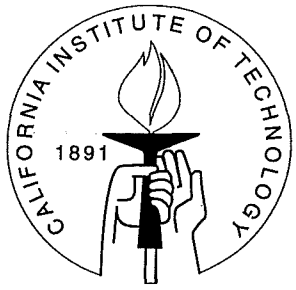
DIVISION OF THE HUMANITIES AND SOCIAL SCIENCES
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA 91125

TWO MEASURES OF DIFFICULTY

Scott E. Page

Forthcoming in *Economic Theory*



SOCIAL SCIENCE WORKING PAPER 927

May 1995

TWO MEASURES OF DIFFICULTY

Scott E. Page*

Summary. This paper constructs two measures of difficulty for functions defined over binary strings. The first of these measures, *cover size*, captures the difficulty of solving a problem in parallel. The second measure, *ascent size*, captures the difficulty of solving a problem sequential. We show how these measures can help us to better understand the performance of genetic algorithms and simulated annealing, two widely used search algorithms. We also show how disparities in these two measures may shed light on the organizational structure of firms.

1 Introduction

This paper addresses the measurement of difficulty for functions defined over discrete domains and demonstrates how formalizing notions of difficulty can improve our understanding of economic phenomena. The principal aims of this paper are fourfold: to introduce two measures of difficulty for functions defined over several discrete variables, to employ these measures to explain the performance of search algorithms, to contrast these measures with existing measures of difficulty, and, finally, to (preliminarily) apply these measures to an economic problem, the organizational structure of firms. We have selected this application because of its inherent difficulty and because it has not proven tractable using standard techniques. Though much of this work relies on the formal statement and proof of claims, a significant portion consists of computational experiments, particularly when we are explaining the performance of search algorithms and comparing measures of difficulty.

Each of the two measures we construct characterizes the difficulty of optimizing a nonlinear function using a class of search algorithms. The first of the measures, *cover size*, captures the difficulty of breaking a large problem into subproblems which can be

*Division of Humanities and Social Sciences 228-77, California Institute of Technology, Pasadena CA 91125. This paper has benefited from comments by seminar participants the NBER\CEME Decentralization Conference, UC–Irvine, and UC–San Diego. The author would like to thank Stan Reiter for his guidance and for his many suggestions in helping refine the concepts developed in this paper. Art Devany, Gordon Green, Jim Jordan, Nick Vriend, and an anonymous referee made helpful comments on earlier versions of this paper.

solved in parallel, while the second measure, *ascent size*, captures the difficulty of solving a problem using an ascent algorithm. We offer these measures hoping to complement, rather than supplant, standard measures of difficulty such as NP hard and decomposition size (Liepins and Vose 1990) as well as to encourage economists to study difficulty and complexit theory. We construct our measures so that they satisfy several normative criteria which we discuss at length in the next section. To mention just one criterion, each of our measures is defined relative to the upper contour sets of the function. If, for example, we are trying to maximize a function, then difficulty in the upper contour sets is more relevant than difficulty in lower contour sets.

Before continuing with this introduction, several caveats are in order. To begin with, the term difficulty is nonstandard. We use it, rather than the more common term complexity, because a function with randomly assigned values would be difficult to optimize, but not necessarily complex. In common usage, a complex problem possesses structure. Often this structure results from the combination of many simple components (Langton 1990 and Gell–Mann 1994). Given that our concern here is the problem of maximizing a function defined over several variables and not necessarily with characterizing a system created by interacting parts, difficulty is the more appropriate term. Another issue we should mention is that in the model that follows, we stress optimizing a function. In many, if not most, practical applications, search algorithms fail to locate global optima. The concepts that we develop, such as string dominance, can be relaxed so that instead of being true for all elements of the domain, they are true with high probability. Applications of the measures developed within this paper to actual problems may require less severe assumptions then those we present.

There are several reasons why economists should be interested in measures of difficulty. In the remainder of this introduction, we focus on the importance of difficulty measurements within the neoclassical paradigm. One criticism often leveled against neoclassical models is that they oversimplify economic environments and overestimate the abilities of human and organizational decision makers (Leijonhufvud 1992). In response to this criticism, economists have renewed their interest in bounded rationality, emphasizing the works of Simon (1969) and Marshall (1961). Recent attempts to include bounded rationality have incorporated psychology, experimentation with human subjects, computational experiments, and the theory of finite state computing machines. This emphasis on more realistic behavioral assumptions has led to a better understanding of the importance of economic institutions as well as the limitations and strengths of economic theory. Yet, many of these models, we might even say most, consider boundedly rational agents confronting relatively simple problems, such as the repeated prisoners' dilemma.[1] With many players, games such as the prisoners' dilemma may create a *complex* structure of interactions, but much of complexity results from coordinating on one of many equilibria.[2]

[1]Notable exceptions include the work of Brewer and Plott (1994) and Ledyard, Porter and Rangel (1994) in which the experimental environments are rather complex.

[2]We deliberately use the word "complexity" here because the interaction of simple decisions creates a complex environment.

In this paper, we are interested in single period problem defined over many variables which are complex, such as investing savings, choosing a career, buying a house, formulating a business strategy, etc...We focus on characterizing measures of the difficulty of such problems, This research intendly complements much of the aforementioned literature on bounded rationality, for, although the study of bounded rationality has contributed significantly to economics, it only partially addresses the criticism that neoclassical economics considers hyper intelligent agents in simple decision making environments. In order to formulate a complete response, we must also consider complex and difficult environments. To do so, we must have a theory of what constitutes a difficult or simple environment.

To summarize, we want to advance the notion that difficulty is the flip side of bounded rationality: each matters most in conjunction with the other. On the one hand, if economic agents have unbounded capabilities then neither the difficulty of their decisions nor the complexity of the economy of interacting agents is important. On the other hand, if the decisions facing agents are not complicated, then bounded rationality will be less relevant. Agents may still make optimal choices. The perspective of this paper is that while Robinson Crusoe might optimally allocate his time between leisure and banana production, economic agents in the modern economy, confronted with a vast array of commodities, careers, and constraints, rarely choose optimally. Further, while two person firms may be organized optimally, two hundred thousand person firms assuredly are not.[3]

The remainder of this paper consists of four parts followed by a brief discussion. In the first part, we discuss normative criteria by which a measure of difficulty might be judged, we reveal shortcomings of the NP or P classification scheme predominant in computer science,[4] and we mention a distinction between difficulty and randomness and discuss how it applies to the economic problems we consider and economics in general. In the second part, we construct two measures of difficulty for nonlinear functions defined over several discrete variables. One of our measures, *cover size*, captures the difficulty of solving a problem in parallel. The other measure, *ascent size*, computes the number of decisions which must be coordinated if the problem is to be solved serially. In this part, we also compare these measures of difficulty to existing measures. In the third part, we demonstrate how cover size and ascent size contribute to our understanding of the performance of genetic algorithms and simulated annealing on nonlinear search problems. In the fourth part, we apply our measures to an economic problem: optimal organizational design. Though, we do not come to any definitive conclusions, our analysis provides possible intuition about how firms should be divisionalized and about differences in the depth of hierarchies. Finally, in the discussion at the end of this paper, we discuss other economic applications.

[3]We make no claims to the originality of this idea. Many economists have called into question the knee jerk upgrading of decision makers' abilities so that they can handle the complexity of their environments.

[4]A problem is NP hard if the time required to solve the problem increases non–polynomially in the number of variables.

