Finding Building Blocks through Eigenstructure Adaptation

Danica Wyatt[1] and Hod Lipson[2]

[1] Physics Department
[2] Mechanical & Aerospace Engineering, Computing & Information Science
Cornell University, Ithaca, NY 14853, USA

Abstract. A fundamental aspect of many evolutionary approaches to synthesis of complex systems is the need to compose atomic elements into useful higher-level building blocks. However, the ability of genetic algorithms to promote useful building blocks is based critically on genetic linkage - the assumption that functionally related alleles are also arranged compactly on the genome. In many practical problems, linkage is not known *a priori* or may change dynamically. Here we propose that a problem's Hessian matrix reveals this linkage, and that an eigenstructure analysis of the Hessian provides a transformation of the problem to a space where first-order genetic linkage is optimal. Genetic algorithms that dynamically transforms the problem space can operate much more efficiently. We demonstrate the proposed approach on a real-valued adaptation of Kaufmann's NK landscapes and discuss methods for extending it to higher-order linkage.

1 Introduction

A fundamental aspect of many evolutionary algorithms is the need to compose atomic elements into higher-level building blocks. This compositional process should continue recursively to generate increasingly complex modules from lower level components, until the desired solution is attained. The importance of discovery of partial building blocks was initially stressed by Holland in "The building block hypothesis" [5] that described how Genetic Algorithms (GAs) work. GAs promote useful building blocks represented as schemata and compose them through the process of crossover. As the Schema Theorem shows, however, the ability of GAs to promote useful building blocks is based critically on genetic linkage - the assumption that functionally related alleles are also arranged compactly on the genome. If genetic linkage is poor (i.e., there is little correlation between functional dependency and genetic proximity) then the crossover operator is more likely to break useful building blocks than it is likely to compose them.

The effect of poor genetic linkage can be dramatic. Before proceeding to describe previous work and our proposed solution, we demonstrate the grave effect of poor linkage in Fig. 1. The graph shows the best fitness of a GA running on a hard test problem. The test problem consists of 16 real-valued dimensions,

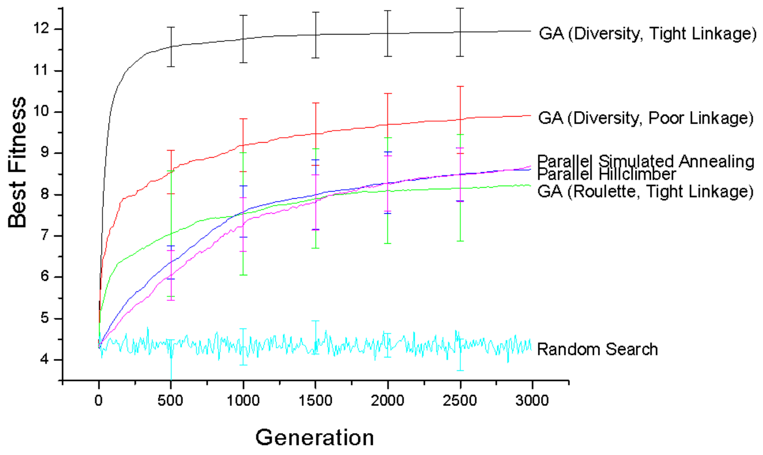

Fig. 1. Performance of a GA on a 16 dimensional problem with poor and tight linkage, with and without diversity maintenance. A parallel Simulated Annealing optimizer, a parallel hillclimber, and a random search are provided for comparison. All methods perform equal number of evaluations per generation. Error bars of one standard deviation are based on 10 runs on the same problem

each of which is deceptive (gradients lead in the wrong direction) and contains significant coupling between the variables. The test problem will be described in detail later; for now, it will suffice to notice the difference in performance of the GA between the top curve, where the genome is ordered so that coupled variables are close to each other (tight linkage) versus the lower curve, where the genome is shuffled so that coupled variables are far from each other (poor linkage.) In both of these cases, diversity maintenance techniques were also used. Without diversity, the crossover operator has little or no effect, and the GA's performance is inferior even to standard optimizers such as a well-tuned parallel simulated-annealing optimizer and a parallel random mutation hillclimber (a basic gradient optimizer.) A random search process is shown for reference. All methods perform an equal number of samplings per generation.

