University Library



A gateway to Melbourne's research publications

Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
Alpcan, T;Everitt, T;Hutter, M

Title:
Can we measure the difficulty of an optimization problem?

Date:
2014-12-01

Citation:
Alpcan, T., Everitt, T. & Hutter, M. (2014). Can we measure the difficulty of an optimization problem?. 2014 IEEE Information Theory Workshop Itw 2014, pp.356-360. IEEE. https://doi.org/10.1109/ITW.2014.6970853.

Persistent Link:
https://hdl.handle.net/11343/241549

Can we measure the difficulty of an optimization problem?

Tansu Alpcan
Dept. of Electrical and Electronic Engineering
The University of Melbourne, Australia
Email: tansu.alpcan@unimelb.edu.au

Tom Everitt
Department of Mathematics
Stockholm University
Email: everitt@math.su.se

Marcus Hutter
Research School of Computer Science
Australian National University
Email: marcus.hutter@anu.edu.au

***Abstract*—Can we measure the difficulty of an optimization problem? Although optimization plays a crucial role in modern science and technology, a formal framework that puts problems and solution algorithms into a broader context has not been established. This paper presents a conceptual approach which gives a positive answer to the question for a broad class of optimization problems. Adopting an information and computational perspective, the proposed framework builds upon Shannon and algorithmic information theories. As a starting point, a concrete model and definition of optimization problems is provided. Then, a formal definition of optimization difficulty is introduced which builds upon algorithmic information theory. Following an initial analysis, lower and upper bounds on optimization difficulty are established. One of the upper-bounds is closely related to Shannon information theory and black-box optimization. Finally, various computational issues and future research directions are discussed.**

I. Introduction

Considering the broad applicability of optimization as a discipline, it is not surprising that the difficulty of optimization problems has been investigated before, e.g. [1], [2]. Likewise, the (No) Free Lunch Theorems which explore the connection between effective optimization algorithms and the problems they solve have generated a lot of interest in the research community [3]–[5].

While the work [1] has presented an excellent overview of various aspects which make an optimization problem "difficult", it has not presented a unifying conceptual framework for measuring "difficulty". The complexity of optimization problems has been discussed in [6]. The discussion on (No) Free Lunch Theorems [3]–[5], [7] is more focused on performance of algorithms for various problems rather than analyzing difficulty. Moreover, Free Lunch Theorems focus on black-box optimization problems where the objective function is not known to the optimizer unlike their counterpart *"open-box" problems considered in this paper where the objective function is known and its properties are actively used to find the solution*. The paper [4] has partly discussed complexity-based ranking of problems following an algorithmic information approach similar to the one proposed here, however, does not aim to create a broader framework. The paper [8] has proposed the use of instance and Kolmogorov complexities as an estimator of difficulty for black-box optimization problems.

The goal of this paper is to develop a framework for measuring difficulty of optimization problems. The conceptual framework introduced builds upon Shannon and Algorithmic Information Theories [9], [10] and establishes a link between probabilistic and algorithmic approaches to optimization. *A distinctive feature of the proposed framework is the explicit focus on algorithmic information for measuring (open-box) optimization difficulty.*

The outline of the paper is as follows. The next section presents the preliminary definitions and the adopted model. Section III contains the main definitions and results, including upper and lower bounds on the optimization difficulty measure introduced. The paper concludes with a discussion on future research directions in Section IV.

II. Definitions and Model

This paper studies mathematical optimization problems [11] that are commonly expressed as

$$\max_x f(x) \text{ subject to } g_i(x) \leq 0,\ i = 1, \ldots, m, \qquad (1)$$

where the n-dimensional real-valued vector $x \in \mathbb{R}^n$ is the decision variable, the function f is the objective function and g_i are the constraint functions. The list of constraints, denoted by c, define the solution space $\mathcal{A}$ which is a assumed to be a *compact* subset of the n-dimensional real space, $\mathcal{A} \subset \mathbb{R}^n$.