2 Difficulty Criteria

No single measure of difficulty is likely to be appropriate for all settings. The standard approach in computer science classifies problems according to whether they can be solved in polynomial time (P hard) or whether they require non–polynomial time (NP hard) as n, the number of decision variables, increases. Though this approach to measuring the difficulty of a problem has been helpful in categorizing problems, economists will notice at least three problems with a wholesale application of the NP classification scheme to economic problems. First, it considers worst case scenarios. Rather than characterizing the *expected* difficulty given a probability density function over a class of functions, it measures the worst case. Second, as mentioned, a problem is NP hard if the computation grows exponentially *as the number of decision variables increases*, which may be of only secondary concern to economists. In most economic problems, the number of decisions variables remains unchanged or else does not vary much.[5] Finally, the distinction between NP and P hard is a blunt instrument.[6] Dare we suggest that people can solve problems which can be solved in polynomial time but cannot solve problems which require non-polynomial time? Moreover, problems with relatively few variables by computer science standards, say between twenty and sixty, may be relatively large by economic standards, and in these problems, the constant term may outweigh the higher order terms: *a P hard problem may require more computation time than an NP hard problem.*

The inadequacy of the NP, P hard classification scheme for economic problems partly motivates our construction of new measures and recommends developing measures motivated by economic concerns as opposed to borrowing measures from another discipline. If we are to construct measures of difficulty for economic problems, we would like them to satisfy several normative criteria. First, these measures should create a finer classification scheme than the binary NP or P distinction. Second, each ought to measure the difficulty of solving problems given a class of optimization algorithms. Karp (1977) divides search algorithms into two classes: *ascent* and *divide and conquer.* The latter approach is more commonly referred to as *decomposition* which is the term we employ here. In accordance with Karp's taxonomy, the two measures of difficulty developed in this paper: *ascent size* and *cover size* capture the difficulty of optimizing a function using ascent algorithms and decomposition algorithms respectively. Third, a measure of difficulty should capture regularities in classes of problems. Examples are the works of Lin (1965) and Reiter and Sherman (1965) which find that traveling salesperson problems can often be solved by interchanging routes between three or fewer cities. For less than forty cities, the benefits of switching more than three routes appear to be outweighed by the computational costs. The perspective of this paper is that an accurate measure of ascent difficulty would classify the traveling salesperson problem as being of size three.

[5]Though it seems that each visit to the grocery store reveals yet another new Kelloggs' cereal, there is no reason to believe that the number of cereal brands is heading off to infinity any time in the near future.

[6]To be fair, functions can be classified by their exponent, and many are, but the typical distinction is whether a problem is NP or P hard.

Fourth, each measure individually should help explain the performance of search algorithms. We mean not simply that more complex problems should be harder to solve, but that the measures should explain the performance of nonlinear search algorithms such as simulated annealing and genetic algorithms. In addition, for some classes of functions the two measures may differ substantially in their characterization of difficulty. We would like this information to help guide the search for a successful algorithm for the class of functions. If, for example, a class of functions has a large cover size and a small ascent size, then we should have some hint as to which among a genetic algorithm, simulated annealing, or an ascent algorithm would perform best on average. Fifth, we want to measure difficulty relative to the function's contour sets. Reiter (1991) has advocated using measures of difficulty which distinguish between difficulty at lower contour sets and difficulty near the optimum. This point is demonstrated in figure 1.

Place Figure 1 Here

Each landscape consists of four local peaks and a unique global peak. In landscape A these local peaks have relatively low values and may not pose much of a problem in locating the highest point. Once the function value has exceeded a threshold equal to the value of the highest local but non–global local optima, any ascent algorithm locates the optimum. In landscape B, the local peaks have values close to the optimal value. On this landscape, a local ascent method may become stuck on a local optimum. Our intuition suggests that landscape B is more difficult than landscape A despite the fact that they have the same number of local optima. In section 4 of this paper, we investigate the performance of genetic algorithms on two functions, one of which forms a landscape similar to landscape A. The other function forms a landscape similar to landscape B. Our final criterion is for the measures of difficulty to be applicable to economic problems. We apply cover size and ascent size to the organization of firms in section 5.

3 Cover Size and Ascent Size

We construct two measures of difficulty for functions defined over several variables: *cover size* and *ascent size*. The entire analysis is done with respect to functions whose domains are binary strings. Extending the analysis to more general domains is straightforward. Both of our measures rely on the construction of *string dominant hyperplanes* for the function, which are a special class of subsets of strings. We assume throughout that our objective is to maximize a function V, and we define the string dominant hyperplanes with respect to V's upper contour sets. We begin this section with some basic definitions and then define string dominant hyperplanes and *covers*. A cover is a collection of string dominant hyperplanes, such that every decision variable is determined by one of the string dominant hyperplanes.

3.1 Binary Strings

3.1.1 Preliminaries

Each binary decision variable will be referred to as a *bit* and each element in the domain will be referred to as a *string.*

Def'n: The set of *bits* $N = \{1, 2, 3, .., n\}$

There are several economic problems which can be embedded in this framework. For instance, each bit may be thought of as the choice as to whether or not to undertake a public project (Page 1994b), to include someone in a group (De Vany 1994), or to locate a plant in one of two cities (Koopmans and Beckman 1961).

Def'n: The set of *strings*, $S = \{s : s = s_1 s_2 .. s_n \text{ with } s_i \in \{0, 1\}\}$

Without loss of generality, we assume that our objective is to maximize a function, V which belongs to F, the set of all functions whose domain can be encoded as binary strings of length n and whose range is the real numbers.

Def'n: The set of *objective functions*, $F = \{V : V : S \rightarrow \Re\}$

Of particular importance to our analysis are a class of subsets of N called hyperplanes. A hyperplane can by represented by a ternary string of length n over the set {0,1,*}. The ternary variables h_i are also referred to as bits.

Def'n: The set of *hyperplanes*, $H = \{h : h = h_1 h_2 .. h_n \text{ with } h_i \in \{0, 1, *\}\}$

We say that a bit in a hyperplane is *defined* if it assumes a value in the set $\{0, 1\}$ and *undefined* otherwise.

Def'n: The *defined bits* of h, $d(h) = \{i : h_i \in \{0, 1\}\}$

A string belongs to a hyperplane if the string and the hyperplane have identical values

on the defined bits of the hyperplane. We let $S(h)$ equal the set of binary strings which belong to h.

Def'n: The *set of binary strings belonging to* h, $S(h) = \{s \mid s_i = h_i \, \forall i \text{ if } h_i \in \{0,1\} \}$

Example: S(0*1*) = {0010, 0011, 0110, 0111}

The size of h equals the number of defined bits of h. According to this measure, a hyperplane's size equals its co-dimension. A hyperplane with a larger size contains fewer strings.

Def'n: The *size* of h, $\sigma(h) = \mid d(h) \mid$

3.2 String Dominant Hyperplanes

In creating a cover for a function, we must accomplish two tasks: to identify hyperplanes whose strings have high values and to combine these hyperplanes to form good strings. We address the latter of these tasks first by defining the *projection* operator $\wedge$ which combines hyperplanes.

Def'n: The *projection* operator $\wedge : H \times H \rightarrow H$ according to the following rule: $h \wedge \tilde{h} = y$, where

$$y_i = \begin{cases} \tilde{h}_i & if \;\; h_i = * \\ h_i & if \;\; h_i \in \{0,1\} \end{cases}$$

Example: 0** $\wedge$ 1*1 = 0*1

To simplify notation, we also define the *projection product* over a set of hyperplanes.

Def'n: Given a set of hyperplanes $I = \{h^1, h^2, \ldots h^m\}$, the *projection product* of I, $\wedge(I) = h^1 \wedge h^2 .. \wedge h^m$

The set of strings, S, is contained in H, therefore, $\wedge$ is also a map from the Cartesian product of H and S into S. We can think of $h \wedge s$ as moving the string s into the hyperplane h making the minimal number of changes in bit values. Page (1994a) shows that the operator $\wedge$ is associative but not symmetric.

To accomplish the second task, the identification of hyperplanes whose strings have high values under V, we rely on the notion of *string dominance*. A hyperplane h is said to be string dominant on a subset of strings, T, if for all s belonging to T, the value of $h \wedge s$ under V is at least as large as the value of s.

Def'n: A hyperplane, h, is *string dominant* for V on T iff $V(h \wedge s) \geq V(s)\ \forall s \in T$

Claim 3.1 states that if h and $\hat{h}$ are string dominant on T then $h \wedge \hat{h}$ is also string dominant on T, provided that $\hat{h} \wedge T \rightarrow T$.

Claim 3.1 *If h and $\hat{h}$ are string dominant for V on T and $\hat{h} : T \rightarrow T$, then $h \wedge \hat{h}$ is string dominant on T.*

pf. $\hat{h}$ string dominant for V on T implies $V(\hat{h} \wedge s) \geq V(s)\ \forall s \in T$. By assumption $\hat{h} \wedge s \in T$. Therefore, given that h is string dominant for V on T, it follows that $V(h \wedge (\hat{h} \wedge s)) \geq V(\hat{h} \wedge s)$, and by the associativity of $\wedge$, $V((h \wedge \hat{h}) \wedge s) \geq V(s)$, which completes the proof.

