mathcal{F})} \cdot \sum_{G \in \wp(\mathcal{F})} \frac{1_G(f)}{|G|} \\ &= \frac{1}{\wp(\mathcal{F})} \cdot \sum_{i=1}^{|\mathcal{F}|} \frac{1}{i} \binom{|\mathcal{F}|-1}{i-1} \end{aligned}$$

As the last term is independent of f, the lemma is proven. □

Thus, when all problem classes are equally likely, all problem instances are too. Consider the situation where we have fully sampled the functions in the set $G \subset \mathcal{F}$. W.l.o.g., let $F \supset G$ be the objective set. Let $u \in (F - G)$ be our next problem instance

to be solved. As information has been gathered about F under idealistic assumptions, performance of OEFF over it must improve. However, as a consequence, all problem classes that shares no elements with G must decrease. $F - G$ is such a set.

This means that given some arbitrary objective set F, after sampling instances in G from it, though we face increased performance over F on average, we face worse performance than random search over every function in F that has not been seen so far. By coupon collector's problem, we know that we need $O(|F| \log(|F|))$ instances from F (sampled with repetition) to see all instances at least once. It is clear that to receive performance improvements over F fairly evenly, we must sample the problem class much more than its size. To improve performance over the majority of the problem class, even if we ignore repeated problem instances, we need to solve at least $|F|/2$. Thus, it is unrealistic to start with no domain knowledge and depend on problem learning alone to acquire sufficient domain knowledge for reasonable performance.

Next, using examples, we argue that performance improvement due to problem learning can be significant, which we feel is what can be expected in most practical situations. We argue that only in rare situation is improvement impossible. However, a more rigorous proof is needed to settle the issue.

Consider the problem classes F and G over a binary space of length $n + 1$.

$$F = \{f_{y_1}, f_{y_2}, \cdots, f_{y_n}\}$$

$$G = \{g_{y_1 y_1}, \ldots, g_{y_1 y_n}, g_{y_2 y_1}, \ldots, g_{y_n y_n}\}$$

where y_i is an n-bit binary string whose numerical value is i and,

$$f_y(x) = \begin{cases} 2n & \text{if } \forall i \leq n-1, x[i] = y[i] \text{ and } x[n] = 1 \\ n - H(x[0, n-1], y) & \text{if } x[n] = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$g_{yy'}(x) = \begin{cases} n - H(x[0, n-1], y) & \text{if } x[n] = 1 \\ 2n & \text{if } \forall i \leq n-1, x[i] = y'[i] \text{ and } x[n] = 0 \\ 0 & \text{otherwise} \end{cases}$$

where H is the hamming distance function.

In this problem, the search space is broken into two halves. One half where the last bit has value 1 is structured like a Generalized ONEMAX problem. For function $f_y \in F$, the other half has only 0s. Thus the solution will be $y.1$ and this sample will have fitness $2n$. This fitness value of $2n$ in the generalized ONEMAX half identifies this function as a member of F.

For the function $g_{yy'} \in G$, the generalized ONEMAX portion is based on y. However, the string $y.1$ only has fitness value n. In the other section of the search space, all fitness values are 0 except the point $y'.0$, which has fitness value $2n$ and is the solution. This half is similar to Needle-in-a-haystack (NIHS) landscape. Having the solution in this half identifies this function as a member of G.

Assume our prior knowledge is that the problem class is F, G, or $F \cup G$ with equal probability. We want to answer the question: Using only the prior knowledge, can we

accurately predict whether gathering domain knowledge from new instances is going to give us higher performance?

Recall from section 6 that we design our subjective prior into the optimization algorithm for best performance. To keep things simple, assume that we can solve the n-bit generalized ONEMAX problem in n samples, and that it takes on average 2^{n-1} samples to solve the NIHS portion. Given an unknown instance u, there are two broad strategies we can choose.

