What can we learn from No Free Lunch? A First Attempt to Characterize the Concept of a Searchable Function

Steffen Christensen

Computer Science Dept.
Carleton University
Ottawa, ON K1S 5B6
steffen@scs.carleton.ca
(613) 520-2600 x 1857

Franz Oppacher

Computer Science Dept.
Carleton University
Ottawa, ON K1S 5B6
oppacher@scs.carleton.ca
(613) 520-2600 x 3520

Abstract

The No Free Lunch theorem has had considerable impact in the field of optimization research. A terse definition of this theorem is that no algorithm can outperform any other algorithm when performance is amortized over all functions. Once that theorem has been proven, the next logical step is to characterize how effective optimization can be under reasonable restrictions. We operationally define a technique for approaching the question of what makes a function searchable in practice. This technique involves defining a scalar field over the space of all functions that enables one to make decisive claims concerning the performance of an associated algorithm. We then demonstrate the effectiveness of this technique by giving such a field and a corresponding algorithm; the algorithm performs better than random search for small values of this field. We then show that this algorithm will be effective over many, perhaps most functions of interest to optimization researchers. We conclude with a discussion about how such regularities are exploited in many popular optimization algorithms.

1 INTRODUCTION

The No Free Lunch Theorem (NFL) states that no single algorithm outperforms random search (equivalently, systematic linear search) when amortized over all functions (Wolpert and Macready, 1995). This is easy to see and, indeed, unsurprising: if nothing is known about the structure of a domain, then all problems in it are equally likely; and any particular search or learning algorithm can be expected to perform better than chance on some randomly selected problems and worse than chance on others, and on average it can be expected to do no better (and no worse) than random guessing.

The apparently disheartening conclusion of NFL depends on the assumption that nothing is known about the structure of a domain; by the same token, the theorem may be taken to point out the importance of assumptions about non-uniformities in a domain for an understanding of the success of search and learning algorithms.

In this paper we try to specify general and minimal non-uniformities that seem to be realized in nearly all application domains, and that enable certain algorithms to outperform random search. In particular, we believe that virtually all functions that a practitioner would consider searching exhibit the requisite non-uniformity and are thus unaffected by NFL.

In the following, we consider what makes a function searchable, and then give an algorithm that outperforms random search on functions thus characterized.

2 WHAT MAKES A FUNCTION SEARCHABLE?

We all know that so-called "general-purpose" optimization algorithms are at least modestly effective on a wide range of combinatorial optimization problems. In fact, most general-purpose optimization algorithms greatly outperform random search on functions commonly used in testing these algorithms. Indeed, while random and linear search find minima in time linear in the size of the search space, these algorithms are running in time exponential in the input.

Most "general-purpose" optimization techniques rely on some sort of hill climbing at the lowest level. These techniques include golden section search, Brent's method, the downhill simplex method, direction-set methods, conjugate gradient methods, quasi-Newton methods, simulated annealing, the genetic algorithm, evolution strategies, evolutionary programming and various types of

hill climbers themselves (Press, 1992; Fogel, 1995; Mitchell, 1996; Schwefel, 1977). However, NFL guarantees even hill climbers will not outperform random search when amortized over all functions.

If all these methods are exploiting similar general and minimal sorts of regularities in the functions that they successfully search, then perhaps these regularities are encountered more often in functions of interest in the real world than the uniform distribution over which NFL applies would suggest. How can we reconcile our intuition and our experience with the efficacy of different types of search algorithms with the conclusion of NFL?

2.1 EXPERIMENTAL FRAMEWORK

We define a framework for talking about optimization problems in general, adopting the notation given in Macready and Wolpert (1996).

In general optimization problem, we are given a cost function $f : X \rightarrow Y$ to optimize. Without loss of generality, we assume that we are seeking the global minima of f. Since optimization techniques will in general be implemented on digital computers, we can take X and Y to be finite sets. (For instance, Y might be the set of floating-point numbers representable in a computer's native hardware representation, while X may be an ordered set of such floating-point numbers.) These sets are of size $|X|$ and $|Y|$, respectively.

A "population" d_m is defined as an ordered sample of m successive points chosen by a given algorithm from the cost function. $d_m \equiv \{d_m(i)\} \equiv \{d_m^x(i), d_m^y(i)\}$. Note that this definition differs from the concept of population used in the evolutionary computation literature, in that it includes *all* of the points ever evaluated, while in evolutionary computation we normally consider a population to be a working set of points in the domain that may or may not overlap with points selected in previous generations.

Let D_m be the set of all populations of size m, and let $D = \bigcup_m D_m$ be the set of all populations of any size. An algorithm that attempts to find minima of f can be made maximally efficient if it does not revisit any points. (This can be implemented in any standard optimization algorithm by caching the function evaluations, and by looking up subsequent evaluations from the cache.) We can therefore define an algorithm as a mapping $a: d \in D \rightarrow \{x \mid x \notin d^x\}$.

Let $\omega(f)$ be a vector of length $|Y|$ such that each element ω_i is the number of points $x \in X$ such that $f(x) = \omega_i$. That is, $\omega(f)$ is the histogram of the y-values of f. Define $\Omega(f)$ as the vector of the successive nonzero components of $\omega(f)$. $\Omega(f)$ thus corresponds to the ordered non-zero histogram of the distribution of values of f. f_1 and f_2 are then said to be in the same equivalence class iff $\Omega(f_1) = \Omega(f_2)$. That is, two functions are in the same equivalence class if the same number of points in X map to each corresponding point in Y. Macready and Wolpert make several important points about trying to compare the relative hardness of functions taken from different equivalence classes in (Macready and Wolpert, 1996); we refer the reader to their discussion therein for more information.

It is our belief that general-purpose optimization of a particular random function about which nothing is known *a priori* can only hope to be successful if there is a degree of "self-similarity" in the function. By self-similarity, we mean that function values of points that are "near" to a given point are "related" to that point's value. Such self-similarity is, in general, difficult to characterize completely; it seems that a truly accurate definition would have to rely on the notion of Kolmogorov complexity such as "what is the ratio of the size of the Turing machine of minimum length that can reproduce the patterns observed in the function to the size of the function's domain?" However, such a measure would be computationally intractable. The approach we choose in this paper is to define a technique that allows us to specify, investigate and analyze our own operational measure of self-similarity. Thus, if you have in mind a specific kind of regularity or pattern that your optimization technique effectively exploits, you may be able to operationally define that regularity and make concrete statements about the effectiveness and generality of your optimization technique.

We begin by requiring a total order of all points in the domain and in the range. For traditional multidimensional optimization in digital computers, we can simply take the order of the complete binary representation of the points in memory. For fixed-point and floating-point discretizations, this is equivalent to saying that larger values in the first dimension sort higher than lower values, and that relative values in succeeding dimensions break ties.

Without loss of information, we can define a function isomorphism that maps the elements of the domain of our original function f to the integers in $[1..\delta]$, and maps the elements of the range to the integers in $[1..\rho]$. (δ and ρ are used to suggest "size of domain" and "size of range", respectively. Note that δ is the size of the preimage of f, while ρ is the size of the image of f.) The simplest such isomorphism is to index the elements of X and Y, and in our new function, preserve the mapping of the indices of X to the indices of Y. We designate this generated function $r(f)$, and call the set of all such functions R, $R : \mathbb{N}_\delta \rightarrow \mathbb{N}_\rho$. The *successor* graph of such a function r is

the matrix $G(r)$ whose elements g_{ij} are defined by[1]: $g_{ij} = \sum_{i=0}^{\delta-1} [r(i+1) = j]$.

Note that $\sum_{j=1}^{\rho} g_{ij}$ = the i-th component of the previously introduced index function $\Omega(f)$. Furthermore, invertible functions have $\rho = \delta$, and therefore all components of $\Omega(f)$ are 1. A random function will, in general, exhibit no relation between $r(i)$ and $r(i+1)$; therefore the corresponding matrix $G(r)$ will reveal few or no patterns to exploit.

2.2 TOWARDS UNDERSTANDING OPTIMIZATION

We now begin a general process to classify the functions taken from a function space such as R with respect to the NFL theorem. As Wolpert and Macready state (1995), algorithms that do better in some regions of R must do correspondingly worse in others. To attempt to characterize which functions are "searchable", we adopt the following procedure: we define a computable scalar field S over R; we then show a systematic relation between the values of this field on functions in R and the performance of some corresponding algorithm A. (Note that this concept may readily be extended to include vector fields over R.)

If we can then show that the performance of this algorithm A will be better in areas of R where $S(r)$ is high, we will have effected a partition of the function space R based on searchability. Thus, we can begin to operationally address the question of how well we *can* do in general-purpose optimization, *given* that NFL holds.

By way of explanation, we here offer a definition of one such field and a corresponding algorithm. The goal of this algorithm, SUBMEDIAN-SEEKER, is to choose points in the search space that lie below the median of r as early in the selection process as possible.

We define the field $M(r)$ as follows:

Let $M(r) = \sum_{i=0}^{\rho/2} \sum_{j=\rho/2+1}^{\rho} g_{ij}$, where g_{ij} is the i, j-th component of $G(r)$. Thus $M(r)$ is the number of function values that lie below the median whose successors' values lie above the median.

We now define a general-purpose optimization algorithm, SUBMEDIAN-SEEKER, which attempts to use information gathered on the points observed thus far to predict future points. In the following, we express the algorithm in terms of a random function f, for generality. Because of the possibility of constructing a function isomorphism between any real-valued function f and the corresponding ranked function r, the algorithm applies equally in both cases. However, the analysis is simplified if we continue to work in the uniform function space, R.

2.3 ALGORITHM

SUBMEDIAN-SEEKER(f):

$s \leftarrow \{0,0\}$ -- *will count the number of sampled points with successors below the median, for the first and second halves of the domain.*

$z \leftarrow \{0,0\}$ -- *will count the number of sampled points with successors above the median*

$i \leftarrow 0$

$j \leftarrow 0$

Until j = *Number of points to sample* do:

  Choose a next point for the algorithm to consider as follows:

    $window \leftarrow [f(x_i) < median(f)]$

    If $s_{window} > z_{window}$ then

      $x_{i+1} \leftarrow x_i + 1$

    Else

      $x_{i+1} \leftarrow$ *Random unchosen point*

  $i \leftarrow i + 1$

  Evaluate $r(x_i)$.

  $j \leftarrow j + 1$.

  If $j \neq$ *Number of points to sample* then

    If $f(x_i + 1)$ has not been evaluated then evaluate $f(x_i + 1)$ and set $j \leftarrow j + 1$.

  Add the $f(x_i)$ vs. $f(x_i + 1)$ information to the graph $\boldsymbol{G}$ as follows:

    If $f(x_i) < med(f)$, then increment $s_{[f(x_i)>=med(f)]}$

    Else increment $z_{[f(x_i)>=med(f)]}$.

2.4 ALGORITHM SUMMARY

This algorithm basically builds up information about the function by sampling. s and z store information on the number of values that map to successor values below and above the median, respectively. This information is used to guide further progress in the search. This algorithm can be expected to perform well if previous performance is an accurate indicator or predictor of future performance. There is, of course, no guarantee that this is the case; however, if many points in the function have values and successor values that happen to co-occur below the median, then SUBMEDIAN-SEEKER will be able to exploit these regularities. We will see later that many functions that we might attempt to search will have this characteristic.

[1]Note that here and elsewhere in this paper, we use Knuth's shorthand notation for predicate functions (Graham, Knuth, and Patashnik, 1994): $[x = y]$ is a terse way of specifying the function $\begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$.

2.5 EXPERIMENTATION ON SUBMEDIAN-SEEKER

We define the performance $\phi(a, r)$ of an algorithm a as the number of points discovered in m samples that are below the median. In Wolpert and Macready's notation, (Wolpert and Macready, 1995):

$$\varphi(a \mid r, m) = \sum_{i=1}^{m} [d_i^y < \rho / 2]$$

We would expect that any differences between the distribution of points selected by SUBMEDIAN-SEEKER and that of a random-search algorithm would be maximized at $m = \delta/2$; after this point, the requirement to sample only previously unselected points would force the algorithm to begin making unfavourable choices. We therefore run the algorithm until it has chosen exactly half the points in the domain - it is expected (and verified by experiment) that this will maximize any performance difference between random search and SUBMEDIAN-SEEKER.