3.3 Covers

Before formally defining a cover, we first construct the contour sets for the objective function, V. To simplify the analysis, we assume that the function V is injective. This assumption allows us to define the upper contour sets with ordinal notation. The extension to cardinal characterization of the upper contour sets, and non–injective objective functions, is straightforward provided a technical condition is met.[7]

Assumption 1: $\forall s, \hat{s} \in S$, s.t. $s \neq \hat{s}$, $V(s) \neq V(\hat{s})$

Assumption 1 allows us to ordinally rank the strings from 1 to 2^n according to their value under V.

[7] A sufficient condition for the proofs to hold as stated is that there exists a unique string of highest value.

Def'n: *S ordered by* $V = \{s^1, ..s^{2^n}\}$ where $V(s^i) > V(s^{i+1})$ for $i = 1$ to $2^n - 1$

Def'n: The *upper contour set including* s^α, $T(\alpha) = \{s^\beta : \beta \leq \alpha\}$

Claim 3.2 states that applying the projection operator with a string dominant hyperplane maps an upper contour set into itself.

Claim 3.2 *h string dominant for V on* $T(\alpha)$ *implies* $h \wedge s \in T(\alpha)$ $\forall s \in T(\alpha)$

pf: h string dominant for V on $T(\alpha)$ implies $V(h \wedge s) \geq V(s)\, \forall s \in T(\alpha)$. It follows that $h \wedge s \in T(\alpha)$.

Corollary 3.1 states that combining two hyperplanes which are string dominant on an upper contour set with the projection operator forms a new string dominant hyperplane on the upper contour set.

Corollary 3.1 *For any* $h, \hat{h}$ *string dominant for V on* $T(\alpha)$, $h \wedge \hat{h}$ *is string dominant for V on* $T(\alpha)$.

pf. Follows directly from Claim 3.1 and Claim 3.2.

A *cover* for V is a finite set of string dominant hyperplanes, the union of whose defining bits contains all variables.

Def'n: The collection of hyperplanes, $C = \{h^1, h^2, ..h^m\}$, forms a *cover* for V on T iff (i) and (ii) hold:

(i) h^i is string dominant for V on T $\forall i$

(ii) $\bigcup_{i=1}^{m} d(h^i) = N$

This definition allows for two hyperplanes in a cover to be defined on the same bit. The example below shows that a cover does not have to be a partition.

Example: n=3 and $V(s) = 3s_1 + s_2 + s_3 - 2s_1s_2 - 2s_1s_3$. It is straightforward to show that $C = \{10*, 1*0\}$ is a cover for V on S.

Note that if C is a cover for V on an upper contour set, $T(\alpha)$ then it is also a cover for V on any upper contour set contained in $T(\alpha)$.

Claim 3.3 *If C is a cover for V on $T(\alpha)$ then C is a cover for V on $T(\beta)$ $\forall\, \beta \leq \alpha$.*

pf: If h^i is string dominant for V on $T(\alpha)$, then h^i is also string dominant for V on $T(\beta)$ $\forall\, \beta \leq \alpha$, which completes the proof.

Claim 3.4 below states that any string belonging to every hyperplane in a cover for V must optimize V. A consequence of Claim 3.4 is that the order in which the hyperplanes comprising a cover are located is irrelevant.

Claim 3.4 *If $C = \{h^1, h^2, ..h^m\}$ is a cover for V on $T(\alpha)$ and $\tau \in \Phi$, the permutation group on m elements, then $h^{\tau(1)} \wedge (h^{\tau(2)} \wedge (...h^{\tau(m)} \wedge (s)..)) = s^1 \;\; \forall s \in S$.*

pf. By Claim 3.3, C is a cover for V on $T(1) = \{s^1\}$. By Corollary 3.1, h^i string dominant for V on $T(\alpha)$ implies $V(h^{\tau(1)} \wedge (h^{\tau(2)} \wedge (...h^{\tau(m)} \wedge (s^1)..))) = V(s^1)$. It follows that:

$$h^{\tau(1)} \wedge (h^{\tau(2)} \wedge (...h^{\tau(m)} \wedge (s^1)..) = s^1$$

Therefore, by (ii) in the definition of a cover:

$$h^{\tau(1)} \wedge (h^{\tau(2)} \wedge (...h^{\tau(m)} \wedge (s)..) = h^{\tau(1)} \wedge (h^{\tau(2)} \wedge (...h^{\tau(m)} \wedge (s^1)..) \, \forall s \in S$$

which completes the proof.

Claim 3.4 implies that we can simplify the expression $h^{\tau(1)} \wedge (h^{\tau(2)} \wedge (...h^{\tau(m)} \wedge (s^1)..)))$ as $\wedge(C)$.

3.3.1 Cover Size

A cover can be interpreted as a decomposition into subproblems. If these subproblems are solved in parallel, then the time required to solve every subproblem is determined by the most times consuming subproblem. Therefore, we define a cover's *size* to be the maximal number of defined bits in any hyperplane which belongs to the cover.

Def'n: The *size* of a cover, $C = \{h^1, h^2, ..h^m\}$, for V on S, $Z(C) = max_i\{\sigma(h^i)\}$.

Example: $C = \{1**, *00*, **01\}$ is a cover of size 2.

We define $\alpha_j(V)$ as the number of strings in the largest upper contour set which has a cover of size j. Each $\alpha(\cdot)$ can be thought of as a functional which maps functions defined over binary strings into the set $\{1, .., 2^n\}$.

Def'n: $\alpha_j(V) = max\{\alpha : \exists C$, a cover for V on $T(\alpha)$, s.t. $Z(C) \leq j\}$

For example, if $\alpha_1(V) = 2^n$, then there exists a cover of size 1 for V on all of S. Such functions have been characterized by Liepins and Vose (1990) as *easy*. The $\alpha_j(\cdot)$'s can be combined to form the *decomposability vector*. The decomposability vector measures the cardinality of the upper contour sets which have covers of various sizes.

Def'n: The *decomposability vector* $\alpha(V) = (\alpha_1(V), \alpha_2(V), ..., \alpha_n(V))$

The decomposability vector, $\alpha(V)$, can be considered as a functional mapping the set of all functions defined on S into integer valued vectors of length n. Functions mapped to decomposability vectors with larger values are less complex, as measured by cover size, than those mapped to vectors with smaller values. Claim 3.5 states that for any function $V \in F$, $\alpha_j(V)$ is weakly increasing in j. In other words, as the function value improves, the cover size decreases.

Claim 3.5 *For all $V \in F$, the following hold:*

(i) $\alpha_{j+1}(V) \geq \alpha_j(V)$

(ii) $\alpha_n(V) = 2^n$

pf. (i) Let $\hat{C}$ be a cover for V of size j on $T(\alpha_j(V))$. Thus, $Z(\hat{C}) = j < j + 1$.

(ii) $C = \{s^1\}$ is a cover of size n on $T(2^n)$.

Claim 3.5 implies that covers distinguish between *irrelevant* nonlinear interactions,

those that may affect optimization, and *relevant* nonlinear interactions, those that do.[8] This distinction is of substantial importance. If nonlinear effects do not create problems for optimization, such as those shown in landscape A in figure 1, then a heuristic, mechanism, or institution constructed to overcome the nonlinear effects may be unnecessarily complicated.

3.4 Ascent Size

Cover size measures the number of decision variables in the largest subproblem in a decomposition which forms a cover. Therefore, cover size can be interpreted as a proxy for the minimal time required to solve the problem in parallel. We now show through an example that cover size may overstate the minimal number of bit decisions which must be coordinated in order to locate the optimal string using an ascent algorithm. We then develop an alternative measure of the difficulty of a function called *ascent size*. Prior to defining the ascent size of a function, we define the *ascent size of a cover*, which is at least as large as the ascent size of the function. As in our formulation of cover size, both measures of difficulty are made with respect to the upper contour sets of the function.

We begin by defining an algorithm applied to a function V and then define a class of ascent algorithms, the ALGO(r)s. We describe an algorithm as a functional which maps a value function and an initial string into the set of strings.

Def'n: An *algorithm* $A : F \times S \rightarrow S$.

