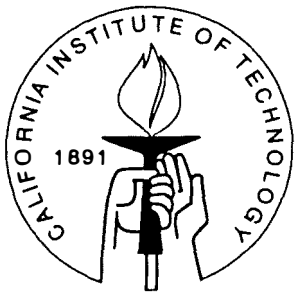
DIVISION OF THE HUMANITIES AND SOCIAL SCIENCES
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA 91125

COVERS

Scott E. Page



SOCIAL SCIENCE WORKING PAPER 872

December 1993

COVERS

Scott E. Page

Abstract

This paper introduces the theory of covers for functions defined over binary variables. Covers formalize the notion of decomposability. Large complex problems are decomposed into subproblems each containing fewer variables, which can then be solved in parallel. Practical applications of the benefits from decomposition include the parallel architecture of supercomputers, the divisionalization of firms, and the decentralization of economic activity. In this introductory paper, we show how covers also shed light on the choice among public projects with complementarities. Further, covers provide a measure of complexity/decomposability with respect to contour sets, allowing for nonlinear effects which occur near the optimum to receive more weight than nonlinear effects arbitrarily located in the domain. Finally, as we demonstrate, covers can be used to analyze and to calibrate search algorithms.

COVERS

Scott E. Page*

1 Introduction

This paper introduces the theory of covers for functions defined over binary variables. Cover theory serves three purposes. First, covers formalize the familiar notion that complex problems defined over many variables can be decomposed into simpler subproblems each containing fewer variables. Each of the subproblems can then be solved in parallel, thereby decreasing computation time. Practical applications of the benefits from decomposition include the parallel architecture of supercomputers, the divisionalization of firms, and the decentralization of economic activity (Simon 1969). In this paper, we show how covers also shed light on the multiple public projects problem.[1] Second, covers provide a measure of complexity/decomposability with respect to contour sets, allowing for nonlinear effects which occur near the optimum to receive more weight than nonlinear effects arbitrarily located in the domain. Third, covers can be used to analyze the performance of and to optimize search algorithms.

Covers can be best explained through an example. Suppose that a city is considering three public projects: an airport (a), a botanical garden (b), and a cable car system (c). Suppose that the city's value function is well-defined and denoted by V, where V maps the power set of {a,b,c} into the real numbers. Let the empty set, $\emptyset$, denote the status quo and let "ab" denote the state of the world where the airport and the botanical garden are provided, but the cable car system is not. Suppose that V satisfies the following inequalities:

$$V(c) < V(\emptyset) < V(bc) < V(b) < V(a) < V(ab) < V(ac) < V(abc)$$

If the objective function, V, is not known ex ante but is revealed through cost-benefit analysis, a complete decomposition of the three project decisions so that each decision is

*Division of Humanities and Social Sciences 228-77, California Institute of Technology, Pasadena CA 91125. The author would like to thank Stan Reiter, David Richardson, and Jim Jordan for their insights.

[1]See Page 1993 for a more complete analysis of the multiple public project problem.

made independently might not lead to the optimal choice of projects. In this scenario, the decision on the cable car system is problematic: the cable car system should not be built if the airport is not built:

$$V(c) < V(\emptyset) \text{ and } V(bc) < V(b)$$

However, if the airport is built, then the cable car system should be built as well. It follows that coordination on decisions is required to guarantee the optimal choice over subsets of projects. Suppose that we decompose the set of projects into the sets {a,c} and {b} and make decisions on these two sets independently. Consider first the decision on the botanical garden. The inequalities below follow from above:

$$V(\emptyset) < V(b) \quad V(c) < V(bc) \quad V(a) < V(ab) \quad V(ac) < V(abc)$$

These inequalities show that regardless of the decision on the other projects, the botanical garden should always be provided. Consider second, the decision on the other set of projects:

$$max\{V(\emptyset), V(a), V(c)\} < V(ac)$$

$$max\{V(bc), V(ab), V(b)\} < V(abc)$$

These two inequalities show that providing both the airport and the cable car system is the preferred alternative, regardless of the decision on the botanical garden. Combining the decisions on the subproblems {a,c} and {b} leads to the optimal decision over the set of all projects, i.e. providing all three projects. A decomposition into subproblems *forms a cover* if (i) each decision (variable) belongs to at least one subproblem and (ii) the optimal decisions on the subproblems "agree" with the optimal decision on the larger problem. In the example, {a,c} and {b} form a cover and also partition the set of decisions. The latter need not be true in general, the same variable may belong to more than one subproblem.

This example also can be used to demonstrate how covers can measure complexity/decomposability relative to the function's contour sets. Suppose that the airport's value is known to be greater than the status quo value and that the airport's external effect on the other two projects is thought to be positive. Suppose further that preliminarily, the airport is assumed to be provided so that the starting point for optimization is "a." In this case, the problem can be decomposed into the sets {a}, {b}, and {c}. To understand why, first consider the decision on the cable car system.

$$V(a) < V(ac)$$

$$V(ab) < V(abc)$$

The value of the airport together with the cable car system, V(ac), exceeds the value of the airport alone, V(a). And the value of all three projects, V(abc), exceeds the value of the airport and the botanical garden, V(ab). It follows that, the optimal decision on the subproblem {c} is to provide the cable car system. A similar argument establishes that the optimal decision on the botanical garden is to provide that project as well. Finally, the optimal decision on the airport is to not reverse the earlier decision. Regardless of the decisions on the other two projects, the airport is always worth providing. Thus, we say that the sets {a}, {b}, and {c} form a cover for V on the contour set above "a."