1. Assume $u \in F$. Solve the generalized ONEMAX problem using n samples. Success probability is $\frac{1}{2^n}$. If fail, solve the NIHS portion in 2^{n-1} samples. Expected time to solve is

$$\text{Expected time} = n.\frac{1}{2^n+1} + (n + 2^{n-1})\frac{2^n}{2^n+1} = h(n) + \frac{n2^n}{2^n+1}$$

2. Assume $u \in G$. Solve the NIHS problem using 2^{n-1} samples. If fail, solve the generalized ONEMAX problem. Expected time to solve is

$$\text{Expected time} = 2^{n-1}.\frac{2^n}{2^n+1} + (2^n + n)\frac{1}{2^n+1} = h(n) + \frac{2^n}{2^n+1}$$

From this, we conclude that solving the NIHS problem first rather than the generalized ONEMAX problem is likely to solve u sooner.

Now, we examine three different possibilities for the actual objective set from which u is being drawn

Case 1: $F \cup G$ **is the objective set**: In this case, the collapsed distribution of our subjective prior matches the subjective set exactly. Thus, trying to update the subjective prior from seen instances is not going to bear fruit.

Case 2: G **is the objective set**: In this case, if we gather knowledge from new instances, we can update our subjective prior to match problem class G. This improves our expected performance slightly by eliminating chance of failure. However, note that even though the expected performance of strategy 2 is slightly worse than what is possible with complete domain knowledge, because it solves the NIHS portion first, if the problem class turned out to be G, its actual performance is 2^{n-1}. Thus gathering domain knowledge will not benefit performance as it cannot get any better than 2^{n-1}.

This effect of improvement in expected performance with no improvement in actual performance is due to expected performance being calculated based on the subjective prior while the latter being calculated based on the objective set. This discrepancy between expected and actual performance can be steeper if strategy 1 was chosen and the objective set is F.

Case 3: F **is the objective set**: The performance we get on strategy 2 if F is the problem class is $2^n + n$ because our attempt to find solution in the NIHS section always fails. And if we learned that F is the problem class, we can skip the NIHS section entirely and our performance would be n. Compared to $2^n + n$, this is a huge improvement and underlines the benefits of domain knowledge gathering under some circumstances.

Thus, we show that depending upon the case, gathering domain knowledge may result in no improvement, some improvement and even great improvement.

Our second question is whether we can differentiate between these cases based on just our prior. From the above examples, it is clear that all three of these cases follow from the same subjective prior. Also, based on just the prior, we can make no conclusions about which case we are going to have. Thus, in this case, we cannot be sure one way or the other that gathering domain knowledge will deliver performance improvements.

However, consider the case where current domain knowledge and any future domain knowledge we may gather are all going to result in the same behavior from OEFF. For example consider various problem classes which are variations of Generalized ONEMAX differing only in increases to the fitness value of the solution. If all the problem classes had the same probability of being the true problem class, then it is no use to learn which one exactly is the true problem class. This is because no matter which one it turns out to be, OEFF is going to solve it like a Generalized ONEMAX problem class.

6.4.2 Suggestions for practical optimization algorithms

Based on the theory that has been presented so far, and the points we have made previously regarding the blackbox framework in section 5.8, we make the following rea-

soned recommendations for a practical optimization algorithm. As some of these recommendations are seen separately implemented in practice already, we provide examples.

1. *Be capable of using additional instance knowledge sources.* Due to the importance of instance knowledge sources in solving problem instances, we must build algorithms that are capable of using more than just the objective function. With any new source, we face the problem of interpreting it: what can the values from this source tell us about the problem instance? Interpreting this information requires domain knowledge which can be gathered through problem learning and/or be made available by practitioner. The fact that using additional instance knowledge sources violates the fitness value restriction over the problem class means that it can result in performance exceeding OEFF. An example is the multi-objectivization approach that uses "helper functions" based on the target objective function.

2. *Do problem learning based on all instance knowledge sources.* By definition, an objective function is only required to define solutions. For this, a needle-in-a-haystack function is adequate, and this function is not a useful instance knowledge source.

   An objective function can be used as an instance knowledge source only if the algorithm has domain knowledge — knows something about the values of non-solution samples — that can improve performance. This requirement of domain knowledge to give context to fitness values applies equally to additional instance knowledge sources. The more we know about how the values returned by these sources re-

late to the problem instance, the more domain knowledge we have and the more informative these sources will be.