We want to restrict attention to ascent algorithms, algorithms whose image given any function, V, and any initial string, s, has a greater value under V than the values of neighboring strings. In the class of algorithms we define, the ALGO(r)'s, a neighborhood consists of all strings which differ by fewer than a fixed number of bits. One bit mutation (Kauffman 1988), a sequential algorithm which compares a string's value to the values of all strings which differ on exactly one bit and moves to any string with a strictly higher value, satisfies this condition.

Def'n: An algorithm $A \in$ ALGO(r), iff $V(A(V,s)) \geq V(h \wedge A(V,s))$ for all h s.t. $\sigma(h) \leq r$ for all $s \in S$ and for all $V \in F$.

Obviously, if A belongs to ALGO(r), then it also belongs to ALGO(k) for all k less than or equal to r. Another consequence of the definition is that if the function V has

[8] Buchanan and Stubblebine (1962) draw a similar distinction between nonlinearities in economic problems.

a cover of size less than or equal to r on all of S, then A locates the global optimum regardless of the starting point.

Claim 3.6 *If $A \in ALGO(r)$ and C is a cover for V on S with $Z(C) \leq r$, then $A(V,s) = s^1$ for all $s \in S$.*

pf: Let $C = \{h^1, .., h^m\}$ be a cover for f on S. By above, $V(A(V,s)) \geq V(h^i \wedge A(V,s))$ for all h^i. The result follows immediately.

3.4.1 Ascent Size of a Cover

The previous claim guarantees that if ascent size is measured by the minimal size r such that ALGO(r) optimizes V for all initial points, then ascent size cannot be larger than cover size. As the next example demonstrates, if a bit belongs to more than one hyperplane in a cover, then the ascent size of a cover may in fact be less than the cover size.

Example: $N = \{1,2,3,4\}$ $V : S \rightarrow \Re$. Let $C = \{10**, 1**, **00, ***0\}$ be the unique cover for V on S of size 2. By Claim 3.4, the string 1000 optimizes V. Suppose that we apply A which belongs to ALGO(1) to V. By Claim 3.6, A locates the string dominant hyperplanes $1**$ and $***0$. Once those two hyperplanes have been located, the resulting string must lie in the hyperplane $1**0$. By assumption $10**$ is string dominant, therefore $V(1010) > V(1110)$ and $V(1000) > V(1100)$. Thus $A(V,s) \in 10*0$ for any $s \in S$. By a similar argument, $A(V,s) = 1000$. Thus, if A belongs to ALGO(1), it locates the optimal string, even though V has a cover size of two.

The *ascent size of a cover* can be defined recursively. The presentation of the definition is simplified if we adopt the convention of referring to a hyperplane by its defined bits, $d(h)$. Recall that if $h^i = 101**1$, then $d(h^i) = \{1,2,3,6\}$. The definition is constructed as follows: first all string dominant hyperplanes in the cover of size one are located. The defined bits of these hyperplanes are subtracted from the set N. If any hyperplanes in the cover have only one defined bit on the remaining elements of N, then their defined bits are also subtracted. This process continues until either all bits have been subtracted or until all remaining hyperplanes have at least two defined bits on the remaining elements of N. If the latter occurs, then hyperplanes with two defined bits on the remaining elements of N are identified and their defined bits are subtracting from the remaining elements of N. This process continues until all elements have been subtracted from the set N.

Def'n: Given a cover, $C = \{h^1, h^2, ..h^m\}$, for V on $T(\alpha)$, the *ascent size of a cover* C, $AS(C)$ is defined recursively:

Step 0: $t = 0$, $m = 0$, $N_0 = N$, $d_{i,t} = d(h^i)$ for all $i \in N$

Step 1: $m = m + 1$

Step 2: $\acute{N} = N_t \backslash \{ \bigcup_{|d_{i,t}|=m} d_{i,t} \}$

Step 3:
- If $\acute{N} = \emptyset$ go to Step 4
- If $\acute{N} = N_t$ go to Step 1
- If $\acute{N} \subset N_t$ then begin
  - $N_{t+1} = N_t$
  - $d_{i,t+1} = d_{i,t} \cap N_{t+1}$
  - go to Step 2

Step 4: $AS(C) = m$

3.4.2 Test Functions

We now compare the cover size for a function with the ascent size of the minimal covers for the same function. We consider two classes of functions, each is a simple spin glass model. In the first case we consider, the nonlinear effects are symmetric, and in the second case the nonlinear effect between bit i and bit j differs from the nonlinear effect between bit j and bit i.

A function f_1 which belongs to the class of functions F_1 is described as follows:

$$f_1(s) = \sum_{i=1}^{8} \gamma_i \cdot s_i \; + \; \sum_{i=1}^{8} \sum_{j=i}^{8} \gamma_{ij} \cdot s_i \cdot s_j \qquad\qquad \gamma_i \in [-1, 1], \;\; \gamma_{ij} \in [-\delta, \delta]$$

The data shown are from one hundred trials with $n = 8$. We vary the parameter δ to demonstrate that both cover size and the ascent size of the covers increases as the expected size of the nonlinear effects increases. (Estimated standard errors of the distributions are given in parenthesis.)

Symmetric Nonlinear Effects, F_1		
δ	**Cover Size**	**Ascent Size of Cover**
1.2	5.000 (.163)	2.240 (.100)
1.4	5.220 (.151)	2.770 (.109)
1.6	5.200 (.143)	2.880 (.109)
1.8	5.320 (.154)	3.000 (.118)
2.0	5.430 (.157)	2.970 (.104)

The second class of functions allow for asymmetric nonlinear effects. A function f_2 which belongs to the class of functions F_2 is described as follows:

$$f_2(s) = \sum_{i=1}^{8} \gamma_i \cdot s_i \; + \; \sum_{i=1}^{8} \sum_{j=1}^{8} \gamma_{ij} \cdot s_i \cdot s_j \qquad\qquad \gamma_i \in [-1, 1], \;\; \gamma_{ij} \in [-\delta, \delta]$$

The data shown are also from 100 trials and again $n = 8$. In these computations, the δ parameter was varied as in the symmetric case.

Asymmetric Nonlinear Effects, F_2		
δ	**Cover Size**	**Ascent Size of Cover**
1.2	5.210 (.174)	2.680 (.150)
1.4	5.870 (.169)	3.350 (.167)
1.6	5.860 (.171)	3.260 (.148)
1.8	5.900 (.172)	3.360 (.152)
2.0	5.810 (.177)	3.460 (.159)

3.4.3 Ascent Size of a Function

We now define the *ascent size of a function*, which differs from the ascent size of a cover. There are two reasons to draw this further distinction. First, there may exist string dominant hyperplanes not belonging to a particular cover, which nevertheless allow the ascent size to be reduced. And second, as string dominant hyperplanes are located the function value increases, which implies that the ordinal rank of the current string, α, decreases. As α decreases, the set of string dominant hyperplanes increases, creating the potential for a further reduction in the ascent size. Therefore, the ascent size of a function is weakly less than the ascent size of a cover. Before formally defining the ascent size of a cover, we first define the set of string dominant hyperplanes on $T(\alpha)$ which we denote H_α.

Def'n: The *string dominant hyperplanes on* $T(\alpha)$, $H_\alpha(V) = \{h : h \text{ is string dominant for } V \text{ on } T(\alpha)\}$

Claim 3.7 states that as α decreases, the set of H_α weakly increases with respect to set inclusion.

Claim 3.7 *If* $\alpha \leq \beta$, *then* $H_\alpha(V) \subseteq H_\beta(V)$

pf: If h is string dominant for V on $T(\beta)$, then h is string dominant for V on $T(\alpha)$.

Claim 3.8 states that if two hyperplanes belong to H_α then any defined bits common to both hyperplanes must have the same bit value.

Claim 3.8 *Given* $V \in F$, *suppose* $h, \hat{h} \in H_\alpha(V)$. *If* $i \in d(h) \cap d(\hat{h})$, *then* $h_i = \hat{h}_i$

pf: By assumption $h, \hat{h} \in H_\alpha(V)$. By Claim 3.7 $h, \hat{h} \in H_1(V)$, the upper contour set consisting of s^1, where s^1 is the highest valued string under V. Therefore, $h \wedge (\hat{h} \wedge s^1) = \hat{h} \wedge (h \wedge s^1) = s^1$, The result follows from the defintion of the projection operator $\wedge$.