By starting optimization from a better initial set of projects ($V(a) > V(\emptyset)$), the cover was simplified, in that the maximum number of decisions in any one subproblem was reduced from two to one. In the formal model presented later in this paper, the *size of a cover* equals the number of variables in the largest subproblem. Cover size measures the complexity of the smallest subproblem remaining after maximal decomposition. Though there are other possible measures of complexity, cover size is most relevant if the subproblems are to be solved in parallel: the time required to solve the problem equals the time required to solve the largest subproblem. In the example, the cover size decreases as the initial point of search improves. Interpreting cover size as a complexity measure, we can say that the problem becomes less complex near the optimum. In the paper, we show that complexity decreases as the initial point of search improves for *any function* defined over binary strings.

The remainder of this paper is organized as follows. In Sections 2 and 3, we define covers for functions defined over binary strings and construct a complexity vector. We also compare cover size with other measures of complexity and present data from test functions. In Section 4, we use covers to select optimal parameters for a class of hillclimbing algorithms and to offer an alternative explanation for the performance of genetic algorithms. The latter discussion focuses on the "building block hypothesis" and its interpretation through the lens of covers. In the conclusion, we discuss a more general notion of covers mentioned by Richardson (1991).

2 Binary Strings

A cover decomposes a problem into subproblems which can then be solved in parallel. This decomposition is relative to the objective function's upper contour sets. We begin with some basic definitions.

2.1 Preliminaries

We refer to each binary variable as a ***bit*** and to a decision on each binary variable as a ***string.***

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

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

We assume that we are trying to maximize 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.

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

A class of subsets of N called hyperplanes play a prominent role in the analysis. A hyperplane can by represented by a ternary string of length n over the set {0,1,*}.

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

For ease of exposition, the ternary variables h_i are also referred to as bits. A bit in a hyperplane in defined if it takes either the value 0 or 1.

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

A string lies in a hyperplane if the string and the hyperplane have identical values on the defined bits of the hyperplane.

A string s *belongs to* h, $s \in h$ iff $s_i = h_i \; \forall i$ s.t. $h_i \in \{0, 1\}$

Example: $0^*1^* = \{0010, 0011, 0110, 0111\}$

The size of h equals the number of defined bits of h.

The *size* of h, $\sigma(h) = \mid d(h) \mid$

According to this measure, a hyperplane's size equals its co-dimension. A hyperplane with a larger size contains fewer strings.

2.2 String Dominant Hyperplanes

The idea which underpins the definition of a cover is that "good" hyperplanes can be combined to form good strings. This same idea is the basis for the Schema Theorem which we describe in Section 4. We begin by defining the operator $\wedge$ which combines hyperplanes.

$\wedge : H \times H \rightarrow H$ 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

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.

Claim 2.1 *The operator $\wedge$ is associative but not symmetric.*

pf: $\wedge$ *associative:* $h \wedge (\tilde{h} \wedge \hat{h}) = y$ where

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

It is straightforward to show that $y = (h \wedge \tilde{h})) \wedge \hat{h}$, which completes the proof.

$\wedge$ *not symmetric:* Let $h = 00*$ and $\tilde{h} = *1*$. $h \wedge \tilde{h} = 00*$, but $\tilde{h} \wedge h = 01*$.

Recall that a motivation for covers is that "good" hyperplanes can be combined to form "good" strings. A strong notion of "good" hyperplane is Greffenstette and Baker's (1989) *dominant hyperplane.*

A hyperplane, h, is *dominant* for V iff $\forall s \in h$ and $\forall \hat{s} \notin h$, $V(s) \geq V(\hat{s})$

A hyperplane h is *strictly dominant* for V iff $\forall s \in h$ and $\forall \hat{s} \notin h$, $V(s) > V(\hat{s})$

Claim 2.2 below states that given two strictly dominant hyperplanes, one must be a subset of the other. This implies that strictly dominant hyperplanes cannot be combined. They can only be extended.

Claim 2.2 *For any h, $\hat{h}$ strictly dominant for V, s.t. $h \neq \hat{h}$ and $\sigma(h) \leq \sigma(\hat{h})$, $\{s : s \in \hat{h}\} \subset \{s : s \in h\}$*

pf. by contradiction. Suppose $\exists\, \hat{s} \in \hat{h}$ s.t. $\hat{s} \notin h$. It follows that $\exists s \in h$ s.t. $s \notin \hat{h}$. h strictly dominant for V implies $V(s) > V(\hat{s})$, and $\hat{h}$ strictly dominant for V implies $V(\hat{s}) > V(s)$, a contradiction.

A consequence of Claim 2.2 is that if h and $\hat{h}$ are dominant but not strictly dominant hyperplanes for V, then the function V must take an identical value for all strings that belong to exactly one of the hyperplanes. It appears then, that requiring a hyperplane to be dominant is too restrictive for our purposes. As an alternative to dominant hyperplanes, we propose the weaker notion of *string dominance*, which is sufficiently weak to allow for hyperplanes to be combined but strong enough to ensure that the hyperplanes combine to form the optimal string. A hyperplane h is said to be string dominant on a subset of strings, T, if the value of a string in T does not decrease when moved into the hyperplane h by the operator $\wedge$. Formally,

A hyperplane, h, is *string dominant* for V on T iff $V(h \wedge s) \geq V(s)\ \ \forall s \in T$

A hyperplane h is *strictly string dominant* for V on T iff $V(h \wedge s) > V(s)\ \ \forall s \in T \backslash h$, where $T \backslash h = \{s :\ s \in T,\ s \notin h\}$.