Therefore, in every test in the present work, we ran the algorithm until exactly half of the points in the domain were selected. (It can be inferred from this that we generally chose to investigate function with domains of modest size, unlike those used in serious optimization research!) For simplicity, we worked only over the invertible functions I_n, with $\delta = \rho = n$.

Since SUBMEDIAN-SEEKER is a stochastic algorithm (i.e. the points it chooses are selected at random if no other information is available), it will generate a distribution of outcomes for any given source function r. However, this distribution is fixed for a given r. We are therefore free to probe the distribution of outcomes by rerunning the algorithm as many times as is necessary to ascertain the true distribution as precisely as is required.

There are many choices for how one could evaluate the performance of SUBMEDIAN-SEEKER, particularly since its performance will be contrasted with that of random search, another stochastic algorithm. The measure used herein is to compare the mean performance of SUBMEDIAN-SEEKER with the (known) mean performance of random search using the Student's t-test. In general, the use of a t-test to ascertain the difference in means of two random discrete distributions is not valid, because of the assumptions that the t-test makes of normality and homoscedasticity of the variables (see Neter et al, 1996, pp. 407-408). Therefore the accuracy of this simplification was explicitly tested using a Monte Carlo simulation technique. 10 000 simulation runs of 100 experiments each were performed, and the run data were used to estimate parameters of the observed distributions, such as the mean and standard deviation. T-tests were then performed for each run to attempt to reject the incorrect hypothesis that the population mean was greater than a predetermined threshold. The agreement between the observed confidence levels and those estimated from observed distributions using the Student's t-test were excellent. In addition, we used very large confidence intervals to compensate for this uncertainty.

Note that the only test that SUBMEDIAN-SEEKER performs is to see whether the value of the cost function is above or below the median. (For the moment, we assume that there are no points exactly at the median of r; i.e. δ is even.) Therefore, the performance of our algorithm can only depend on the particular sequence of sub- and super-median points in r. We can formalize this observation by defining the 0-1 sequence $s(r)$, where $s_i(r) = [r(i) > median(r)]$. Thus the performance of SUBMEDIAN-SEEKER will only depend on $s(r)$, and not on the specific values that r assumes. In other words, SUBMEDIAN-SEEKER only uses part of the information available to it in deciding where to search next.

It was hoped that $M(r)$ would completely describe the performance of SUBMEDIAN-SEEKER on a function r; however, simulation showed that even when $M(r)$ was fixed, there was a significant dependence on the structure of $s(r)$. This can be seen in Figure 1, where the distribution of SUBMEDIAN-SEEKER performance over a fixed $M(r)$ of 8 is shown[2].

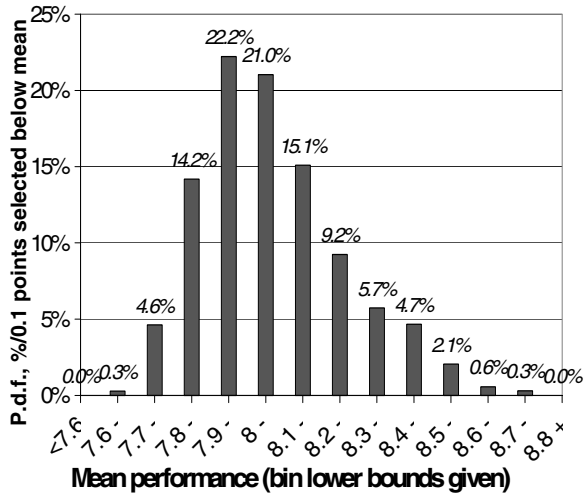

Figure 1: Distribution of mean performance of SUBMEDIAN-SEEKER on 5000 random functions with $n = 32$ and $M(r) = 8$. Performance is defined as the number of evaluated points (out of 16) whose values were below the mean. Mean performance were determined in each case by taking the mean of 1 000 000 runs of SUBMEDIAN-SEEKER. The standard error of the means ($2\ \sigma_{\bar{x}}$) are all less than 0.005.

[2] For details on how one generates random functions satisfying a given value of $M(r)$, see section 5, Appendix.

Consider a function with a small value for $M(r)$: what are its properties? Since $M(r)$ measures the number of submedian values of r that have successors with supermedian values, we know immediately that points below the median are likely to be followed by points below the median. Therefore, we might expect that an algorithm that uses sampling to determine where to go next to perform well on such functions. Indeed, this is observed in practice[3], as demonstrated in Figure 2, which is a representative scatter plot of mean performance obtained over 10 runs as we vary $M(r)$ from 0 to 15.

We might further expect the worst-case performance of SUBMEDIAN-SEEKER over many functions to be monotonic decreasing in $M(r)$; this too was observed (data not shown here). There must therefore be a "critical value" of M, below which all functions r in $\{r \in I_n \mid M(r) < M_{crit}\}$ will perform better than random search, on average.

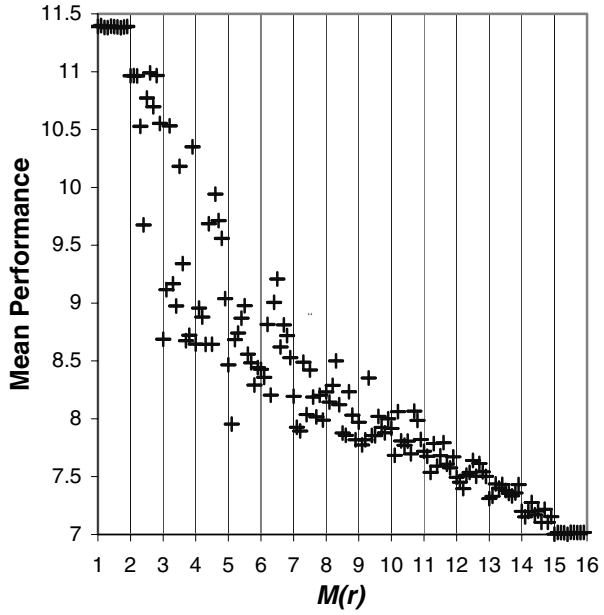

Figure 2: Average Performance of SUBMEDIAN-SEEKER with $n = 32$. Successive columns correspond to increasing values of $M(r)$; within a column, values are replicates with different randomly chosen functions. The standard errors of the means ($2\sigma_{\bar{x}}$) are all less than 0.009.

To understand the worst-case behaviour of SUBMEDIAN-SEEKER, we examined which patterns in s gave the poorest mean outcome. Careful experimentation, involving examining 10 000 random functions over 5 levels of M (not detailed here) revealed that the worst outcomes happened when the supermedian values were as unevenly distributed as possible, subject to the constraint that they occur in pairs; and when the submedian values were as symmetrically distributed as possible[4]. For instance, in the case where $n = 32$, $M = 3$, the worst outcomes occured with the pattern 12+; 6-; 2+; 5-; 2+; 5-.

Once the specific nature of this worst-case input was characterized, an algorithm was developed that generated and tested this worst case as the number of points in the function, n, and $M(r)$ were varied. In each case, 100 runs of SUBMEDIAN-SEEKER were performed, and a t-test of the hypothesis "the mean of this observed distribution is larger than that expected by chance" was performed at a significance level of α=0.001. If the t-test was inconclusive, the number of runs was doubled and the process repeated until it was conclusively proven either that the mean of the distribution is greater than or less than that expected by chance. A replicate of these data was obtained with α=0.000 01; no change in the results was observed. This test procedure was used to determine M_{crit}. This process was repeated for several values of n; the results are graphed in Figure 3.

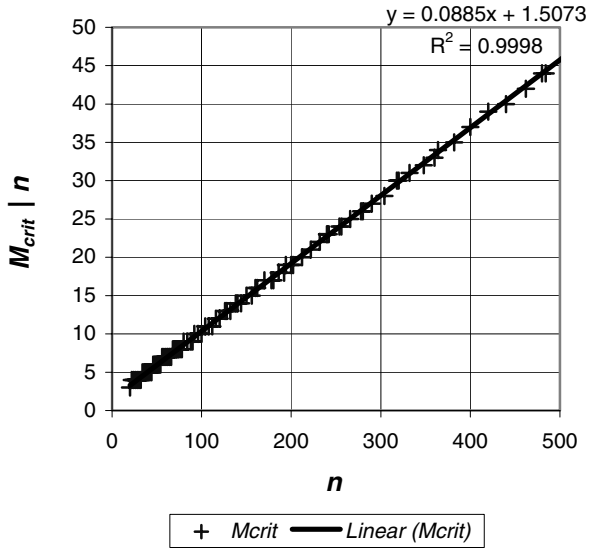

Figure 3: Graph of M_{crit} vs. n, for several values of n

As is evident from the graph, a clear linear trend to the data is observed. In fact, if we approximate the value of M_{crit} as $M_{crit} = \left\lfloor \frac{10n}{113} \right\rfloor + 1$, we are only incorrect in 2 of the cases experimentally sampled, wherein it is off by exactly 1.

[3] Note also that we would expect for random search to find that half of the sampled points fall below the median. For the case of figure 2, where 16 points were sampled, chance would lead us to expect 8 points below the median.

[4] The notation 12+; 6-; refers to the case where 12 consecutive points were above the median, followed by 6 below the median, and so on.

We shall designate the quantity $\lim_{n\to\infty}\frac{M_{crit}}{n}$ as fr_{crit}. Note that the data do not permit selection of one particular value for fr_{crit}; 10/113ths is only the simplest such fraction that fits the data - all that is known is that the true multiplier is in the interval [0.088452, 0.088542].

Therefore, for all functions $r \in R \mid M(r) < M_{crit}$, we have an algorithm that can be expected to perform better than random chance at selecting points below the median. However, since the only property that we have used is that points in a function are above or below its median, the same argument applies for any function $f \in F$.

3 ANALYSIS

Such a result is only interesting if commonly searched functions have this property. We will show now that, because of the definition of $M(f)$ and SUBMEDIAN-SEEKER, many functions indeed do have this property.

If we restrict ourselves to 1-dimensional functions of a single real variable, we can say the following:

1. If $p \in P^k$, $p \neq 0$, is a uniformly sampled polynomial of degree at most k, and if $M_{crit} > k/2$, then SUBMEDIAN-SEEKER will perform better than random search on p.

Proof: There are most k solutions to $p(x) = b$. If we choose b = med p, then we know that there can be at most k median-crossings in total over the sampled interval, and specifically there can be at most $k/2$ crossings from submedian values of p to supermedian values of p. Since $M(p)$ measures exactly the number of such crossings, if $M(p) < M_{crit}$ then SUBMEDIAN-SEEKER will perform better than random search, on average, and the stated result follows.

2. If $f(x) = \sum_{j=0}^{k} a_j e^{-2\pi i j x}$ is a truncated Fourier series of at most k harmonics uniformly sampled over [0,1) at n locations, and if $M_{crit} > k/2$, then SUBMEDIAN-SEEKER will perform better than random search on f.

Proof: Write $f = \sum_{j=0}^{k} a_j e^{-2\pi i j x} = \sum_{j=0}^{k} a_j \left(e^{-2\pi i x}\right)^j$. This is a polynomial of degree k in $e^{-2\pi i x}$, and therefore can have at most k solutions for $e^{-2\pi i x} = b$. If we choose $e^{-2\pi i x}$ = med f, then we have that over the one period of the interval sampled, there can be at most k median crossings. The rest of the proof follows as in (1).

3. General valley case: If each extremum of a sampled function f is represented by $\frac{2}{fr_{crit}} \doteq \frac{113}{20}$ = 5.65… points on average, SUBMEDIAN-SEEKER will perform better than random search on f.

Proof: Suppose that there are k extrema of a sampled function. Since there can be at most 1 crossing of any central line between any pair of extrema, there can be at most $k/2$ crossings from submedian values of f to supermedian values of f. The rest of the proof follows as in (1).

4. Multidimensional functions: Let $p \in P_d^k(\bar{x})$ be a multivariate polynomial of dimension d and of degree at most k in each variable, regularly sampled l times in each dimension over a multidimensional rectangular interval. Assuming that we use the indexing function $In_p(\bar{x}) = \sum_{i=1}^{d} rank(\bar{x}_i) l^{i-1}$ to index the vector $\bar{x}$, if $k < \frac{2M_{crit}}{n} l - 2$, SUBMEDIAN-SEEKER will perform better than random search on p.

