Upper Bounds on the Time and Space Complexity of Optimizing Additively Separable Functions

Matthew J. Streeter

Computer Science Department and
Center for the Neural Basis of Cognition
Carnegie Mellon University
Pittsburgh, PA 15213
matts@cs.cmu.edu

Abstract. We present upper bounds on the time and space complexity of finding the global optimum of additively separable functions, a class of functions that has been studied extensively in the evolutionary computation literature. The algorithm presented uses efficient linkage discovery in conjunction with local search. Using our algorithm, the global optimum of an order-k additively separable function defined on strings of length ℓ can be found using $O(\ell \ln(\ell) 2^k)$ function evaluations, a bound which is lower than all those that have previously been reported.

1 Introduction

A large amount of evolutionary computation research deals with formally defined classes of problems intended to capture certain aspects of real-world problem difficulty. One prominent example of such a class of problems is N-K landscapes [5]. Finding the global optimum of such functions has been shown to be an NP-complete problem for $K \geq 2$ [10, 13]. Another widely studied problem class is additively separable functions, defined by

$$f(s) = \sum_{i=1}^{m} f_i(s)$$

where each subfunction $f_i(s)$ depends on at most k string positions, and the number of subfunctions that depend on a particular position j is at most one. This class of problems plays a central role in the study of *competent* genetic algorithms [1], and is the focus of this paper. We will assume we are seeking the maximum of an additively separable function of order k defined on strings of length ℓ.

Optimizing such functions is primarily a problem of *linkage discovery*: determining which pairs of positions belong to the same subfunction. There is much research in the evolutionary computation literature on linkage discovery, including work on messy codings and operators [2], the linkage-learning genetic algorithm [3], and probabilistic model-building genetic algorithms [6, 8, 9]. Once the linkage structure of the function is known, the global optimum can be found in $O(2^k \ell / k)$ evaluations by greedy search.

Previous work has shown that additively separable problems of order k defined on strings of length ℓ can be solved using $O(2^k\ell^2)$ function evaluations [4, 7][1,2]. Empirical evidence as well as facetwise decompositional models suggest that sub-quadratic ($O(2^k\ell^\beta)$ for $\beta < 2$) performance is possible. For example, the Bayesian Optimization Algorithm (BOA) is estimated to require $O(2^k\ell^{1.65})$ function evaluations [8]. One of the contributions of this paper is to rigorously prove that this and much better performance is in fact possible.

This paper addresses the problem of optimizing additively separable functions from a traditional algorithms perspective. We present an algorithm, prove its correctness, and prove upper bounds on the number of function evaluations it requires. For now our sole concern is finding the global optimum of order-k additively separable functions; we are not worried about whether the resulting algorithm also performs well on real optimization problems such as MAXSAT. The algorithm presented in this paper will be shown to find a global optimum of an order-k additively separable function using $O(2^k\ell\ln(\ell))$ function evaluations, a value which is $O(2^k\ell^{1+\varepsilon})$ for any $\varepsilon > 0$. This performance claim is confirmed experimentally. We obtain more than an order of magnitude improvement over BOA on a set of problems previously studied by Pelikan [9].

The following subsection introduces definitions and notation that are used throughout the paper. Section 2 gives an overview of how our optimization and linkage discovery algorithms work. Section 3 presents the linkage discovery algorithm in full detail and analyzes its time complexity. Section 4 presents our optimization algorithm in full detail, and section 5 presents experiments illustrating it in action. Section 6 discusses the implications and limitations of this work, and section 7 concludes the paper.

Due to space limitations, some of the proofs of the lemmas and theorems given in this paper have been omitted from the text. These proofs are available online at [11], where a Java implementation of our algorithm is also available.

1.1 Definitions and Notation

In this paper we are interested in maximizing a function f defined over binary strings of some fixed length ℓ. We adopt the following notation:

$s.i \equiv$ the value of the i^{th} character of string s
$s[i \rightarrow x] \equiv$ a copy of string s with the i^{th} character set to x

Definition 1. $\Delta f_i(s) \equiv f(s[i \rightarrow (1\text{-}s.i)]) - f(s)$. ■

In other words, $\Delta f_i(s)$ represents the change in fitness that results from flipping the i^{th} bit of s. This notation was introduced by Munemoto [7].

---

[1] The bound was derived for linkage discovery, but clearly holds for optimization as well.

[2] This bound assumes that the probability of discovering all links within a particular subfunction, and not all links within all subfunctions, is to remain constant as ℓ increases.