As a starting point for developing an algorithmic and information-theoretic characterization of optimization difficulty, it is assumed here that the function f is *Lipschitz-continuous* on $\mathcal{A}$ and the optimization problem (1) admits a feasible *global solution* $x^* = \arg\max_{x \in \mathcal{A}} f(x)$.

For a given scalar $\varepsilon > 0$ and compact set $\mathcal{A}$, let $\mathcal{A}(\varepsilon)$ be an *ε-discretization of $\mathcal{A}$* constructed using the following procedure. Let $\mathcal{C}$ be a finite covering of $\mathcal{A}$ with hypercubes of side length at most ε. For each cube $C \in \mathcal{C}$, let $x_C \in C \cap A$. Finally, let $\mathcal{A}(\varepsilon)$ be the set of all x_C, $C \in \mathcal{C}$. Thus, $\mathcal{A}(\varepsilon)$ is a finite subset of $\mathcal{A}$, with the same cardinality as $\mathcal{C}$. If the cubes in $\mathcal{C}$ do not overlap, we call discretization based on $\mathcal{C}$ *non-overlapping*.

To allow for a computational treatment of optimization problems, an encoding of problems as (binary) strings must be chosen. The "standard calculus symbols" we base these descriptions on are: finite precision real numbers $0,\ 1.354, \ldots$; variables $x_1, x_2, \ldots$; elementary functions $+, \cdot, \exp, \ldots$; parenthesis; relations $\leq, =, \ldots$. A *function* $\mathbb{R}^n \to \mathbb{R}$ is an expression formed by elementary functions, real

numbers and the variables $x_1, \ldots, x_n$, and a *constraint on* $\mathbb{R}^n$ is a formula on the form $g(x_1, \ldots, x_n) \leq 0$ with $g : \mathbb{R}^n \to \mathbb{R}$. Binary representations of functions and constraints may then be reached in a straightforward manner by giving each symbol a binary encoding (e.g. ASCII). Through concatenation of the symbol encodings, functions and constraints receive natural binary encodings as well. If e is an expression, let $\ell(e)$ denote the length of its binary encoding.

Based on the model and assumptions introduced, a formal definition of the optimization problem (1) is provided next.

Definition II.1 (Optimization problem)**.** A *(discretizable) optimization problem* on $\mathbb{R}^n$ is a tuple $\langle f, c, \varepsilon \rangle$ where $f : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*, c is a list of constraints on $\mathbb{R}^n$ expressed using functional (in)equalities, and $\varepsilon > 0$ is a *discretization parameter*.

The constraints delineate the *solution or search space* $\mathcal{A}$, over which f is to be optimized. For simplicity, assume that $\mathcal{A}$ is non-empty and compact and that f is Lipschitz-continuous over $\mathcal{A}$. Let $\mathcal{A}(\varepsilon)$ be an ε-discretization of $\mathcal{A}$.

An *argmax* of f on $\mathcal{A}$ is a point $x^* \in \mathcal{A}$ satisfying $\forall x \in \mathcal{A} : f(x) \leq f(x^*)$. A *discrete argmax* (or δ_ε-argmax) is a point $\hat{x}$ satisfying $\forall x \in A : f(x) \leq f(\hat{x}) + \delta_\varepsilon$ where $\delta_\varepsilon = \max\{|f(x) - f(\hat{x})| : \|x - \hat{x}\| < \varepsilon\}$ and $\|\cdot\|$ is the maximum norm.

Discretized and Approximate Solutions

The definition above accepts δ_ε-argmax (rather than true argmax) as a solution to the constrained optimization problem. The discretization parameter ε then effectively states how close the discrete argmax needs to be to the true argmax. The somewhat involved definition of δ_ε-argmax also allows points further than ε away from the correct answer, as long as those points differ less in their target value than some point within ε of the argmax. Under the adopted Lipschitz-continuity assumption on objective functions, if the Lipschitz constant is k, then the desired solution differs at most δ in target value from the optimum, if one chooses $\varepsilon = \delta/k$.

