What Makes an Optimization Problem Hard?

William G. Macready
David H. Wolpert

SFI WORKING PAPER: 1995-05-046

SANTA FE INSTITUTE

What Makes An Optimization Problem Hard?

SFI-TR-95-0?-???

William G. Macready (wgm@santafe.edu)
David H. Wolpert (dhw@santafe.edu)
The Santa Fe Institute
1399 Hyde Park Road
Santa Fe, NM, 87501

April 22, 1995

Abstract

We address the question, "Are some classes of combinatorial optimization problems intrinsically harder than others, without regard to the algorithm one uses, or can difficulty only be assessed relative to particular algorithms?" We provide a measure of the hardness of a particular optimization problem for a particular optimization algorithm. We then present two algorithm-independent quantities that use this measure to provide answers to our question. In the first of these we average hardness over all possible algorithms for the optimization problem at hand. We show that according to this quantity, there is no distinction between optimization problems, and in this sense no problems are intrinsically harder than others. For the second quantity, rather than average over all algorithms we consider the level of hardness of a problem (or class of problems) for the algorithm that is optimal for that problem (or class of problems). Here there are classes of problems that are intrinsically harder than others.

1 Introduction

Optimization is an important task in many fields of engineering and operations research. The optimization task consists in locating global extrema of a mapping, or cost function. Though simple to state, the optimization task is difficult and much effort has been, and will continue to be, devoted to the design of good optimization algorithms, that is algorithms which find extrema or near-extrema reliably and quickly. Accordingly, it is important to identify how difficult these algorithms are to construct for particular optimization problems, *i.e.*, for particular cost functions.

We say that a cost function is "hard" for a particular algorithm if that algorithm can not quickly find a near-extremum of that cost function. A natural question then is whether some classes of cost functions are intrinsically harder than others in some algorithm-independent sense. The alternative is that the hardness of a class of cost function can only be measured relative to a particular optimization algorithm.

As a first step at answering this question, we formally define a hardness measure. This quantity tells us how well any particular algorithm has performed on any particular cost function after some fixed number of iterations of the algorithm. Smaller hardness measures indicate better performance of the algorithm. Given the myriad of ways one might wish to exploit an optimization algorithm, determining a good measure of hardness is not at all trivial. We believe ours to be quite reasonable, and in particular it avoids some problems we have identified in alternative measures. Nonetheless, it may be that there are other reasonable measures besides ours which give different results from those associated with our measure.

Given our measure, we examine two ways to characterize the hardness of a class of cost functions in an algorithm-independent way. In the first we average our hardness measure over all possible algorithms on the class of cost functions of interest. As we show below, such an expected hardness is identical for any two classes of cost functions. So by this measure, no set of optimization problems is harder than another, and to distinguish between the two sets of optimization tasks we must resort to a comparison of how well particular algorithms perform over the two sets.

Our second way of characterizing hardness of a class of cost functions bears more similarity to the usual computational complexity characterizations of hardness than does our first way. In this second approach we first find the algorithm that minimizes the average hardness over our class of cost functions, and in this sense is "optimal" for that class. We then use that minimal average hardness as our characterization of the hardness of the class. As we show below, according to this second approach, there are classes of cost functions that are intrinsically harder than others.

We begin in section 1 by considering appropriate measures of gauging the hardness of an algorithm across a set of cost functions and motivate the measure we use in this paper. In section 2 we go on to use that measure to characterize the difficulty of a class of cost functions, by averaging the measure across that class and across all algorithms. We prove that this average is the same for all classes of cost functions. In section 3 we refine our analysis by comparing, for different classes of cost functions, the hardness of that class for the optimal algorithm for that class. According to this measure there *are* some classes of problems that intrinsically harder than others. We end by presenting open issues.

2 What is hard?

Our notation follows that used in [1]. Formally, the optimization problem consists in locating the global extrema of a single valued mapping $f : \mathcal{X} \to \mathcal{Y}$, which we call the "cost function". We assume $\mathcal{X}$ and $\mathcal{Y}$ are finite and of size $|\mathcal{X}|$ and $|\mathcal{Y}|$ respectively. Without loss of generality we assume we are seeking the global minima of f.

We call an ordered sample of m distinct points from the cost function a "population" of size m and denote it by d_m. In particular, in this paper we consider "search algorithms" like simulated annealing [2], genetic algorithms [3], hill-climbing, *etc.* For this broad class algorithms, the output of m iterations of the algorithm is a population. In such cases, populations are ordered according to the time at which the algorithm generated the points.