The ascent size for V on $T(\alpha)$, which we denote by $A_Z(V, \alpha)$ is defined recursively. As in the definition of the ascent size of a cover, the bits which belong to the set of defined bits of a string dominant hyperplane are removed from consideration. The definition is simplified if we first define the set of string dominant hyperplanes on $T(\alpha)$ of size m with reduced support, $\hat{N}$, a subset of N.

Def'n: Given a subset of bits $\hat{N} \subseteq N$ and an integer m, the *set of string dominant hyperplanes for* V *on* $T(\alpha)$ *of size* m *and support* $\hat{N}$:

$$I(\alpha, m, \hat{N}) = \{h : h \in H_\alpha, \mid d(h) \cap \hat{N} \mid \leq m\}$$

We can now define the *ascent size* of a function V. There are two differences between this definition and definition of the ascent size of a cover. First, the set of hyperplanes being considered consists of all string dominant hyperplanes on H_α rather than just the hyperplanes in the cover. Second, as hyperplanes are located, the minimal value of a string which belongs to the hyperplanes may exceed the value of s^α. When this occurs,

we increase the set of string dominant hyperplanes from H_α to $H_{\hat{\alpha}}$, where $\hat{\alpha}$ is the rank of the string of minimal value belonging to the hyperplanes located.

Def'n: The *ascent size* of V on $T(\alpha)$, $A_Z(V,\alpha)$ is defined recursively:

Step 0: $t = 0$, $m = 0$, $\hat{N} = N$
$h_i^A = *$ for all $i \in N$

Step 1: $m = m + 1$

Step 2: $\hat{\alpha} = \max\{\beta : s^\beta \in S(h^A)\}$

Step 3: $D = \bigcup_{h \in I(\hat{\alpha}, \hat{N}, m)} d(h)$

Step 4: $\acute{N} = \hat{N} \backslash D$

Step 5:

If $\acute{N} = \emptyset$	go to Step 6
If $\acute{N} = \hat{N}$	go to Step 1
If $\acute{N} \subset \hat{N}$	then begin
	$\hat{N} = \acute{N}$
	$h^A = \wedge(I(\hat{\alpha}, \hat{N}, m))$
	go to Step 2

Step 6: $A_Z(V,\alpha) = m$

Recall that given any function V, the cover size for V on $T(\alpha)$ cannot be smaller than the cover size for V on $T(\beta)$ if $\beta < \alpha$. Less formally, this says that if the upper contour set decreases, i.e. if it contains fewer strings, the cover size over that upper contour set must also decrease weakly. Claim 3.9 states that a similar result holds for ascent size: the ascent size varies directly with the size of the upper contour set.

Claim 3.9 *For all $V \in F$, if $\alpha \leq \beta$, then $A_Z(V,\alpha) \leq A_Z(V,\beta)$.*

pf: Follows directly from Claim 3.8 and the definition of $A_Z(V,\alpha)$.

Claim 3.9 provides a partial theoretical explanation for the performance of simulated annealing on nonlinear search problems and may guide search for an optimal annealing schedule. These topics are a focus of section 4 of this paper.

3.4.4 Decomposition Size

We have remarked several times that a unique feature of the two measures that we have constructed: cover size and ascent size, is their focus on difficulty relative to upper contour sets. An additional advantage of these measures is that they do not consider the number and size of encoded nonlinear effects (Kauffman 1989, Liepins and Vose 1990). These other types of measures, which we call *domain based* are useful in creating test functions but can be misleading in their characterizations of difficulty. They can both over and underestimate a problem's difficulty.

Here we define a simple domain based measure, *decomposition size*, for functions defined over binary strings. Before defining decomposition size, we introduce the decomposition basis coefficients, which attach a value to each subset of N (Liepins and Vose 1991). If $O(s)$ denotes the subset of bits in s which assume the value 1, then the value of a string equals the sum of the values of the subsets contained in $O(s)$.

Given $V \in F$, the *decomposition basis coefficients* $(\beta_{V,\emptyset}, ..\beta_{V,I}, ..\beta_{V,N}) \in \Re^n$ satisfy:

$$V(s) = \sum_{I \subseteq O(s)} \beta_{V,I} \qquad \text{where } O(s) = \{i : \ s_i = 1\}$$

The decomposition size equals the size of the largest subset I which has a nonzero coefficient in the decomposition basis.

The *decomposition size* of V, $size_d(V)$ =max $\{\mid I \mid : \beta_{V,I} \neq 0\}$

In previous work (Page 1994a), we have shown that there exist functions with decomposition sizes of two and cover sizes of n for arbitrary n and that there also exist functions with decomposition sizes of n which have covers of size one. An example of the latter would be a function in which all subsets, I, have strictly positive coefficients in the decomposition basis. Examples of the former are more involved. The fact that decomposition size may understate difficulty is evident in the tables reporting cover sizes earlier in this paper. By construction, these classes of functions have decomposition sizes equal to two, but their cover sizes are much larger.

4 Applications to Nonlinear Search

In this section, we evaluate our measures in terms of their ability to explain the performance of search algorithms. To address this criteria, we examine theoretical explanations

of two search algorithms in light of our measures of difficulty. The two search algorithms we have selected, simulated annealing and genetic algorithms, have been widely used in computer science and engineering.

4.1 Genetic Algorithms

A genetic algorithm is a population based search algorithm which mimics evolution.[9] A genetic algorithm begins with a finite population of randomly generated binary strings. For explanatory purposes, we assume that there are thirty strings throughout our description. From this initial population, thirty strings are *reproduced* from the initial population by a process of natural selection –strings with higher values are more likely to be reproduced than those strings with lower values.[10] Following reproduction, two genetic operators are applied to the strings. First, the thirty strings are randomly collected in fifteen pairs of strings and with some probability, usually around 0.5, a *crossover* operator is applied. If crossover occurs, then the strings switch their values for some subset of bits. For example, if the strings to be crossed are 111111 and 000000 and the strings switch bits two and four then the resulting strings are 010100 and 101011. Following crossover, a bit mutation operator is applied the strings. In bit mutation, each bit randomly changes its value with small probability, usually below 0.1. An application of reproduction and the two genetic operators to the thirty strings counts as a generation. The population at the end of generation i becomes the entering population for generation $t+1$.[11]

4.1.1 The Schema Theorem

The extant theory explaining the performance of genetic algorithms relies on the schema theorem. The schema theorem focuses not on the strings that are being reproduced but on the hyperplanes to which the strings belong.[12] A binary string of length n belongs to 2^n hyperplanes, each of which is brought along as informative baggage when a string is reproduced. The schema theorem provides a lower bound on the number of strings belonging to a particular hyperplane h in generation $t+1$, $N_{t+1}(h)$, as a function of the number of strings in generation t belonging to h, $N_t(h)$, the relative value of strings belonging to h, and the susceptibility of h to crossover and mutation.[13] For hyperplanes whose strings have above average values which are not likely to be destroyed by mutation and crossover, then the expected number of strings which belong to the hyperplane in the next generation increases:

[9]Genetic algorithms were invented by John Holland (1975). For an understandable, general introduction to genetic algorithms, see Goldberg (1989).

[10]The reproduction operator typically has a random component which allows for low valued strings to survive if they are fortunate.

[11]A formal description of a genetic algorithm is contained in appendix 1.

[12]*Schema* is another word for hyperplane.

[13]For a hyperplane not to be susceptible to the operators it must have few defined bits.

$$E[N_{t+1}(h)] \geq N_t(h) \cdot (\text{ relative value of strings in } h) \cdot (\text{prob } h \text{ disrupted})$$

An implication of the schema theorem is that if the strings belonging to a hyperplane have values which are *consistently* above average and are not likely to be disrupted by the genetic operators, then the number of strings in the population belonging to this hyperplane grows over time. Such hyperplanes are often called *building blocks*. In one interpretation of the schema theorem, if h is a building block, then, eventually, most of the strings in the population will belong to h. Let B be the set of all building blocks. Using similar logic, eventually most strings belong to most elements of B. Provided that combining building blocks leads to a string of high value, the genetic algorithm locates a good string, and in some cases, locates the optimal string.

The schema theorem and its derivatives capture the intuition that reproduction of the fittest strings implies the survival of better hyperplanes and that the combination of better hyperplanes should form good strings. However, the schema theorem by itself does not explain why genetic algorithms have been successful across a variety of problems. The fact that the population should drift towards higher valued strings does not imply that a genetic algorithm will outperform an ascent algorithm. Why cannot a genetic algorithm become stuck in a region of the space which does not contain the optimum? The answer is that it can and often does. Nevertheless, a genetic algorithm generally performs satisfactorally on functions regardless of their difficulty. The question remains as to why.