Claim 2.3 below states that the operator $\wedge$ preserves string dominance.

Claim 2.3 *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 $V(h \wedge (\hat{h} \wedge s)) \geq V(\hat{h} \wedge s)$, which by the associativity of $\wedge$ implies $V((h \wedge (\hat{h}) \wedge s) \geq V(s)$, which completes the proof.

3 Covers and Complexity

3.1 Covers

In this section we formally define a cover. Before doing so, we need to define the contour sets for the objective function, V. To simplify the analysis, we assume that no two strings have the same value under V. This assumption allows us to ordinally define the upper contour sets. The extension to cardinal characterization of the upper contour sets, and non-injective objective functions, is straightforward.

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

Given Assumption 1, the strings can be ordered from 1 to 2^n according to their value under V.

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

The *upper contour set including* s^α, $T(\alpha) = \{s^\beta : \beta \leq \alpha\}$

Claim 3.1 below states that string dominant hyperplanes map an upper contour set into itself.

Claim 3.1 *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 below follows directly from Claim 2.3 and Claim 3.1.

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.

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

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 $C = \{11*, 1*1\}$ is a cover for V on S.

It can be shown given Assumption 1 that two hyperplanes in a cover must agree on any bits which are defined for both hyperplanes. Another consequence of this definition is that if C is a cover for V on $T(\alpha)$, then it is also a cover for V on $T(\beta)$ for $\beta < \alpha$.

Claim 3.2 *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 3.3 below states that any string belonging to every hyperplane in a cover for V must optimize V. In other words, the optimal string can be located by forming a cover for V. A consequence of Claim 3.3 is that the order in which the hyperplanes comprising a cover are located is irrelevant.

Claim 3.3 *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 3.2, 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.

A cover should be easy to find (and therefore the optimum easy to find) if the hyperplanes which compose it are defined on a small number of bits. For example, if all of the hyperplanes in a cover on S are of size one, then each bit value can be determined in isolation, and the problem can be solved quickly. If, on the other hand, several hyperplanes in a cover have a large number of defined bits, then the time required to solve the subproblems may be significant. In order to capture the intuition that a problem is as difficult as its largest subproblem, we define a cover's size to be the maximal number of defined bits in any hyperplane which belongs to the cover.

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 let $\alpha_j(V)$ equal 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 $\{1, .., 2^n\}$.

$\alpha_j(V) = max\{\alpha : \exists C$, a cover for V on $T(\alpha)$, s.t. $Z(C) \leq j\}$

Example: $\alpha_1(V) = 2^{(n-1)}$ implies there exists a cover of size 1 for V on the upper contour set consisting of all strings with function values above the median.

Claim 3.4 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 amount of relevant complexity, as measured by cover size, decreases.

Claim 3.4 $\forall V \in F$, *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 $T(\alpha_j(V))$. Trivially, $Z(\hat{C}) \leq j+1$. The result follows.

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

An implication of Claim 3.4 is that covers distinguish between potential nonlinear interactions, those that may affect optimization, and relevant nonlinear interactions, those that do. A similar distinction between potential and relevant nonlinearities has been made in economics by Buchanan and Stubblebine (1962). If an encoded nonlinear effect does not create problems for optimization, then heuristics, optimization techniques, mechanisms, and algorithms developed to overcome the nonlinear effect may be inefficient.

The following claim addresses the simplest cover:

Claim 3.5 $\alpha_1(V) = 2^n$ *iff* $\exists C = \{h^1, h^2, .., h^n\}$ *forming a cover for* V *on* S, *which satisfies:*

(i) $h_i^1 \in \{0, 1\}$

(ii) $h_i^j = *$ *for* $i \neq j$

pf. Suppose $\alpha_1(V) = 2^n$. Let $\hat{C} = \{\hat{h}^1, \hat{h}^2, .., \hat{h}^n\}$ form a cover of size one for V on S. Choose $\tau \in \Phi$, the permutation group on m elements, s.t. $\hat{h}_i^{\tau(i)} \in \{0, 1\}$ It follows that $C = \hat{C}$.

The other direction follows immediately from the definition.

Claim 3.5 can be interpreted as a decentralization (or parallel processing) result. Beginning with any string, maximizing each bit with respect to that string leads to the optimal string. Decisions as to which values to assign to bits need not be coordinated. This does not mean that V contains no nonlinear effects. In the next section, we construct functions with nonlinear terms which nonetheless satisfy the assumptions of Claim 3.5. Such functions have been characterized by Liepins and Vose (1990) as *easy*.

The $\alpha_j(\cdot)$'s can be combined to form the ***complexity vector***. The complexity vector measures the size of the upper contour sets which have covers of various sizes.

The ***complexity vector*** $\alpha(V) = (\alpha_1(V), \alpha_2(V), ..., \alpha_n(V))$

The complexity 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 complexity vectors with larger values are less complex, as measured by cover size, than those mapped to vectors with smaller values. Some simple examples demonstrate how $\alpha(V)$ measures complexity.