The following is a sufficient condition for a point $\hat{x}$ in the discrete search space $\mathcal{A}(\varepsilon)$ being a discrete argmax: If it holds for all $x \in \mathcal{A}(\varepsilon)$ that $f(x) \leq f(\hat{x})$, then $\hat{x}$ must be a δ_ε-argmax.

Next, a formal definition of non-trivial problems is provided. Non-trivial problems will be the focus of this paper.

Definition II.2 (Non-trivial problems)**.** A problem $\langle f, c, \varepsilon \rangle$ is *non-trivial* if every ε-discretization $\mathcal{A}(\varepsilon)$ of $\mathcal{A}$ contains at least one point that is not a δ_ε-argmax.

As a remark on approximation and discretization of solutions, it is worth noting that all numerical optimization software packages yield only a discretized and approximate solution.

Correctness of Solutions

To ensure correctness, any alleged δ_ε-argmax solution is paired with a polynomially verifiable certificate s of the the correctness. In general, the trace (step-by-step reporting) of a correct optimization algorithm forms one example of a (linearly verifiable) certificate of the returned argmax. To verify such a certificate, it suffices to check that each step of the trace corresponds to the definition of the algorithm, and that the final step of the trace outputs the proposed argmax. A general type of certificates (not specific to a particular class or optimization algorithm) may for example be based on formal proofs in first-order logic or type-theory [12]. Indeed, many automated theorem proving systems have developed formalizations of analysis [13], which could potentially form the basis of a suitable proof system.

Definition II.3 (Verifiable Solutions)**.** Consider the optimization problem in Definition II.1, and a suitable proof system $\mathcal{T}$ offering polynomially verifiable certificates s of candidate solutions. A *solution of the optimization problem* $\langle f, c, \varepsilon \rangle$ is defined as a pair $\langle x^*, s \rangle$ where s is a certificate in $\mathcal{T}$ that x^* is a δ_ε-argmax for $\langle f, c, \varepsilon \rangle$.

It is beyond the scope of this paper to describe a suitable proof system $\mathcal{T}$ in detail. The paper will instead rely on semi-formal proof sketches in examples, and polynomial verifiability in abstract arguments. For concreteness, we will assume that certificates in $\mathcal{T}$ can be verified in time dn^q. That is, we assume the existence of a verifier for certificates in $\mathcal{T}$ with runtime at most dn^q for certificates of length n.

Illustrative Example

Consider the optimization problem

$$\langle f(x) = 3x^2 + x; x \geq -1, x \leq 1; \varepsilon = 0.001 \rangle . \qquad (2)$$

A solution is δ_ε-$\operatorname{argmax} = 1$. The following certificate sketch establishes the correctness of the solution. Note that the certificate below is not formal. However, a suitable formal system should yield certificates of similar size.

1) $df/dx = 6x + 1$ (derivative)
2) $6x+1 = 0 \iff x = -1/6$ (properties of real numbers)
3) $\text{roots}(df/dx) = \{-1/6\}$ (from 1 and 2)
4) $\text{boundary} = \{-1, 1\}$ (from c)
5) $x \notin \text{roots}(df/dx) \wedge x \notin \text{boundary}(c) \implies \neg \operatorname{argmax}(x)$ (calculus)
6) $\operatorname{argmax} = -1/6 \vee \operatorname{argmax} = -1 \vee \operatorname{argmax} = 1$ (from 3–5)
7) $f(-1) \leq f(1) \implies \operatorname{argmax} \neq -1$
8) $f(-1/6) \leq f(1) \implies \operatorname{argmax} \neq -1/6$
9) $\operatorname{argmax} = 1$ (from 6–8)
10) δ_ε-$\operatorname{argmax} = 1$ (from 9)

The relative shortness of the certificate indicates that the solution to the problem is indeed quite simple. Proposition III.4 makes this claim more formal.

