The Computational Complexity of N-K Fitness Functions

Alden H. Wright — wright@cs.umt.edu
Richard K. Thompson — rkt@selway.umt.edu
Jian Zhang
Computer Science Dept.
The University of Montana
Missoula, MT 59812-1008

ABSTRACT

The N-K fitness landscapes have been widely used as examples and test functions in the field of evolutionary computation. Thus, the computational complexity of these landscapes as optimization problems is of interest. We investigate the computational complexity of the problem of optimizing the N-K fitness functions and related fitness functions.

We give an algorithm to optimize adjacent-model N-K fitness functions which is polynomial in N. We show that the decision problem corresponding to optimizing random-model N-K fitness functions is NP-complete for $K > 1$ and is polynomial for $K = 1$. However, if the restriction that the ith component function depends on the ith bit is removed, then the problem is NP-complete even for $K = 1$.

We also give a polynomial-time approximation algorithm for the arbitrary-model N-K optimization problem.

1 Introduction

Kauffman [Kau93] introduced a class of stochastically defined fitness landscapes over bit strings called the N-K landscapes. These have a parameter K that can be tuned to adjust the "ruggedness" of the landscapes. When $K = 0$, the landscapes are linear, and when $K = N - 1$ (where N is number of bits) the landscapes are random.

Many evolutionary computation papers make reference to these landscapes either as examples or as test functions. Thus, it is of interest to know the computational complexity of the corresponding optimization problems.

An N-K function is the sum of functions, where each summand function depends on $K + 1$ of the N bits of the bit string. There are two variations of the N-K model. In the *adjacent* model, the summand function f_i depends on bit i and on K adjacent bits. In the *random* model, f_i depends on bit i and K other randomly chosen bits. Kauffman [Kau93] and Weinberger [Wei96] show that these two variations have very similar statistical properties. For example, the correlation of the fitness values of adjacent bit strings is almost the same.

We show that the two variations have quite different computational complexities.

In Kauffman's definition, the values of the summand functions are chosen randomly from a uniform distribution. The bit dependencies for the random model N-K functions are randomly chosen. In our complexity analysis, we assume that these choices are made arbitrarily. In other words, our class of N-K functions includes any function that can result from some random choice in Kauffman's definition of the N-K functions. In view of this, we describe the *random* model of N-K landscapes as *arbitrary* rather than *random.*

Some of our results also apply to N-K landscapes over higher arity alphabets. See section 2 for more details.

To summarize our results:

1. We give an dynamic programming algorithm that optimizes the class of adjacent N-K fitness landscapes. The algorithm is polynomial in N and exponential in K.
2. We give a polynomial algorithm for the class of arbitrary N-K fitness landscapes with $K = 1$.
3. We show that the class of arbitrary N-K fitness landscapes for $K \geq 2$ is NP-complete.
4. We show that the class of arbitrary N-K fitness landscapes for $K \geq 1$ where the component function f_i is not required to depend on bit (or symbol) i is NP-complete.
5. We show that there is a polynomial-time ϵ-approximation algorithm for the class of arbitrary N-K fitness landscapes with $\epsilon = 1 - (1/2)^K$.

The results 1, 2, 3, and 4 are from the Master's thesis [Tho95] which was done under the first author's supervision. The result 5 is from the Master's thesis [Zha97] which was also done under the first author's supervision.

Weinberger [Wei96] independently discovered results 1 and 3 listed above.

2 Formalization

We give a formal description of our extensions to the class of N-K fitness functions.

Let Σ denote a (finite) alphabet. To specify an N-K fitness function $f = \sum f_i$, we must specify the positions that influence each term f_i, and we must specify the f_i functions.

We specify the positions through projection functions p_i, where each p_i is a mapping from Σ^N to Σ^{k_i}. Each p_i is defined by a cardinality k_i subset of $\{0, 1, \ldots, N-1\}$. For example, if $k_i = 3$ and if p_2 is defined by the subset $\{1, 3, 6\}$, then p_2 is defined by $p_2(a_0, a_1, \ldots, a_{N-1}) = (a_1, a_3, a_6)$.