Proof: The values of the indexed function will vary most rapidly in the first dimension. There can therefore be at most $k/2$ crossings of the median from below for every sequence of l points from the polynomial, plus at most 1 more jump discontinuity in advancing the higher dimensional indices. So if $M_{crit} < l^{d-1}(\frac{k}{2}+1)$, SUBMEDIAN-SEEKER will outperform random search. Since $n = l^d$, the stated result follows.

The obvious extensions to multidimensional truncated Fourier series and limits on extrema also hold.

4 CONCLUSION

The No Free Lunch theorem has been very seriously considered and is very useful, especially in light of some of the sometimes-outrageous claims that had been made of specific optimization algorithms. However, once that theorem has been proven, the next logical step is to characterize how effective optimization can be under modest restrictions. We have operationally defined a technique for characterizing what makes a function "searchable". We have demonstrated the effectiveness of this technique by example - SUBMEDIAN-SEEKER will outperform random search over many functions of interest.

We emphasize that this is not merely one particular algorithm performing well in a particular domain; but instead a rather general and widely applicable set of conditions over which a given algorithm has been demonstrated to surpass random search. Let us demonstrate this point with an example. Consider the case of an optimization algorithm attempting to solve a multidimensional optimization problem using a vector of single-precision floating-point numbers as the domain. Such a representation has on the order of $l = 2^{23}$ points in any given octave in the domain. In particular, if we take the domain to be the hypercube $[x, 2x]^d$, then by (4)

above, SUBMEDIAN-SEEKER will perform better than random search on any multivariate polynomial function of 1484707^{th} degree or lower.

We finally note that such modest non-uniformity of a source function is all that is needed for some optimization algorithms to outperform random search. While it may be useful and indeed is likely to improve efficiency to specialize a general-purpose algorithm towards the characteristics of a particular problem, this point is unrelated to the challenge posed by the No Free Lunch theorem.

Acknowledgments

The authors would like to thank the members of Carleton University's Artificial Life and Evolutionary Computing group for their valuable feedback during early versions of these results. Special thanks to Dr. S. Melkovian for his assistance in the proof of the Fourier result, and to Dr. Mark Wineberg for his advocacy of achieving understanding before publishing.

This work is supported in part by grants from the National Science and Engineering Research Council of Canada.

References

D.B. Fogel, *Evolutionary Computing: Toward a New Philosophy of Machine Intelligence,* Piscataway, New Jersey, IEEE Press, 1995.

R.L. Graham, D.E. Knuth, O. Patashnik, *Concrete Mathematics; a foundation for Computer Science*, 2nd Ed., Reading, Mass., Addison-Wesley, 1994.

W.G. Macready, D.H. Wolpert, *What Makes An Optimization Problem Hard,* SFI-TR-95-05-046, Oper. Res., (1996).

M. Mitchell, *An Introduction To Genetic Algorithms,* Cambridge, Mass., London, England, MIT Press, 1996.

J. Neter, M.H. Kutner, C.J. Nachtsheim, W. Wasserman, *Applied Linear Statistical Models*, 4th Ed., Boston, Mass., McGraw-Hill, 1996.

W.H. Press, *Numerical Recipes in C; the Art of Scientific Computing,* 2nd Ed., Cambridge, England, U.P., 1992.

H.-P. Schwefel, *Numerische Optimierung von Computermodellen mittels der Evolutionsstrategie*, Basel, Germany, Birkhaeuser, 1977.

D.H. Wolpert, W.G. Macready, *No Free Lunch Theorems For Search,* SFI-TR-95-02-010, Oper. Res., (1995).

5 APPENDIX

While the main text of the paper describes the experimental setup and experimentation thereon, there are a few details required to duplicate the experiments that the authors felt were best left outside of the main flow of the text. Inasmuch as is possible, we present those details here.

5.1 GENERATING AN UNBIASED RANDOM FUNCTION WITH A GIVEN $M(r)$

We consider only the case where $\delta = \rho = n$ (that is, the functions generated by this algorithm are invertible). We give a sketch of an algorithm to generate a random function (call it f) that has the property that $M(f)$ is the given input parameter.

By way of example:

Suppose that n is 32 and M is 4. There are therefore 4 points in r that are below the median that have successors that lie above the median. Consider the graph G of the values of f versus the values of their successors (as in Figure 4). We can characterize such a graph by noting that it divides nicely into 4 quadrants. Along the x-axis, there are points whose values are below the median, and points whose values are above the median. Along the y-axis, there are successor values both below and above the median. Numbering the quadrants as usual in mathematics, we find that M describes the number of points in the second quadrant. By symmetry relations (exactly half the points are below the median), we can immediately see that there are $n - M$ points in quadrants I and III, and also M points in quadrant IV.

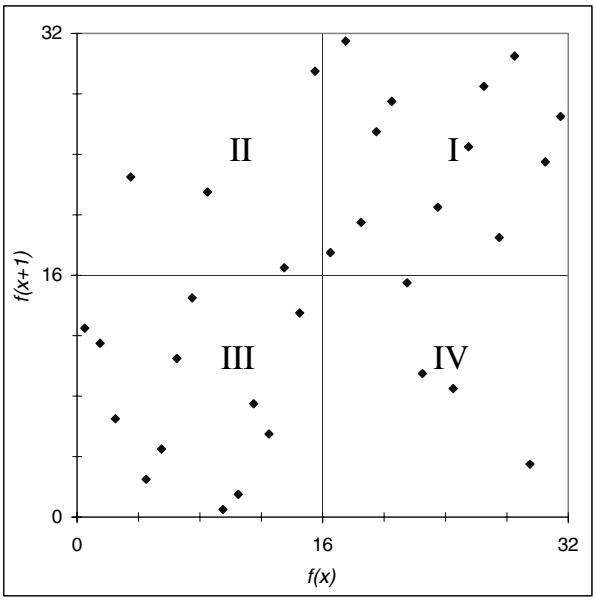

Figure 4: Sample graph of function values plotted against their successors, with $n = 32$ and $M = 4$.

With these observations in hand, we can go about developing an algorithm[5] to randomly generate such a function that satisfies a particular value of M.

We go about generating such a function in reverse; first we develop the graph of $f(x)$ vs. $f(x+1)$, then we build a function from this graph.

We begin by placing the aforementioned number of points randomly in each quadrant. Coordinates are chosen at random; if any two points in any quadrant have the same x- or y-coordinate, we generate all the points anew. (It can be shown that this will occur with very low probability, if we have a modest value of n and we use 8-byte IEEE floating point numbers to store the coordinates).

We then replace each point's x- and y-coordinate by the relative ranks of their coordinates – thus converting the domain and range of our function to $\mathbb{N}_n$. We can represent this information as a vector which, for each value of $f(x)$, store $f(x+1)$. Call this vector v. At this point, it remains only to determine the values for the function f thus generated.

We choose the value for $f(0)$ at random from $[1, n]$.

We then look up successive values $f(1)$, $f(2)$... from our vector v. There are two possibilities:

- We encounter a cycle
- We encounter no cycle

If we don't encounter any cycles, we are done; so we continue assuming that there are cycles present.

We make a list of all the cycles that we've found, and of their consistuent members. For each cycle we find, we then attempt to break the cycle, while still preserving the property that exactly the desired number of points ends up in each quadrant in G.

For each member m of the cycle c, we try the following:

Note the quadrant of m. We then search for a point that lies in the same quadrant of G as m, but is not a member of the c. If we can find such a point (call it p), then swap m and p, and we've just linked two disjoint cycles. If no such point exists, try again with another member m of c. If we run out of members, then we have a cycle that cannot be merged with other cycles; at this point we give up and generate points randomly from the beginning.

As an optimization, after enumerating all the cycles, we attack the cycles in order of size, from smallest to largest. Since a smaller cycle has fewer elements that might be replaced, it is thought that this would improve overall performance, by forgoing analysis of those graphs that can not possibly generate a function.

---

[5] As an aside, it seems likely that elements of this algorithm have been published elsewhere. However, as the authors independently developed this technique to solve the present problem, we do not know to whom to attribute original discovery of such original elements. We would appreciate any guidance in this matter.

The Phase Transition in NK Landscapes is Easy

Yong Gao
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada, T6G 2H1
ygao@cs.ualberta.ca

Joseph Culberson
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada, T6G 2H1
joe@cs.ualberta.ca

Abstract

In this paper, we analyze the decision version of the NK landscape model from the perspective of threshold phenomena and phase transitions under two random distributions, the uniform probability model and the fixed ratio model. For the uniform probability model, we prove that the phase transition is easy in the sense that there is a polynomial algorithm that can solve a random instance of the problem with the probability asymptotic to 1 as the problem size tends to infinity. For the fixed ratio model, we establish several upper bounds for the solubility threshold, and prove that random instances with parameters above these upper bounds can be solved polynomially. This, together with our empirical study for random instances generated below and in the phase transition region, suggests that the phase transition of the fixed ratio model is also easy.

1 INTRODUCTION

The *NK landscape* is a fitness landscape model devised by Kauffman [Kauffman, 1989]. An appealing property of the NK landscape is that the "ruggedness" of the landscape can be tuned by changing some parameters. Over the years, the NK landscape model itself has been studied from the perspectives of statistics and computational complexity [Weinberger, 1996, Wright *et al.*, 1999]. In the study of genetic algorithms, NK landscape models have been used as a prototype and benchmark in the analysis of the performance of different genetic operators and the effects of different encoding methods on the algorithm's performance [Altenberg, 1997, Hordijk, 1997, Jones, 1995].

In the field of combinatorial search and optimization, one of the interesting discoveries is of threshold phenomena and phase transitions. Roughly speaking, a phase transition in combinatorial search refers to the phenomenon that the probability that a random instance of the problem has a solution drops abruptly from 1 to 0 as the order parameter of the random model crosses a critical value called the *threshold*. Closely related to this phase transition in solubility is the hardness of solving the problems. There has been strong empirical evidence and theoretical arguments showing that the hardest instances of the problems usually occur around the threshold and instances generated with parameters far away from the threshold are relatively easy. Since the seminal work of Cheeseman et al. [1991], many NP-complete combinatorial search problems have been shown to have the phase transition and the associated easy-hard-easy pattern of hardness. The hardest instances usually occur around the solubility threshold [Cook and Mitchell, 1997, Culberson and Gent, 2000, Gent *et al.*, 1998, Kirkpatrick and Selman, 1994, Vandegriend and Culberson, 1998].

In this paper, we analyze the NK landscape model from the perspective of threshold phenomena and phase transitions. We establish two random models for the decision problem of NK landscapes and study the threshold phenomena and the associated hardness of the phase transitions in these two models.

The rest of the paper is organized as follows. In Section 2, we introduce the NK fitness landscape and our probabilistic models, the uniform probability model and the fixed ratio model. In Section 3, the threshold phenomena and phase transitions in NK landscapes are analyzed. For the uniform probability model, we prove that the phase transition of the uniform probability model is easy in the sense that there is a polynomial algorithm that can solve a random instance of the problem with the probability asymptotic to 1 as

the problem size tends to infinity. For the fixed ratio model, we establish several upper bounds for the solubility threshold, and prove that random instances with parameters above these upper bounds can be solved polynomially. This, together with our empirical study for random instances generated below and in the phase transition region, suggests that the phase transition of the fixed ratio model is also easy. In section 4, we report our experimental results on typical hardness of the fixed ratio model. In Section 5, we conclude our investigation and discuss implications of our results.

2 NK Landscapes and their Probabilistic Models

An NK landscape $f(x) = \sum_{i=1}^{n} f_i(x_i, \Pi(x_i))$, is a real-valued function defined on binary strings of fixed length, where $n > 0$ is a positive integer and $x = (x_1, \cdots, x_n) \in \{0,1\}^n$. It is the sum of n *local fitness functions* f_i, $1 \leq i \leq n$. Each local fitness function $f_i(x_i, \Pi(x_i))$ depends on the *main variable* x_i and its neighborhood $\Pi(x_i) \subset \{x_1, \cdots, x_n\} \backslash \{x_i\}$. The main parameters of an NK landscape are the number of variables n, and the size of the neighborhood $k = |\Pi(x_i)|$.