Definition 2. Two positions i and j are *linked* w.r.t. a function f (written $\mathcal{L}_f(i, j)$) if there is some string s such that $\Delta f_i(s[j\rightarrow 0]) \neq \Delta f_i(s[j\rightarrow 1])$. ■

In other words, i and j are linked if, in the context of some string s, i's affect on fitness can depend on the value of j. Note that $\mathcal{L}$ is symmetric but not reflexive or transitive.

Definition 3. Two positions i and j are *grouped* w.r.t. a function f (written $\mathcal{G}_f(i, j)$) if $i=j$, if $\mathcal{L}_f(i, j)$, or if there is some sequence of positions $i_1, i_2, ..., i_n$ such that $\mathcal{L}_f(i, i_1)$, $\mathcal{L}_f(i_1, i_2)$, ..., $\mathcal{L}_f(i_{n-1}, i_n)$, and $\mathcal{L}_f(i_n, j)$. ■

$\mathcal{G}$ is thus the reflexive, transitive closure of $\mathcal{L}$. We shall omit the subscripts on $\mathcal{L}$ and $\mathcal{G}$ when the choice of f is obvious.

Definition 4. A *linkage group* of a function f is a set γ of one or more integer positions such that if $i \in \gamma$ then $j \in \gamma$ iff. $\mathcal{G}(i, j)$. ■

Definition 5. Let f be a function with m linkage groups $\gamma_1, \gamma_2, ... \gamma_m$. Then $\Gamma=\{\gamma_1, \gamma_2, ... \gamma_m\}$ is the *linkage group partition* of f. ■

It is easy to see that each string position i belongs to exactly one linkage group, so that a linkage group partition as just defined is in fact a partition of the set of positions $\{1, 2, ..., \ell\}$. We adopt the notation:

$$\Gamma[j] \equiv \text{the linkage group } \gamma \in \Gamma \text{ such that } j \in \gamma.$$

The following example will serve to illustrate these definitions. Let

$$f(s) = (s.1)(s.2) + (s.2)(s.3) + (s.4)(s.5) .$$

Then f is an additively separable function of order 3 with the two subfunctions $f_1(s) = (s.1)(s.2) + (s.2)(s.3)$ and $f_2(s) = (s.4)(s.5)$. The linkage relationships that exist w.r.t. f are:

$$\mathcal{L}_f(1,2), \mathcal{L}_f(2,3), \mathcal{L}_f(4,5),$$
$$\mathcal{L}_f(2,1), \mathcal{L}_f(3,2), \mathcal{L}_f(5,4)$$

the linkage groups are $\gamma_1=\{1,2,3\}$ and $\gamma_2=\{4,5\}$, and the linkage group partition Γ is $\{\gamma_1, \gamma_2\}$.

Intuitively, it should be clear that there is a one-to-one relationship between subfunctions and linkage groups. Namely, if f has m linkage groups $\gamma_1, \gamma_2, ... \gamma_m$ then we can write f as a sum of subfunctions $f_1, f_2, ..., f_m$ where each f_i depends only on the characters at the positions in γ_i. We prove this formally in theorem 2.

2 Overview of the Algorithm

This section gives a high-level overview of how our algorithm works.

The algorithm uses a randomized test that is guaranteed to succeed with probability at least 2^{-k} to detect linkage (interaction) between a given position i and some other position in the string. When the test succeeds it reveals two strings s and s' such that $\Delta f_i(s) \neq \Delta f_i(s')$, and the algorithm then uses binary search to isolate a specific position j that is linked to i. Each newly discovered link is used to update a linkage group partition maintained by the algorithm. The algorithm then performs a local search centered around the best-so-far known string that requires at most 2^k steps, and once the linkage group partition is fully discovered this local search is guaranteed to find a globally optimum string. The algorithm never discovers the same link twice, and so long as it is run long enough to discover the complete linkage group partition it is guaranteed to return a globally optimum string. Figure 1 presents high-level pseudocode for this algorithm.

Two previous algorithms for linkage discovery used the same randomized test for linkage as used in this paper [4, 7], but instead of using binary search restricted each instance of the test to detect linkage between a specific pair of positions i and j, resulting in a requirement[2] of $O(2^k\ell^2)$ rather than $O(2^k\ell\ln(\ell))$ function evaluations.

```
Procedure ASFOPTIMIZE(f, t):
1. Initialize Ω to a random string.
2. Initialize Γ to {{1},{2}, ..., {ℓ}}.
3. Use local search to make Ω optimal with respect to one-bit perturbations.
4. Do t times:
    For i from 1 to ℓ:
        Perform randomized test for linkage between i and some other position.
        If test succeeds then:
            Use binary search to find j such that L(i, j).
            Update linkage group partition Γ.
            Use local search to make Ω optimal with respect to newly discovered
            linkage group Γ[i] ∪ Γ[j].
5. Return Ω.
```

Fig. 1. High-level overview of the optimization algorithm presented in this paper.