For the adjacent model, we choose the subset of $\{0, 1, \ldots, N-1\}$ that defines p_i to be $\{i, i+1, \ldots, i+K\}$, where the indices are taken $\bmod N$. For the arbitrary model, the subset that defines p_i is arbitrarily chosen. Sometimes we may specify that i is in the subset that defines p_i. Then we assume that the component functions f_i map Σ^{K+1} into $R^{\geq 0}$. Then f is written more precisely as

$$f = \sum_{i=0}^{N-1} f_i \circ p_i$$

where the domain of each f_i is Σ^{K+1}. When it is clear from the context, we will omit the projection functions.

The input to the optimization problem consists of the subsets defining the p_i and a set of tables defining the f_i.

3 Polynomial Algorithms

3.1 The $K = 1$ adjacent case

We demonstrate a polynomial dynamic programming algorithm to find the optimum fitness of the adjacent model. We start with an algorithm for the $K = 1$ case.

The idea of the algorithm is to reduce the problem of size N to a problem of size $N - 1$ by defining an fitness function $f' : \Sigma^{N-1} \longrightarrow R^{\geq 0}$, where $R^{\geq 0}$ denotes the nonnegative reals. The function f' is defined as

$$f' = \sum_{i=0}^{N-2} f'_i$$

where $f'_i = f_i$ for $0 \leq i < N - 1$, and

$$f'_{N-2}(a_{N-2}, a_0) = \max\{f_{N-2}(a_{N-2}, b) + f_{N-1}(b, a_0) : b \in \Sigma\}$$

Theorem 3.1 *Under the definition of f' given above,*

$$\max\{f'(\mathbf{a}') : \mathbf{a}' \in \Sigma^{N-1}\} = \max\{f(\mathbf{a}) : \mathbf{a} \in \Sigma\}$$

Proof. Define $p : \Sigma^N \longrightarrow \Sigma^{N-1}$ by $p(a_0, \ldots, a_{N-2}, a_{N-1}) = (a_0, \ldots, a_{N-2})$. Then for any $\mathbf{a} \in \Sigma^N$,

$$f(\mathbf{a}) - f'(p(\mathbf{a})) = f_{N-2}(a_{N-2}, a_{N-1}) + f(a_{N-1}, a_0) - f'(a_{N-2}, a_0)$$

Since

$$\begin{aligned} f'_{N-2}(a_{N-2}, a_0) &= \max\{f_{N-2}(a_{N-2}, b) + f_{N-1}(b, a_0) : b \in \Sigma^N\} \\ &\geq f_{N-2}(a_{N-2}, a_{N-1}) + f(a_{N-1}, a_0) \end{aligned}$$

we have

$$f(\mathbf{a}) \leq f'(p(\mathbf{a})) \tag{1}$$

Let $\mathbf{c}' \in \Sigma^{N-1}$ be such that $f'(\mathbf{c}') = \max\{f'(\mathbf{a}') : \mathbf{a}' \in \Sigma^{N-1}\}$. Then

$$f'_{N-2}(c'_{N-2}, c'_0) = \max\{f_{N-2}(c'_{N-2}, b) + f_{N-1}(b, c'_0) : b \in \Sigma\}$$

Choose $c_{N-1} \in \Sigma$ to achieve this maximum, and let $c_i = c'_i$ for $i < N - 1$ so that $\mathbf{c}' = p(\mathbf{c})$, and

$$f'_{N-2}(c'_{N-2}, c'_0) \quad = \quad f_{N-2}(c_{N-2}, c_{N-1}) + f_{N-1}(c_{N-1}, c_0)$$

which implies that $f(\mathbf{c}) = f'(p(\mathbf{c})) = f'(\mathbf{c}')$. This, along with equation (1), implies the statement of the theorem. ■

If we know the string $\mathbf{c}' \in \Sigma^{N-1}$ that maximizes f', and if we know the element $b \in \Sigma$ which realizes the maximum in the definition of f'_{N-2}, then we can construct the element $\mathbf{c} \in \Sigma^N$ which maximizes f. It is not hard to see that if $\mathbf{c}' = (c_0, c_1, \ldots, c_{N-2})$, then $\mathbf{c} = (c_0, c_1, \ldots, c_{N-2}, b)$.