In an NK landscape, the neighborhood $\Pi(x_i)$ can be chosen in two different ways: the *adjacent neighborhood*, where k variables with indices nearest to i (modulo n) are chosen, and the *random neighborhood*, where the k variables are randomly chosen from the set $\{x_1, \cdots, x_n\} \backslash \{x_i\}$. Once the variables in the neighborhood are determined, the local fitness function f_i is determined by a fitness lookup table which specifies the function value f_i for each of the 2^{k+1} possible assignments to the variables x_i and $\Pi(x_i)$. See [Altenberg, 1997] for a detailed discussion on the fitness lookup table.

Throughout this paper, we consider NK landscapes with random neighborhoods. To simplify the discussion, we further assume that the local fitness functions take on binary values. Given an NK landscape f, the corresponding decision problem is stated as follows: Is the optimum of $f(x)$ equal to n? An NK landscape decision problem is insoluble if there is no solution for it.

It has been proved in [Weinberger, 1996, Wright *et al.*, 1999] that the NK landscape model is NP complete for $k \geq 2$. The proofs were based on a reduction from SAT to the decision problem of NK landscapes. To study the typical hardness of the NK landscape decision problems in the framework of thresholds and phase transitions, we introduce two random models. In both of the models defined below, the neighborhood set $\Pi(x_i)$ of a variable x_i is selected by randomly choosing without replacement $k = |\Pi(x_i)|$ variables from $x \backslash \{x_i\}$.

Definition 2.1 *The Uniform Probability Model $\overline{N}(n,k,p)$: In this model, the fitness value of the local fitness function $f_i(x_i, \Pi(x_i))$ is determined as follows: For each assignment $y \in Dom(f_i) = \{0,1\}^{k+1}$, let $f_i(y) = 0$ with the probability p and $f_i(y) = 1$ with the probability $1-p$, where this is done for each possible assignment and each local fitness function independently.*

Definition 2.2 *The Fixed Ratio Model $N(n,k,z)$: In this model, the parameter z takes on values from $[0, 2^{k+1}]$. If z is an integer, we specify the local fitness function $f_i(x_i, \Pi(x_i))$ by randomly choosing without replacement z tuples of possible assignments $Y = (y_1, \cdots, y_z)$ from $Dom(f_i) = \{0,1\}^{k+1}$, and defining the local fitness function as follows:*

$$f_i(y) = \begin{cases} 0, & \text{if } y \in Y; \\ 1, & \text{else.} \end{cases}$$

For a non-integer $z = (1-\alpha)[z] + \alpha[z+1]$, we choose randomly without replacement $[(1-\alpha)n]$ local fitness functions and determine their fitness values according to $N(n,k,[z])$. The rest of the local fitness functions are determined according to $N(n,k,[z]+1)$.

3 Threshold Phenomena and Phase transitions in NK Landscapes

In this section, we study the threshold phenomena and phase transitions of the two random models established in Section 2. We prove that the phase transition in the probability model is easy by showing that there is a polynomial algorithm that can solve a random instance of the problem with the probability asymptotic to 1 as the problem size tends to infinity. For the fixed ratio model, we establish several upper bounds for the solubility threshold, and prove that random instances with parameters above these upper bounds can be solved linearly or polynomially.

3.1 The Uniform Probability Model

In the uniform probability model $\overline{N}(n,k,p)$, the parameter p determines how many zero values a local fitness function can take. We are interested in how the solubility and hardness of the NK landscape decision problem change as the parameter p increases from 0 to 1. The main result on the uniform probability model is summarized in the following theorem.

Theorem 3.1 *For any $p(n)$ such that $\lim_n p(n)n^{\frac{1}{2^k+1}}$ exists, k fixed, there is a polynomial time algorithm that successfully solves a random instance of $\overline{N}(n,k,p)$ with probability asymptotic to 1 as n tends to infinity.*

The proof of Theorem 3.1 can be found in the appendix.

3.2 The Fixed Ratio Model

As has been discussed in the previous section, the uniform probability model $\overline{N}(n,k,p)$ of NK landscapes is asymptotically trivial. This is largely due to the fact that if the parameter p does not decrease very quickly with n, then asymptotically there will be at least one local fitness function that takes the value 0 for all the possible assignments, making the whole decision problem insoluble. In this section, we study the fixed ratio model $N(n,k,z)$. In this model, we require that each local fitness function has fixed number of zero values so that the trivially insoluble situation in the uniform probability model is avoided. We note that the same idea has been used in the study of the *flawless CSP* [Gent *et al.*, 1998].

We will establish several upper bounds on the solubility threshold of the parameter z, and theoretically prove that random instances generated with the parameter z above these upper bounds can be solved with probability asymptotic to 1 by polynomial (even linear) algorithms.

3.2.1 The Upper Bound of $z = 3.0$

The derivation of this upper bound is based on the concept of a conflicting pair of local fitness functions. We say that two local fitness functions f_i and f_j conflict with each other if (1)f_i and f_j have exactly one common variable x and (2)for any assignment $s \in \{0,1\}^n$, we have $f_i(s)f_j(s) = 0$. It is obvious that an instance of the NK decision problem is insoluble if there exists a pair of conflicting local fitness functions. Based on the second moment method in the theory of probability [N.Alon and J.H.Spencer, 1992], we can prove the following result.

Theorem 3.2 *Define X to be the event that there is a conflicting pair of local fitness functions in $N(n,2,z)$. For the fixed ratio model $N(n,2,z)$ with $z = 3.0 + \varepsilon$, we have*

$$\lim_n Pr\{X\} = 1$$

and thus is insoluble with probability asymptotic to 1.

Since it takes linear time to check if there is a pair of conflicting local fitness functions, we conclude that the fixed ratio model $N(n,2,z)$ is linearly solvable when $z > 3.0$.

3.2.2 2-SAT Sub-problems in $N(n,2,z)$ and a Tighter Upper Bound

In this subsection, we establish a tighter upper bound $z > 2.837$ for the threshold of the fixed ratio model $N(n,2,z)$ by showing that asymptotically $N(n,2,z)$ contains an unsatisfiable 2-SAT sub-problem with probability 1 for any value of z greater than 2.873. This also gives us a polynomial time algorithm which determines that $N(n,2,z)$ is insoluble with probability asymptotic to 1 for $z > 2.837$.

Recall from Section 2 that each instance of $N(n,2,z)$ has an equivalent 3-SAT instance. The idea is to show that with probability asymptotic to 1, an instance of $N(n,2,z)$ will contain a set of special structured 3-clauses, called a t-3-module [Franco and Gelder, to appear]:

$$\mathcal{M} = \{M_1, \ldots, M_{3p+2}\}$$

where

$$\begin{aligned}
M_1 &= (\bar{u}_1 \vee u_2 \vee z_1, \bar{u}_1 \vee u_2 \vee \bar{z}_1);\\
&\cdots\\
M_{p-1} &= (\bar{u}_{p-1} \vee u_p \vee z_{p-1}, \bar{u}_{p-1} \vee u_p \vee \bar{z}_{p-1});\\
M_p &= (\bar{u}_p \vee \bar{u}_0 \vee z_p, \bar{u}_p \vee \bar{u}_0 \vee \bar{z}_p);\\
M_{p+1} &= (\bar{u}_{p+1} \vee u_{p+2} \vee z_{p+1}, \bar{u}_{p+1} \vee u_{p+2} \vee \bar{z}_{p+1});\\
&\cdots\\
M_{3p-1} &= (\bar{u}_{3p-1} \vee u_{3p} \vee z_{3p-1}, \bar{u}_{3p-1} \vee u_{3p} \vee \bar{z}_{3p-1});\\
M_{3p} &= (\bar{u}_{3p} \vee u_0 \vee z_{3p}, \bar{u}_{3p} \vee u_0 \vee \bar{z}_{3p})\\
M_{3p+1} &= (\bar{u}_0 \vee u_1 \vee z_{3p+1}, \bar{u}_0 \vee u_1 \vee \bar{z}_{3p+1});\\
M_{3p+2} &= (u_0 \vee v_{p+1} \vee z_{3p+2}, u_0 \vee u_{p+1} \vee \bar{z}_{3p+2});
\end{aligned}$$

and $u_1, \cdots, u_{3p+1}, z_1, \cdots, z_{3p+1}$ are binary variables. Notice that a t-3-module can be reduced to a 2-SAT problem containing two contradictory cycles and hence is unsatisfiable.

The result is proved in two steps. In the first step, it is shown that for $z > 2.837$ the average number of t-3-modules contained in $N(n,2,z)$ tends to infinity as n increases. In the second step, we use a result of Alon and Spencer [N.Alon and J.H.Spencer, 1992] on the second moment method to prove that for $z > 2.837$ the probability that $N(n,2,z)$ contains at least one t-3-module tends to 1. The two steps are summarized in the following two theorems and the proof can be found in [Gao, 2001].

Theorem 3.3 *Let A_t be the number of t-3-modules*

contained in $N(n,2,z)$ *and* $t = \Theta(\ln^2 n)$. *Then, if* $z = 2 + \alpha > 2.837$, $\lim_{n\to\infty} E\{A_t\} = \infty$.

Theorem 3.4 *If* $z = 2 + \alpha > 2.837$, *then* $N(n,2,z)$ *is insoluble asymptotically with probability 1.*

Since both of the tasks of converting a NK landscape instance to a 3-SAT instance and identifying a t-3-module can be done in polynomial time, it follows that the fixed ratio model $N(n,2,z)$ is asymptotically polynomially solvable for $z > 2.837$.

4 Experiments

Our study of the threshold phenomena in NK landscapes started with an experimental investigation. Many of the theoretical results in the previous section are motivated by the observations made in our experiments. In this section, we describe the approach and methods we used in the experimental study, and report the results and observations we have made.

In our experiments, an instance of the NK landscape decision problem is converted to an equivalent 3-SAT problem, and then the 3-SAT problem is solved using Roberto's relsat—an enhanced version of the famous Davis-Putnam algorithm for SAT problems implemented in C^{++}. The source code of relsat can be found at http://www.cs.ubc.ca/ hoos/SATLIB/solvers.html.

In the experiments, we generated random instances of the NK landscape decision problem from the random model $N(n,2,z)$. As a result, the equivalent SAT problem for each random NK landscape instance is a 3-SAT problem with n variables and (on average) zn clauses. By definition, the parameter z is between 0 and 8. For $z \leq 1$, the 3-SAT instance can be solved easily by setting the literals that correspond to the main variables of the local fitness function to true. As z increases, we get more and more clauses and the 3-SAT problem becomes more and more constrained. The aims of the experiments are threefold:(1)Investigating if there exists a threshold phenomenon in the random NK landscape model; (2) Locating the the threshold of the parameter z; and (3)Determining if there are any hard instances around the threshold.

4.1 Experiments on the Original Fixed Ratio Model

In this part of the experiments, we generate 100 random instances of $N(n,2,z)$ for each of the parameters $n = 2^9 \cdots 2^{16}$ and $z = 2.71 + offset, 0 \leq offset \leq 0.29$. These instances are then converted to 3-SAT instances and solved by *relsat*. Figure 1 shows the fraction of insoluble instances as a function of the parameter z. It can be seen that there exists a threshold phenomenon and the threshold is around 2.83. This shows that our upper bound $z = 2.837$ is very tight.

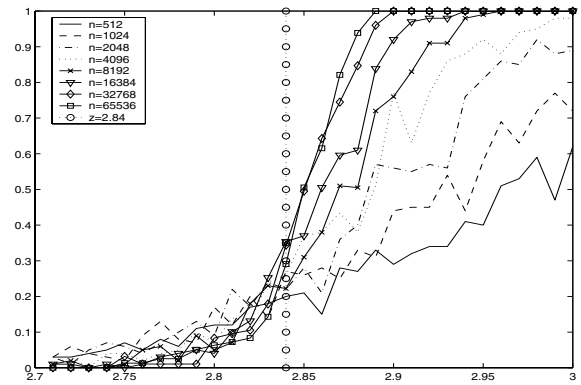

Figure 1: Fractions of insoluble instances(Y-axis) as a function of z (X-axis).