Consequently, based on theory explained so far, problem learning can help us use these sources more effectively to identify the problem instance. In fact, the impact of problem learning is proportional to the number of additional sources and their informativeness.

3. *Accept significant domain knowledge in a form that is easy for human beings to understand and use.* The domain knowledge that a practitioner or a subject matter expert can provide to an optimization algorithm can make a significant impact on performance. Thus, it is critical to the performance of an optimization algorithm that it can accept domain knowledge from practitioners and experts in a form most convenient for them.

4. *Restrict the domain knowledge representation for efficiency.* In addition to restricting the domain knowledge for human input, it is also required to restrict it in ways that allow it to be used with a practical optimization algorithm. Even though efficiency is not the first priority for practitioners resorting to optimization and problem learning, as OEFF is in fact impractical, sacrifices must be made on the space of domain knowledge for the sake of practicality. Most algorithms restrict domain knowledge input to a handful of parameters.

5. *Use the information maximizing sample selection strategy if possible.* This sample selection strategy, which is introduced in this dissertation, is not optimal, but we believe it can be good enough to be an effective practical strategy. However, its applicability may depend on the chosen domain knowledge representation.

6. *Solve by building a model of the problem instance and using it to find at least a partial solution.* This is in line with expected behavior from an algorithm that aims to approximate OEFF. This is an important requirement which also restricts the domain knowledge representation as it must be informative for both problem identification as well as problem solving. Continuous and Embedded Learning algorithm from section 2.4.2 is a good example of this model building approach.

7. *Optionally, favor problem abstraction by maximizing scope of domain knowledge representation.* Based on existing theory, increasing the scope of the domain knowledge representation is likely to increase overhead, reducing efficiency. However, as stated in Chapter 2, the increase in problem abstraction may be more desirable to practitioners as it allows them to use off-the-shelf algorithms to solve related but new problem classes. This point brings to light the efficiency-abstraction trade-off that all algorithm designers must make.

The theory presented in this dissertation is insufficient to decide the best implementation of these suggestions in practice. For that, further research is required. Once

that is accomplished, we can implement these suggestions into a theoretically sound algorithm capable of problem learning. We leave this to future work.

6.4.3 Suggestions for Transfer Learning approaches

The problem learning theory we have developed is specifically for the black-box scenario where we learn about the problem class by sampling it. The transfer learning scenario is different from black-box scenario. In transfer learning, there are two problem classes of interest say, A and B. One of these is our objective problem class, say B and unlike the blackbox scenario, we are given no access to it in the beginning. Instead, we are given access to A which is expected to be "similar" to B. We gather domain knowledge from A with the hope that this domain knowledge improves performance over B. Note that transfer learning can be followed by regular problem learning using B. Such a scenario of using domain knowledge from one problem class to solve another is justified under many circumstances such as when B is too expensive to evaluate, but we were able to devise a cheaper problem class A that is similar.

Despite being a different scenario, we can apply the theory we have developed as both scenarios involve problem learning. Also, this theory is applicable whether the transfer learning is automated or manual. First, we try to determine the best prior bias we can choose for this problem learning task. Because we are attempting problem learning, it is clear we do not know much about A or B. Nevertheless, as our goal is to learn B,

it makes sense to choose our initial knowledge about B as the prior bias. It is especially important to ignore domain knowledge about A and focus on the differences between A and B in this prior bias.

Next, we update this bias by learning about the problem class A. If we have a good prior bias that suppresses the aspects of A that is not present in B, then as we learn about A, the prior bias will slow us down from learning too much about A that is not relevant to B. However, it is clear that if we spend enough time learning about A, we can over-learn it, making our original prior bias irrelevant. As a result, we will end up with complete domain knowledge about A, losing any knowledge we had about B. The differences between A and B form inaccurate domain knowledge about B and consequently can result in "negative transfer", which is the situation where transfer learning leads to lower performance than with no transfer learning. As concluded by Pan et al. [9], negative learning is an open problem in transfer learning.