We can now write the above in the form of an algorithm. We assume that the tables that define the components of the given fitness function f are given in the form of a 3-dimensional array $F[0..N-1, \Sigma, \Sigma]$. In other words, the function $f_i : \Sigma^2 \longrightarrow R^{\geq 0}$ is given by $f_i(a, b) = F[i, a, b]$. The algorithm repeatedly applies the process of Theorem 3.1 to reduce the problem to a problem of size $N = 2$. The solution of the $N = 2$ problem is found by direct search, and then a solution to the original problem is reconstructed. The F array is used as temporary storage by the algorithm, so the original contents of F are destroyed. At the end of the ith stage, $F[0..i-2, \Sigma, \Sigma]$ contains the definition of the current component functions, and $F[i-1..N, \Sigma, \Sigma]$ contains the elements of Σ that realize the maximum in the definition of f' above. For example, if $F[N-1, a, c] = d$, then $f_{N-2}(a, d) + f_{N-1}(d, c) = \max\{f_{N-2}(a, b) + f_{N-1}(b, c) : b \in \Sigma\}$.

```
OPTIMIZE(F)
/// Assumes that K = 1.
/// The given array F[0..N − 1, Σ, Σ] which defines the fitness function, is destroyed.
/// v[Σ, Σ] and u[Σ, Σ] are temporary arrays.
/// Reduce the problem size to N = 2.
  for n from N − 1 downto 2 do
    for a ∈ Σ do
      for c ∈ Σ do
        Choose b_max so that
            F[n − 1, i, b_max] + F[n, b_max, k] = max{F[n − 1, a, b] + F[n, b, c] : b ∈ Σ}
        v[a, c] ← b_max
        u[a, c] ← F[n − 1, i, b_max] + F[n, b_max, k]
    for a ∈ Σ do
      for c ∈ Σ do
        F[n − 1, a, c] ← u[a, c]
        F[n, a, c] ← v[a, c]

/// Problem size is now N = 2. Find the max fitness string for N = 2
  Choose a_max and c_max so that
      F[0, a_max, c_max] + F[1, c_max, a_max] = max{F[0, a, c] + F[1, c, a] : a ∈ Σ, c ∈ Σ}

/// Construct an optimal string S for the whole problem.
  S[0] ← i_max
  S[1] ← k_max
  for i from 2 to N − 1 do
    S[i] ← F[i, S[i − 1], S[0]]

  return S
```

Algorithm 1

Clearly, the time complexity of this algorithm is $\Theta(N|\Sigma|^3)$ if we assume that binary operations on real numbers (such as addition and maximizaton) require $O(1)$ time.

The space required is that required for the arrays F, u, and v. Thus the space used is $O(N|\Sigma|^2)$ on the assumption that every real number used in the algorithm can be stored in $O(1)$ space.

3.2 The $K > 1$ adjacent case

First, suppose that N is divisible by K. Then we can view a string of length N over Σ as a string of length N/K over alphabet Σ^K. Any component function f_i which depends on at most $K+1$ positions of the string over Σ will depend on at most 2 positions of the string over alphabet Σ^K. If we let $\widetilde{f}_i = \sum_{j=Ki}^{Ki+K-1} f_i$, then $\widetilde{f}_i$ depends on only symbols $\widetilde{a}_i$ and $\widetilde{a}_{i+1}$ of the string over the alphabet Σ^K. Then we can apply the Algorithm 1 of the previous section to achieve an algorithm of complexity $\Theta(N|\Sigma|^{3K})$.