The point of this section is not to suggest that the schema theorem is incorrect but to provide additional explanations for the ability of genetic algorithms to locate good solutions. We focus on two applications of covers to genetic algorithms though there are certainly more. The first is obvious: the elements of a cover can be thought of as potential building blocks. If the cover size is small enough that the constituent hyperplanes are not susceptible to the genetic operators and likely to be building blocks, then the genetic algorithm should easily find the optimum. We might be tempted to think that if the cover size is large, then a genetic algorithm would perform poorly. Though this may be true in some cases, it need not be true in general. This is the essence of the second application of cover theory.

Recall from Claim 3.5 that the cover size decreases as the function value improves regardless of the function. However, the rate of decrease of the cover size depends upon the specific function. Among the implications of this claim are that a successful search algorithm should reduce the number of bits switched as it moves up the contour sets and this reduction in bits should differ across functions. We would not expect an algorithm which switched a prespecified number of bits as a function of the number of trials to perform as well as an algorithm which adapted the number of bits switched in response to the function values encountered during search. Page (1994a) analyzes the performance of a genetic algorithm on some test functions to test this theory and finds that not

only does a genetic algorithms reduce the number of bits switched as the function value increases, but also that it adapts the number of bits switched in response to search. For a problem with a large cover size, the genetic algorithm switches many bits even late into the search, whereas for a problem with a small cover size, within a few generations, the genetic algorithm only switches a few bits.

4.2 Simulated Annealing

The second search algorithm that we discuss is simulated annealing (Metropolis, et al. 1953) which grew out of research connecting statistical mechanics to multi-variate combinatoric optimization (Otten and van Ginneken, 1989). Simulated annealing is perhaps best understood as a Markov process. Given any string s, there is a transition function, $\tau : S \times S \rightarrow [0,1]$, where $\tau(s, \hat{s})$ gives the probability of testing string $\hat{s}$ given that the current string s. In an ascent algorithm, the tested string $\hat{s}$ would be accepted as the current string if it has a higher value. In simulated annealing, this is also the case. Where simulated annealing differs from an ascent algorithm is that it also accepts strings with lower values with some probability. The probability of accepting a string of a lower value depends on the *temperature* of the system and the difference in function values between s and $\hat{s}$. We can loosely interpret the temperature as the probability of making a mistake, although the mistakes only occur in one direction: all improvements are accepted.[14] The term annealing refers to the fact that during an application of simulated annealing the temperature lowers according to an exogenous schedule.

We define the transition function τ as giving equal probability to all strings which differ by exactly one bit and zero probability to all strings which differ by more than one bit.

Def'n: The *transition function* $\tau : S \times S \rightarrow [0,1]$ according to the following rule:

$$\tau(s, \hat{s}) = \begin{cases} \frac{1}{n} & \text{if } \mid s - \hat{s} \mid = 1 \\ 0 & \text{if } \mid s - \hat{s} \mid \geq 2 \end{cases}$$

The *annealing schedule* determines the probability that the algorithm accepts a string with a lower function value. This feature of simulated annealing reduces the probability that the algorithm becomes stuck at a local optimum.

Def'n: The *annealing schedule* $\{T_t\}_{t=0}^{\theta}$ satisfies

[14] I have some preliminary notes and simulation data on a double annealing algorithm which makes mistakes in both directions. Double annealing appears to be less effective than standard simulated annealing.

(i) $T_t \geq 0$ for all t
(ii) $T_t \geq T_{\acute{t}}$ for $t < \acute{t}$
(iii) $T_\theta = 0$

The simulated annealing algorithm determined by the transition function τ and the annealing schedule $\{T_t\}_{t=0}^{\theta}$ can now be defined.

Def'n: The *simulated annealing algorithm* given the transition function τ, annealing schedule $\{T_t\}_{t=0}^{\theta}$, and initial string s consists of five steps:

Step 1: $t = 0$ and $s^* = s$

Step 2: Choose $\hat{s}$ according to $\tau(s, \hat{s})$

Step 3: If $V(\hat{s}) \geq V(s)$ then $s^* = \hat{s}$
If $V(\hat{s}) < V(s)$ then $s^* = \hat{s}$ with probability $P(s, \hat{s}, t) = e^{-\frac{\{V(s^*) - V(\hat{s})\}}{T_t}}$

Step 4 If $t = \theta$ goto step 5
If $t < \theta$, then $t = t + 1$, goto step 2

Step 5 end

Notice that if the difference between the value of s^* and $\hat{s}$ increases then the numerator of the negative exponent increases, which decreases the probability of accepting the string $\hat{s}$. And as the temperature decreases, the denominator decreases, which decreases the probability of accepting a string $\hat{s}$. To summarize, the probability of accepting a lower valued string is increasing in the temperature and decreasing in the difference in string values.

There are several theoretical explanations for the performance of simulated annealing. Many of these theorems show that if the annealing temperature is chosen properly then the global optimum is reached (Hajek 1988). Underpinning these proofs is the idea that if the annealing schedule lowers slowly enough than search visits every string, including, of course, the optimal string. If the annealing algorithm is interpreted as a Markov process then if the temperature is lowered quickly, the system quickly reaches equilibrium but the equilibrium value is generally not of very high value. Alternatively, if the temperature is lowered slowly, then the equilibrium state is likely to place substantial weight on the optimal string at a cost of many searches (Otten and van Ginneken 1989).

The explanation that we provide here differs from the standard one. In applying

simulated annealing to large problems, it has been found to perform well relative to other algorithms though it may not always locate the optimal string (Davis 1987). More interesting, for our purposes, is the fact that different functions have different annealing schedules, even in physical systems. To evaluate the simulated annealing algorithm using insights gained from our ascent size measure, we rely on Claim 3.9 which states that as the function value improves, the ascent size decreases. When simulated annealing is applied to a function the value of the current string generally drifts upward, which implies that the minimal size ALGO(r) required to locate the optimal string weakly decreases. If the transition function is biased towards strings which differ by a few bits, then the only way that several bits can be switched is through a combination of switches of a few bits. When the temperature is high, combinations of bit switches are likely, because even descents are accepted with substantial probability. A high temperature offers the potential of switching many bits even if no subset of bit switches increases the value. A low temperature makes such combinations of bit switches improbable. This argument suggests a strong relationship between an optimal annealing schedule for a function and the function's ascent sizes on its upper contour sets.[15]

5 Applications to Economics

We have demonstrated that the construction of measures of difficulty can contribute to our understanding of the difficulty of solving problems using various search algorithms, and may help to explain the performance of genetic algorithms and simulated annealing. What remains to be shown is whether these measures have any relevance to economics. In this section, we apply these measures to the problem of organizational design. The informal discussion which follows focuses on the divisionalization of firms and the depth of hierarchies.

5.1 The Divisionalization of Firms

Underpinning our analysis is an assumption that firms confront problems defined over many variables. A firm's problem may be to maximize profits and its decision variables may be production technologies, advertising plans, or input suppliers. Intuitively, a firm's organizational structure - be it divisionalization by task, product, technology, or geography must depend upon the external relationships among decisions and the costs of coordination. In sum, a firm's organizational structure characterizes the extent and nature of the coordination among decisions and determines the firm's ability to maximize profits.

To apply cover theory, we want to think of a situation in which a firm selects an organizational structure to maximize their expected revenues minus costs. We include

[15] In fairness, some of the formal proofs that simulated annealing locates the optimal string determine an optimal annealing schedule as a function of the depth of the function. Depth is a measure of ruggedness (Kern 1993)

among a firm's costs, the costs of computing the values of various decisions, the *computation costs*, as well as the costs of transmitting information between levels of the firm, the *communication costs*. To include these costs in a formal model of organizations goes beyond the current analysis.[16] For now, we can think of these costs as either delay as in Radner's (1993) model or as being administrative costs. What we wish to explore here is whether the notion of a cover can be of any use in understanding how firms are divisionalized. For this exploration to lead to interesting questions, we assume that the cover size for a firm's problem exceeds one – that the firm's decisions are inter–related.