Examples:

$$V_1(s) = 8s_1 + 4s_2 + 2s_3 + s_4 - 10s_1 \cdot s_4$$

$$\alpha(V_1) = (4, 16, 16, 16)$$

$$V2(s) = s_1 + s_2 + s_3 + s_4 + 8(1 - s_1) \cdot (1 - s_2) \cdot (1 - s_3)$$

$$\alpha(V_2) = (2, 2, 16, 16)$$

$$V_3(s) = 8(s_1 \cdot s_2 \cdot s_3 + s_1 \cdot s_2 \cdot s_4 + s_2 \cdot s_3 \cdot s_4) - s_1 - s_2 - s_3 - s_4$$

$$\alpha(V_3) = (5, 5, 16, 16)$$

Four features merit attention. First, V_1 has a cover of size two on all of S while the other two functions have covers of size three on S. Thus, at least according to this measure, V_1 appears least complex or most decomposable. Second, the function V_3 has a cover of size one on a larger upper contour set than either V_2 or V_1, which implies that as the function value improves, V_3 becomes easiest. Third, while V_2 has a unique cover of size 3 on S, V_3 has multiple covers of size 3. Finally, although the vector $\alpha(\cdot)$ does not create a complete ordering of functions, a function V might be said to be less complex

than a function $\hat{V}$ if $\alpha_i(V) \geq \alpha_i(\hat{V})$, for $i = 1$ to n. According to this criterion, V_1 and V_3 could be said to be less complex than V_2, but no comparison can be made between V_1 and V_3.

3.2 Measuring Complexity

Covers differ from other measures of complexity by focusing on a function's upper contour sets. Standard nonlinearity/complexity measures count the number and size of encoded nonlinear effects (Kauffman 1989, Liepins and Vose 1990). We refer to these measures as *domain based.* In this section, we show that domain based measures can be misleading. On the one hand, simple nonlinear interactions can combine to form complex problems. On the other hand, complicated nonlinear interactions can collapse to form easy problems.Decomposition size is perhaps the simplest domain based complexity measure for functions defined over binary strings. Before we define decomposition size, we need to introduce the decomposition basis coefficients, which attach a value to each subset of N (Liepins and Vose 1991). If $O(s)$ equals the subset of bits in s which have 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:

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

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

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

Using decomposition size as a measure of complexity, it is possible for simple nonlinear effects to combine to form problems of extreme complexity. In the example below, we construct a function with decomposition size equal to two which forms a problem with a cover of size n. This example can be understood in the context of a multiple public projects model (Page 1993). Suppose there are n potential public projects and that project values are interdependent as in the introductory example with the airport, botanical garden, and cable car system. Decisions on public projects can be modelled as discrete choices, where "yes" is denoted by 1 and "no" is denoted by 0. If we let s_i represent the decision on project i, then a string represents a decision on each project.

In the example below, each individual project has a negative isolated value and each pair of projects has a positive complementarity. It can be shown that the decomposition

size of this problem equals two. However, only when all n projects are undertaken does the combined value of the complementarities outweigh the negative project values, i.e. the problem has a cover of size n.

Example: Assume $n > 2$, and choose the decomposition basis coefficients as follows:

$$\begin{array}{ll} \beta_{V,I} = -1 & if \mid I \mid = 1 \\ \beta_{V,I} = \frac{1}{n-1} + \frac{1}{n-2} & if \mid I \mid = 2 \\ \beta_{V,I} = 0 & if \mid I \mid > 2 \end{array}$$

Choose $s \in S$ s.t. $s \cdot s = k < n$. It is straightforward to show:

$$\begin{aligned} V(s) &= -n + \tfrac{k\cdot(k-1)}{2} \cdot \tfrac{1}{n-1} + \tfrac{1}{n-2} \\ &\leq -n + \tfrac{(n-1)\cdot(n-2)}{2} \cdot \tfrac{1}{n-1} + \tfrac{1}{n-2} \\ &= -n + \tfrac{n-2}{2} + \tfrac{n-1}{2} \\ &= -\tfrac{3}{2} \end{aligned}$$

Choose $s \in S$ s.t. $s \cdot s = n$, i.e. $s_i = 1 \; \forall i$. It is straightforward to show:

$$\begin{aligned} V(s) &= -n + \tfrac{n\cdot(n-1)}{2} \cdot \tfrac{1}{n-1} + \tfrac{1}{n-2} \\ &= -n + \tfrac{n}{2} + \tfrac{n}{2} + \tfrac{1}{2n-4} \\ &= \tfrac{n}{2n-4} \\ &> 0 \end{aligned}$$

Therefore, the highest valued string of projects equals the set of all projects, and the second highest valued string of projects equals the set of no projects. It follows that n equals the minimal cover size for any upper contour set containing more than *two strings*. Along similar lines, Goldberg (1990) has shown that complex problems can arise from simple interactions. Using the Walsh Basis, Goldberg combines nonlinear interactions of decomposition size two and three to form problems which are deceptive for genetic algorithms. Roughly speaking, a problem is deceptive if hyperplanes whose strings have above average value do not contain the optimal string.

Measuring complexity by encoded effects may also *overstate* the amount of relevant complexity. In the decomposition basis, maximum complexity occurs when the coefficient

of the subset of all variables has a positive coefficient, i.e. $size_d(V) = n$. The example below is a function of decomposition size n which has a cover of size 1.

Example: Choose $V \in F$ s.t. $\beta_{V,I} > 0 \ \forall I \subset N$. It is straightforward to show that $size_d(V) = n$ but that V has a cover of size 1.