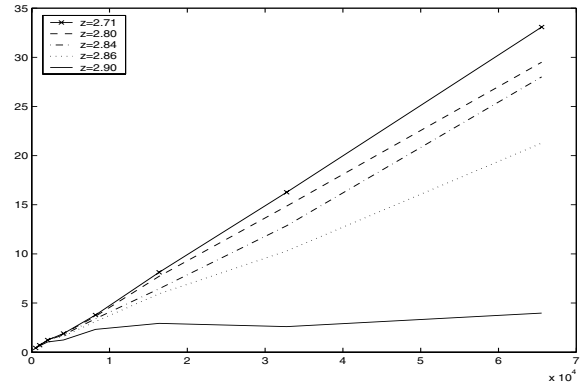

Figure 2: Square root of the average search cost (Y-axis, in seconds) as a function of n (X-axis).

In Figure 2, we plot the square root of the average search cost as a function of the parameter n. The figure indicates that the average search is in $O(n^2)$ for any parameter z. We have also observed that more than 99 percent of the insoluble instances are solved quickly in the preprocessing stage of relsat. This indicates that there must be some "small" structures that make the instances insoluble. For the detailed experimental results, see [Gao, 2001].

4.2 Experiments on the 2-SAT sub-Problem

This is the part of the experiments that motivated our theoretical analyses in Section 3.2.2. The idea can be

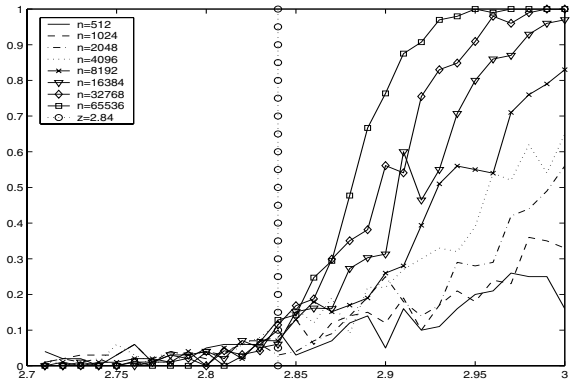

Figure 3: Fractions of insoluble instances(Y-axis) as a function of z (X-axis) for 2-SAT sub-problems.

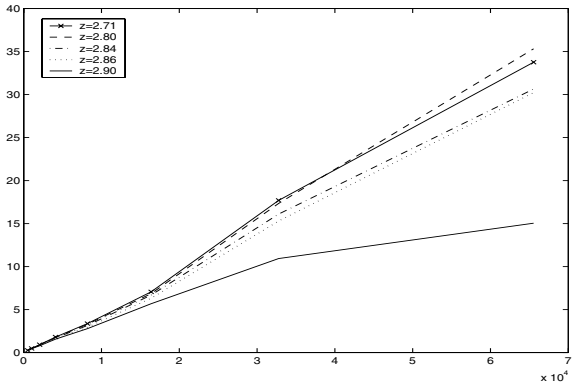

Figure 4: Square root of the average search cost (Y-axis, in seconds) as a function of n (X-axis) for 2-SAT sub-problems.

explained as follows. Let $f(x) = \sum_{i=1}^{n} f_i(x_i, \Pi(x_i))$ be an instance of the decision problem of NK landscape and $\varphi = C_1 \bigwedge C_2 \cdots \bigwedge C_n$ the equivalent 3-SAT problem where C_i is the set of 3-clauses equivalent to the local fitness function f_i. For each i, there is a set of 2-clauses D_i(possibly empty) implied by C_i. For example, if C_i has three 3-clauses $((x, y, z), (x, \bar{y}, z), (x, y, \bar{z}))$, then the set of 2-clauses D_i would be $((x, z), (x, y))$. The conjunction of D_i, denoted by $\bar{\varphi}$, is a 2-SAT problem. It is obvious that the original 3-SAT problem φ is satisfiable only if the 2-SAT sub-problem $\bar{\varphi}$ is satisfiable.

The experimental settings are the same as those in the experiment on the original problem. The results are shown in Figures 3-4, in parallel to the Figures 1-2 of the results on the original 3-SAT problems in Section 4.1. We see that the patterns of insoluble fractions and search cost are similar to those we found in the original 3-SAT problems. There is a soluble-insoluble phase transition occurring around 2.83, but the fraction of unsatisfiable instances is lower than the fraction in the original 3-SAT problems.

We also observed that the average search cost for the 2-SAT sub-problems remains the same as that for the original 3-SAT problems. This tells us that the difficulty of solving a soluble instance of NK landscape is almost the same as that of solving a 2-SAT problem, and hence is easy. Therefore, on average the NK landscape $N(n, 2, z)$ is also easy at parameters below the threshold where almost all of the instances are soluble.

5 Implications and Conclusions

One of the questions that arises about this work is its implications to the design and analysis of genetic algorithms. NK landscapes were initially conceived as simplified models of evolutionary landscapes which could be tuned with respect to ruggedness and epistatic interactions [Kauffman, 1989]. In the study of genetic algorithms, NK landscape models have been used as a prototype and benchmark in the analysis of the performance of different genetic operators and the effects of different encoding methods on the algorithm's performance [Altenberg, 1997, Hordijk, 1997, Jones, 1995]. Kauffman, (1993, pages 40ff) points out that the parameters that primarily affect a number of ruggedness measures are n and k. Nevertheless, the fact that for $k \geq 2$ the discrete NK landscape is NP-complete [Wright *et al.*, 1999] when the neighbors are arbitrarily chosen could be construed as implying that random landscapes with fixed k are in practice hard.

The results in this paper should serve as a cautionary note that this may not be the case. Our analyses show that for fixed k the uniform probability model is trivially solvable as the problem size tends to infinity. For the fixed ratio model, we have derived two upper bounds for the threshold of the solubility phase transition, and proved that the problem with the control parameter above the upper bounds can be solved in polynomial time with probability asymptotic to 1 due to the existence of easy sub-problems such as 2-SAT. A series of experiments has also been conducted to investigate the hardness of the problem with the control parameters around and below the threshold. From the experiments, we have observed that the problem is also easy around and below the threshold.

Our proofs hold only for the decision version of the problem where the component functions are discrete on $\{0, 1\}$. The proofs are obtained by noticing that the clustering of functions, or clauses, on selected subsets of variables implies that the overall problem is

decomposable into independent subproblems, or that the problem contains small substructures that identify the solution. The subproblems are the components of the connection graph defined in the proof in the appendix.

In response to the question 'what are the implications for GAs?' we suggest the following speculative line of enquiry. For the discrete model we use, the soluble instances are readily solved by a standard algorithmic approach based on recognizing the components of the connection graph. A similar connectivity can be developed for real valued distributions, for example by capping the minimum value which we allow a subfunction to take. We can speculate that the clustering imposed by fixed values of k would also generate localized structures when real values are applied and when considering optimization instead of decision, but perhaps with fuzzy boundaries. In fact, this observation is just the flip side of limited epistasis. Genetic algorithms, designed to mimic natural evolution, are supposed to take advantage of this situation. So, to the extent that NK landscapes are an accurate reflection of the features exploited by evolutionary algorithms, we pose the following question. Is it possible to identify these fuzzy components if they exist, and in doing so design an algorithm that exploits the same landscape features that the evolutionary algorithms do, but far more efficiently, as we have done for the uniform discrete decision problem?

These landscapes were designed with the intent of studying limited interactions, and our results can also be seen as a confirmation that indeed limited epistasis leads to easier problems. In another domain, that of the more traditional research into search and optimization, there is a need for test bed problems with real world connections which are tunable with respect to difficulty. NK landscapes might have been such a domain for generating 3-SAT instances. It is disappointing that for restricted k the instances generated are easy with high probability.

Acknowledgements

This research supported in part by Natural Sciences and Engineering Research Council Grant No. OGP8053.

References

[Altenberg, 1997] L. Altenberg. NK fitness landscapes. In T. Back, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Oxford University Press, New York, 1997.

[Cook and Mitchell, 1997] Stephen A. Cook and David G. Mitchell. Finding hard instances of the satisfiability problem: A survey. In Du, Gu, and Pardalos, editors, *Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1997.

[Culberson and Gent, 2000] J. Culberson and I.P. Gent. Frozen development in graph coloring. Technical Report APES-19-2000, APES Research Group, 2000. To appear in Theoretical Computing Science.

[Franco and Gelder, to appear] J. Franco and A. Van Gelder. A perspective on certain polynomial time solvable classes of satisfiability. *Discrete Applied Mathematics*, to appear.

[Gao, 2001] Yong Gao. Threshold phenomena in nk landscapes. Master's thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, 2001.

[Gent *et al.*, 1998] Ian Gent, Ewan MacIntyre, Patrick Prosser, Barbara Smith, and Toby Walsh. Random constraint satisfaction: Flaws and structure. Technical Report APES-08-1998, APES Research Group, 1998.

[Hordijk, 1997] W. Hordijk. A measure of landscapes. *Evolutionary Computation*, 4(4):335–360, 1997.

[Jones, 1995] T. C. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Albuquerque, NM, 1995.

[Kauffman, 1989] Stuart Kauffman. Adaptation on rugged fitness landscapes. In Daniel L. Stein, editor, *Lectures in the Sciences of Complexity*, Santa Fe Institute Studies in the Sciences of Complexity, pages 527–618. Addison Wesley, 1989.

[Kauffman, 1993] Stuart A. Kauffman. *The Origins of Order: Self-organization and Selection in Evolution*. Oxford University Press, Inc., 1993.

[Kirkpatrick and Selman, 1994] S. Kirkpatrick and B. Selman. Critical behavior in the satisfiability of random boolean expressions. *Science*, 264:1297–1301, 1994.

[N.Alon and J.H.Spencer, 1992] N.Alon and J.H.Spencer. *The Probabilistic Method*. Wiley, New York, 1992.

[Vandegriend and Culberson, 1998] B. Vandegriend and J. Culberson. The $G_{n,m}$ phase transition is not

hard for the Hamiltonian Cycle problem. *Journal of Artificial Intelligence Research*, 9:219–245, 1998.

[Weinberger, 1996] Edward D. Weinberger. Np completeness of kauffman's N-K model, a tunable rugged fitness landscape. Technical Report Working Papers 96-02-003, Santa Fe Institute, Santa Fe, 1996.

[Wright *et al.*, 1999] Alden H. Wright, Richard K. Thompson, and Jian Zhang. The computational complexity of N-K fitness functions. Technical report, Department of Computer Science, University of Montana, 1999.

Appendix: Proof of Theorem 3.1

We consider two cases: (1) $\lim_n p(n)n^{\frac{1}{2^{k+1}}} = +\infty$ and (2) $\lim_n p(n)n^{\frac{1}{2^{k+1}}} < +\infty$.

(1) The Case of $\lim_n p(n)n^{\frac{1}{2^{k+1}}} = +\infty$.

Let A_i be the event that $f_i(y) = 0$ for each possible assignment $y \in \{0,1\}^{k+1}$ and let $A = \bigcup_{i=1}^{n} A_i$ be the event that at least one of the A_i's occurs. We have

$$\begin{aligned}\lim_{n\to\infty} Pr\{A\} &= 1 - \lim_{n\to\infty} Pr\{\bigcap_{i=1}^{n} A_i^c\} \\ &= 1 - \lim_{n\to\infty} (1 - p(n)^{2^{k+1}})^n.\end{aligned}$$

It can be shown that if k is fixed and $\lim_n p(n)n^{\frac{1}{2^{k+1}}} = +\infty$, then $\lim_{n\to\infty} Pr\{A\} = 1$. It follows that with probability asymptotic to one, there is at least one local fitness function which takes on values 0 for any possible assignments. We can therefore show that in this case, the NK decision problem is insoluble by checking the local fitness functions one by one. And this only takes linear time.

(2) The Case of $\lim_n p(n)n^{\frac{1}{2^{k+1}}} < +\infty$.

To prove that the theorem is true in this case. We need to introduce the concept of a connection graph for the NK landscape model and some related results.