If N is not divisible by K, let $Q = \lfloor N/K \rfloor$ and $r = N \bmod K$. We view a string of length N over Σ as a string of symbols $\widetilde{a}_0\widetilde{a}_1 \ldots \widetilde{a}_Q$, where $\widetilde{a}_0 \in \Sigma^K$, $\widetilde{a}_1 \in \Sigma^r$, and $\widetilde{a}_2, \ldots, \widetilde{a}_Q \in \Sigma^K$. We can write $f = \sum_{i=0}^{Q} \widetilde{f}_i$ where each $\widetilde{f}_i$ depends on $\widetilde{a}_i$ and $\widetilde{a}_{i+1}$, except for $\widetilde{f}_0$ which depends on $\widetilde{a}_0$, $\widetilde{a}_1$, and $\widetilde{a}_2$. We apply Algorithm

1 of the previous section to reduce the problem to a problem over a string of length $2K + r$ over Σ, or a string of length 3 over $\Sigma^K \times \Sigma^r \times \Sigma^K$. Note that Algorithm 1 does not work with component function $\widetilde{f}_0$. Each major iteration of the first part of Algorithm 1 takes $\Theta(|\Sigma|^{3K}|)$ steps. We solve the reduced problem by enumerating all solutions, which takes $|\Sigma|^{2K+r} \in O(|\Sigma|^{3K})$ steps. This gives the following theorem:

Theorem 3.2 *Let f be an adjacent-model N-K fitness function over the alphabet Σ. Then a string corresponding to the optimal fitness value can be found in time $\Theta((Q+1)|\Sigma|^{3K})$, where $Q = \lfloor N/K \rfloor$ (assuming that real-number operations can be done in $O(1)$ time).*

Corollary 3.3 *Let $C > 0$ be a constant, and consider the family of adjacent-model N-K fitness functions over the alphabet Σ such that $K \le C \log N$. There is a polynomial algorithm to find the optimal string for this family of fitness functions.*

Proof. The time complexity of the algorithm of Theorem 3.2 is $O(N|\Sigma|^{3C \log N}) = O(N^{3C \log |\Sigma| + 1})$. ■

3.3 A $K = 1$ arbitrary case

In this section we show that for $K = 1$ and the arbitrary model where component function f_i of f depends on position i of the string, then there is a polynomial algorithm to optimize f.

We define a graph whose vertices are the integers from 0 to $N-1$, and whose edges are the terms f_i. The edge f_i connects the vertices of the set of indices corresponding to the projection p_i. In other words, the edge f_i connects the indices corresponding to those components of the string that f_i depends on. Note the graph corresponding to a $K = 1$ adjacent-model N-K fitness function is a cycle.

Let us rearrange the indices $\{0, 1, \ldots, n-1\}$ so that the indices corresponding to the connected components of G are adjacent. Thus, we assume that the vertices $\{n_0 = 0, 1, \ldots, n_1 - 1\}$ correspond to the first connected component of G, the vertices $\{n_1, n_1 + 1, \ldots, n_2 - 1\}$ correspond to the second, etc. The let

$$g_j = \sum_{i=n_j}^{n_{j+1}} f_i$$

where g_j depends only on the components $n_j, \ldots n_{j+1} - 1$, so the g_j can be optimized independently, and the sum of the optimal values for the g_j is the optimal value for f. Further, the optimal string for f is the concatenation of the optimal strings for the g_j.

Theorem 3.4 *Let f be an N-K fitness function with $K = 1$, where each f_i depends on position i of the string and one other position. Then there is a polynomial algorithm to optimize f.*

Proof. Our assumption is that the edge f_i must be incident on vertex i. Thus, every component of the graph has the same number of edges as vertices, and every component of the graph has exactly one cycle.

If a component of the graph G consists of a cycle, then we can apply Theorem 3.1 and use Algorithm 1.

If a component of G is not a cycle, then there must be some vertex of degree 1. We show how the function f can be replaced by a function f' which does not depend on this string position, and which has the same optimum.

Without loss of generality, we assume that index $N-1$ corresponds to a vertex of G of degree 1. Then f_{N-1} is the only term of f which depends on this position of the string. Without loss of generality, we can also assume that the other position of the string that f_{N-1} depends on is $N-2$.