3.3 Test Functions

In this section, we report data from numerical simulations on test functions. We created two classes of test functions each with a decomposition size of two and also looked at functions with random values. We then measured the cover size of randomly drawn functions from the three classes of functions. The first two classes of functions have the same functional form. The difference is in the relative size of the nonlinear effects. Note that with probability one, any function drawn from either class has a decomposition size of two. In the first class of test functions, the nonlinear terms are half as large as the linear terms:

$$V_1(s) = \sum_{i=1}^{n} \beta_i \cdot s_i + \sum_{i=1}^{n} \sum_{j=1}^{n} \beta_{ij} \cdot s_i \cdot s_j \qquad \beta_i \in [-2, 2],\ \beta_{ij} \in [-1, 1]$$

In the simulations both β_i and β_{ij} were uniformly distributed. We varied n, the number of bits, and found that cover size tended to increase with n. The table below shows that for $n = 6$ that only 10% of the randomly generated functions of type V_1 had a cover size of six or greater, but that for $n = 10$, 88% of the functions had a cover size of six or greater. This occurs because the number of nonlinear effects increases more than linearly with n. The data from one hundred simulations are given below:

% of functions of type V_1

# *Bits*	*Cover Size*					
n	1	2	3	4	5	≥ 6
5	1	25	34	30	10	0
6	1	11	25	31	22	10
7	0	2	10	27	34	27
8	0	0	3	24	29	44
9	0	0	2	8	9	81
10	0	0	2	2	8	88

100 trials in each cell

In the second class of functions, the nonlinear terms are only one fourth as large as the linear terms:

$$V_2(s) = \sum_{i=1}^{n} \beta_i \cdot s_i + \sum_{i=1}^{n} \sum_{j=1}^{n} \beta_{ij} \cdot s_i \cdot s_j \qquad \beta_i \in [-4,4],\ \beta_{ij} \in [-1,1]$$

As before, β_i and β_{ij} are uniformly distributed. The data from test functions randomly drawn from the second class of function are given below:

% of functions of type V_2

# *Bits*	*Cover Size*					
n	1	2	3	4	5	≥ 6
5	13	37	27	17	6	0
6	4	33	33	17	12	1
7	2	16	19	30	21	12
8	0	7	27	25	20	21
9	0	2	11	24	24	39
10	0	2	4	7	30	57

100 trials in each cell

As might be expected, the smaller nonlinear effects result in smaller cover sizes and as before, the probability of the cover size being greater than or equal to six increases with the number of bits. Two hypothesis may be formulated from these data. First, we see that decomposition size and cover size may differ substantially. Second, for a fixed n, functions drawn from the same class have similar cover sizes. Late we discuss how these similarities may be exploited.

Finally, we briefly consider functions whose values are randomly determined. We let $V_r(s) = \beta_s$ where β_s was randomly drawn from the uniform distribution on $[0,1]$. The data below suggest that cover size is nearly equal to string length.

% of functions of type V_r

# *Bits*	*Cover Size*					
n	1	2	3	4	5	≥ 6
4	0	0	18	82	0	0
5	0	0	1	6	93	0
6	0	0	0	0	0	100
7	0	0	0	0	0	100

100 trials in each cell

4 Covers and Optimization Theory

In this section, we show how covers can be used to select optimal parameters for a class of hillclimbing algorithms. We also provide an alternative explanation for the performance of genetic algorithms in terms of covers.

4.1 ALGO(r,p)'s

In this section, we define a class of hillclimbing algorithms called $ALGO(r,p)$ where r equals the maximum number of bits selected in one iteration and p equals the probability that each bit is switched. The term *ALGO* is borrowed from the work of Reiter and Sherman (1965). The algorithms used here differ slightly from their algorithms which learned regions of the domain from which to select initial strings. In our formulation, $ALGO(r,p)$ begins with a randomly generated string.

The search algorithm $ALGO(r,p)$ is defined as follows:

Step 1:	Choose $s^A \in S$		
Step 2:	Randomly select $I \subset N$ s.t. $\mid I \mid = r$		
Step 3:	Create s^t as follows:		
	$s_i^t = s_i^A$		if $i \notin I$
	$s_i^t = (1 - s_i^A)$	with probability p	if $i \in I$
	$s_i^t = s_i^A$	with probability $(1-p)$	if $i \in I$
Step 4:	If $f(s^t) > f(s^a)$ then $s^A = s^t$		
Step 5:	Goto Step 2		

$ALGO(r,p)$ compares the current best string, s^A, to a chosen string, s^t, which differs by at most r bits. A string s^* belongs to the set of local optima w.r.t. $ALGO(r,p)$ if any string which differs by r bits or fewer has a strictly lower value under f.[2]

A string s^* belongs to the *set of local optimum* w.r.t ALGO(r,p), LO(r,p) iff $\forall s$ s.t. $\mid \{i : s_i \neq s_i^*\} \mid \leq r$, $V(s) < V(s^*)$

Claim 4.1 below provides a sufficient condition for an ALGO(r,p) to locate a string dominant hyperplane.

[2]Recall that we assume that no two strings have the same value.

Claim 4.1 *$\forall h \in H$ s.t. $\sigma(h) \leq r$ and h is string dominant for V on$S$, it follows that $s^* \in h$ $\forall s^* \in LO(r,p)$.*

pf. $\sigma(h) \leq r$ implies $\{i : h \wedge s_i \neq s_i^*\} \leq r$. By assumption, if $h \wedge s^* \neq s*$, then $V(h \wedge s*) > V(s^*)$, which completes the proof.

A corollary of Claim 4.1 is that $ALGO(r,p)$ locates any cover of size less than or equal to r.