3 Detecting Linkage

This section formally defines the procedures used by our optimization algorithm to discover the linkage group partition for an arbitrary function. Figure 2 and lemmas 1-2 concern our binary search procedure. Figure 3 and lemmas 3-4 concern our randomized test for linkage. Figures 4-5, lemmas 5-7, and theorem 1 concern our algorithm for finding the linkage group partition of an arbitrary function.

Procedure FINDLINKEDPOSITION(f, s, s', i):
1. assert($\Delta f_i(s) \neq \Delta f_i(s')$)
2. Let $j_1, j_2, ..., j_\delta$ be the δ distinct positions for which the values of s and s' differ.
3. If δ=1, return j_1.
4. Let $s_2 = s'[j_1 \rightarrow s.j_1][j_2 \rightarrow s.j_2]...[j_{\lfloor \delta/2 \rfloor} \rightarrow s.j_{\lfloor \delta/2 \rfloor}]$.
5. If $\Delta f_i(s) \neq \Delta f_i(s_2)$ return FINDLINKEDPOSITION(s, s_2, i) else return FINDLINKEDPOSITION(s', s_2, i).

Fig. 2. Procedure FINDLINKEDPOSITION. Given strings s and s' such that $\Delta f_i(s) \neq \Delta f_i(s')$, the procedure performs a binary search to return a position j such that $\mathcal{L}(i, j)$.

Lemma 1. If $\Delta f_i(s) \neq \Delta f_i(s')$, FINDLINKEDPOSITION($f$, s, s', i) returns a value j such that $\mathcal{L}(i, j)$ is true.

Proof: By induction on the hamming distance, δ, between s and s'.

Case δ=1: If s and s' differ only in one position there must be some j such that $s' = s[j \rightarrow s'.j]$. Assume without loss of generality that $s'.j = 0$. Then $s.j = 1$, so $s = s[j \rightarrow 1]$ while $s' = s[j \rightarrow 0]$. Thus by our assumption that $\Delta f_i(s) \neq \Delta f_i(s')$ we have $\Delta f_i(s[j \rightarrow 1]) \neq \Delta f_i(s[j \rightarrow 0])$, so by definition $\mathcal{L}(i, j)$ is true.

Case δ>1: In step 4, FINDLINKEDPOSITION creates a string s_2 that differs from s' in $\lfloor \delta/2 \rfloor$ positions and from s in $\lceil \delta/2 \rceil$ positions. Suppose $\Delta f_i(s) \neq \Delta f_i(s_2)$. In this case, the value j returned in step 5 is FINDLINKEDPOSITION(s, s_2, i), so $\mathcal{L}(i, j)$ is true by the induction hypothesis. Otherwise if $\Delta f_i(s) = \Delta f_i(s_2)$, then because $\Delta f_i(s) \neq \Delta f_i(s')$ (by assumption) it must be that $\Delta f_i(s') \neq \Delta f_i(s_2)$. In this case the value j returned in step 4 is FINDLINKEDPOSITION(s', s_2, i), so by the induction hypothesis we have $\mathcal{L}(i, j)$. ■

Lemma 1 establishes that FINDLINKEDPOSITION performs a binary search to isolate a position j such that $\mathcal{L}(i, j)$ is true. Lemma 2 then follows from the fact that binary search on a set of δ positions requires at most $\lceil \lg(2\delta) \rceil$ iterations.

Lemma 2. If $\Delta f_i(s) \neq \Delta f_i(s')$, FINDLINKEDPOSITION($f$, s, s', i) performs no more than $4*(\lceil \lg(2\ell) \rceil - 1)$ function evaluations. ■

Procedure TESTFORLINK(f, Γ, i):
1. s := RANDOMSTRING().
2. s_{wrk} := RANDOMSTRING().
3. Let $j_1, j_2, ..., j_n$ be the n positions in $\Gamma[i]$.
4. s' := $s_{wrk}[j_1 \rightarrow s.j_1][j_2 \rightarrow s.j_2]...[j_n \rightarrow s.j_n]$
5. If $\Delta f_i(s) \neq \Delta f_i(s')$, return FINDLINKEDPOSITION(f, s, s', i).
6. Otherwise return -1.

Fig. 3. Procedure TESTFORLINK. The procedure performs a randomized test for linkage and returns either a position j such that $\mathcal{L}(i, j)$ is true or returns -1 if the test is inconclusive.