Let

$$f'_{N-2}(a_{N-1}, a_{N-2}) = f_{N-2}(a_{N-2}, a_{N-1}) + \max\{f_{N-1}(a_{N-2}, c) : c \in \Sigma\}$$

Clearly, if $a_0 \ldots a_{N-2}a_{N-1}$ is optimal for f, then $a_0 \ldots a_{N-2}$ is optimal for f'. ■

4 NP-Completeness

In this section we show that when the restriction that f_i depends on position i of the string is removed, then the $K = 1$ arbitrary-model optimization problem is NP-complete. We now assume that the function f maps into the nonnegative integers rather than the nonnegative reals.

Theorem 4.1 *The problem of optimizing an N-K fitness function $f = \sum f_i$ with $K = 1$ and no restrictions on the dependence of the f_i on string positions is NP-complete.*

Proof. To show NP-completeness, we must show that a solution to the corresponding decision problem can be checked in polynomial time, and that some problem known to be NP-complete can be reduced to our problem. The decision problem corresponding to the problem of maximizing f is: Given a fixed natural number k, decide if there exists a string $\mathbf{a} \in \Sigma^N$ such that $f(\mathbf{a}) \geq k$. Since f can be computed in polynomial time, a solution can be checked in polynomial time.

The known NP-complete problem that we will reduce to our problem is the MAXIMUM 2-SAT problem [GJ79]: Given a Boolean formula in conjunctive normal form (CNF) with 2 literals per clause, the problem is to maximize the number of true clauses. A 2-CNF formula is a conjunction of clauses of the form $u_i \vee u_j$, where u_i and u_j are literals, where a literal is either a Boolean variable or a negation of a Boolean variable.

Given a 2-CNF formula, we will construct a corresponding N-K fitness function f over the alphabet $\{0, 1\}$ so that the value of f is the number of true clauses in the 2-CNF formula. Let N be the maximum of the number of variables and the number of clauses in the CNF formula. To each Boolean variable of the formula, we associate a string position, and to each clause of the formula, we associate a term f_i of f. If there are string positions that are not associated with Boolean variables, then the symbols in these positions will not affect the value of f, and if there are f_i which are not associated with clauses, then we set these f_i to have value zero.

We assume that the ith Boolean variable of the formula is associated with the ith string position, and that the ith clause of the formula is associated with f_i. Thus, each f_i depends on two string positions. Let a string position value of 1 correspond to a *true* value of the corresponding Boolean variable, and a value of 0 correspond to *false*. The value of f_i will be defined to be 1 if the corresponding clause of the formula is *true* and 0 if the corresponding clause is *false*. For example, if f_i corresponds to the clause $(x_2 \vee \overline{x_4})$, then f_i will depend on string positions 2 and 4, and the values of f_i are given by

a_2	a_4	f_i
0	0	1
0	1	0
1	0	1
1	1	1

Clearly, the value of f is the number of *true* clauses in the 2-CNF formula. ■

Under Kauffman's model, f_i was required to depend on string position i. If $K = 2$ and this is the only restriction on the dependence of f_i, then clearly Theorem 4.1 shows that optimizing such an f (with integer values) is an NP-complete problem. Weinberger [Wei96] has independently discovered this result.

5 A polynomial-time approximation algorithm

5.1 Definitions

For this section, we assume a binary alphabet.

Although all NP-complete problems share exponential worst-case complexity (unless $P = NP$), they have little else in common. When seen from almost any other perspective, they have interesting, unique diversity. In this paper, we want to assess the difficulty of the optimization of the arbitrary model N-K functions. We will accomplish this by studying the problem against several complexity classes [Pap94].