For convenience we index the m points of the population by $1 \leq i \leq m$ and write $d_m \equiv \{d_m(i)\} \equiv \{d_m^x(i), d_m^y(i)\}$ where d_m^x and d_m^y are respectively the $\mathcal{X}$ and $\mathcal{Y}$ components of the population. $\mathcal{D}_m$ is the set of all populations of size m and $\mathcal{D} = \cup_m \mathcal{D}_m$ is the set of populations of all sizes.

Search algorithms of the type considered in this paper rely on extrapolating from m distinct samples of the cost function, $(x, f(x))$, to a new point in $x \in \mathcal{X}$. They are mappings from populations to new points in $\mathcal{X}$. To simplify our exposition we impose the restriction that the new point is selected deterministically (for the same population the algorithm always chooses the same "new point") and is not already in the population. However, as discussed in [1] most of our results apply equally well to algorithms which are stochastic and which potentially re-visit the elements of the population. Given these restrictions, an algorithm is a mapping $a : d \in \mathcal{D} \to \{x \,|\, x \notin d_{\mathcal{X}}\}$.

We wish to compare algorithms with the same number of distinct cost evaluations, m (i.e., with the same size populations). Often we are not interested in the time-dependent nature of the algorithm's output but rather simply in the cost values obtained. Accordingly, in this paper we consider the histogram $\vec{c}$ of numbers of times each cost value occurs in the population generated by running the algorithm on the cost function for m steps.[1] When time-dependencies (*e.g.* how quickly during the m samples did the algorithm get to the optima) are not of interest, the value of $\vec{c}$ fixes the value of any measure one might use to judge the effectiveness of the search. In particular, it fixes the minimum cost value present in the population. When time-dependencies are important we consider not the histogram $\vec{c}$ but rather the time ordered population, d_m. As can be seen from the structure of our proofs all our results are equally applicable to these time-dependent notions of hardness.

We seek a "hardness" function, ϕ, of $\vec{c}$ and f, which measures the quality of the search that produced $\vec{c}$ on cost function f. Note that since we are restricting

[1] Formally, the y'th component of $\vec{c}$ is given by $\vec{c}_{y \in \mathcal{Y}} \equiv \sum_{i=1}^{m} \delta(d_{\mathcal{Y}}(i), y)$, where $\delta(.,.)$ is the Kronecker delta function.

ourselves to deterministic algorithms, the value of ϕ is equally as well specified by (a, f, m) as by $(\vec{c}, f)$.

Unfortunately, there are scenarios which do not seem to have a preferred natural hardness measure. For example, say there are four possible cost values, which we list as $\{y_1, y_2, y_3, y_4\}$ where $y_1 < y_2 < y_3 < y_4$. Say that for the cost function f_1 the successive fractions of $x \in \mathcal{X}$ such that $f(x) = y_1, y_2, y_3, y_4$ are 0.1, 0.1, 0.4, and 0.4, respectively. However for f_2, the four fractions are 0.4, 0.4, 0.1, and 0.1. The goal is to minimize cost, and assume algorithm 1 run on f_1 gets down to y_3, but algorithm 2 run on f_2 gets down to y_2. Algorithm 2 got to a lower cost value than algorithm 1. But for algorithm 1, there are only 20% of the $x \in \mathcal{X}$ with a lower cost, whereas for algorithm 2 there are 40%. It's not clear that there is any obvious way to decide which algorithm did better. As another, more general example, how should one compare search across a largely flat cost function with search across one which has no plateaus?

The traditional computational complexity approach circumvent these issues by solely considering how close the algorithm got to the minima, in this case y_1. However this measure varies with monotonic transformations of $\mathcal{Y}$. In many cases this is clearly an inappropriate measure of hardness.

In this paper, to circumvent these difficulties, when comparing hardnesses we restrict attention to sets of f's all of which are in the same equivalence class. To define our equivalence structure, define $\omega(f)$ as the $|\mathcal{Y}|$-vector whose i'th value is the number of x such that $f(x)$ is the i'th y value, y_i. Define $\Omega(f)$ as the vector — whose dimension can vary with f — of the successive nonzero components of $\omega(f)$. Then f_1 and f_2 are in the same class iff $\Omega(f_1) = \Omega(f_2)$. As an example, if f_1 is f_2 "shifted" upwards, they are in the same class. Similarly f_1 is in the same class as f_2 if it is produced from f_2 by any monotonic transformation of the cost values.