Thus, we suggest an alternate strategy. Start with prior bias about A and problem learn about A as much as possible resulting in plenty of domain knowledge about A. But instead of using this domain knowledge directly, we first "water it down" to reflect the fact that we are transfer learning. Then, we update this watered down bias using our prior bias about B. This of course requires that we use an algorithm with a domain knowledge representation that allows these operations. OEFF allows them, and we can demonstrate these operations as follows. Let Θ_B be our prior bias about B and Θ_A be the domain knowledge that we have learned about A and c be a positive real value that

reflects our distrust of the domain knowledge we have gathered about A. Thus, we can compute Θ'_B, the final bias about B as follows.

$$\forall F \in \mathcal{E}, \Theta'_B(F) = \textsc{Normalize}\left((\Theta_A(F) + c) \cdot \Theta_B(F)\right) \tag{6.10}$$

This approach avoids over-learning A as we explicitly reduce the emphasis on the domain knowledge of A. Furthermore, we need not worry about losing what we know about B as it is added only in the last step. If equation 6.10 is approximated properly with an adequately conservative c, the theory we have developed tells us that we can assuredly avoid any negative learning.

CHAPTER 7
CONCLUSION

The blackbox framework is a formalization of optimization in a scenario where the information available to optimization algorithm is partially restricted. As per the modern view of blackbox optimization, optimization algorithms have access to the broad class of objective functions from which the problem instance is drawn. In this dissertation, we examine the blackbox scenario, focus on blackbox performance in combinatorial blackbox optimization, and ask how we can make the best of the blackbox restriction. We present theory that answers some of these questions and discuss its implications on the blackbox framework and practical optimization algorithms.

Popular blackbox optimization approaches such as evolutionary algorithms encode information about the nature of the objective/fitness functions it may be asked to solve, aiming to improve performance over those problems. These approaches use objective values from the functions, often only for ranking, in a way that helps find the solution. We argue for a different approach, where we sample the search space to identify the objective function in the problem class and use its identity to determine the solution.

This function identification is done by eliminating other fitness functions, and we call this the *Elimination of Fitness Functions* approach.

We prove that an algorithm called Optimal Elimination of Fitness Functions, based on EFF has optimal blackbox performance, a popular performance metric for the blackbox framework that counts samples of the fitness function. We also present a sampling strategy that does a thorough search of potential samples before making an optimal sample for OEFF. OEFF is also optimal from a problem abstraction point of view.

This maximizing of blackbox performance by OEFF makes it a natural candidate for proving performance bounds. We explain why this can be done and demonstrate it on the Generalized ONEMAX problem class. We bound the performance of OEFF using a random sample sequence to $O(\frac{n}{\log n})$, which was original research within the community and tightens the previously proven bound of $\Omega(\frac{n}{\log n})$. We also show how we can prove lower bounds by providing an alternate proof for the No Free Lunch theorem.

Another aspect of OEFF is that as a result of having to process its many parameters for each sample, it is inefficient from a space and time complexity point of view. We argue that this is inevitable due to OEFF being optimal as well as having a large degree of problem abstraction. In fact, we conclude that any algorithm can maximize only two out of the three key characteristics of optimization algorithms viz. performance, efficiency and problem abstraction.

We show that we can trade-off problem abstraction for efficiency without sacrificing performance by modeling the behavior of OEFF over a problem class. For this modeling process, we develop formal techniques that reduce states and exploit symmetry on the game-tree induced by OEFF when operating over a specific problem class. We apply this technique to derive an optimal algorithm for the Generalized LEADINGONES problem class. We chose this problem class because it is well known and its optimal strategy is easy to understand. Its game-tree is also very suitable for the modeling process.

We then examine a further generalization of Generalized LEADINGONES called LO^{**} that has infinite problem instances. Despite the fact that OEFF operates over only a finite problem class, we were able apply our technique to derive the optimal algorithm for LO^{**} and tighten existing bounds. We build on this result and tighten known bounds for another generalization called LO^{***}. We briefly discuss the LONGPATH problem class and explain that this singleton class which is known to be $O(n^3)$ for a (1+1)EA is trivial for OEFF. Thus, we develop the STEEPPATH problem class that has the opposite property — over it, a (1+1)EA is no worse than OEFF. We then present the OEFF modeling process over STEEPPATH problem class and derive the performance bounds.