Lemma 3. If TESTFORLINK(f, Γ, i) returns a value j then either (a) $j = -1$ or (b) $\mathcal{L}(i, j)$ and $j \notin \Gamma[i]$.

Proof: If TESTFORLINK returns at line 6 the lemma is trivially satisfied. Otherwise the procedure must return at line 5, and by lemma 1 it must return a value j such that $\mathcal{L}(i, j)$ is true. To see that $j \notin \Gamma[i]$, note that line 4 guarantees that s and s' do not differ in any of the positions in $\Gamma[i]$, so it is sufficient to show that if FINDLINKEDPOSITION(f, s, s', i) returns a value $j \neq -1$ then $s.j \neq s'.j$. This can easily be shown by induction on the number of times n that FINDLINKEDPOSITION recurses. For n=0, we see by inspection of line 3 of FINDLINKEDPOSITION that $s.j \neq s'.j$. For n>0, the induction hypothesis gives either $s.j \neq s_2.j$ or $s'.j \neq s_2.j$, and it follows from inspection of lines 2 and 4 that $s.j \neq s'.j$. ∎

In other words, TESTFORLINK only finds links that allow us to make a meaningful update to Γ.

Lemma 4. If there exists a j such that $\mathcal{L}(i, j)$ and $\Gamma[i] \neq \Gamma[j]$, then TESTFORLINK(f, Γ, i) returns a value other than -1 with probability at least $2^{-|\gamma|}$, where γ is the linkage group to which i belongs.

Proof: Suppose there exists a j such that $\mathcal{L}(i, j)$ and $\Gamma[i] \neq \Gamma[j]$. By definition of $\mathcal{L}$, there must be some string $s_{\mathcal{L}}$ such that $\Delta f_i(s_{\mathcal{L}}[j \rightarrow 0]) \neq \Delta f_i(s_{\mathcal{L}}[j \rightarrow 1])$. Let S denote the set of strings that agree with $s_{\mathcal{L}}$ on all positions in $\Gamma[i]$. Letting s and s' denote the strings created on lines 1 and 4, respectively, of TESTFORLINK, we note three things. First, if $s \in S$ then $s' \in S$ by inspection of line 4. Second, if $s \in S$ then there must be some $r \in S$ such that $\Delta f_i(s) \neq \Delta f_i(r)$. This is true because $s_{\mathcal{L}}[j \rightarrow 0]$ and $s_{\mathcal{L}}[j \rightarrow 1]$ give different Δf_i and both belong to S. Third, because Δf_i depends only on the positions in γ (see lemma 8), for any string r' that agrees with r on these $|\gamma|$ positions we have $\Delta f_i(s) \neq \Delta f_i(r')$. The probability that $s \in S$ is $2^{-|\Gamma[i]|}$. Given that $s \in S$, the probability that s' agrees with r on the remaining $|\gamma|-|\Gamma[i]|$ positions in γ is $2^{-(|\gamma|-|\Gamma[i]|)}$. So the overall probability that $\Delta f_i(s) \neq \Delta f_i(s')$ is at least $2^{-|\Gamma[i]|}2^{-(|\gamma|-|\Gamma[i]|)} = 2^{-|\gamma|}$. ∎

```
Procedure FINDLINKAGEGROUPS(f, t)
1. Γ := {{1},{2}, ..., {ℓ}}.
2. Do t times:
     For i from 1 to ℓ:
        j := TESTFORLINK(f, Γ, i)
        If j ≠ -1 then Γ := (Γ - {Γ[i]} – {Γ[j]}) ∪ {Γ[i] ∪ Γ[j]}.
3. Return Γ.
```

Fig. 4. Procedure FINDLINKAGEGROUPS. If run for a sufficiently long time, the procedure returns the linkage group partition for an arbitrary function f.

Lemma 5 follows from lemma 3 and the manner in which Γ is updated in FINDLINKAGEGROUPS.

Lemma 5. For any t, FINDLINKAGEGROUPS(f, t) returns a partition Γ such that for any i and j, if $j \in \Gamma[i]$ then $\mathcal{G}(i, j)$. ■