III. Optimization Difficulty

The model introduced in the previous section helps formalizing the broad research question investigated in this paper, which is "*how to measure the difficulty of an optimization problem?*"

The **proposed solution concept** for measuring the difficulty of an optimization problem is formulating it as a "knowledge or "information" problem. Before solving $\langle f, c, \varepsilon \rangle$ it is only known that the solution has to be in the search domain, $\mathcal{A}$, which itself may require considerable work to identify. Solving the optimization problem yields *knowledge* about the location of x^* up to a certain precision. Hence, it is natural to establish a link between "solving an optimization problem" and the "knowledge obtained about the solution location". In other words, **solving an optimization problem is equivalent to obtaining knowledge about the location of the solution**.

If the problem $\langle f, c, \varepsilon \rangle$ is "simple" then solving it corresponds to discovering a small amount of knowledge. Likewise, a difficult problem means a lot of knowledge is produced in solving it. Thus, we will argue that **the difficulty of an optimization can be quantified using concepts from Shannon and algorithmic information theories** which provide a strong mathematical foundation of information [9], [10].

Definition III.1 (Optimization Algorithm)**.** An algorithm p *solves the optimization problem* $\langle f, c, \varepsilon \rangle$ if $p(\langle f, c, \varepsilon \rangle) = \langle x^*, s \rangle$ with s a certificate that x^* is a δ_ε- argmax of f on $\mathcal{A}$. That is, p should output a solution $\langle x^*, s \rangle$ when fed $\langle f, c, \varepsilon \rangle$ as input.

With U a universal Turing-machine (aka programming language), let the *description length* $\ell_U(p)$ be the length of the binary string-encoding of p on U, and let the *runtime* $t_U(p(\langle f, c, \varepsilon \rangle))$ be the number of time steps it takes for p to halt on input $\langle f, c, \varepsilon \rangle$ on U.

Deep invariance theorems [10, p. 578] show that the choice of U is inessential. Subsequently, assume that some universal Turing-machine U has been chosen as a *reference machine*, and simply write $\ell(p)$ and $t(p)$ to quantify description length and runtime of the algorithm p, as is custom in Algorithmic Information Theory.

Building upon this definition, a main contribution of the paper is presented next.

Definition III.2 (Optimization difficulty)**.** The *optimization difficulty* of a given optimization problem $\langle f, c, \varepsilon \rangle$ characterized in Definition II.1, is defined as

$$D_{\text{opt}}(\langle f, c, \varepsilon \rangle) := \min_p \{\ell(p) + \log_2(t(p)) : p \text{ solves } \langle f, c, \varepsilon \rangle\}$$

The optimization difficulty defined in III.2 refers to *instances* of optimization problems rather than classes. Classes is the standard object when analyzing difficulty in for example computational complexity theory [14], but an instance version has also been considered [15]. Considering the difficulty of instances has several advantages. Firstly, instances are what we ultimately need to solve in practice. Secondly, instance-difficulty naturally generalizes to class-difficulty, through average- or worst-case definitions. Thirdly, using instances allows us to formalize the information-theoretic intuition that the difficulty of optimization problems corresponds to how much knowledge is gained by finding the maximum, which turns out to be hard to formulate from a class-oriented perspective.

One challenge in considering the difficulty of instances rather than classes is how to take into account special algorithms that are well-tailored to problem instances. For example, a correct argmax to the problem in Example 1 is returned by the algorithm `print 1`. For this reason, it would be misleading to identify the difficulty of a problem with the length or runtime of the shortest algorithm returning a solution. This issue is addressed by using *verifiable solutions* (Definition II.3). In order to qualify as a proper solution algorithm, the program should include enough information (a certificate) to verify the correctness of its output.