Even given this restriction on the set of f's we will consider, it is not immediately obvious what the measure of hardness, ϕ, should be. One possible measure is the minimal y such that $\vec{c}_y$ is nonzero, which we write as $\phi(\vec{c}, f) = \min(\vec{c})$. This measure is actually a poor indicator of the quality of the search though, because it is not invariant under a translation of the $\mathcal{Y}$ values. (Note that such a translation does not knock us out of our equivalence class.) To get around this difficulty we might imagine instead defining ϕ in terms of $|\min(\vec{c}) - E(f)| \ / \ \sigma(f)$, where $E(f)$ is the average (across $\mathcal{X}$) cost value of f, and $\sigma(f)$ is the standard deviation of these cost values. With this ϕ we would be measuring the number of standard deviations $\min(\vec{c})$ is from the average cost in f. However this measure is also not without difficulties. For example, according to it algorithms would be judged to perform better on f's for which the algorithm produces the same $\vec{c}$ but which have smaller standard deviations. It is not at all clear that we wish to favor such f.

To get around these problems, in this paper we have defined ϕ be the "sta-

tistical weight" of the set of $x \in \mathcal{X}$ such that $f(x) > \min(\vec{c})$. Formally

$$\phi(\vec{c}, f) = \phi(a, f, m) \equiv 1 - \frac{\sum_{x \in \mathcal{X}: f(x) > \min(\vec{c})}(1)}{|\mathcal{X}| - \mu(f)}, \tag{1}$$

where $\mu(f)$ is the number of distinct $x \in \mathcal{X}$ that give the same global minimum of f. (By $\sum_{x \in \mathcal{X}: f(x) > \min(\vec{c})}$ we mean a sum over all x values such that $f(x) > \min \vec{c}$.) This ϕ is normalized so that it equals 1 if $\min(\vec{c})$ is the global maximum and equals 0 if $\min(\vec{c})$ is the global minimum.

We next turn to the question of measuring the hardness of a class of cost functions, or more generally of a distribution over cost functions, for a particular algorithm. We have a number of options for how to do this. The simplest choice is the average performance of a according to the distribution at hand,

$$\Phi_1(a, m) \equiv \sum_f \phi(a, f, m) P(f) . \tag{2}$$

There are a number of alternatives to this choice. For example, we might measure the difficulty of a class of cost functions as the difficulty of the hardest instance within the class:

$$\Phi_2(a, m) \equiv \max_{f \in F} \phi(a, f, m) . \tag{3}$$

This second measure could be appropriate where one is unsure of $P(f)$ (except that its support is F), and wants to be conservative. For sake of space though, in this paper we restrict our attention to Φ_1.

Now that we have determined a reasonable scheme for measuring the hardness of an f or a $P(f)$ for a particular algorithm, we can move on to determining the intrinsic hardness of $P(f)$ by itself, without any a priori specification of the algorithm. This is done in the next two sections.

3 Average behavior over all algorithms

Perhaps the most obvious algorithm-independent measure of the hardness of $P(f)$ is the uniform average over all possible algorithms a of $\Phi_1(a, m)$. This can be written as

$$\sum_{a, f, c} \phi(\vec{c}, f) P(f) P(\vec{c} \mid a, f, m) = \sum_{f, c} \phi(\vec{c}, f) P(f) \sum_a P(\vec{c} \mid a, f, m). \tag{4}$$

(Note that in the current context, where all algorithms are deterministic, the distribution $P(\vec{c} \mid f, m, a)$ is a delta function in the space of $\vec{c}$ values, with the argument of the delta function set by f, m and a.)

We begin by calculating

$$P(\vec{c} \mid f, m) \equiv \sum_a P(\vec{c} \mid a, f, m).$$

(Our notation is motivated by taking $P(a)$ to be uniform.) If we could prove that this quantity is independent of f, that would mean that the average $\Phi(a, m)$ is independent of $P(f)$ (see Eq. 4). Of course, in general $P(\vec{c} \mid f, m)$ can not be independent of f, since different f's have different distributions over cost values, $\omega(f)$. However this need not be the case in our current context, where we are implicitly restricting $P(f)$ so that its support lies in a single Ω equivalence class; the relevant question for us is whether $P(\vec{c} \mid f, m)$ can vary as f is varied within an equivalence class.