Corollary 4.1 *If $s^A \in T(\alpha_j(V))$ and $r \geq j$, then $s^* \in LO(r,p)$ optimizes V.*

pf. Suppose $s^* \in$ LO(r,p). By definition, of $T(\alpha_j(V))$, $\exists C$, a cover for V on $T(\alpha_j(V))$ s.t. $Z(C) \leq j$. By assumption, $s^A \in T(\alpha_j(V))$, therefore, $s^* \in T(\alpha_j(V))$. By Claim 4.1, $\forall h \in C, s^* \in h$.

From Corollary 4.1 we see that the vector $\alpha(V)$ reveals information about how r should be chosen. A function V_1 which satisfies $\alpha_1(V_1) = 2^n$ can be solved with $ALGO(1,p)$ for any positive p. Alternatively, a function V_2, such that $\alpha_{20}(V) = 1$ probably would not be optimized by $ALGO(r,p)$ for r small.

An implication of Corollary 4.1 is that if the distribution of cover sizes for a particular class of functions is known or approximated, then $\alpha(V)$ can be used to compute the probability that a local optimum is global. In the previous section, the data suggested that 99% of functions of type V_2 have covers of size 5 or less. Suppose further that 99% of the functions of type V_2 satisfy $\alpha_2(V) > \frac{|S|}{10}$. A search algorithm which randomly generates thirty strings and applies $ALGO(2,p)$, where $r > 0$ to the string with the highest value would yield a global optimum with a probability strictly greater than (.99)y[1-(.9)30] = .948. This lower bound is strict given that each new local optima moves the search to a higher contour set and increases the probability that $ALGO(2,r)$ finds the global optimum.

Cover size may help to explain why simple algorithms are often effective at finding solutions. For example, finding an optimal tour for a travelling salesperson problem (TSP) is NP hard. Reiter and Sherman (1965) and Kauffman and Levin (1987) attempted to solve TSPs using algorithms similar to our $ALGO$'s. Their algorithms switch cities within a tour. Both studies found that increasing the number of cities switched in a given iteration improved performance but that these improvements dropped off sharply. For instance, Kauffman and Levin showed for TSPs that a variant of $ALGO(3,1)$ performed almost as well as a variant of $ALGO(4,1)$. In other words, their TSPs may have had covers of size 3 for large portions of the domain.

The performance of an $ALGO(r,p)$ depends both on the parameter p and on r, the maximal number of bits switched. The following example shows how changing r changes the probability of locating a particular hyperplane.

Example: Let $n = 5$, and $h = 11 * **$, be string dominant for V on S, and $s^A = 00000$. Consider the performances of the following two $ALGO(r,p)$'s:

ALGO($3,\frac{1}{2}$):

In Step 2, prob$\{d(h) \subset I\} = 0.3$

In Step 3, prob$\{s_i^t = 1 \Leftrightarrow i \in d(h)\} = 0.25$

Therefore, after Step 3, prob$\{s_i^t \in h\} = 0.075$

ALGO($2,\frac{1}{2}$):

In Step 2, prob$\{d(h) \subset I\} = 0.1$

In Step 3, prob$\{s_i^t = 1 \Leftrightarrow i \in d(h)\} = 0.25$

Therefore, after Step 3, prob$\{s_i^t \in h\} = 0.025$

Claim 4.3 states a more general result. The idea of the claim is that to maximize the probability of switching bit values on a given subset of size k, that the optimal r for a given $ALGO(\cdot,p)$ varies directly with k and inversely with p. We first define the probability of switching a given subset using $ALGO(r,p)$.

The *probability of switching* h in $ALGO(r,p)$,
pr$(h,ALGO(r,p))$ = prob$\{$ $d(h) = \{i : s^t{}_i \neq s^A{}_i\}$: ALGO(r,p)$\}$

Claim 4.2 *If* $\sigma(h) = k$ *and* $r \geq k$, *then* $pr(h,ALGO(r,p)) = p^k \cdot (1-p)^{r-k} \cdot \frac{r!\cdot(n-k)!}{n!\cdot(r-k)!}$

pf. In Step 2 of $ALGO(r,p)$:

prob$\{\sigma(h) \subset I\} = \frac{r!\cdot(n-r)!\cdot(n-k)!}{n!\cdot(n-r)!\cdot(r-k)!}$

$= \frac{r! \cdot (n-k)!}{n! \cdot (r-k)!}$

In Step 3 of $ALGO(r,p)$: $\text{prob}\{s_i^t = s_i^A \Leftrightarrow i \in d(h)\} = p^k \cdot (1-p)^{(r-k)}$, which completes the proof.

Claim 4.3 *If $\sigma(h) = k$, then argmax pr(h,ALGO$(\cdot,p)$) = $\lfloor k/p \rfloor$, where $\lfloor a \rfloor$ equals the greatest integer less than or equal to a.*

pf: From Claim 4.2, pr(h,ALGO(r,p)) $= p^k \cdot (1-p)^{r-k} \cdot \frac{r! \cdot (n-k)!}{n! \cdot (r-k)!}$

Let $g(r,k,p) = p^k \cdot (1-p)^{r-k} \cdot \frac{r!}{(r-k)!}$.

It follows that pr(h,$ALGO(r,p)$)= g(r,k,p)$\cdot \frac{(n-k)!}{n!}$

Therefore, it suffices to maximize $g(r,k,p)$ with respect to r.

Let $G(r) = \frac{g(r,k,p)}{g(r-1,k,p)} = r \cdot \frac{1-p}{r-k}$.

Therefore, $G(r) \geq 1 \Leftrightarrow r \leq \frac{k}{p}$, and further, $G'(r) < 0$, which completes the proof.