The two terms $\ell(p)$ and $\log_2(t(p(\langle f, c, \varepsilon \rangle)))$ in D_{opt} deserve a remark. They represent a tension between generally applicable solvers on the one hand, and highly specialized ones on the other. A very general solver is *exhaustive search*; a very specialized one is of the form `print` $\langle x^*, s \rangle$, with $\langle x^*, s \rangle$ being a verifiable solution. For most problems there will also be a range of solvers of increasing specialization in between the two extremes. From our perspective, optimization difficulty is closely related to the *knowledge required to find the maximum*. The general kind of solver starts with a small amount of knowledge about the problem and typically pays a price in longer runtime, whereas the specialized kind already has information about the problem encoded in its source code, and thus needs less computation time to find the (verifiable) solution. Symmetries between the "runtime-knowledge" and the "source code"-knowledge are discussed in connection to the upper bounds below. These symmetries justify the combination of description length and runtime used in D_{opt}.[1]

A. Bounds on Optimization Difficulty

Optimization is closely related to "search". A given constrained optimization problem with a finite search space $\mathcal{A}(\varepsilon)$ can always be solved as a "search problem" by ignoring the properties of the objective function f. This coincidentally reduces the problem to well-known black-box optimization where the objective function is unknown.

Assuming lack of any a priori knowledge (i.e., a uniform prior over argmax-locations in $\mathcal{A}(\varepsilon)$), the argmax of the function could be in any location $\mathcal{A}(\varepsilon)$ with equal probability. Therefore, once x^* is found, the amount of a posteriori knowledge obtained is $\log_2(|\mathcal{A}(\varepsilon)|)$ bits from Shannon information theory [9], where $|\cdot|$ denotes cardinality of a set. This immediately follows from the definition of discrete entropy:

$$\sum_{x \in \mathcal{A}} \frac{1}{|\mathcal{A}|} \log_2(|\mathcal{A}|) = \log_2(|\mathcal{A}|).$$

This information measure provides a quantitative way of comparing such search problems, and provides a fundamental upper bound on the optimization difficulty.

[1] Readers familiar with Algorithmic Information Theory may note that D_{opt} is tightly related to Levin's Kt-complexity. Indeed, D_{opt} may equivalently be defined as $Kt(\text{solution}|\text{problem})$.

Proposition III.3 (Upper Bound 1)**.** *There is a computational constant $k \in \mathbb{N}$ such that for any optimization problem $\langle f, c, \varepsilon\rangle$ with ε-discretization $\mathcal{A}(\varepsilon)$,*

$$D_{\text{opt}}(\langle f, c, \varepsilon\rangle) \leq k + \log_2(|\mathcal{A}(\varepsilon)| + C)\,,$$

where C is the runtime cost of obtaining the discretization $\mathcal{A}(\varepsilon)$, and $|\cdot|$ denotes cardinality.

Proof: Let p be an optimization algorithm that starts with obtaining $\mathcal{A}(\varepsilon)$ from c, and then searches $\mathcal{A}(\varepsilon)$ exhaustively (finding a δ_ε-argmax $\hat{x}$). Thereafter p prints a proof starting with an establishment of $\mathcal{A}(\varepsilon)$, followed by a proof of the exhaustive search. The proof will be of the following form (with $m = |\mathcal{A}(\varepsilon)|$ and $x^1, \ldots, x^m$ an enumeration of the elements of $\mathcal{A}(\varepsilon)$):

1.	
$\vdots$	Establish $\mathcal{A}(\varepsilon)$
n.	ε-discret $\left(x^1, \ldots, x^m, \mathcal{A}\right)$ (from 1 to $n-1$)
$n+1$.	$f(x^1) \leq f(\hat{x})$
$\vdots$	
$n+m$.	$f(x^m) \leq f(\hat{x})$
$n+m+1$.	δ_ε-$\operatorname{argmax}(\hat{x})$ (from n to $n+m$)