Having understood how the performance of algorithms is impacted by the black-box framework, we approach optimization algorithms with a fresh perspective. We infer from common practice that when it comes to optimization algorithms, practitioners care about problem abstraction, performance and efficiency in that order. Our attempts to create optimization algorithms should take full advantage of the freedoms allowed us by

the practical scenario or context to improve these metrics. We find that while we can use the blackbox framework for analytical purposes, it is unsuitable as an algorithm design guideline as some of its restrictions are not warranted in many of the practical scenarios that we face. We argue that the blackbox restriction artificially limits performance and is not required to confer problem interface unification to blackbox algorithms. We also suggest that the framework fails to account for the form in which the knowledge about the problem class is made available. Only OEFF can directly use the domain knowledge in the form assumed by the blackbox framework. Practical algorithms are as limited by their inability to transform the domain knowledge to a suitable form as they are by the lack of domain knowledge. Lastly, we stress the importance of the fact that majority of domain knowledge in a practical algorithm is transformed and placed there by a practitioner.

We propose problem learning as a critical approach to improving problem abstraction and performance at the cost of efficiency. Though limited problem learning has been attempted often, a theoretical understanding of problem learning in optimization is lacking. We present theory where we extend the OEFF algorithm to handle incomplete domain knowledge. We also derive the formulas we can use to update this domain knowledge based on experience. This general theory for problem learning is used to make some suggestions for doing problem learning in practice.

We use this theory to conclude that it is unrealistic to start with no domain knowledge and depend on problem learning alone to acquire sufficient domain knowledge for reasonable performance. We prove that barring some practically rare situations, it is im-

possible to predict the benefit of problem learning. We speculate that due to the relatively knowledge-light nature of optimization algorithms, there should nevertheless often be significant performance benefit.

We apply this theory to practical optimization algorithms and make some recommendations. We suggest ignoring the blackbox restriction by designing algorithms that can accept instance knowledge through sources other than the objective functions. These algorithms should be made capable of problem learning as well as accepting domain knowledge from the practitioner to better understand these sources. We highly recommend that domain knowledge from practitioners be accepted in a form that favors the practitioner instead of the algorithm. Relative to the theoretical ideal, we would have to restrict the scope of problem learning with the intention of reducing overhead to achieve a minimum practical efficiency. However, increasing the scope of problem learning confers problem abstraction, a priority for most practitioners. If possible, we recommend an information maximizing sample selection strategy. Finally, the optimization algorithm should approximate the problem modeling approach of OEFF by using the available domain knowledge to do partial problem identification as well as problem solving.

We are also able to apply the problem learning theory to the transfer learning scenario where we are attempting to learn a target task by learning shared aspects from a cheaper related task. Though we are attempting to learn a related task, we should have some knowledge about the target task. We suggest that to avoid negative transfer, we should "water down" the domain knowledge learned from the related task and update

it with this prior knowledge. We present a formula for this update that should be approximated in practical algorithms and argue that this formula, properly applied, avoids negative transfer in transfer learning.

7.0.4 Future Research

As future research, we can extend the research presented here in the following directions.

1. *Use our theory to derive more theoretical bounds.* These results may include new blackbox complexity bounds based on OEFF performance bounds. Many problem classes including infinite problem classes may be attempted. We expect many opportunities for improving upper bounds of blackbox complexity.

2. *Develop new theory to predict the utility of problem learning in a particular context.* We showed how the benefit of problem learning can range from significant to very little. Developing theory to predict the benefit can help practitioners make informed decisions about using problem learning.

3. *Use our theory to derive additional practical algorithms.* We presented a theoretical approach to modeling OEFF over specific problem classes. This theory can be used to derive practical algorithms for other problem classes, extending our understanding

of solving them. In addition, we can also identify new patterns in the game-tree that we can use to simplify it.

4. *Use our practical recommendations to create frameworks for algorithm design.* We can also take advantage of the practical recommendations as well as the presented theory to develop a framework for designing practical optimization algorithms or learning them using problem learning. This approach should be backed by theory that justifies problem learning from the knowledge sources that are used.