Definition 5.1 *[Pap94] Suppose that A is an optimization problem. This means that for each instance x we have a set $F(x)$ of* feasible solutions, *and for each solution $s \in F(x)$ we have a positive integer cost $c(s)$ (we use the term* cost *and notation $c(s)$ even in the case of maximization problems). The optimal cost is $OPT(x) = min_{s \in F(x)} c(s)$ (or $max_{s \in F(x)} c(s)$, if A is a maximization problem). Let M be an algorithm which, given any instance x, returns a feasible solution $M(x) \in F(x)$. We say that M is an ϵ-approximation algorithm, where $\epsilon > 0$, if for all x we have*

$$\frac{|c(M(x)) - OPT(x)|}{max\{OPT(x), c(M(x))\}} \leq \epsilon$$

Intuitively, a heuristic is ϵ-approximate if the "relative error" of the solution found is at most ϵ. For a maximization problem, an ϵ-approximate algorithm returns solutions that are never smaller than $1 - \epsilon$ times the optimum. For a minimization problem, the solutions returned are never more than $(1 - \epsilon)^{-1}$ times the optimum. Evidently, the ϵ here is used to measure how far away the approximate solution is from the optimum.

For each NP-complete optimization problem A we shall be interested in determining the smallest ϵ for which there is a *polynomial-time* ϵ-approximation algorithm for A. Sometimes no such ϵ exists, but there are also approximation algorithms that achieve arbitrarily small error ratios.

Definition 5.2 *The* approximation threshold *of A is the greatest lower bound of all $\epsilon > 0$ such that there is a polynomial-time ϵ-approximation algorithm for A.*

The approximation threshold of an optimization problem can be anywhere between zero (arbitrarily close approximation is possible) and one (essentially no approximation is possible) (see [Pap94]). Of course, any optimization problem that has a polynomial-time algorithm has approximation threshold zero.

Our approximation algorithm for the arbitrary model N-K problem is best described in terms of the K-MAXGSAT problem. This is a generalization of the K-MAXSAT problem. In the K-MASGSAT problem, we are given a set of Boolean expressions in N variables, where each expression is a function of at most K of the variables. The Boolean expressions can be any any functions of their variables. (In the K-MAXSAT problem, the Boolean expressions must be disjunctions of literals.)

5.2 The Approximation Algorithm

The arbitrary-model N-K problem is similar to the MAXGSAT problem and we can solve it in a similar way.

There are N bits in the input string. Since each bit could only be 0 or 1, there are 2^N possible bit assignments. If we calculate the value of the N-K fitness function corresponding to each assignment, what is the average value of these values?

We will denote the set of all N-bit strings as S, the average value of a function f on the set S as $AVG(f^S)$, and the average value of $f_i \circ p_i$ on the set S as $AVG(f_i^S)$.

From the definition of N-K fitness function, we have

$$f = \sum_{i=1}^{N-1} f_i.$$

Therefore, we have

$$AVG(f^S) = \sum_{i=0}^{N-1} AVG(f_i^S).$$

To calculate $AVG(f^S)$, we have to remember that each function f_i depends on exactly $K+1$ bits of the string; therefore, if we identify the $K+1$-bit strings with the integers from 0 to $2^{K+1}-1$, there are only 2^{K+1} different values:

$$f_i[j] \text{ for } j = 0, 1, \ldots, 2^{K+1}-1$$

for each function. Then we have

$$AVG(f_i^S) = \frac{1}{2^{K+1}} \times \sum_{j=0}^{2^{K+1}-1} f_i[j].$$

Let m_i be the maximum value of f_i. Since the sum of a sequence of nonnegative numbers is always greater than its maximum, we have

$$\sum_{j=0}^{2^{K+1}-1} f_i[j] \geq m_i$$

We may thus obtain a lower bound for $AVG(f_i^S)$:

$$AVG(f_i^S) \geq \frac{1}{2^{K+1}} \times m_i$$

Let the optimal value of f be MAX, then it is obvious that $\sum_{i=0}^{N-1} m_i \geq MAX$, and we have the following conclusion:

Lemma 5.3 *The average value of the N-K fitness function satisfies*

$$AVG(f^S) \geq \frac{1}{2^{K+1}} \times MAX.$$

Using the result above, we will try to reduce the size of the string set to 1 without reducing the average value of the N-K fitness function corresponding to the set; this will allow us to find an approximate solution for the N-K fitness optimization problem.