The contributions to p's runtime are: (1) find (and prove) $\mathcal{A}(\varepsilon)$ to a cost of C, (2) search $\mathcal{A}(\varepsilon)$ exhaustively to a cost linear in $|\mathcal{A}(\varepsilon)|$, (3) print the final part of the proof of the exhaustive search. In total, this yields a running time of $k|\mathcal{A}(\varepsilon)| + C \leq k(\mathcal{A}(\varepsilon) + C)$. By taking the logarithm and including the description length of p in a new constant $k' = \ell(p) + \log_2(k)$, the bound $D_{\text{opt}} \leq k' + \log_2(|\mathcal{A}(\varepsilon)| + C)$ is established. ∎

Given a discretization $\mathcal{A}(\varepsilon)$, the δ_ε-argmax $\hat{x} \in \mathcal{A}(\varepsilon)$ found in the proof of Proposition III.3 could also have been directly encoded into the source code of p. This way, p would not have had to perform the exhaustive search, reducing its runtime considerably. However, a standard result in algorithmic information theory is that the typical description length of an element $x \in \mathcal{A}(\varepsilon)$ is of order $\log_2(|\mathcal{A}(\varepsilon)|)$. Thus, the effect (on the bound on) D_{opt} would have been the same for a "typical" argmax. This symmetry between information encoded in the algorithm, and information found searching, provides one deep justification of the particular combination "description-length plus the binary log of the runtime" used in the definition of D_{opt}.

The difficulty of optimization is also bounded by the shortest solution.

Proposition III.4 (Upper bound 2)**.** *There is a (small) computational constant k such that if $\langle f, c, \varepsilon\rangle$ is an optimization problem with shortest solution $\langle x^*, s\rangle$, then*

$$D_{\text{opt}}(\langle f, c, \varepsilon\rangle) \leq k + \ell(\langle x^*, s\rangle)$$

Proof: The proof is immediate: Let p be the program `Print '`$\langle x^*, s\rangle$`'`. ∎

Note that the program in the proof of Proposition III.4 is not as short as it initially looks. The program fits the entire solution s in the source code. That this is possible may appear like a weakness in the instance-oriented definition of difficulty. However, a similar bound can be established through exhaustive search of all solutions:

Let the algorithm p' be constructed to through all strings 0, 1, 00, 01, … in order. For each string, p' checks whether it encodes a tuple $\langle x^*, s\rangle$, and, if it does, checks whether $\langle x^*, s\rangle$ is a valid solution to the present problem. If $\langle x^*, s\rangle$ is a solution, then p' prints $\langle x^*, s\rangle$ and halts.

Assuming $\langle f, c, \varepsilon\rangle$ has a shortest solution $\langle \hat{x}, s\rangle$ of length m, the string representing $\langle \hat{x}, s\rangle$ will be around the 2^mth string that p' checks (somewhere between string 2^m and 2^{m+1}, to be precise). The certificates s were assumed to be verifiable in polynomial time (in the length of the certificate). The algorithm p' thus has to make less than 2^{m+1} checks, each to a runtime-cost of at most dm^q (the cost of verifying proofs in $\mathcal{T}$). The description-length of p is a constant independent of the problem, and the runtime contributes $\log_2(2^{m+1} \cdot dm^q) = m + 1 + \log(d) + q\log(m) \leq k + \ell(\langle \hat{x}, s\rangle) + q\log_2(\ell(\langle \hat{x}, s\rangle))$ to D_{opt}. Save for an extra logarithmic term, this gives a bound on D_{opt} similar to Proposition III.4. This second symmetry – on the level of verifiable solutions instead of the search space – further justifies the choice of D_{opt}.

The difficulty of optimization may also be bounded from below.

Proposition III.5 (Lower bound)**.** *Assume that $d \cdot n^q$ bounds the running time of the proof-verifier, and that $\langle f, c, \varepsilon\rangle$ is a non-trivial problem. Then*

$$D_{\text{opt}}(\langle f, c, \varepsilon\rangle) \geq \frac{1}{q}\log_2(\ell(\langle f, c\rangle)) - \log_2(d)/q$$