Suppose that a firm produces and markets three goods or services (hereafter referred to as goods) denoted by 1,2, and 3 respectively. Suppose also that these goods require substantial research and development. The firms decision vector might consist of nine decision variables:

- p_i production of good i
- m_i marketing for good i
- r_i research and development for good i

The firm's objective function is given by:

$$\max\ V(p_1, p_2, p_3, m_1, m_2, m_3, r_1, r_2, r_3)$$

A firm without computation and communication costs could locate and select the optimal decision vector through exhaustive search. Alternatively, a firm could be omniscient and know the optimal decision vector. We find neither story convincing and believe instead that a firm is likely to evaluate some small subset of the possible decision vectors and choose the decision vector which has the highest expected value given the information gathered. In our interpretation, the decision vectors evaluated by a firm depend in a fundamental way on the firm's organizational structure.

Expanding on this example, we assume that there are external effects among the decisions for the individual products: decisions about the production, marketing, and research and development of product i may have an impact on one another, and external effects between similar tasks: marketing decisions for products i and j might be related, and so on. These two types of external effects generate two focal organizational structures: divisionalization by product and divisionalization by task. If the first type of external

[16] Radner (1993) has a model in which he determines optimal organizational forms given computation costs, where the costs are in terms of delay. In his model, the problem that the firm solves is equivalent to the addition of a long string of numbers, which we would argue is not complex.

effects dominate the latter type then the function may have a cover which breaks the larger problem into the subproblems:

$$(p_1, m_1, r_1) \quad (p_2, m_2, r_2) \quad (p_3, m_3, r_3)$$

In this case, the firm would be divisionalized by product. Alternatively, if the latter type of external effect dominates the external effects among products, then the function V may have a cover which breaks the problem into the subproblems:

$$(p_1, p_2, p_3) \quad (m_1, m_2, m_3) \quad (r_1, r_2, r_3)$$

Now, the firm would be divisionalized by task. In each case, the firm would reduce the computation costs: fewer decision vectors must be evaluated; and reduce the communication costs: less information is passed between decision makers.

In this framework, we must assume that the firm does not confront the exact same problem each period. Otherwise, it would make the same decision each period. A more reasonable assumption is that the firm's problem is drawn from some class of problems. For our theory to apply, problems within this class must possess sufficient regularity that the defined bits of the hyperplanes in their covers must be approximately the same. The bit values on the defined may differ.[17] As we have already stated, a firm's organizational structure determines its ability to make good decisions. If decisions m_1 and m_2 belong to different components in firm's structure, then external effects between these decisions would be ignored by the firm. Cover theory may provide the foundation for a framework in which to evaluate and interpret the organizational structures as decompositions of decision variables.

In addition to providing a framework for the problem of optimal organizational structure for a firm, cover theory may also contribute to the study of how firms choose their organizational structures. The class of problems confronting firms in a given industry probably drifts over time due to changes in technology, shifts in input factor prices, or changes in consumer preferences. In responding to this drift, firms may have to alter their organizational structures.[18]

There are several competing theories as to how firms choose their organizational structure ranging from the choices being optimal to being made through "a long string of incalculable and disreputable accidents" (Edgeworth 1881). Alchian (1950) and Winter (1975) advocate an adaptive theory in which firms change their structure incrementally.

[17]Thus, we really need a theory of covers for *classes of functions*, which we have not included in this paper.

[18]For example, as purchasers of automobiles have become more concerned with safety, automobile producers' marketing departments have increased their emphasis on safety features and presumably their interaction with safety engineers.

Among the proponents of this view are Dahl and Lindblom (1963) who provide a litany of reasons as to why large organizations must change slowly. We now investigate a consequence of this incrementalist assumption. We show with an example that it is possible that a small change in the parameterization of a function can result in a large change in the set of string dominant hyperplanes.[19] Obviously, any such finding depends upon the choice of metrics over both the space consisting of the class of problems confronting the firm and the space of possible organizational structures. Therefore, any inferences which can be drawn from simple examples have a large subjective component. Nevertheless, we feel that the computational findings presented below are informative.

We return to the example above in which the firm produces three goods. To simplify notation, let $m = (m_1 + m_2 + m_3)$, $r = (r_1 + r_2 + r_3)$, $p = (p_1 + p_2 + p_3)$, $o = (p_1 + r_1 + m_1)$, $w = (p_2 + r_2 + m_2)$, and $h = (p_3 + r_3 + m_3)$. The computational experiments assume the following functional form:

$$f(p_1, p_2, p_3, r_1, r_2, r_3, m_1, m_2, m_3) = \alpha_m \cdot m + \alpha_r \cdot r + \alpha_p \cdot p + \alpha_o \cdot o + \alpha_w \cdot w + \alpha_h \cdot h$$

where the $\alpha_i \in [-4, 4]$ and are distributed uniformly. This functional form was chosen because two focal decompositions are divisionalization by product and by task. In the computational experiments, we randomly chose α_is and located a minimal size cover. We then simulated drift in the parameters by setting

$$\hat{\alpha}_i = \alpha_i + \beta \cdot \epsilon$$

where ϵ is uniformly distributed in $[-1, 1]$ and β is a parameter of the model. We then calculated how many of the string dominant hyperplanes in the original cover were no longer string dominant given the drift in parameters.

Number of Changes in Cover		
$[-\beta, \beta]$	*# hyps lost*	*(s.d)*
$[-0.005, 0.005]$	0.563	(1.01)
$[-0.050, 0.050]$	1.884	(1.81)
$[-0.100, 0.100]$	2.337	(1.94)
$[-0.150, 0.150]$	2.940	(2.06)
$[-0.200, 0.200]$	2.990	(2.29)
$[-0.250, 0.250]$	3.111	(2.15)

[19]This example suggests also that two firms with similar objective functions may have radically different optimal organizational structures.

As can be seen from the data, the number of hyperplanes in the original cover which are no longer string dominant is quite large for even small changes in the parameters of the model. To the extent that this is a generic phenomenon, it merits further study. Though, we run the risk of over–interpretation of findings, we feel that these simple computations shed light on a potential problem if firms do in fact change their organizational structures incrementally. Imagine a firm which has an optimal divisionalization given some set of parameters representing technology, taste, etc ... If, as appears possible from these computations, a small change in parameters leads to a massive change in the optimal organizational structure. For example, if the optimal structure shifts from divisionalization by product to divisionalization by task, then a firm which adapts incrementally may not be able to switch to the optimal organizational form. Perhaps the more important question is whether a suboptimal organizational structure, such as divisionalization by task, may be a *local* optimum in the sense that it is better than any structure which can be achieved by an incremental change. A firm might then be trapped with a non–optimal organizational structure and be susceptible to competition from an entrant which can assume any organizational form.[20]

5.2 The Depth of Hierarchies

The second application of our measures of difficulty to the organization of firms pertains to the depth of hierarchies. Clearly, the depth of a firm's hierarchy must be influenced by the costs of transmitting information within the organization. Here we pursue a complementary explanation: that the depth of the hierarchy depends upon the difficulty of the firm's objective function. In this preliminary analysis, we consider only the number of strings which have to be evaluated in order to locate the optimal string. We are counting the structural costs here entirely in terms of delay.

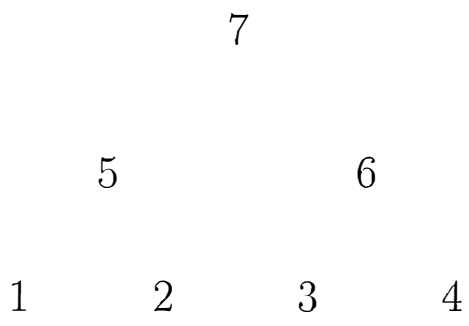
Suppose that a firm's objective function is defined over seven binary variables and has a cover of size three. To better elucidate the argument we refer to a string dominant hyperplane by its defined bits; the string dominant hyperplane 11**1** is written as $\{1,2,5\}$. Suppose that the cover for the firm's objective function, V, is given by:

$$\{1\},\ \{2\},\ \{3\},\ \{4\},\ \{1,2,5\},\ \{3,4,6\},\ \{5,6,7\}$$

The function V has a cover of size three. The firm's problem could be decomposed into three problems each consisting of three decision variables. Three binary variables implies that there are eight distinct combinations of values, therefore we can use eight as a rough measure of the time required to compute the optimal string. Suppose that instead the firm has a hierarchical structure as shown in figure 2:

[20]This could be viewed as an example of Schumpeterian creative destruction at the level of organizational form (Schumpeter 1950).