Suppose that we set bit 0 to 1 in all string assignments; then we have a set $S1$ of string assignments which only involve bit 1 through bit $N-1$, and we can again calculate the average value $AVG(f^{S1})$ of the N-K fitness function for this set of string assignments. Similarly if we set bit 0 to 0, then we have a set $S0$ of string assignments. Let the average of f for this set be $AVG(f^{S0})$. Since sets $S0$ and $S1$ have the same size, now it is very easy to see that

$$AVG(f^S) = \frac{1}{2}(AVG(f^{S1}) + AVG(f^{S0}))$$

This equation shows that, if we set bit 0 to 0 when $AVG(f^{S1}) < AVG(f^{S0})$ and 1 otherwise and we replace S by $S0$ in the first case ($S1$ in the second), then we end up with a string set with average value at least as large as the original $AVG(f^S)$.

We can continue like this, always splitting the string into two subsets and assigning to the next bit the value that maximizes the average function value on the resulting string set. In the end, all bits have been assigned values. However, since our average value never decreases in the process and the last set we have will only have one member left, we know that the value of the N-K fitness function corresponding to the final string (since now we have only one string in the set, it is the same as the average value of the set now) is at least as large as

$$\frac{1}{2^{K+1}} \times MAX.$$

These remarks suggest the following algorithm for approximating a solution for the N-K fitness optimization problem.

```
N-K_OPTIM(S)
/// s is the approximately optimal string.
/// S is initialized to the set of all N-bit strings
/// and evolves as the bits of S are assigned;
/// eventually, S = {s}.
  for i from 0 to N − 1 do
    S0 ← subset of S where the ith bit is 0
    M0 ← average of f over S0
    S1 ← subset of S where the ith bit is 1
    M1 ← average of f over S1
    if M0 > M1 then
      s[i] ← 0
      S ← S0
    else
      s[i] ← 1
      S ← S1
/// The approximately optimal string is now in s
  return s
```

Algorithm 2

Notice that inside the "for" loop, each step needs to calculate at most $2^{K+1} \times N$ values (each f_i depends on at most $K+1$ bits). Therefore, the complexity of the algorithm is

$$N \times 2^{K+1} \times N = N^2 \times 2^{K+1},$$

which is a polynomial in N.

Now we have an approximation threshold for the algorithm:

Theorem 5.4 *The approximation threshold for the algorithm with $K \geq 2$ is at most $1 - \frac{1}{2^{K+1}}$.*

Proof: The approximation threshold for the algorithm is at most

$$\frac{MAX - \frac{1}{2^{K+1}} \times MAX}{MAX} = 1 - \frac{1}{2^{K+1}}.$$

■

In fact, the string given by this algorithm is no more than average, but nothing more than that is guaranteed. There might be a way, such as hill-climbing, to optimize it, but no better approximation threshold has been found so far. However, since this problem is closely related to the $(K+1)$-*MAXSAT* problem, and the best known approximation threshold for the latter is $1 - \frac{1}{2^{K+1}}$ (see [Pap94]), it is unlikely that the approximation threshold of the arbitrary model class of N-K functions can be better than $1 - \frac{1}{2^{K+1}}$. This is not a very good approximation, since the approximation threshold we have established is quite close to 1, especially when K is large, but it is still possible that one could do some better kind of approximation for this problem.

5.3 Classifying the Problem by Approximation Threshold

Depending on how large the approximation threshold is, we can roughly divide the approximation problem into three categories (from the most difficult to the least difficult):

1. The approximation threshold for the problem is 1. Many approximation problem fall into this category. The unrestricted INDEPENDENT SET and CLIQUE problems belong to this category, but they are right on the edge because we have the following result from [Pap94]: unless $P = NP$, the approximation threshold of INDEPENDENT SET and CLIQUE is one. As we mentioned before, it is thought to be impossible to do any polynomial-time approximation for these problems.

2. The approximation threshold for the problem is at most $\varepsilon \in (0, 1)$. Some well-known problems such as NODE COVER, MAXSAT and Maximum Cut belong to this category. This is another common case among the approximation problems. There is a special class called MAXSNP and many approximation problems with a threshold less than one belong to the class MAXSNP.