Proof: The proof builds upon two bounds on the verification time of solutions for an optimal polynomial verifier v for the proof system $\mathcal{T}$. First, the runtime of v is bounded by dn^q for solutions of size n, which gives, $t(v(\langle \hat{x}, s\rangle)) \leq d\ell\langle \hat{x}, s\rangle^q$ for any $\langle \hat{x}, s\rangle$. Second, $\ell(\langle f, c\rangle) \leq t(v(\langle \hat{x}, s\rangle))$, since v cannot correctly verify certificates without reading at least the object function and the constraints (the last constraint always threatens to remove the alleged argmax from the domain). Combined, this gives $\ell(\langle f, c\rangle) \leq d \cdot \ell\langle \hat{x}, s\rangle^q$ or

$$\sqrt[q]{\ell(\langle f, c\rangle)/d} \leq \ell\langle \hat{x}, s\rangle$$

for any solution $\langle \hat{x}, s\rangle$ of $\langle f, c, \varepsilon\rangle$. Finally, observe that any solver p of $\langle f, c, \varepsilon\rangle$ must print a solution $\langle \hat{x}, s\rangle$, which yields the bound

$$\sqrt[q]{\ell(\langle f, c\rangle)/d} \leq \min_p\{t(p(\langle f, c, \varepsilon\rangle)) : p \text{ solves } \langle f, c, \varepsilon\rangle\}\,.$$

Taking the logarithm of both sides and adding $\ell(p)$ completes the proof. ∎

B. Problem Instances versus Classes

A common situation is that an algorithm p is available that provably outputs the right argmax on all instances in a class S of problems. This leads to a third bound on

D_{opt} (Proposition III.6). Note that this connects the instance-difficulty introduced in this paper with the commonly-known class-difficulty from computational complexity theory [14]. The latter is defined as the best (asymptotic) runtime of an algorithm outputting the correct argmax on all instances. The following proposition show that this class-difficulty essentially bounds D_{opt} for all instances in the class. The proposition assumes that the proof system $\mathcal{T}$ is powerful enough to allow traces of provably correct algorithms as (part of) a certificate.

Proposition III.6 (Upper bound 3)**.** *There is a computational constant $k \in \mathbb{N}$ allowing the following bound: Let p output the correct argmax for all instances in a class S of optimization problems, and let s_p b a certificate for this. Let $\langle f, c, \varepsilon\rangle$ be a problem in S. Then*

$$D_{\text{opt}}(\langle f, c, \varepsilon\rangle) \leq 2\ell(s_p) + \log_2(t(p(\langle f, c, \varepsilon\rangle))) + k + C$$

where C subsumes the cost of proving $\langle f, c, \varepsilon\rangle \in S$.

Proof sketch: An optimizer p' may be constructed that (1) establishes that $\langle f, c, \varepsilon\rangle \in S$, and then (2) uses p to find the argmax. Finally, (3) a certificate of the argmax is produced based on (1), s_p and the trace of p.

Step (1) contributes C in combined runtime and description length. Step (2) contributes $t(p(\langle f, c, \varepsilon\rangle))$ to the runtime. Step (3) contributes $k_1 t(p) + \ell(s_p)$ to the runtime. Both Step (2) and (3) may use s_p in the code, yielding a total contribution $\ell(s_p) + k_2$ to the description-length of p' for Step (2) and (3). Letting $k = k_1 + k_2 + 1$ and summing up the contributions finishes the proof. ■

Note that for a fixed class S, the only terms depending on the instance are $\log_2(t(p(\langle f, c, \varepsilon\rangle)))$ and C. The former will typically dominate, which shows the connection between the class-difficulty and instance-difficulty discussed above.

IV. Conclusion and Research Directions

The conceptual framework presented constitutes merely a first step in developing a deeper understanding of optimization problems from an information and algorithmic perspective. Hence, several important issues have not been addressed and left for future analysis. The first issue is the intricate relationship between the description of the algorithm, its runtime, and the computing resources it requires. While the proposed definition of optimization difficulty captures the first two to some extent, it does not model the computing resources required. Considering the increasing importance of parallel and distributed computing, it might be useful to incorporate this third aspect to achieve a more elaborate definition of optimization difficulty.