Definition 5.1 *The connection graph of an NK landscape instance* $f(x) = \sum_{i=1}^{n} f_i(x_i, \Pi(x_i))$ *is a graph* $G = G(V, E)$ *satisfying*

(1) Each vertex $v \in V$ *corresponds to a local fitness function; and*

(2) There is an edge between v_i, v_j *if and only if the corresponding local fitness functions* f_i, f_j *share variables and both of them have at least one zero value.*

Definition 5.2 *Let* $f(x) = \sum_{i=1}^{n} f_i(x_i, \Pi(x_i))$ *be an NK landscape instance with the connection graph* $G = G(V, E)$*. Let* $G_1, \cdots, G_l$ *be the connected components of* G*. Since the vertices of* G *correspond to local fitness functions, we can regard* G_i *as a set of local fitness functions. For each* $1 \le i \le l$*, let* $U_i \subset x = (x_1, \cdots, x_n)$ *be the set of variables that appear in the definition of the local fitness functions in* G_i*.*

It's easy to see that $(U_1, \cdots U_l)$ excluding independent vertices forms a disjoint partition of (a subset of) the variables $x = (x_1, \cdots, x_n)$, and that the local fitness functions in G_i only depend on the variables in U_i. Furthermore, the NK decision problem is soluble if and only if for each $1 \le i \le l$, there is an assignment $s_i \in \{0,1\}^{|U_i|}$ to the variables in U_i such that for each local fitness function $g \in G_i$, $g(s) = 1$.

Now, let us consider an algorithm that first finds the connected components G_i, $1 \le i \le l$ of the connection G of the NK model, and then uses brute force to find an assignment $s_i \in \{0,1\}^{|U_i|}$ to the variables in U_i such that for each local fitness function $g \in G_i$, $g(s) = 1$. The time complexity of this algorithm is $O(n^2 + n * 2^{\mathcal{M}(n,k,p)})$ where $\mathcal{M}(n,k,p) = \max(|U_i|, 1 \le i \le l)$ is the maximum size of the subsets $(U_i, 1 \le i \le l)$ associated with the connected components of the connection graph. To prove the theorem, we only need to show that $\mathcal{M}(n,k,p) \in O(\log n)$. In the following, we will show that for $\lim_n p(n)n^{\frac{1}{2^{k+1}}} < +\infty$,

$$\lim_{n\to\infty} Pr\{\mathcal{M}(n,k,p) \le 2^k + 2\} = 1$$

Consider the connection graph $G = G(V, E)$ of the NK model. It is a random graph and there is an edge between two nodes if and only if the two corresponding local fitness functions share a variable and both of the local fitness functions take at least one zero as their fitness value. However, under this definition the edge probabilities are not independent. If $vx \in E$ then we know that f_x has at least one zero and so the probability that xw is in E is greater than if there were no other edge on x.

To deal with this we resort to the following proof construction. Let $C_m = \{v_1, \ldots, v_m\}$ be a subset of V of size m. Let π be an ordering (permutation) of $v_1 \ldots v_m$. We say that C_m *is variable connected with respect to the ordering* π, denoted as $\mathcal{C}(C_m, \pi)$, if for each $i, 2 \le i \le m$ there is either

1. a $j < i$ such that f_j and f_i share a variable or
2. there is a $j, 1 \leq j \leq m$ such that the variable x_j is one of the k random variables in f_i.

Lemma *If the induced subgraph $G[C_m]$ is connected then there exists at least one ordering π of $v_1 \ldots v_m$ such that $\mathcal{C}(C_m, \pi)$.*

As proof, consider the ordering of vertices of any depth first search of a connected subgraph. In this case, the connections are all by case 1.

The expected number of permutations π for which $\mathcal{C}(C_m, \pi)$ is

$$E_c = \mathrm{E}[|\{\pi : \mathcal{C}(C_m, \pi)\}|] = m! \Pr\{\mathcal{C}(C_m, \pi)\}$$

We then observe that the expected number of connected induced graphs on m vertices is less than $p_0^m \binom{n}{m} E_c$, where p_0 is the probability that at least one variable is zero in f_i. We show this value goes to zero in the limit if $m \geq 2^k + 2$. Finally, since if there is a connected connected subgraph on m vertices then there must be one for each $i < m$, it follows that the largest connected component is at most $2^k + 1$.

For a randomly generated permutation π of C_m, let C_i be the set of the first i vertices of the permutation. For $i \geq 2$ define P_i to be the probability that $f_{\pi(i)}$ shares at least one variable with $f_{\pi(j)}$ for some $j < i$ given that $\mathcal{C}(C_{i-1}, \pi)$. Let $P_1 = 1$. (A one vertex subgraph is always a connected.)

For $i > 1$ we have $P_i = \Pr\{\exists j < i, f_{\pi(i)}$ and $f_{\pi(j)}$ share a variable, given $\mathcal{C}(C_{i-1}, \pi)$ or one of the k random variables in $f_{\pi(i)}$ is in $\{x_1 \ldots x_m\} - \{x_i\}\}$.

$$\Pr\{\mathcal{C}(C_m, \pi)\} = \prod_{i=2}^{m} P_i$$

Finally, for $i > 1$ we note that C_{i-1} has at most $(i-1)k$ distinct other variables. If C_{i-1} is connected then the number of variables may be less than this. Thus,

$$P_i \leq 1 - \frac{\binom{n-k(i-1)-m}{k}}{\binom{n-1}{k}}$$

The combinatorial part reduces to

$$\frac{(n-k(i-1)-m)\ldots(n-k(i-1)-m-k+1)}{(n-1)\ldots(n-k)} \geq \left(\frac{n-ki-m+1}{n-1}\right)^k$$

So, $\Pr\{\mathcal{C}(C_m, \pi)\}$ is

$$\begin{aligned} &\leq \prod_{i=2}^{m} \left(1 - \left(\frac{n-ki-m+1}{n-1}\right)^k\right) \\ &\leq \left(1 - \left(1 - \frac{km+m-2}{n-1}\right)^k\right)^{m-1} \\ &\in O\left(\left(\frac{1}{n}\right)^{m-1}\right), m, k \text{ fixed} \end{aligned}$$

Noting that $p_o^m \in O\left(n^{\frac{-m}{2^k+1}}\right)$, we see that the expected number of connected subgraphs of size m is bounded by

$$p_0^m \binom{n}{m} E_c \in O\left(n^m n^{\frac{-m}{2^k+1}} \left(\frac{1}{n}\right)^{m-1}\right)$$

which goes to zero if $m = 2^k + 2$.

It follows that $\mathcal{M}(n, k, p)$ is less than $2^k + 2$ with probability asymptotic to 1. This completes the proof.

An Empirical Analysis of Collaboration Methods in Cooperative Coevolutionary Algorithms

R. Paul Wiegand
George Mason University
Computer Science Department
Fairfax, VA 22030
paul@tesseract.org

William C. Liles*
Central Intelligence Agency
Washington, DC 20505
wliles@gmu.edu

Kenneth A. De Jong
George Mason University
Computer Science Department
Fairfax, VA 22030
kdejong@gmu.edu

Abstract

Although a variety of coevolutionary methods have been explored over the years, it has only been recently that a general architecture for cooperative coevolution has been proposed. Since that time, the flexibility and success of this cooperative coevolutionary architecture (CCA) has been shown in a variety of different kinds of problems. However, many questions about the dynamics of this model, as well as the efficacy of various CCA-specific choices remain unanswered. One such choice surrounds the issue of how the algorithm selects collaborators for evaluation. This paper offers an empirical analysis of various types of collaboration mechanisms and presents some basic advice about how to choose a mechanism which is appropriate for a particular problem.

1 Introduction

In recent years there has been a growing interest in coevolutionary algorithms as interesting and useful extensions to the more traditional Evolutionary Algorithms (EAs). The important difference in moving to coevolutionary algorithms is that the fitness of an individual is a function of the other individuals in the population. Two basic classes of coevolutionary algorithms have been developed: competitive coevolution in which the fitness of an individual is determined by a series of competitions with other individuals (see, for example, Rosin and Belew (1996)), and cooperative coevolution in which the fitness of an individual is determined by a series of collaborations with other individuals (see, for example, Potter and De Jong (2000)). Both types of coevolution have been shown to be useful for solving a variety of problems.

In this paper our focus is on cooperative coevolutionary algorithms (CCAs). A standard approach to applying CCAs to a problem is to identify a natural decomposition of the problem into subcomponents. Each component is assigned to a subpopulation, such that individuals in a given subpopulation represent potential components to the greater problem. Then each component is evolved simultaneously, but in isolation to one another. In order to evaluate the fitness of an individual from a given subpopulation, collaborators are selected from the other subpopulations in order to form a complete solution.

While the CCA has shown definite promise on various problems, there is still a lot that we do not know about how the model works. One major question is the issue of how collaborators are chosen. It is clear from early work that problem characteristics are connected with this choice. For instance, problem landscapes with strong inter-activity between components seem to require less greedy methods for selection of collaborators.

In this paper we examine this choice by looking at three main aspects of collaboration: collaborator selection pressure, the number of collaborators for a given evaluation, and the assignment of fitness in evaluation when using multiple collaborators. These attributes are examined by a series of experimental studies on a variety of different function optimization problems.

In the next section we will discuss some background about coevolutionary approaches. The third section will describe the architecture in more detail, as well as illustrating the various choices surrounding collaboration. Then we will describe the experiments that were run and the results obtained. Finally, we will discuss our conclusions and offer some practical advice about how to do collaboration in the CCA in light of particular problems.

*This material has been reviewed by the CIA. That review neither constitutes CIA authentication or information nor implies CIA endorsement of the author's views.

2 Background

Although evolutionary algorithms (EAs) have been with us for half a century, significant research into the use of coevolutionary systems did not really begin until the early 1990's. From the start early research focused on applications of complex tasks, such as competitive approaches using a parasite-host relationship as a model to coevolve sorting networks and problem sets (Hillis, 1991), and cooperative approaches for coevolving job-shop schedules (Husbands and Mill, 1991).

However, although both cooperative and competitive approaches were explored from the beginning, most research since that time has dealt with applications of competitive approaches. Most popularly competitive coevolution has been applied to game playing strategies (Rosin and Belew, 1995, 1996; Rosin, 1997; Pollack and Blair, 1998). Additionally Angeline and Pollack (1993) demonstrate the effectiveness of competition for evolving better solutions by developing a concept of competitive fitness to provide a more robust training environment than independent fitness functions. Competition was also successfully harnessed by Schlierkamp-Voosen and Mühlenbein (1994) to facilitate strategy adaptation in breeder genetic algorithms.

More recently a variety of coevolutionary methods have been applied to machine learning problems. There has been particular attention to neural network coevolution (Paredis, 1994; Juillé and Pollak, 1996; Mayer, 1999; Potter and De Jong, 2000). Additionally, some work in using coevolutionary algorithms for concept learning has been done (Potter and De Jong, 1999).

Moreover, there have been recent attempts to lay out general frameworks for coevolutionary models (Potter and De Jong, 1994; Paredis, 1996). However, attempts to understand the dynamics of these frameworks have been few and far between. Some basic empirical work is laid out by Potter and De Jong (1994) and Potter (1997), indicating that there is a possible link between variable inter-activity and collaboration selection. Even more recently, some basic theoretical work has been done to take ideas from simple genetic algorithm theory provided by Vose (1999), and apply it to coevolution (Ficici and Pollack, 2000). This work explores the mechanics of a simple competitive coevolutionary algorithm from a game theoretic viewpoint.

These questions of coevolutionary dynamics are not academic. The question of selecting collaborators for evaluation, for instance, has not only been an issue for our application activities, but has also cropped up with other researchers who are applying the techniques to problems such as inventory control optimization Eriksson and Olsson (1997). Indeed, since this is a key element of the success of applying this cooperative coevolution architecture (CCA), we believe it merits particular attention in order to improve our ability to apply CCAs in the future.

3 Coevolution and Collaboration

When applying a CCA to a particular problem, a standard approach is to decompose the problem into subcomponents and assign each subcomponent to a subpopulation. These subpopulations may or may not be homogeneous with respect to the representation used or the EA being used to evolve a particular subcomponent.