We do not analyze the behavior of FINDLINKAGEGROUPS directly. Instead we analyze a simpler algorithm called MARKPOSITIONSA, and prove that its performance provides a lower bound on that of another algorithm called MARKPOSITIONSB, which we prove has exactly the same performance as FINDLINKAGEGROUPS.

```
Procedure MARKPOSITIONSA(π, t)
1. Π:={}.
2. Do t times:
   2.1. For i from 1 to ℓ:
        2.1.1. With probability π^-1, do:
               2.1.1.1. Π := Π ∪ {i}.
3. Return Π.
```

```
Procedure MARKPOSITIONSB(f, t, Γ_f)
1. Π := {}; Γ := {{1},{2}, ..., {ℓ}}.
2. Do t times:
   2.1. For i from 1 to ℓ:
      2.1.1. j := TESTFORLINK(f, Γ, i).
      2.1.2. If j ≠ -1 then:
         2.1.2.1. Π := Π ∪ {i}.
         2.1.2.2. Γ := (Γ - {Γ[i]} – {Γ[j]})
         ∪ {Γ[i] ∪ Γ[j]}.
         2.1.2.3. If Γ[j] = Γ_f[j], then Π :=
         Π ∪ Γ_f[j].
4. Return Π.
```

Fig. 5. Procedures MARKPOSITIONSA and MARKPOSITIONSB, which are used in the analysis of FINDLINKAGEGROUPS.

Lemma 6. The probability that MARKPOSITIONSA(π, t) returns a set containing all ℓ positions is $(1-(1-\pi^{-1})^t)^\ell$. To ensure that this probability is at least p, we must set t to at least $\ln(1-p^{1/\ell})/\ln(1-\pi^{-1})$, and this expression is $O(\pi\ln(\ell))$ for constant p.

Proof: By inspection, the probability that $j \in \Pi$ for any particular j is $1-(1-\pi^{-1})^t$. Because each position is independent, the probability that Π contains all ℓ positions is $(1-(1-\pi^{-1})^t)^\ell$. For this quantity to exceed p, t must satisfy $(1-(1-\pi^{-1})^t)^\ell \geq p$, and solving this inequality for t yields:

$$t \geq \frac{\ln\left(1-p^{\left(\frac{1}{\ell}\right)}\right)}{\ln\left(1-\pi^{-1}\right)} .$$

To see that this expression is $O(\pi\ln(\ell))$, observe that $-\ln(1-\pi^{-1})$ is $\Theta(\pi^{-1})$ because:

$$\lim_{\pi\to\infty}\frac{-\ln\left(1-\pi^{-1}\right)}{\pi^{-1}} = \lim_{x\to 0}\frac{-\ln(1-x)}{x} = \lim_{x\to 0}\frac{1}{1-x} = 1 .$$

where in the first step we have made the change of variable $x = \pi^{-1}$ and in the second step we have used l'Hôpital's rule. Thus $-1/\ln(1-\pi^{-1})$ is $O(\pi)$.

A similar argument shows that $-\ln(1-p^{1/\ell})$ is $O(\ln(\ell))$ [12], from which the lemma follows. ■

Lemma 7. Let f be an order-k additively separable function with linkage group partition Γ_f. Then for any t, the probability that MARKPOSITIONSB(f, t, Γ_f) returns a set

containing all ℓ positions (i) is at least as high as the probability that MARKPOSITIONSA(2^k, t) returns such a set and (ii) is equal to the probability that FINDLINKAGEGROUPS(f, t) returns Γ_f.

Proof: (ii) The code for MARKPOSITIONSB is identical to that for FINDLINKAGEGROUPS except for the additional bookkeeping needed to maintain Π. So to prove (ii) it suffices to show just before MARKPOSITIONSB returns, $\Gamma=\Gamma_f$ iff. $\Pi=\{1, 2, ..., \ell\}$. To prove this it is sufficient to show that for all i' ($1 \le i' \le \ell$), $\Gamma[i']=\Gamma_f[i']$ iff. $\Gamma_f[i'] \subseteq \Pi$. Suppose $\Gamma_f[i'] \subseteq \Pi$. If any $j' \in \Gamma_f[i']$ was added to Π by line 2.1.2.3 then by inspection $\Gamma[i']=\Gamma_f[i']$. The other possibility is that all $j' \in \Gamma_f[i']$ were added to Π by line 2.1.2.1. In this case let $J'(\Gamma) = |\{\gamma: \gamma \in \Gamma \text{ and } \gamma \subseteq \Gamma_f[i']\}|$. Initially $J'(\Gamma)$ is $|\Gamma_f[i']|$. Each time line 2.1.2.1 adds a member of $\Gamma_f[i']$ to Π, the following line decreases $J'(\Gamma)$ by 1. But because $J'(\Gamma)$ must be at least 1 this can happen at most $|\Gamma_f[i']|$-1 times, so it must be that at least one $i' \in \Gamma_f[i]$ was added to Π by line 2.1.2.3. The converse (that $\Gamma[i']=\Gamma_f[i']$ implies $\Gamma_f[i'] \subseteq \Pi$) follows from inspection of lines 2.1.2.2 and 2.1.2.3.