A second and related issue is the practical computability of optimization difficulty. While certain theoretical computability results can be obtained, developing practical methods for approximating optimization difficulty using finite resources and in finite runtime is of interest. The third open issue is the availability of "information" about the optimization problem itself. The presented framework has interesting implications for gray and black-box optimization problems where learning and pattern recognition methods and modeling play a natural role [16] A fourth and final future research direction is related to approximations and noise. A unique feature of optimization difficulty compared to descriptive complexity is the fact that it is not always necessary to use the full description of the problem if the solver takes a certain amount of risk on the precision of the solution, which may lead to a potentially interesting line of research.

Acknowledgment

This research was supported in part by the Australian Research Council Discovery Projects (DP140100819). The first author thanks Iman Shames for helpful suggestions.

References

[1] T. Weise, M. Zapf, R. Chiong, and A. Nebro, "Why Is Optimization Difficult?" in *Nature-Inspired Algorithms for Optimisation*, ser. Studies in Computational Intelligence, R. Chiong, Ed. Springer Berlin Heidelberg, 2009, vol. 193, pp. 1–50. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00267-0_1

[2] J. He, C. Reeves, C. Witt, and X. Yao, "A Note on Problem Difficulty Measures in Black-Box Optimization: Classification, Realizations and Predictability," *Evolutionary Computation*, vol. 15, no. 4, pp. 435–443, 2007. [Online]. Available: http://dx.doi.org/10.1162/evco.2007.15.4.435

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

[4] T. Lattimore and M. Hutter, "No Free Lunch versus Occam's Razor in Supervised Learning," in *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence*, ser. Lecture Notes in Computer Science, D. Dowe, Ed. Springer Berlin Heidelberg, 2013, vol. 7070, pp. 223–235. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-44958-1_17

[5] A. Auger and O. Teytaud, "Continuous Lunches Are Free Plus the Design of Optimal Optimization Algorithms," *Algorithmica*, vol. 57, no. 1, pp. 121–146, 2010. [Online]. Available: http://dx.doi.org/10.1007/s00453-008-9244-5

[6] M. W. Krentel, "The complexity of optimization problems," *Journal of Computer and System Sciences*, vol. 36, no. 3, pp. 490–509, 1988. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022000088900396

[7] T. Everitt, T. Lattimore, and M. Hutter, "Free Lunch for Optimisation under the Universal Distribution," in *Proceeding of IEEE Congress on Evolutionary Computation (CEC'14)*. IEEE, 2014, pp. 167–174.

[8] Y. Borenstein and R. Poli, "Kolmogorov complexity, optimization and hardness," in *Proceedings of the IEEE Congress on Evolutionary Computation CEC'06*. IEEE, 2006, pp. 112–119. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1688297

[9] T. M. Cover and J. A. Thomas, *Elements of information theory*. New York, NY, USA: Wiley-Interscience, 1991.

[10] M. Li and P. Vitanyi, *Kolmogorov Complexity and its Applications*, 3rd ed. Springer Verlag, 2008.

[11] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[12] W. M. Farmer, "The Seven Virtues of Simple Type Theory," *Journal of Applied Logic*, no. December, pp. 1–37, 2007.

[13] S. Boldo, C. Lelay, and G. Melquiond, "Formalization of Real Analysis: A Survey of Proof Assistants and Libraries," INRIA, Tech. Rep., 2013. [Online]. Available: http://hal.inria.fr/docs/00/94/89/11/PDF/article.pdf

[14] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[15] P. Orponen, K.-I. Ko, U. Schöning, and O. Watanabe, "Instance complexity," *Journal of the ACM (JACM)*, vol. 41, no. 1, pp. 96–121, 1994. [Online]. Available: http://dl.acm.org/citation.cfm?id=174648

[16] T. Alpcan, "A framework for optimization under limited information," *Journal of Global Optimization*, pp. 1–26, 2012.