Evolution proceeds independently, except for evaluation. Since any given individual from a subpopulation represents only a subcomponent of the problem, *collaborators* will need to be selected from the other subpopulations in order to assess fitness. Each generation, all individuals belonging to a particular subpopulation have their fitness evaluated by selecting some set of collaborators from other subpopulations to form complete solutions. Afterward, the CCA proceeds to the next subpopulation, which will in turn draw collaborators from each of the other subpopulations. A simple algorithm of this process is outlined below in figure 1.

$gen = 0$
for each species s **do**
  $Pop_s(gen) =$ initialized population
  evaluate($Pop_s(gen)$)
while not terminated **do**
  $gen++$
  for each species s **do**
    $Pop_s(gen) \leftarrow$ select($Pop_s(gen-1)$)
    recombine($Pop_s(gen)$)
    evaluate($Pop_s(gen)$)
    survive($Pop_s(gen)$)

Figure 1: The structure of a Cooperative Coevolutionary Algorithm (CCA).

Computing the fitness of individuals in a coevolutionary system can be done in a variety of ways. Some competitive coevolutionary algorithms perform bipartite evaluations, applying each individual in one population to each in the other (Hillis, 1991). Additionally, it is not uncommon for single population competitive fitness models to perform exhaustive pair-wise evaluations (Axelrod, 1989). Such approaches can be computationally expensive in multi–population models, since the number of objective function evaluations used for each assessment of fitness grows exponentially by the number of species. Less expensive approaches, such as single elimination tournaments have also been addressed (Angeline and Pollack, 1993).

Alternatively, early work in the CCA model (Potter and De Jong, 1994) suggests two methods for selecting collaborators for the purpose of fitness evaluation:

CCA-1 Choose the best individuals from alternative subpopulations, as defined by fitness obtained from the last evaluation process of that group.

CCA-2 Select two individuals: the best and a random individual. Evaluate both with the current individual and use the higher objective function value for the current individual's fitness score.

We could of course imagine many more such methods. Rather than speculate about potential collaboration choices, however, it is more useful to define some basic attributes of this choice. In our opinion, there are three such attributes:

- The degree of greediness of choosing a collaborator (*collaborator selection pressure*).
- The number of collaborators per subpopulation to use for a given fitness evaluation (*collaboration pool size*).
- The method of assigning a fitness value given multiple collaboration-driven objective function results (*collaboration credit assignment*).

We will examine all three of these attributes in this paper.

3.1 A Clearer Picture of Collaboration

Since in a CCA an individual represents only a *subcomponent* of a problem solution, collaborator subcomponents from each of the other subpopulations must be assembled to form a complete solution. We call the process of choosing these collaborator subcomponents *collaborator selection*. To do this, we can use the last evaluated fitness scores of the individuals in the alternative subpopulations to bias how we choose these collaborators. The degree of bias in this choice is what we are calling *collaborator selection pressure*.

This newly assembled complete candidate solution (a *collaboration*) can now be plugged into the objective function and assigned a collaboration score. If only one collaborator from each subpopulation is selected, there will only be a single collaboration score, and this score may be used as the fitness value.

However, we may choose to try different combinations of collaborators from the other subpopulations. In which case evaluation of an individual will consist of multiple collaborations. The number of collaborators selected from each subpopulation is what we call the *collaboration pool size*. Since each of these collaborations will have their own collaboration score from the objective function, these multiple scores must be resolved in some way to a single fitness value (*collaboration credit assignment*).

4 Experimental Methods

4.1 Fitness Landscapes

For our initial experimental studies on collaboration methods we elected to study simple function optimization problems. These types of problems are well–suited for the CCA, since a natural decomposition of the problem is very straightforward: each subpopulation represents a particular variable of the function. In all cases, we chose to examine only two variable landscapes, since increasing the dimensionality creates a combinatorial problem and raises questions about how multiple collaborators are applied. Future work on this matter is needed.

We use three fitness functions: Rosenbrock, Rastrigin, and a quadratic which is not directly aligned with the axes. The first two were chosen to be consistent with the Potter and De Jong (1994) study. They represent two difficult problems, one of which is not linearly separable (Rosenbrock) and the other which is linearly separable (Rastrigin). The third function was chosen because, although it is a simple problem conceptually, the lack of axis alignment has been shown to introduce problems for certain kinds of evolutionary algorithms (Salomon, 1996). Table 1 summarizes the functions used, as well as shows the constraints of those functions. In all cases the functions were to be minimized.

4.2 EA Characteristics

As previously stated, subpopulations of the CCA are homogeneous in our study. Again, where possible EA characteristics remain similar to previous studies. In all cases, the details of the evolutionary mechanisms are as follows:

representation:	binary (16 bits per function variable)
selection:	fitness proportionate with linear scaling
genetic operators:	two-point crossover & bit-flip mutation
mutation probability:	1 / chromosome length
crossover probability:	0.6
(sub)population size:	100
termination criteria:	100,000 function evaluations

Table 1: Function Test Suite

Function	Bounds	Name
$f_1(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$-2.048 \leq x_i \leq 2.048$	Rosenbrock
$f_2(x_1, x_2) = 6 + \sum_{i=0}^{2} x_i^2 - 3.0cos(2\pi x_i)$	$-5.12 \leq x_i \leq 5.12$	Rastrigin
$f_3(x_1, x_2) = x_1^2 + (x_1 + x_2)^2$	$-65.536 \leq x_i \leq 65.536$	Off-axis quadratic

5 Experimental Results

We used the experimental setup described above to run a large number of experiments in order to see if there were clear patterns indicating how best to design a CCA collaboration method. Recall that designing a collaboration method involves three decisions: how to assign fitness when there are multiple collaborations, how to choose collaborators, and how many collaborators to choose. Each of these issues is explored in the following subsections.

5.1 Collaboration Credit Assignment

We explored three methods for assigning an eventual fitness score to individuals who have had multiple collaborative function evaluations. These methods are as follows:

Optimistic: The more traditional method of assigning an individual a fitness score equal to the value of its best collaboration.

Hedge: Assign an individual a fitness score equal to the *average* value of its collaborations as is generally done in competitive coevolution.

Pessimistic: Assigning an individual a fitness score equal to the value of its worst collaboration.

The intuition for the latter option is that perhaps it might be best to use a "safe" credit assignment that rewards an individual only as well as its weakest collaboration. However, in all of our experiments this never turned out to be the case. In fact both the pessimistic and hedge strategies consistently resulted in significantly poorer performance in our studies, generally by several orders of magnitude. Some typical examples of these results can be seen in Table 2, involving minimizing a two variable Rosenbrock function when selecting two and three collaborators.

As a consequence of this consistent pattern, the remainder of the experiments discussed in this paper will only use the optimistic approach for credit assignment.

5.2 Collaboration Selection Pressure

The next attribute for deciding collaboration mechanics is the degree of greediness of choosing a collaborator.

Table 2: Example collaboration credit assignment results for the Rosenbrock minimization problem. The number represent averages across 50 trials. Collaborators are chosen at random.

Collaborator Poolsize	Credit Assignment	Result
2	Optimistic	43.35
2	Hedge	10,413.6
2	Pessimistic	7,063.22
3	Optimistic	25.82
3	Hedge	76,870
3	Pessimistic	46,843.5

CCA-1 uses a very greedy method, selecting the best individual from the previous generation. CCA-2, however, weakens this pressure somewhat by using a two collaborator mechanism in which the second collaborator is chosen at random. It is still quite greedy however, since the best individual is still used. We can imagine weakening this pressure even more by allowing for different combinations of selection mechanisms aside from selecting the best. Tables 3, 4, and 5 show results for all three functions with various combinations of collaborator pool size two using the three selection mechanisms: best, random, and worst.

Notice that no improvements are obtained on the Rosenbrock function *regardless* of whether the best individual is included as one of two collaborators. This finding is consistent with ANOVA at 95% confidence. In the Rastrigin function and the quadratic functions however, there is a clear significant advantage to using the best individual as one of the collaborators. We will return to this later in the paper.

We can weaken selection even further, as well as provide ourselves with a way of controlling the degree of collaborator selection pressure by using previous generation evaluation results to bias a non-deterministic choice of current collaborations. We use a method similar to tournament

selection, varying tournament sizes from 1 (random selection) to 4, which is a fairly strong bias. More extreme sizes were used, but are not presented in detail in this paper.

Table 3: Rosenbrock minimization results using various combinations of selection choices for each of two collaborators: best, worst, and random. These show averages of 50 trials, as well as 95% confidence intervals.

Rosenbrock	**Random**	**Best**
Worst	56.51 ±44.24	57.67 ±48.49
Random	37.82 ±24.29	68.28 ±88.43

Table 4: Rastrigin minimization results using various combinations of selection choices for each of two collaborators: best, worst, and random. These show averages of 50 trials, as well as 95% confidence intervals.

Rastrigin	**Random**	**Best**
Worst	6.43 ±1.93	0.54 ±0.03
Random	3.46 ±1.06	0.54 ±0.05

Table 5: Off-Axis quadratic minimization results using various combinations of selection choices for each of two collaborators: best, worst, and random. These show averages of 50 trials, as well as 95% confidence intervals.

Quadratic	**Random**	**Best**
Worst	295.07 ±199.19	2.85 ±5.45
Random	136.63 ±82.96	1.21 ±0.93

This tournament–like method should not be confused with that of selecting multiple collaborators. In the case where the collaboration pool size is only one, we can still have large tournament sizes without affecting the fact that only a single objective function application is required to evaluate an individual. This tournament-style collaborator selection method is used merely to leverage previous generation fitness values as a way to bias our search for a collaborator for the current evaluation. Of course the method is still reasonable for larger collaboration pool sizes, and again the size of the tournament will not impact the number of collaborations that are ultimately formed.

Figure 2 shows the results for the minimization experiments of the Rosenbrock function. The x-axis represents fitness scores. The points on the plot are averages of 50 trials for each experimental group. The whiskers show the 95% confidence intervals of these groups. Results for the basic GA, CCA-1, and CCA-2 groups are shown at the top for baseline comparison. The remaining four panels show the results for groups which were run using increasing collaboration selection pressure.

Notice that varying the pressure with this tournament-like collaborator selection method makes very little difference in the overall performance of the CCA. The f_1, f_2, and f_3 graphs show a similar story in this respect. Although the graphs in this paper show results for conservative ranges of this parameter, a range of extreme values (such as tournament sizes limiting toward the population sizes, and tournament selection of collaborators with the worst fitness score, etc.) were also run. These trends bear out even at extreme values.

More interesting perhaps is the fact that with the non-linear case, collaborator selection method neither helps, nor creates a performance degradation unless the selection bias is so strong that it effectively reduces the number of collaborators used (for extreme tournament sizes). Indeed, recall from Table 3 that using one best and one random (or worst) individual as collaborators in a two-way collaboration resulted in no significant performance difference.

Clearly it is not entirely the case that collaboration selection pressure is unimportant, however, since CCA-1 and CCA-2 do quite well against the linearly separable Rastrigin and the off-axis quadratic function compared to those CCA algorithms employing random selection.

One hypothesis is that the CCA-1 and CCA-2 algorithms' use of the most extreme collaboration selection pressure creates a search which is similar in behavior to a line-search. Therefore Rosenbrock non-linearities create a problem for which this bias gives us little or no advantage. In the case of the other two functions, this simple line-search type of behavior provides a strong bias which is well suited to solving these problems.

5.3 Collaboration Pool Size

The most dramatic effect on the success of the CCA was clearly the number of collaborators one uses. To some extent, computationally speaking this is an unfortunate, though not surprising, result since increasing the number of collaborators can significantly increase overall computation time—a problem which is combinatorial with the number of subpopulations.

Again look at figures 2, 3, and 4. In almost all cases increasing the number of collaborators used assisted the performance of the algorithm. In fact, although the relaxation of the greedy collaboration method hinders the CCA with respect to the GA on the Rastrigin function, increasing the collaborative pool size to 5 with random collaborator selec-

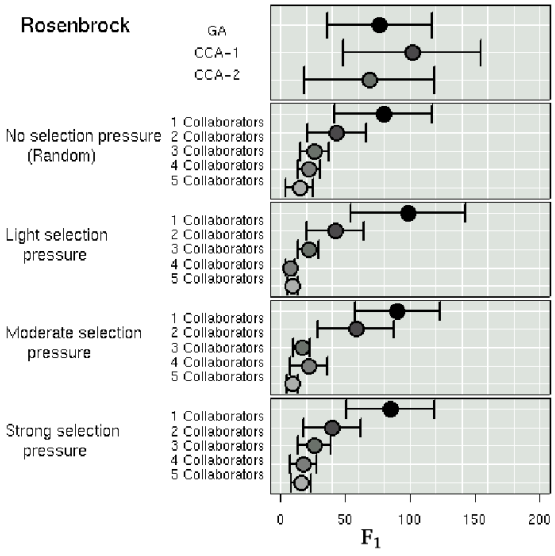

Figure 2: Results for Rosenbrock (f_1) minimization experiments. The x-axis represents the final reported result from the EA after 100,000 function evaluations. The points plotted are averages of 50 trials, and the whiskers show the 95% confidence intervals.