(i) The difference between MARKPOSITIONSA and MARKPOSITIONSB is in the actions they perform within the inner loop that begins on line 2.1. When this inner loop is executed for some position i, each of the two algorithms will add i to Π with some probability. If $i \in \Pi$ already, then executing $\Pi := \Pi \cup \{i\}$ has no affect, so we may concern ourselves only with the case $i \notin \Pi$. Whether or not $i \notin \Pi$, MARKPOSITIONSA adds i to Π with probability exactly 2^{-k}. When $i \notin \Pi$, MARKPOSITIONSB adds i to Π with a probability that depends on Γ. Thus to prove part (i), it suffices to show that when $i \notin \Pi$ this probability is always at least 2^{-k}, independent of Γ. But if $i \notin \Pi$, then by the previous paragraph it must be that $\Gamma[i] \neq \Gamma_f[i]$ and lemma 4 guarantees that TESTFORLINK(f, Γ, i) returns a value other than -1 with probability at least $2^{-|\gamma|}$ where $\gamma=\Gamma_f[i]$. That $|\Gamma_f[i]| \le k$ follows from the definition of $\mathcal{G}$.■

Theorem 1 follows immediately from lemmas 6 and 7.

Theorem 1. Let f be an order-k additively separable function with linkage group partition Γ_f. The probability that FINDLINKAGEGROUPS(f, t) returns Γ_f is at least $(1-(1-2^{-k})^t)^\ell$. To find Γ_f with probability p we must invoke FINDLINKAGEGROUPS(f, $\ln(1-p^{1/\ell})/\ln(1-2^{-k})$), which will require $O(2^k \ell \ln(\ell))$ evaluations of f. ■

4 Optimizing Additively Separable Functions

Our optimization algorithm uses TESTFORLINK to discover the linkage group partition Γ in exactly the same manner as FINDLINKAGEGROUPS. Additionally, whenever a new linkage group γ is discovered, the optimization algorithm performs a local search that guarantees discovery of a string Ω that is optimal w.r.t. the linkage group γ. The result is that whenever the optimization algorithm discovers the linkage group partition Γ, it also discovers a globally optimum string.

Figures 6 and 7 present the code for our algorithm. Lemma 8 and theorem 2 are presented without proof, and formally establish the connection between the subfunctions that make up an additively separable function and the function's linkage group partition. Full proofs are available online [11]. Lemma 9 shows that our optimization algorithm finds a globally optimum string provided that it discovers the linkage group partition, and theorem 3 gives its time complexity.

```
Procedure OPTIMIZEWRTGROUP(f, Ω, γ):
1. Let i_1, i_2, ..., i_|γ| be the positions in γ.
2. For each of the 2^|γ| tuples <x_1, x_2, ..., x_|γ|> where x_a ∈ {0,1} for 1≤ a ≤ |γ|:
     Ω' := Ω[i_1→x_1][i_2→x_2]...[i_|γ|→x_|γ|].
     If f(Ω') > f(Ω) then Ω := Ω'.
3. Return Ω.
```

Fig. 6. Procedure OPTMIZEWRTLINKAGEGROUP. Performs a local search to return a string Ω that is guaranteed to be optimal with regard to a linkage group γ.

```
Procedure ASFOPTIMIZE(f, t):
1. Ω := RANDOMSTRING().
2. Γ := {{1},{2}, ..., {ℓ}}.
3. For i from 1 to ℓ:
     Ω := OPTIMIZEWRTGROUP(f, Ω, Γ[i]).
4. Do t times:
     4.1. For i from 1 to ℓ:
          j := TESTFORLINK(Γ, i)
          If j ≠ -1 then
               γ := Γ[i] ∪ Γ[j].
               Ω := OPTIMIZEWRTGROUP(f, Ω, γ).
               Γ := (Γ - {Γ[i]} – {Γ[j]}) ∪ {γ}.
5. Return Ω.
```

Fig. 7. Procedure ASFOPTIMIZE. With arbitrarily high probability, returns a global optimum of an order-k additively separable function f using $O(2^k \ell \ln(\ell))$ function evaluations.