LIST OF REFERENCES

[1] H. Kargupta and D. E. Goldberg, "Search, blackbox optimization, and sample complexity," in *Foundations of Genetic Algorithms IV*, pp. 291–324, Morgan Kaufmann, 1997.

[2] G. Wang, E. Goodman, and W. Punch, "Toward the optimization of a class of black box optimization algorithms," in *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on*, pp. 348 –356, nov 1997.

[3] M. Mongeau, H. Karsenty, V. Rouz, and J.-B. Hiriart-Urruty, "Comparison of public-domain software for black box global optimization," tech. rep., Laboratoire Approximation et Optimisation, 1998.

[4] S. Droste, T. Jansen, and I. Tinnefeld, K. Wegener, "A new framework for the valuation of algorithms for black-box optimization," in *Foundations of Genetic Algorithms VII*, pp. 197–214, Morgan Kaufmann, 2002.

[5] I. Wegener, *Complexity Theorey: Exploring the Limits of Efficient Algorithms*. Springer, 2005.

[6] J. E. Rowe and M. D. Vose, "Unbiased black box search algorithms," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, (New York, NY, USA), pp. 2035–2042, ACM, 2011.

[7] C. Winzen, *Toward a Complexity Theory for Randomized Search Heuristics: Black-Box Models*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2011.

[8] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," Tech. Rep. NOTTCS-TR-SUB-0906241418-274, School of Computer Science and Information Technology, University of Nottingham, February 2010.

[9] S. J. Pan and Q. Yang, "A survey on transfer learning," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, pp. 1345 –1359, oct. 2010.

[10] P. Erdős and A. Rényi, "On two problems of information theory," *Magyar Tud. Akad. Mat. Kutató Int. Közl*, vol. 8, pp. 229–243, 1963.

[11] S. Droste, T. Jansen, and I. Wegener, "Upper and lower bounds for randomized search heuristics in black-box optimization," *Theor. Comp. Sys.*, vol. 39, no. 4, pp. 525–544, 2006.

[12] B. Doerr, D. Johannsen, T. Kötzing, P. K. Lehre, M. Wagner, and C. Winzen, "Faster black-box algorithms through higher arity operators," in *Proceedings of the 11th workshop proceedings on Foundations of genetic algorithms*, FOGA '11, (New York, NY, USA), pp. 163–172, ACM, 2011.

[13] B. Doerr and C. Winzen, "Memory-restricted black-box complexity of onemax," *Information Processing Letters*, vol. 112, no. 12, pp. 32 – 34, 2012.

[14] P. K. Lehre and C. Witt, "Black-box search by unbiased variation," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, (New York, NY, USA), pp. 1441–1448, ACM, 2010.

[15] B. Doerr, J. Lengler, T. Kötzing, and C. Winzen, "Black-box complexities of combinatorial problems," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, (New York, NY, USA), pp. 981–988, ACM, 2011.

[16] S. Luke, *Essentials of Metaheuristics*. Lulu, 2009. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[17] K. A. De Jong, *Evolutionary Computation: A Unified Approach*. MIT Press, 2006.

[18] G. Anil and R. P. Wiegand, "Black-box search by elimination of fitness functions," in *FOGA '09: Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms*, (New York, NY, USA), pp. 67–78, ACM, 2009.

[19] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 1, pp. 67–82, apr 1997.

[20] S. Droste, T. Jansen, and I. Wegener, "Perhaps not a free lunch, but at least a free appetizer," in *Proceedings for the 1999 Genetic and Evolutionary Computation Conference*, pp. 833–839, 1999.

[21] E. Zitzler, M. Laumanns, and S. Bleuler, "A tutorial on evolutionary multiobjective optimization," in *Metaheuristics for Multiobjective Optimisation* (X. Gandibleux, M. Sevaux, K. Srensen, and V. T'kindt, eds.), vol. 535 of *Lecture Notes in Economics and Mathematical Systems*, pp. 3–37, Springer Berlin Heidelberg, 2004.