3. The approximation threshold for the problem is 0. That is, for any $\varepsilon > 0$ there is a polynomial ε-approximation algorithm for the problem. There is a sequence of algorithms whose error ratios have limit 0. In other words, this is the kind of problem for which approximation can be arbitrarily close. This is called a *polynomial-time approximation scheme.* The KNAPSACK problem is one example of such a problem. This category is evidently the "easiest" among these three categories.

Since we have already shown that the approximation threshold for the N-K fitness problem is at most $1 - \frac{1}{2^{K+1}}$, this problem might belong to the second category or the third, depending on whether the approximation is limited or unlimited.

6 Conclusion

Our version of the N-K fitness landscapes generalize Kauffman's N-K fitness landscapes by allowing arbitrary component functions rather than stochastically defined component functions. We have given an algorithm for optimizing the adjacent-model N-K landscapes which is polynomial in N. This implies that if $K \in O(\log N)$, then the problem of optimizing adjacent-model landscapes is of polynomial time complexity. The problem of optimizing arbitrary-model N-K landscapes where the ith component function is not required to depend on string component i are NP-complete for $K \geq 1$. (More precisely, the corresponding decision problem is NP-complete.)

From another point of view, the dependency of the component functions of the fitness function on the positions of the domain string is defined by a graph if $K = 1$ (or a hypergraph if $K > 1$). For $K = 1$, if each connected component of this graph contains a single cycle, the optimization problem is polynomial in N. If we remove the restriction that the ith component function depends on the ith symbol, then even with $K = 1$, the above graph can be any graph, and the problem is NP-complete. For $K > 1$ the arbitrary model N-K optimization problem is NP-complete.

In terms of approximation algorithms, the arbitrary model N-K problems is similar to the MAXGSAT (generalized MAXSAT) problem. There is a polynomial-time approximation algorithm, but the guaranteed approximation is not very good.

Kauffman [Kau93] empirically compared the adjacent-model and random-model N-K landscapes using hill-climbing. He compared the mean fitness of local optima under the two models, and found very little difference. He also compared mean walk lengths to local optima, and again found very little difference. Weinberger [Wei96] computed formulas for the correlation between strings of Hamming distance d under both models. For the random-model landscape, he obtained:

$$R(d) = 1 - \frac{d(k+1)}{N} + \frac{d(d-1)k(k+2)}{2N^2} + O\left(\left(\frac{dk}{N}\right)^3\right)$$

and for the adjacent-model landscape, he obtained:

$$R(d) = 1 - \frac{d(k+1)}{N} + \frac{d(d-1)k(k+1)}{2N^2} + O\left(\left(\frac{dk}{N}\right)^3\right)$$

These are clearly very close. Thus, statistically the adjacent and random model N-K landscapes are very close, at least relative to a Hamming-distance topology.

Relative to a crossover topology, we might expect different results. The component functions f_i can be considered as "building blocks", and when K is small relative to N, one-point or two-point crossover preserves most of the values of the f_i from one parent or the other on adjacent-model fitness functions, but does not on random-model fitness functions.

The results of Kauffman and Weinberger [Wei96] make the results of this paper seem somewhat surprising. It may be that the "algorithmically hard" problems are a statistically small subset of the random-model N-K fitness landscapes.

References

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractibility.* W. H. Freeman and Company, San Francisco, 1979.

[Kau93] Stuart A. Kauffman. *The Origins of Order.* Oxford University Press, New York, 1993.

[Pap94] Christos Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, MA, 1994.

[Tho95] Richard K. Thompson. Fitness landscapes investigated. Master's thesis, University of Montana, Mansfield Library, Missoula, MT, 59812, 1995.

[Wei96] Edward D. Weinberger. Np completeness of kauffman's n-k model, a tuneable rugged fitness landscape. Technical Report 96-02-003, Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, New Mexico 87501 USA, 1996. http://www.santafe.edu/sfi/publications/96wplist.html.

[Zha97] Jain Zhang. Nk fitness functions. Master's thesis, University of Montana, Mansfield Library, Missoula, MT, 59812, 1997.