Lemma 8. Let γ be a linkage group of f, and let s_1 and s_2 be two strings such that for all $i \in \gamma$, $s_1.i = s_2.i$. Then $\Delta f_i(s_1) = \Delta f_i(s_2)$. ■

Theorem 2. Any function whose largest linkage group is of size k is order-k additively separable. Specifically, let $\Gamma_f = \{\gamma_1, \gamma_2, ..., \gamma_m\}$ be the linkage group partition for f, and let $\gamma_i = \{\alpha_{i,1}, \alpha_{i,2}, ..., \alpha_{i,k(i)}\}$, where $k(i) = |\gamma_i|$. Let S_0 be a string consisting entirely of zeroes. Then for all s, $f(s) = f_\Sigma(s)$, where:

$$f_\Sigma(s) \equiv \sum_{i=1}^{m} f_i(s.\alpha_{i,1}, s.\alpha_{i,2}, ..., s.\alpha_{i,k(i)}) \text{ , and}$$

$$f_i(\beta_1, \beta_2, ..., \beta_{k(i)}) \equiv f(S_0[\alpha_{i,1}\rightarrow\beta_1][\alpha_{i,2}\rightarrow\beta_2]...[\alpha_{i,k(i)}\rightarrow\beta_{k(i)}]) + f(S_0)(1/m\text{-}1). \quad ■$$

We are now prepared to make the following definition.

Definition 6. A string s is optimal w.r.t. a linkage group $\gamma_i = \{\alpha_{i,1}, \alpha_{i,2}, ..., \alpha_{i,k(i)}\}\}$ if $f_i(s.\alpha_{i,1}, s.\alpha_{i,2}, ..., s.\alpha_{i,k(i)})$ is maximized. ■

Corollary 1. A string s is globally optimal w.r.t. a function f iff. s is optimal w.r.t. each linkage group of f. ■

Corollary 2. If γ is a linkage group for f, then OPTIMIZEWRTGROUP(f, s, γ) returns a string s' that is optimal w.r.t. γ. ■

Lemma 9. If, when ASFOPTIMIZE(f, n) returns, Γ is the linkage group partition (Γ_f) for f, then the string Ω returned by ASFOPTIMIZE(f, n) is globally optimal w.r.t. f.

Proof: If $\Gamma=\Gamma_f$ then ASFOPTIMIZE(n) must have executed the statement Ω := OPTIMIZEWRTGROUP(Ω, γ) for every linkage group $\gamma \in \Gamma_f$. Consider the effect of executing this statement for some particular γ. Immediately after the statement is executed, Ω must be optimal w.r.t. γ by corollary 2. Because of the way in which Γ is updated, and noting γ cannot be merged with any other linkage group if we are to have $\Gamma=\Gamma_f$, all subsequent calls to OPTIMIZEWRTGROUP(Ω, γ') must be such that γ and γ' are disjoint. Such calls do not alter the positions in γ, so once Ω := OPTIMIZEWRTGROUP(Ω, γ) has been executed Ω will remain optimal w.r.t. γ until the function returns. Thus the Ω returned by ASFOPTIMIZE will be optimal w.r.t. all linkage groups $\gamma \in \Gamma_f$, so by corollary 1 Ω will be globally optimal w.r.t. f. ■

Theorem 3. Let f be an order-k additively separable function. ASFOPTIMIZE(f, t) returns a globally optimum string with probability at least $(1-(1-2^{-k})^t)^\ell$. To make this probability at least p we must invoke ASFOPTIMIZE(f, $\ln(1-p^{1/\ell})/\ln(1-2^{-k})$). This requires $O(2^k \ell \ln(\ell))$ evaluations of f and storage of O(1) strings.

Proof: The theorem would follow directly from lemma 9 and theorem 2 if we ignored the function evaluations required by calls to OPTIMIZEWRTGROUP. Thus to prove the theorem it is sufficient to show that these calls cannot account for more than $O(2^k \ell \ln(\ell))$ function evaluations. Each time OPTIMIZEWRTGROUP is called within the loop that begins on line 4, $|\Gamma|$ has just been reduced by 1 by the previous line. Because $|\Gamma|$ is initially ℓ and can never fall below 1, OPTIMIZEWRTGROUP can be called at most $|\Gamma|$-1 times within this loop. Prior to this loop OPTIMIZEWRTGROUP is called exactly ℓ times, so the total number of calls is at most 2ℓ-1. Because each call requires at most 2^k function evaluations, the total number of function evaluations is $2^k(2\ell-1)$, which is $O(2^k \ell \ln(\ell))$. The fact that ASFOPTIMIZE requires storage of O(1) strings is clear by inspection. ■

5 Experimental Results

To make the performance claims concerning our algorithm more concrete, we now present a set of experiments using the ASFOPTIMIZE procedure on additively separable functions. In particular, we examine the performance of ASFOPTIMIZE on order-5 folded trap functions [1]. Using ASFOPTIMIZE we have solved up to 10,000 bit problems; results are presented here for $5 \leq \ell \leq 1{,}000$. Performance data for BOA on this same problem for $30 \leq \ell \leq 180$ was presented by Pelikan [9].