[22] J. Knowles, R. Watson, and D. Corne, "Reducing local optima in single-objective problems by multi-objectivization," in *Evolutionary Multi-Criterion Optimization* (E. Zitzler, L. Thiele, K. Deb, C. Coello Coello, and D. Corne, eds.), vol. 1993 of *Lecture Notes in Computer Science*, pp. 269–283, Springer Berlin / Heidelberg, 2001.

[23] D. Brockhoff, T. Friedrich, N. Hebbinghaus, C. Klein, F. Neumann, and E. Zitzler, "Do additional objectives make a problem harder?," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, (New York, NY, USA), pp. 765–772, ACM, 2007.

[24] M. T. Jensen, "Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation," *Journal of Mathematical Modelling and Algorithms*, vol. 3, pp. 323–347, 2004.

[25] J. Lehman and K. Stanley, "Exploiting open-endedness to solve problems through the search for novelty," *Artificial Life*, vol. 11, p. 329, 2008.

[26] J. Lehman and K. Stanley, "Evolving a diversity of creatures through novelty search and local competition," *Evolutionary Computation*, no. Gecco, pp. 211–218, 2011.

[27] J. Lehman and K. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evolutionary Computation*, vol. 19, no. 2, pp. 189–223, 2011.

[28] J. Doucette and M. Heywood, "Novelty-based fitness: An evaluation under the santa fe trail," in *Genetic Programming* (A. Esparcia-Alczar, A. Ekrt, S. Silva, S. Dignum, and A. Uyar, eds.), vol. 6021 of *Lecture Notes in Computer Science*, pp. 50–61, Springer Berlin / Heidelberg, 2010.

[29] C. Ollion and S. Doncieux, "Why and how to measure exploration in behavioral space," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, (New York, NY, USA), pp. 267–274, ACM, 2011.

[30] M. Hauschild and M. Pelikan, "An introduction and survey of estimation of distribution algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 3, pp. 111 – 128, 2011.

[31] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[32] L. Barnett, "Ruggedness and neutrality: the nkp family of fitness landscapes," in *Proceedings of the sixth international conference on Artificial life*, ALIFE, (Cambridge, MA, USA), pp. 18–27, MIT Press, 1998.

[33] V. K. Vassilev, T. C. Fogarty, and J. F. Miller, "Advances in evolutionary computing," ch. Smoothness, ruggedness and neutrality of fitness landscapes: from theory to application, pp. 3–44, New York, NY, USA: Springer-Verlag New York, Inc., 2003.

[34] I. Harvey and A. Thompson, "Through the labyrinth evolution finds a way: A silicon ridge," in *Proceedings of the First International Conference on Evolvable Systems: From Biology to Hardware*, ICES '96, (London, UK, UK), pp. 406–422, Springer-Verlag, 1996.

[35] E. G. López and K. Rodríguez-Vázquez, "The importance of neutral mutations in GP," in *PPSN* (T. P. Runarsson, H.-G. Beyer, E. K. Burke, J. J. M. Guervós, L. D. Whitley, and X. Yao, eds.), vol. 4193 of *Lecture Notes in Computer Science*, pp. 870–879, Springer, 2006.

[36] M. Huynen, P. Stadler, and W. Fontana, "Smoothness with ruggedness: the role of neutrality in adaptation," in *Proceedings of the National Academy of Sciences*, vol. 93, pp. 397–401, 1996.

[37] M. Kimura, "Evolutionary rate at the molecular level.," *Nature*, vol. 217, no. 129, pp. 624–626, 1968.

[38] E. van Nimwegen, J. Crutchfield, and M. Huynen, "Neutral evolution of mutational robustness," *Proceedings of the National Academy of Sciences*, vol. 96, pp. 9716–9720, 1999.

[39] J. Reisinger and R. Miikkulainen, "Acquiring evolvability through adaptive representations," in *GECCO-2007: Proceedings of the 9th annual conference on Genetic and evolutionary Computation*, (New York), ACM Press, 2007.

[40] A. P. Wagner, *Robustness and evolvability in living systems*. Princeton university press, 2005.