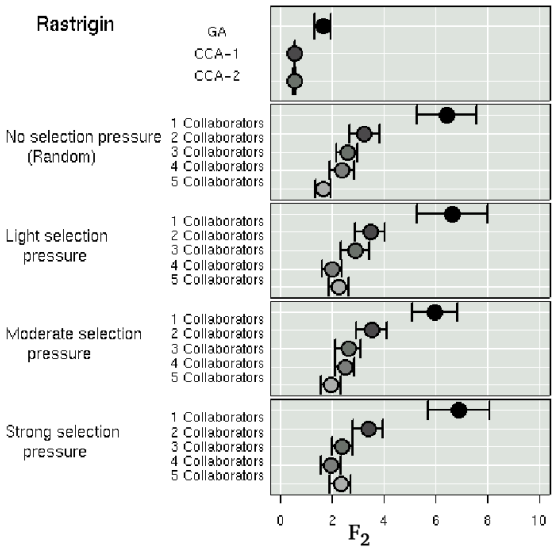

Figure 3: Results for Rastrigin (f_2) minimization experiments. The x-axis represents the final reported result form the EA after 100,000 function evaluations. The points plotted are averages of 50 trials, and the whiskers show the 95% confidence intervals.

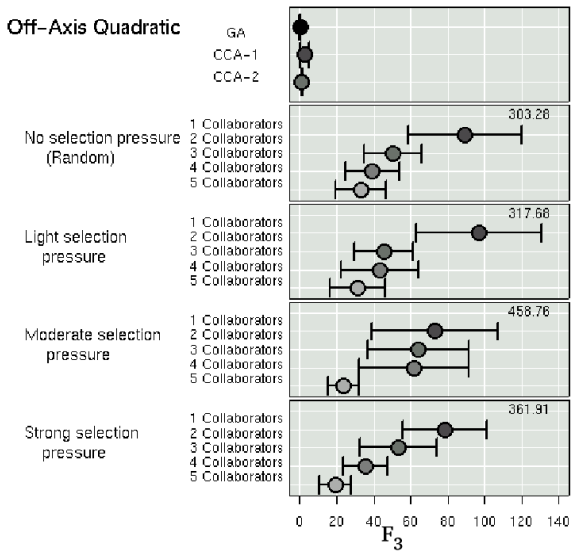

Figure 4: Results for the off-axis quadratic (f_3) minimization experiments. The x-axis represents the final reported result from the EA after 100,000 function evaluations. The points plotted are averages of 50 trials, and the whiskers show the 95% confidence intervals.

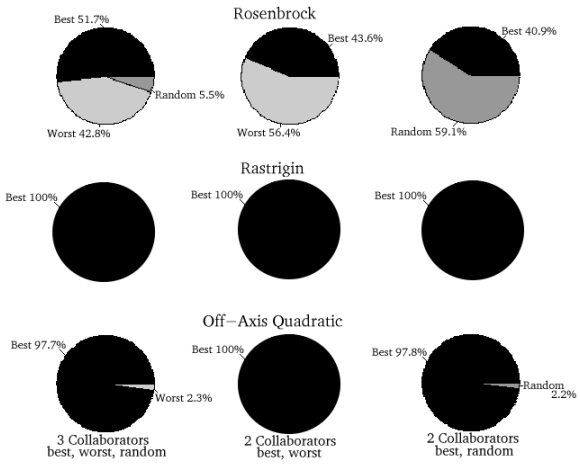

Figure 5: The contributions of collaborators were tracked using three different collaboration mechanisms for each function (best, worst, and random; best and worst; and best and random). The number of times each collaborator is responsible for yielding the better fitness score is illustrated as ratios in the above chart. Averages across 50 trials were used.

tion returns CCA performance to at least an insignificant difference from that of the GA.

While our research does not address the issue of how large the collaboration pool size should be for a given problem, it does suggest that a relatively conservative adjustment from one to two collaborators will frequently yield substantial benefit. Indeed, in all three functions this change is statistically significant for the data shown in our figures.

Of course even such a "conservative" adjustment from one to two collaborators is disappointing news in terms of computational complexity. Whether some sampling technique can be used to reduce the combinatorial problem remains to be seen.

6 Further Analysis

In order to look more closely at what kind of use the CCA is making of its collaborators, we decided to setup some multi-collaborator runs and track the frequency with which the CCA makes use of any particular collaborator for fitness assessment. We did this by setting up three groups for each function, one requiring three collaborators, and two requiring just two. In the first group the best, worst, and random collaborators were selected and the function was evaluated with each. The second group selects the best and worst collaborator and the third selects best and random. Since we are using an optimistic strategy for credit assignment, we kept track of the number of times each collaborator produces the best resulting fitness score. Figure 5 on page 6 shows these ratios for these groups for all three functions.

The Rastrigin function always uses the best collaborator. Given the premise that using the best individual for collaboration gives us something like a simple line-search, it is not surprising that the best collaborator is always the one used by the algorithm during evolution. Even the off-axis quadratic makes predominant use of the best individual, although it seems evident that its own alignment properties create a need for small use of other collaborators.

What is more interesting is that not only does the Rosenbrock make use of all three collaborators, but it doesn't seem to matter whether we use a random collaborator or the worst individual as a collaborator. Curiously, use of the worst individual seems to overshadow use of the random individual in the three collaborator case. We believe this is an artifact of the properties of this particular landscape, though clearly more investigation is needed to fully explain this.

7 Conclusions and Future Work

There are several clear lessons to take home from this research. First and foremost it is evident that using an optimistic approach is generally the best mechanism for collaboration credit assignment. This may not always be true for every type of problem, but it seems that it is a very safe first guess for static objective functions.

The next question a practitioner should ask is how much non-linear interaction there is likely to be among the subcomponents with respect to fitness. If this can be reasonably assessed, it is the key to making the next decisions about collaboration. Clearly if the problem is a simple problem that is linearly separable, a greedy approach to collaborator selection is warranted. Additionally, it may well be that the number of collaborators may be limited (perhaps even to just 1).

For more complicated problems with large degrees of variable interactivity, the selection pressure of collaboration is far less important than the number of collaborators. Moreover, if computationally feasible, increasing the number of collaborators seems to benefit CCA performance in general.

However, it is also clear that there is no magic bullet. The simple off-axis quadratic still perplexes the non-greedy CCA. Combining random (or worst, or arbitrary) collaborators with best collaborators against these problems resulted in no significant degradation of solution quality over the greedy CCA-1. So combining these methods (as in CCA-2 by Potter and De Jong (1994)) may be a good first stab at solving a problem when the degree of variable interactivity is unknown.

The fact that in the non-linear case collaborator selection pressure seems to be unimportant may be a clue for some resolution to the multi–collaborator combinatorial problem. It suggests that *how* you sample the collaboration space is not very important. This encourages the possibility that some simple sampling methods can give us some relief to this problem. As state earlier, more work here is needed.

Although we feel this is a good first step toward understanding how collaboration works in the CCA, much work remains. Ideally it would be nice to have some theory that allows us to find the minimal number of collaborators necessary to solve a given problem, for instance.

Even without such a theory though many questions are raised by this research which deserve attention. First of all, if the CCA makes even use of different collaborators in non-linear problems like Rosenbrock as it seems to, what kind of run-time behavior does this usage have? Is there some sort of periodicity, as one collaborator selection method dominates the other for some time, then trends

reverse? Or is there some punctuation of equilibrium that occurs to cause the method to re-balance? Or perhaps the usage is random and unpredictable. Exploring this question is one of our foremost questions.

Additionally, while increasing from two to three collaborators in the Rosenbrock problem is clearly superior, CCA runs do not seem to make even distributed use of these collaborators. Our observations show that two selection methods can overshadow the third, even though the addition of a third method improves performance. We would like to answer this question in the future, as well.

Finally, one of our most pressing questions is that of how to efficiently deal with the dimensionality problems caused by having multiple collaborators and multiple subpopulations. We are interested in exploring whether or not various sampling methods can be used to address these problems.

References

P. Angeline and J. Pollack. Competitive environments evolve better solutions for complex tasks. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 264–270, San Mateo, CA, 1993. Morgan Kaufmann.

R. Axelrod. Evolution of strategies in the iterated prisoner's dilemma. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*. Morgan Kaufman, 1989.

R. Eriksson and B. Olsson. Cooperative coevolution in inventory control optimisation. In G. Smith, N. Steele, and R. Albrecht, editors, *Proceedings of the Third International Conference on Artificial Neural Networks and Genetic Algorithms*, University of East Anglia, Norwich, UK, 1997. Springer.

S. Ficici and J. Pollack. A game-theoretic approach to the simple coevolutionary algorithm. In *Proceedings from the Sixth Parallel Problem Solving from Nature*, pages 467–476. Springer-Verlag, 2000.

D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Artificial Life II, SFI Studies in the Sciences of Complexity*, 10:313–324, 1991.

P. Husbands and F. Mill. Simulated coevolution as the mechanism for emergent planning and scheduling. In R. Belew and L. Booker, editors, *Proceedings of the Fourch International Conference on Genetic Algorithms*, pages 264–270. Morgan Kaufmann, 1991.

H. Juillé and J. Pollak. Co-evolving interwined spirals. In L. Fogel, P. Angeline, and T. Bäck, editors, *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 461–468. MIT Press, 1996.

H. Mayer. Symbiotic coevolution of artificial neural networks and training data sets. In *Proceedings from the Fifth Parallel Problem Solving from Nature*, pages 511–520. Springer-Verlag, 1999.

J. Paredis. Steps towards co-evolutionary classification networks. In R. A. Brooks and P. Maes, editors, *Artificial Life IV, Proceedings of the fourth International Workshop on the Synthesis and Simulation of Living Systems.*, pages 359–365. MIT Press, 1994.

J. Paredis. Coevolutionary computation. *Artificial Life Journal*, 2(3), 1996.

J. Pollack and A. Blair. Coevolution in the successful learning of backgammon strategy. *Machine Learning*, 32(3): 225–240, 1998.

M. Potter. *The Design and Analysis of a Computational Model of Cooperative CoEvolution*. PhD thesis, George Mason University, Fairfax, Virginia, 1997.

M. Potter and K. De Jong. A cooperative coevolutionary approach to function optimization. In *Proceedings from the Third Parallel Problem Solving from Nature*, pages 249–257. Springer-Verlag, 1994.

M. Potter and K. De Jong. The coevolution of antibodies for concept learning. In *Proceedings from the Fifth Parallel Problem Solving from Nature*, pages 530–539. Springer-Verlag, 1999.

M. Potter and K. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

C. Rosin. *Coevolutionary Search Among Adversaries*. PhD thesis, University of California, San Diego, 1997.

C. Rosin and R. Belew. Methods for comptetitive coevolution: Finding opponents worth beating. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference*, pages 373–380. Morgan Kaufmann, 1995.

C. Rosin and R. Belew. New methods for competetive coevolution. *Evolutionary Computation*, 5(1):1–29, 1996.

R. Salomon. Performance degradation of genetic algorithms under coordinate rotation. In L. Fogel, P. Angeline, and T. Bäck, editors, *Proceedings of the Fifth Annual Conference on Evolutionary Programming V*, pages 153–161. MIT Press, 1996.

D. Schlierkamp-Voosen and H. Mühlenbein. Strategy adaptation by competing subpopulations. In *Proceedings from the Third Parallel Problem Solving from Nature*, pages 199–108. Springer-Verlag, 1994.

M. Vose. *The Simple Genetic Algorithm*. MIT Press, 1999.