Figure 2



Decisions at the bottom level are made first and then passed up to the middle level. The decisions at the middle level are then made contingent upon the decisions made at the lower level. The decisions from both levels are then passed up to the decision maker at the top, who chooses a value for decision seven. The maximal number of strings evaluated at each level equals one and there are three levels. Ignoring communication costs between levels, the total costs equal three which is less than the eight required by a decomposition into a cover.

To summarize, the function has a cover size of three and an ascent size of one. The difference in these two measures captures the advantage of sequential search over decomposition of the larger problem. A firm whose problem has an ascent size much lower than its cover size should have a deeper hierarchy than a firm whose problem has an equal cover size and ascent size.

6 Discussion

In this paper we have constructed two measures of difficulty for functions defined over binary strings of arbitrary length. One of these measures, cover size, captures the difficulty of decomposing a large problem into subproblems which can then be solved in parallel. The other, ascent size, measures the difficulty of solving a problem using an ascent algorithm. We have discussed how these concepts may provide insights into the divisionalization of organizations and the depth of hierarchies. Other possible applications of these techniques include decision making over public projects (Page 1994b) and the problem of location of industries (Koopmans and Beckman 1961). The latter problem has recently been addressed by Arthur (1989) and Krugman (1993)[21]. In these models, a firm must decide whether to locate in one of several cities and there are external effects

[21]De Vany (1994) addresses this problem indirectly.

between firms.[22] Consider the problem of a central planner in these environments. If the central planner's problem has a small ascent size then a decentralized approach in which firms make independent decisions may be quite effective, provided the individual firm's incentives are in line with the planner's. If the ascent size is large, then greater coordination may be necessary.

Finally, in recent years there has been increased use of computational methods in the social sciences. As a rule most of this research has focused on situations in which artificial agents following simple rules combine to form a complex environment (Arifovic 1989, Marimon, McGratten, and Sargent 1990, Holland and Miller 1991). [23] Choosing a career, buying a house, allocating assets in a portfolio, or even a visit to the grocery story requires optimizing a difficult function defined over a discrete set. Current computational models primarily consider agents who choose a uni–dimensional variable, often price or quantity. Eventually, theoretical and computational models in economics must admit more difficult, more realistic objective functions for agents. When they do, the interplay between difficulty and bounded rationality will become a central issue.

[22] Arthur models these external effects as spillovers and Krugman models them as pecuniary externalities.

[23] An exception to this rule is the work of Kollman, Miller, and Page (1992) who examine a computational model of two–party competition. In their model, a challenging political party faces a complicated decision problem: how to locate the platform in a multi–dimensional issue space which obtains the most votes against a fixed incumbent. In economic settings, multi–dimensional choice settings are pervasive.

Appendix 1

In this appendix we construct a genetic algorithm using pseudo–code. A genetic algorithm begins with a procedure that initializes the population of strings.

Initialize Population: Randomly generate m strings of length n and number them from 1 to m.

Each generation of a genetic algorithm consists of reproduction of the best strings followed by two genetic operators: crossover and mutation. Here we describe tournament reproduction, uniform crossover and bit mutation. After these three stages, the resulting m strings become the entering population for a new application of reproduction followed by the genetic operators

Tournament Reproduction: From the set $\{1, 2.., m\}$, randomly select m pairs of numbers.[24] Each pair of numbers denotes two strings. Reproduce the string with the higher value. If the strings have the same value, choose randomly between the strings.

Uniform Crossover: Group the m strings into $\frac{m}{2}$ pairs of strings. With probability p_c apply the following procedure to the strings in the pair s and $\hat{s}$:

$$\text{With prob 0.5 exhange } s_i \text{ with } \hat{s}_i \text{ for } i = 1 \text{ to } n$$

Bit Mutation: For each bit on each string in the popultion, with probability p_m let $s_i = 1 - s_i$

[24] These numbers need not be unique.

References

Arifovic, J., "Learning by Genetic Algorithms in Economic Environments," Santa Fe Institute. Working Paper 90–001 (1989).

Arthur, B., "Competing Technologies, Increasing Returns, and Lock–in by Historical Events", Economic Journal, **99** 116-131 (1989).

Bethke, A., "Genetic Algorithms as Function Optimizers," Doctoral Dissertation, The University of Michigan, Dissertation Abstracts International 41(9) 3503B, 1988.

Brewer, P. and C. Plott "A Binary Conflict Ascending Price (BICAP) Mechanism" SSWP #887, California Institute of Technology, June 1994.

Buchanan, J. and C. Stubblebine, "Externality," Economica. **29** 371-384 (1962)

Dahl, R. and C. Lindblom, "Politics, Economics, and Welfare", Harper, New York, 1963.

Davis, L. "Genetic Algorithms and Simulated Annealing", Morgan Kauffman, London, 1987.

Gell–Mann, M., "The Quark and the Jaguar: Adventures in the Simple and the Complex", Freeman and Co., New York, 1994.

Goldberg, D., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, Reading MA, 1989.

Goldberg, D.,"Construction of High-order Deceptive Functions Using Low-order Walsh Coefficients," IlliGAL Report no. 90002, Department of General Engineering, University of Illinois, 1990.

Greffenstette, J. and J. Baker, "How Genetic Algorithms Work: A Critical Look at Implicit Parallelism," in Proceedings of the Third International Conference on Genetic Algorithms edited by J. Schaffer, Morgan Kauffman, New York, 1989.

Hajek, B., "Cooling Schedules for Optimal Annealing", Mathematics of Operations Research **13** 311-330 1988.

Holland, J., Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI. 1975.

Holland, J. and J. Miller, "Artificial Adaptive Agents in Economic Theory", Proceedings of the American Economic Association, **81** 365–370 (1991)

Karp, R. "Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problem in the Plane", Mathematics of Operations Research, **2**(3) 209-224 (1977).

Kauffman, S. "Adaptation on Rugged Fitness Landscapes," in Lectures in the Sciences of Complexity, Addison Wesley, Reading MA., 1989.

Kern, W. "On the Depth of Combinatorial Optimization Problems", Discrete Applied Mathematics, **43** 115–129 (1993).

Kollman, K., J. Miller, and S. Page, "Adaptive Parties in Spatial Elections", American Political Science Review, **86** 929-937 (1992).

Koopmans T. and M. Beckman, "Assignment Problems and the Location of Economic Activities," Econometrica, 53–76, (1961).

Langton, C., "Computation at the Edge of Chaos: Phase Transitions and Emergent Computation", Physica D. **42** 12–37 (1990).

Ledyard, J., D. Porter, and A. Rangel, "Using Computational Exchange Systems to Solve an Allocation Problem in Project Management", Journal of Organizational Computing forthcoming.

Leijonhufvud, A., "Towards a Not-To-Rational Macroeconomics", Distinguished lecture annual meeting of the Southern Economics Association, 1992.

Liepins, G., and M. Vose, "Representational Issues in Genetic Optimization," Journal of Experimental and Theoretical Artificial Intelligence, **2**(2), 4–30 (1990).

Lin, S., "Computer Solutions of the Traveling Salesman Problem", The Bell System Technical Journal, 2245–2269 (1965).

Marimon, R., E. McGratten, and T. Sargent, "Money as a Medium of Exchange in an Economy with Artificially Intelligent Agents," Journal of Economic Dynamics and Control **14** 329–373 (1990).

Marshall, A., "Principals of Economics", Macmillan, London 1961.

Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," Journal of Chemical Physics, **21** 1087-1092 (1953).

Otten, R. and L. van Ginnekan, "The Annealing Algorithm", Kluwer, Boston, 1989.

Page, S., "Covers: A Theory of Boolean Function Decompositon", Complex Systems **8** 1-24. (1994a).

Page, S., "A Bottom-up Efficient Algorithm for Allocating Public Projects with Positive Complementarities", working paper #885, California Institute of Technology, 1994b.

Reiter, S. and G. Sherman, "Discrete Optimizing," Journal of the Society of Industrial and Applied Mathematics, **13**(3) 864–889 (1965).

Reiter, S., private communication 1991.

Schumpeter, J., "Capitalism, Socialism, and Democracy," Harper, New York, 1950.

Simon, H., "The Sciences of the Artificial," MIT Press, Cambridge MA. 1969.