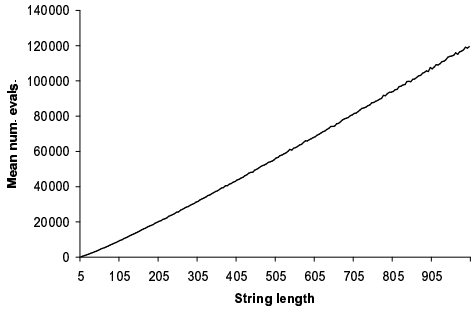

Fig. 8. Average number of function evaluations required by ASFOPTIMIZE to find the global optimum of an order-5 folded trap function, for problem sizes between 5 and 1,000. Each data point is an average of 1,000 runs. Error bars are not shown.

Figure 8 shows the average number of function evaluations required by ASFOPTIMIZE to find the global optimum as a function of problem size. As expected, the curve is near-linear. An average of 2,200 evaluations are required for ℓ=30, while 17,700 are required for ℓ=180. By way of comparison, BOA requires an average of approximately 220,000 evaluations for ℓ=180 [9]. Thus, it addition to its guaranteed $O(2^k \ell \ln(\ell))$ performance as ℓ tends toward infinity, ASFOPTIMIZE appears to be quite efficient for problems of modest size.

6 Discussion

Although the algorithm presented in this paper satisfies the operational definition of competence [1], it is not what one would consider to be a competent problem-solving algorithm. For any problem of practical interest, we expect that there is *some* degree of interaction between every pair (i, j) of positions (i.e., $\Delta f_i(s[j \rightarrow 0]) \neq \Delta f_i(s[j \rightarrow 1])$ for some s and s'), so that by our definitions the linkage group partition will be $\Gamma = \{\{1, 2, ..., \ell\}\}$. For problems with this linkage group partition, our optimization procedure will attempt to search through all 2^ℓ strings for one that is optimal w.r.t. the single linkage group in Γ, and thus will degenerate into an exhaustive search.

Though the algorithm is clearly brittle as currently defined, we do not believe that this brittleness is inherent. As discussed by Munemoto [7], the randomized linkage test employed by our algorithm can employ repeated sampling in order to handle an additively separable function corrupted by (external) additive noise. Handling noise in this manner is a first step toward handling the more systematic departures from a linear model that are characteristic of real optimization problems.

7 Conclusion

We have presented upper bounds on the time and space complexity of optimizing additively separable functions. We exhibited a simple algorithm that optimizes such functions and provided upper bounds on the number of function evaluations it requires to find a globally optimum string with a specified probability. The upper bounds provided in this paper are sharper than all those that have previously been reported. While acknowledging that the algorithm as described is not practical, we have suggested that the algorithm could be modified so as to overcome its existing limitations, and we are optimistic that the ideas presented here can be used to construct more powerful competent genetic algorithms.

References

1. D. E. Goldberg. *The Design of Innovation: Lessons from and for competent genetic algorithms*. Boston, MA: Kluwer Academic Publishers; 2002.
2. D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In *Proc. Fifth Int'l. Conf. on Genetic Algorithms*., 1993, p 56-64.
3. G. R. Harik and D. E. Goldberg. Learning linkage through probabilistic expression. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):295-310, 2000.
4. R. B. Heckendorn and A. H. Wright. Efficient linkage discovery by limited probing. In *Proc. 2003 Genetic and Evolutionary Computation Conf.*, 2003, p 1003-1014.
5. S. A. Kauffman. *The Origins of Order*. New York: Oxford University Press; 1993.
6. P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Boston, MA: Kluwer Academic Publishers. 2001.
7. M. Munemoto and D. E. Goldberg. Linkage identification by non-monotonicity detection for overlapping functions. *Evolutionary Computation*, 7(4):377-398, 1999.
8. M. Pelikan. *Bayesian Optimization Algorithm: From Single Level to Hierarchy.* Ph.D thesis, University of Illinois Urbana-Champaign. 2002.
9. M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. Linkage problem, distribution estimation, and Bayesian networks. *Evolutionary Computation*, 8(3):311-340, 2000.
10. E. D. Weinberger. NP completeness of Kauffman's NK model, a tunable rugged fitness landscape. Santa Fe Institute T.R. 96-02-003. 1996.
11. M. J. Streeter. http://www.cs.cmu.edu/~matts/gecco_2004/index.html.
12. A. H. Wright and R. B. Heckendorn. Personal communication. Jan. 2004.
13. A. H. Wright, R. K. Thompson, and J. Zhang. The computational complexity of N-K fitness functions. *IEEE Transactions on Evolutionary Computation*, 4(4):373-379, 2000.