To prove $P(\vec{c} \mid f, m)$ is independent of f (subject to the above caveat) we will replace our sum over algorithms with a sum over all possible orderings of points sampled from $\mathcal{X}$. To that end, first note that since $|\mathcal{X}|$ and $|\mathcal{Y}|$ are finite, populations are finite, and therefore any (deterministic) a is a huge, but finite, list. That list is indexed by all possible d's (aside from those d's that extend over the entire input space). Each entry in the list is the x the a in question outputs for that d-index.

Now consider any particular ordered set of m x values with no x appearing twice in the set. Such a set is an "ordered path" π_m. (Note that such a π_m is actually the x components of a population, d^x_m, but for clarity we avoid referring to it as such.) A particular π_m is "from" or "in" a particular f if there is an ordered set of m distinct $(x, f(x))$ pairs identical to π_m.

Make the definition that "$\{a : a_m(f) = \pi_m\}$" specifies the set of all algorithms that produce populations d when run on f such that $d^x_m = \pi_m$. Given these definitions, we can write

$$\begin{aligned} \sum_a P(\vec{c} \mid a, f, m) &= \sum_{\pi_m} \sum_{a : a_m(f) = \pi_m} P(\vec{c} \mid a, f, m) \\ &= \sum_{\pi_m} \sum_{a : a_m(f) = \pi_m} P(\vec{c} \mid \pi_m, f, m). \end{aligned} \tag{5}$$

We rewrite our sum as

$$\sum_{x_1} \sum_{x_2 \notin \{x_1\}} \cdots \sum_{x_m \notin \{x_1, \cdots, x_{m-1}\}} \sum_{a : a_m(f) = \{x_1, \ldots, x_m\}} P(\vec{c} \mid x_1, \cdots, x_m, f, m).$$

Now the summand is independent of the inner-most sum. Moreover, regardless of $\{x_1, ..., x_m\}$ or f, there are the same number of terms in that sum.[2] This establishes the following:

[2] This follows immediately from viewing an algorithm as a list (see above); the restriction on a in the sum fixes a total of m entries in that list, and has no effect on the remaining entries.

$$
\begin{aligned}
P(\vec{c} \mid f, m) &= \sum_a P(\vec{c} \mid a, f, m) \\
&\propto \sum_{x_1} \sum_{x_2 \notin \{x_1\}} \cdots \sum_{x_m \notin \{x_1, \cdots, x_{m-1}\}} P(\vec{c} \mid x_1, \cdots, x_m, f, m), \quad (6)
\end{aligned}
$$

where the proportionality constant is independent of f.

Now we expand about all possible d_m^y and note that $\vec{c}$ depends only on d_m^y:

$$
\begin{aligned}
P(\vec{c} \mid f, m) &\propto \sum_{d_m^y} P(\vec{c} \mid d_m^y) \sum_{x_1} \sum_{x_2 \notin \{x_1\}} \cdots \sum_{x_m \notin \{x_1, \cdots, x_{m-1}\}} P(d_m^y \mid f, m, x_1, \cdots, x_m) \\
&= \sum_{d_m^y} P(\vec{c} \mid d_m^y) \sum_{x_1} \delta(d_m^y(1), f(x_1)) \sum_{x_2 \notin \{x_1\}} \delta(d_m^y(2), f(x_2)) \cdots \\
&\quad \times \sum_{x_m \notin \{x_1, \cdots, x_{m-1}\}} \delta(d_m^y(m), f(x_m))
\end{aligned}
$$

The $\sum_{x_i} \delta(d_m^y(i), f(x_i))$ sums collectively contribute a constant which varies with d_m^y but is the same for all f's in the same equivalence class. Therefore for all such f $P(\vec{c} \mid f, m) \propto \sum_{d_m^y} P(\vec{c} \mid d_m^y)$, which is independent of f.

This establishes that any function of $P(\vec{c} \mid f, m)$, and in particular the average over a of $\Phi(a, m)$, is the same for all $P(f)$'s whose supports lie in the same equivalence class. Note that no property of $\phi(\vec{c}, f)$ was used in proving this result. Therefore this result holds for any hardness based solely on $\vec{c}$ and f. (Note though that whereas Φ_1 makes no distinction between different $P(f)$, Φ_2 may if $\sum_a$ and $\max_{f \in F}$ do not commute.)

Subject to the caveats discussed above, any problem or class of problems is as hard as any other, when hardness is averaged over all possible algorithms. To some such lack of distinguish-ability may seem intuitively obvious. However it does not hold for example when we compare the performance of algorithms that are optimal for their associated $P(f)$. This is true even if we make the same equivalence class restrictions on $P(f)$'s that were made here, as the analysis in the next section demonstrates.