[41] J. S. Edwards and B. O. Palsson, "Robustness analysis of the *Escherichia coli* metabolic network," *Biotechnology Progress*, vol. 16, pp. 927–939, 2000.

[42] D. Krakauer and J. Plotkin, "Redundancy, antiredundancy and the robustness of genomes," *Proceedings of the National Academy of Sciences*, vol. 99, pp. 1405–1409, 2002.

[43] D. Krakauer and J. Plotkin, "Robustness in biological systems: A provisional taxonomy.," in *Robust design: A repertiore for biology, ecology and engineering* (E. Jen, ed.), Oxford University Press, NY, 2005.

[44] A. Wagner, "Distributed robustness versus redundancy as causes of mutational robustness," *BioEssays*, vol. 27, pp. 176–188, 2005.

[45] D. A. MacDónaill, "A parity code interpretation of nucleotide alphabet composition," *Chemical Communications*, vol. 18, pp. 2062–2063, 2002.

[46] B. Rost, "Protein structures sustain evolutionary drift," *Folding and Design*, vol. 2, pp. S19–S24, 2002.

[47] R. Hinterding, Z. Michalewicz, and A. Eiben, "Adaptation in evolutionary computation: a survey," in *Evolutionary Computation, 1997., IEEE International Conference on*, pp. 65 –69, apr 1997.

[48] L. Altenberg, "Emergent phenomena in genetic programming," in *Evolutionary Programming — Proceedings of the Third Annual Conference* (A. V. Sebald and L. J. Fogel, eds.), (San Diego, CA, USA), pp. 233–241, World Scientific Publishing, 24-26 Feb. 1994.

[49] J. J. Grefenstette and C. L. Ramsey, "An approach to anytime learning," in *Proceedings of the ninth international workshop on Machine learning*, ML92, (San Francisco, CA, USA), pp. 189–195, Morgan Kaufmann Publishers Inc., 1992.

[50] A. C. Schultz and J. J. Grefenstette, "Continuous and embedded learning in autonomous vehicles - adapting to sensor failures," in *Unmanned ground vehicle technology II; Proceedings of the Conference*, (Orlando, FL), pp. 55–62, Apr 2000.

[51] C. L. Ramsey and J. J. Grefenstette, "Case-based initialization of genetic algorithms," in *Proceedings of the 5th International Conference on Genetic Algorithms*, (San Francisco, CA, USA), pp. 84–91, Morgan Kaufmann Publishers Inc., 1993.

[52] S. J. Louis and J. Johnson, "Solving similar problems using genetic algorithms and case-based memory," pp. 283–290, 1997.

[53] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward, "Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, (New York, NY, USA), pp. 1559–1565, ACM, 2007.

[54] H. Terashima-Marin, C. J. Farias Zarate, P. Ross, and M. Valenzuela-Rendon, "Comparing two models to generate hyper-heuristics for the 2d-regular bin-packing problem," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, (New York, NY, USA), pp. 2182–2189, ACM, 2007.

[55] S. Droste, T. Jansen, and I. Wegener, "On the analysis of the (1+1) evolutionary algorithm," *Theoretical Computer Science*, vol. 276, pp. 51–81, 2002.

[56] C. Schumacher, M. D. Vose, and L. D. Whitley, "The no free lunch and problem description length," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001*, pp. 565–570, Morgan Kaufmann, 2001.

[57] J. Horn, D. E. Goldberg, and K. Deb, "Long path problems," in *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*, PPSN III, (London, UK, UK), pp. 149–158, Springer-Verlag, 1994.

[58] G. Rudolph, "How mutation and selection solve long-path problems in polynomial expected time," *Evol. Comput.*, vol. 4, pp. 195–205, June 1996.

[59] M. Pelikan, D. E. Goldberg, and F. G. Lobo, "A survey of optimization by building and using probabilistic models," *Computational Optimization and Applications*, vol. 21, pp. 5–20, 2002. 10.1023/A:1013500812258.

[60] J. Baxter, "Theoretical models of learning to learn," in *T. Mitchell and S. Thrun (Eds.), Learning*, pp. 71–94, Kluwer, 1997.