Note that in the special case where $p = 1/2$, Claim 4.3 states that $ALGO(2k, 1/2)$ maximizes the probability of switching a hyperplane of size k. Claim 4.3 provides a hint as to how r should be chosen in $ALGO(r,p)$ to find a cover of a given size. Suppose that a function V has a cover on S of size 8, but it has a cover on $T(2^{n-2})$ of size 4. Initially, r should be large to find the hyperplanes of larger size, but once these hyperplanes are located and the function value has improved, r should be decreased. The Corollary below states that p should be set equal to one and r equal to $\sigma(h)$ to maximize pr(h,$ALGO(r,p)$).

Corollary 4.2 *If $\sigma(h) = k$, then pr(h, $ALGO(k,1)$) $\geq$ pr(h,$ALGO(r,p)$) for all $r \in N$ and $p \in [0,1]$.*

pf: From Claim 4.3, pr(h,$ALGO(r,p)$)= g(r,k,p)$\cdot \frac{(n-k)!}{n!}$

where $g(r,k,p) = p^k \cdot (1-p)^{r-k} \cdot \frac{r!}{(r-k)!}$.

Note that $g(r,k,p) = \frac{1}{k!} \cdot f(k:r,p)$, where $f(k:r,p)$ is the standard binomial distribution. It follows immediately that $r = k$ and $p = 1$ maximizes f, which completes the proof.

4.2 Genetic Algorithms

A Genetic Algorithm (hereafter GA) is a constant-size population based search algorithm (Holland 1975, Goldberg 1989). Recently, GAs have been used in economics (Arifovic 1989, Marimon, McGrattan, and Sargent 1990, Holland and Miller 1991), political science (Kollman, Miller, and Page 1992), and in the study of inductive learning (Holland, Holyoak, Nisbett, and Thagard 1989). In this section, we provide a brief introduction into GAs and use cover theory to develop an alternative explanation for their performance.

A GA begins with an population of M strings which it transforms into a new population. Each iteration of a GA is called a generation. We denote the population in generation t by p_t. The transition from p_t to p_{t+1} occurs in three stages: reproduction, crossover, and mutation. Reproduction chooses a new population of M strings from the existing population according to the function values of the strings. The idea is that better (more "fit") strings are reproduced with greater frequency, thus increasing the fitness of the population. Let p_i be the population after reproduction. The reproduction we describe relies on a tournament to select strings.

Tournament Selection: M pairs of strings from p_t are randomly created. The higher valued of the two strings in each pair belongs to p_i.

A crossover operator creates new strings by "crossing" strings from among those reproduced. The analogy to be kept in mind is genetic recombination with the bit values thought of as alleles. The bit value of a string created during crossover may come from either parent. Crossover randomly creates M pairs of strings, and each pair independently crosses bits with probability p (typically $p \in [.25, .75]$). The crossing or exchanging of bit values occurs on a subset of the positions. The positions which cross bit values can be chosen in many ways. Three typical crossover rules applied to strings s and t are *one point*, *two point*, and *uniform crossover*. In the tests described later in this section, we employ uniform crossover. There are two reasons for this decision. First, with one or two point crossover, hyperplanes with large distances between defined bits are more likely to get destroyed, making the distance between defined bits a more natural definition of hyperplane size. With uniform crossover, it can be shown that the more defined bits in a hyperplane, the measure used in cover size, the greater the probability that the hyperplane is destroyed during crossover. Second, uniform crossover treats all bits symmetrically, one and two-point crossover have endpoint bias effects.

Uniform crossover: A "switching rule", x, is created by selecting a string from the set S. If $x_i = 1$, then the strings s and t switch their ith bit values. If $x_i = 0$, then the strings s and t do not switch their ith bit values.

Bit: x_1 x_2 x_3 x_n

Switching rule:	**0**	1	1	**0**
new s	s_1	t_2	t_3	s_n
new t	t_1	s_2	s_3	t_n

Crossover can "destroy" hyperplanes. A hyperplane h is destroyed during crossover, if one of the strings belonged to h prior to crossover, but neither string belongs to h after crossover.

Example: Suppose $h = *0*1$. Let $s = 0011$ and $t = 0110$. It follows that $s \in h$ and $t \notin h$. Suppose that s and t are uniformly crossed using the switching rule 1100. Neither resulting string, $\acute{s} = 0010$ or $\acute{t} = 0111$ belongs to h.

Finally, mutation may occur at the bit level or the string level. Bit mutation switches the value of each bit of each string with a very small probability ($p < 0.1$). Bit mutation is an analogous to biological mutation. Bit mutation creates both short and long term effects. In the short term, a mutant represents a single random search for a better string. In the long term, a bit mutation may spread through the population and slow the rate of convergence of bit values. In contrast to the milder bit mutation, string mutation creates an entirely random string. Typically, the probability of string mutation is also small ($p < 0.05$). String mutation allows for random searches in entirely different regions of the domain, which can enable the GA to avoid getting stuck at a local peak. After mutation, the new population, p_{t+1}, is completed; reproduction, crossover, and mutation are reapplied to create p_{t+2}. The randomness created in the mutation stage guarantees that the population never converges to a single string replicated M times.

The performance of GAs has been the focus of a great deal of research in recent years. To date, the most important result in the theory of GA performance is the Schema Theorem (Holland 1975), which says that hyperplanes whose strings consistently have higher than average function values will increase in number in the population proportionate to their fitness advantage in the population. Decreasing a hyperplane's size (or defining length for one and two-point crossover) increases the probability that a hyperplane survives crossover.