4 Optimal behavior

In the previous section, we measured the algorithm-independent hardness of a $P(f)$ within an equivalence class by averaging $\Phi(a, m)$ over all a. Such an average can be misleading. This is because even though $\sum_a \Phi(a, m)$ is the same for all $P(f)$, the probability of any particular value of $\Phi(a, m)$ need not be. For example, it may be that for a small set of a $P_1(f)$ has a much higher $\Phi(a, m)$ than does $P_2(f)$. To meet the result of the previous section, we would

then have $\Phi(a,m)$ a tiny bit larger for $P_2(f)$ than for $P_1(f)$ for the large set of the remaining algorithms. In such a scenario, one might argue that $P_1(f)$ is "intrinsically harder" than $P_2(f)$, since in the worst case (over algorithms) it is much harder than $P_2(f)$, but in the worst case going the other $P_2(f)$ is only slightly harder than $P_1(f)$.

There are many other ways as well that $P(\Phi(a,m))$ can be used to distinguish the hardness of $P(f)$'s. In this section we examine the following one: Let $\bar{a}_m(P(f))$ be the best possible algorithm for $P(f)$, for populations of size m. Formally, $\bar{a}_m(P(f)) \equiv \text{argmin}_a \Phi$. (Recall that Φ is a function of $P(f)$ — see 2.) Then we can define $\bar{\Phi}_m(P(f))$ as the hardness of $P(f)$ for $\bar{a}_m(P(f))$: $\bar{\Phi}_m(P(f)) \equiv \Phi(\bar{a}_m(P(f)), m)$. So $\bar{\Phi}_m(P(f))$ is how hard $P(f)$ is for the best algorithm for $P(f)$.

For simplicity, restrict attention to those $P(f)$ that are uniform over some set of cost functions F and zero elsewhere. Let $|F|$ denote the number of cost functions, f, in the support of $P(f)$.

The question before us is to characterize the equivalence classes of those F sharing the same $\bar{\Phi}_m(F)$ for certain m. Here we won't answer the question in full, but rather will analyze it for certain small values of m. Our analysis will involve explicit construction of optimal algorithms. These constructions will depend upon m, $|F|$, and the location of the global minima of each $f \in F$. We proceed by considering each of these dependencies.

$|F| = 1$, any m: In this case our class of problems contains only a single instance, f. The optimal algorithm for this problem locates the global minima with the first sample of f. Regardless of f it is always possible to construct this optimal algorithm. So no single problem is inherently harder than any other.

$|F| = 2, m = 1$: Next we consider the case where we have two problems in the class and are allowed only a single guess. Here distinctions between classes appear. Any F such that the two cost functions share the $\mathcal{X}$ value of their global extrema will be easier than any F for which the pair of f's differ in the location of their extrema. In the first case the optimal algorithm guesses the shared value achieving maximal performance while in the second case the algorithm can only maximize the performance of one of the cost functions.

$|F| = 2, m = 2$: In this case the algorithm is allowed as many samples as there are cost functions. The optimal algorithm can always attain the extrema of both cost functions by first visiting the extrema of f_1 and then visiting the extrema of f_2. So no pair of problems is intrinsically harder than any other if we are given two cost evaluations. Of course, this is immediately generalizable to any scenario where $m \geq |F|$.

$|F| = 3, m = 2$: Here we demonstrate through a simple construction that some problems are harder than others even for the optimal algorithms. Consider cost functions $f_1, f_2, f_3 \in F$ with extrema at x_1, x_2, and x_3 respectively as in Fig. 1(a). By symmetry, the best the optimal algorithm, $\bar{a}$, can do is to select two distinct § values on its two guesses.. Without loss of generality assume the guesses are x_1 and x_2 (the order doesn't matter). In this case the optimal

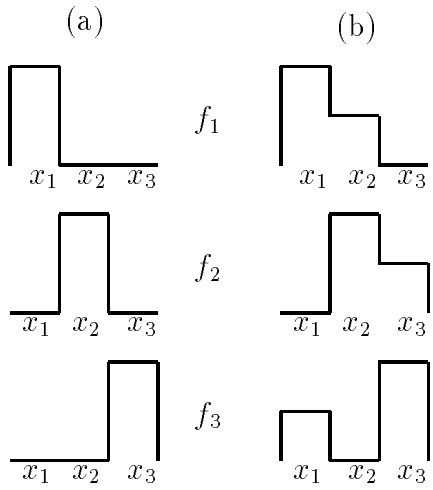

Figure 1: Two sets of three cost functions, f_1, f_2, f_3, with the same optima upon which the optimal algorithm, $\bar{a}$, performs poorly on (a) and well on (b).

algorithm will identify the optima on f_1 and f_2 but not on f_3. The best any optimal algorithm could do is to identify the optima for two of the three cost functions. Can we now construct a simpler set of cost functions for the optimal algorithm? Trivially yes! If all three algorithms have the optima at the same x value then the optimal algorithm simply guesses this x value. More interestingly, can we construct an easier set of cost functions but with the same optima as the original set? The answer is again yes and is shown if Fig. 1(b). If the distinct $\mathcal{Y}$ values in Fig. 1 are 1, 2, and 3, then an optimal algorithm which locates the optima on *all* three cost function is the following:

1: `select` x_1

2: `if` $f(x_1) = \begin{cases} 3 & \texttt{select } x_2 \texttt{ or } x_3 \\ 2 & \texttt{select } x_3 \\ 1 & \texttt{select } x_2 \end{cases}$

Essentially, knowledge of the y value conveys information about the location of the optima in case 1(b) whereas it does not in case 1(a). Intuitively, with the construction of Fig. 1(a) it is difficult even for the optimal algorithm to do well since the information gained with each guess is as small as possible. See [4] for a formalization of this notion in the context of supervised learning. Again, some classes F are harder than others.

As previously mentioned, in the case where $m \geq |F|$ then the first $|F|$ guesses of $\bar{a}_m$ are the extrema of each f_i, and no set of cost functions is harder than any other.

Given the close parallel between supervised learning and combinatorial optimization ([1]), the result of this section shouldn't be too surprising. After all, the (Bayes) optimal supervised learning algorithm performs differently for different priors over target functions.

5 Conclusions

We close on a rather philosophical note. In part, this paper came about as a proof of concept of the optimization framework developed in [1]. In response to the question posed to us, "are some optimization problems intrinsically hard?" we investigated what our previously developed framework had to say about this issue. As such we have barely scratched the surface, much work clearly remains to be done in answering this question and we briefly mention some future research directions. But, in response to our main motivation of exploring the practical utility of our framework we feel we have been amply rewarded.

There are a number of alternatives to the schemes explored in this paper for algorithm-independent measures of the hardness of a $P(f)$. Some were briefly mentioned in the text. Others are more similar to the conventional computational complexity measures than those mentioned in the text. For example, consider the scenario where $P(f)$ is nonzero only over some set F of cost functions. In that case, it is often may be more appropriate for Φ to be given by the worst performance of a on any of the $f \in F$, *i.e.* our $\Phi_2(a, m)$ measure of section 2. Presumably the results we have presented here will change if such changes are made in the definition of hardness. Future work involves exploring such alternative definitions.

Other future work involves carrying the analysis for optimal rather than averaged algorithms in more detail. In particular, the notions of information gain, and of the VC dimension and VC entropy ([4]) of a class —F—, may be helpful in carrying out this analysis.

Another interesting issue is the following question: Given $|F|$ and m, what fraction of classes are not assuredly solvable for the optimal algorithm? I.e., how pervasive are the problems discussed in the previous section? Obviously $|F|$ must be greater than m, or all classes are trivially easy (just look at all $|F|$ extrema). Given that that condition is met though, little is known.

Another question: How common are algorithms that are not optimal for any $P(f)$? What about if we restrict ourselves to stochastic algorithms whose first point is uniformly randomly chosen? (Note that one can use the inverse of this concept — the $P(f)$ that is optimal for a particular algorithm — to measure how close two algorithms are.[3]

Other future work involves perturbing the same-equivalence-class assumption. The idea here is that one can define a metric on the space of Ω values,

[3] Two algorithms are close if the corresponding $P(f)$ are close according to some metric on the space of probability distributions. See [1].

and then bound how much hardness can vary between two equivalence classes as a function of the distance between their respective Ω values.

Acknowledgements
We would like to thank Bill Spears for bringing our attention to the problem addressed by this paper.

References

[1] D.H. Wolpert, W.G. Macready, *No Free Lunch Theorems for Search*, SFI-TR-02-010, (1995).

[2] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, *Science*, **220**, 671, (1983).

[3] S. Forrest, *Science*, **261**, 872, (1993).

[4] D. Haussler, M. Kearns, R. Schapire, *Machine Learning*, **14**, 83, (1994).