One interpretation of the Schema Theorem is the "building block" hypothesis, which says that GAs increase the number of hyperplanes of small size whose strings have above average value and combine them to form above average strings. These hyperplanes commonly are referred to as "building blocks." A hyperplane is above average if the strings in the population which lie in the hyperplane have, on average, higher value than the strings not in the hyperplane.

Greffenstette and Baker (1989), among others, criticize the Schema Theorem. They argue that GA performance can be altered drastically by monotonically transforming the

objective function V. In contrast, the ALGO(r,p)'s described in the previous section depend only upon ordinal rankings of strings. A monotonic transformation of V would have no impact on the performance of an ALGO(r,p).

Covers offer an alternative perspective on GA performance. Claim 4.3 states that to maximize the probability of switching all of the bits in a hyperplane h, the optimal size of r $ALGO(r,p)$ depends upon the number of defined bits of h. Claim 3.4 states that for any function, the cover size decreases as the function value improves. Taken together these two claims suggest that a good search algorithm should switch many bits early to avoid local optima and, as the search moves up the contour sets and the cover size decreases, switch fewer bits. This is a defining principle of another search procedure known as simulated annealing (Davis 1988).

Ideally, a search algorithm would adapt the number of bits that it switches as a function of its climb up the contour sets. Optimal adaptation to a function requires information about the function's contour sets. In a crude way, a GA learns the function's contour sets and adapts the number of bits it switches. If the same hyperplanes of small size consistently obtain above average values, i.e. if building blocks are consistent, then the population converges quickly. If the best hyperplanes vary between generations, a GA's population converges slowly. A function with a cover of small size is likely to have consistent building blocks during the search. A function with a large cover size is likely to have inconsistent building blocks.

To test this hypothesis, we experimented on two functions with a GA with tournament selection uniform crossover, and bit mutation. We ran the GA one hundred times on each function. The first function was linear in the bits:

$$V_1(s) = s_1 + s_2 + ... + s - n$$

Its building blocks should be consistent. The second function was constant:

$$V_2(s) = 10$$

The second function should not have consistent building blocks. Table 1 shows the average number of bits changed per crossover. As the theory above suggests, the crossover operator switched more bits for the constant function than for the linear function. The results described in Table 1 capture a basic relationship between the size of a GA search and the vector $\alpha(V)$. In future research, we hope to identify more subtle effects of $\alpha(V)$ on GA search.

5 Discussion

Covers formalize the idea that large complex functions can be decomposed into simpler subproblems which then can be solved in parallel. Covers can be used to construct a complexity vector which measures a function's complexity relative to its contour sets. Finally, covers can be used in the development and evaluation of search algorithms.

As an extension of covers, Richardson (1991) has advanced the idea of ϵ-covers, a less restrictive construction, which allows the theory to accept anomalous function values. An ϵ-cover consists of "almost" string dominant hyperplanes - hyperplanes which are string dominant except on a small subset. To formalize this notion, define h as ϵ-string dominant on $T(\alpha)$ if the proportion of strings in $T(\hat{\alpha})$ that satisfy $V(h \wedge s) \geq V(s)$ is greater than $(1 - \epsilon)$, where $\hat{\alpha} \leq \alpha$. An ϵ-cover is a collection of ϵ-string dominant hyperplanes the union of whose defined bit is N. A consequence of this definition is that the number of strings for which an ϵ-string dominant hyperplane is not string dominant must decrease as the search moves to higher contours.

References

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

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

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

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

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

J. Greffenstette, 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 1989)

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

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

J. Holland, K. Holyoak, R. Nisbett, and P. Thagard, Induction: Processes of Inference, Learning, and Discovery, (MIT Press 1989)

S. Kauffman, "Adaptation on Rugged Fitness Landscapes," pp 527-619, in Lectures in the Sciences of Complexity, (Addison Wesley, Reading MA. 1989)

S. Kauffman, and S. Levin, "Towards a General Theory of Adaptive Walks on Rugged Landscapes," Journal of Theoretical Biology, 1987 128(11).

K. Kollman, J. Miller, and S. Page, "Adaptive Parties in Spatial Elections", American Political Science Review, forthcoming.

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

G. Liepins, and M. Vose, "Polynomials, Basis Sets, and Deceptiveness in Genetic Algorithms," Complex Systems, 1991 5(1), 45-62.

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

S. Page, and D. Richardson, "Walsh Functions and Schema Variance," Complex Systems, forthcoming.

S. Page, "Public ProjectS," mimeo California Institute of Technology, 1993.

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

D. Richardson, private communication 1991.

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

Table 1
Average Number of Bits Switched
100 Trials
Genetic Algorithm with Uniform Crossover and Tournament Selection

	$V_1(s) = s_1 + s_2 + ..s_n$	$V2(s) = 10$
	Number of	*Number of*
Generation	*Bits Switched*	*Bits Switched*
1	6.89	7.09
2	5.96	6.72
3	5.23	6.28
4	3.91	6.55
5	2.78	6.18
6	1.75	6.01
7	1.02	5.82
8	0.80	5.51
9	0.74	5.56
10	0.76	4.95
11	0.62	5.18
12	0.60	4.70
13	0.51	4.88
14	0.52	4.61
15	0.54	4.05
16	0.64	4.55
17	0.62	4.50
18	0.45	4.35
19	0.43	3.85
20	0.52	3.94

pmut = 0.05 pcross = 0.5