2 Prior Work

Since genetic linkage is not necessarily known in advance and may change dynamically as solution progresses or as the problem varies, new evolutionary algorithms have been proposed that either change linkage dynamically, or that do not rely on genetic linkage at all (at an additional computational cost.) Originally, Holland [5] proposed an inversion operator that would reorder alleles on the genome, with the expectation that genomes with better orderings (tighter genetic linkage) would gradually take over the population. However, this operator turned out to be too slow in practice. A variety of algorithms were proposed that do not assume linkage and build up the genome from scratch starting with the founding building blocks. Goldberg *et al* [4], and later Watson and Pollack [10], developed GAs that co-evolve partial solutions in which linkage can be adapted through new kinds of combination and split operators. Kargupta introduced new operators which search explicitly for relationships between genes [12]. Genetic programming [7] combines building blocks in tree hierarchies, allowing arbitrary branches of solutions to be swapped thereby permitting promotion of useful subcomponents without relying on linkage. More recently, Harik and Goldberg [11] proposed a linkage learning genetic algorithm (LLGA) that intersperses alleles in the partial solutions with variable-sized gaps (introns) allowing the GA more control over linkage tightening and exchange of complete building blocks. A more recent version of this algorithm uses evolved start expressions to assist in the process of nucleation of tightly linked groups [1]. All of these methods increase computational cost through the exploration of many compositional permutations.

3 Identifying Linkage through the Hessian

Here we propose an alternative way of identifying building blocks through dynamic eigenstructure analysis of the problem landscape's Hessian matrix. The Hessian matrix H is defined by measuring the cross-correlation effect of the variation of each of the n variables x_i on the variation effect of each other locus of the genome on the fitness function $F(X)$. Essentially, the Hessian matrix determines the first-order functional dependencies between gene locations. The Hessian is defined as

$$H_{ij} = \frac{\partial^2 F(X)}{\partial x_i \partial x_j} \tag{1}$$

By definition, it is a symmetric matrix. For example, if

$$X = (x_1, x_2, x_3, x_4) \tag{2}$$

and the fitness function to be optimized is

$$F(X) = \sin(2x_1\, x_3) + \sin(3x_2\, x_4) \tag{3}$$

then computing [1] H and evaluating at $X = 0$ would yield

$$H = \begin{bmatrix} 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \\ 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix} \tag{4}$$

[1] Note that the Hessian can easily be computed numerically with the fitness given as a black box.

Highly coupled variable pairs are represented by large magnitude elements in H. Large off-diagonal elements imply coupling between variables that are not adjacent on the genome; to improve linkage, variables can be rearranged so that coupled pairs are proximate on the genome, bringing their corresponding Hessian coefficient closer to the diagonal. Rearranging the order of parameters on the genome is equivalent to swapping rows and columns of H, effectively transforming it by a permutation transformation. To bring the elements of H in the example above closer to the diagonal (effectively tightening the genetic linkage) we can use the permutation matrix T, where

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

to yield a new Hessian H'

$$H' = T^T H T = \begin{bmatrix} 0 & 2 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 3 & 0 \end{bmatrix} \tag{6}$$

and the improved genome ordering X'

$$X' = XT = (x_3, x_1, x_2, x_4) \tag{7}$$

H' above is as close as we can bring the non-zero elements to the diagonal by reordering, and the resulting genome ordering of Eq. (7) has indeed improved its linkage compared to the original ordering of Eq. (2). But is there a way to bring them even closer to the diagonal?

3.1 Generalized Genome Reordering

The optimal linkage ordering is not necessarily sequential. The permutation matrix simply reorders elements on the genome. However, there might not be a perfect sequential ordering if variables are coupled in any way but a linear serial chain. For example, if three variables are coupled equally, there is no way to arrange them in linear progression so that all are equally close to each other (the first will be less proximate to the last than it is to the middle variable.) Similarly, in the previous example of Eq. (3), x_1 is coupled to x_3, but not to x_2 so the genome ordering provided in Eq. (7) is still not optimally tight, and that is why H' is not exactly diagonal.

However, since the permutation matrix is nothing but an orthogonal linear transformation, we can think of any reordering process as merely a linear transformation. In this form, by allowing the transformation to have non-integer elements, we can find even more compact orderings. The optimal transformation is essentially the one that will bring all elements of the Hessian matrix *exactly* to the diagonal. We thus seek the optimal transformation T_0 to a diagonal matrix λ:

$$T_0^T H T_0 = \lambda \tag{8}$$

Solving for T_0 yields the Hessian's eigenvectors. Because the Hessian matrix is always symmetric, the eigenvectors have no imaginary component. In our example,

$$T_0 = \frac{\sqrt{2}}{2}\begin{bmatrix} 0 & -1 & -1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & -1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

and

$$H_0 = T^T H T = \begin{bmatrix} -3 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} \tag{10}$$

and the optimal genome ordering X_0 is given by

$$X_0 = XT_0 = \frac{\sqrt{2}}{2}(< x_2, -x_4 >, < x_3, -x_1 >, < -x_1, -x_3 >, < x_2, -x_4 >) \tag{11}$$

To understand why the linkage-optimal vector X_0 above has even less coupling than the compactly ordered vector $X' = (x_3, x_1, x_2, x_4)$ of Eq. (7), consider the following. A small positive mutation δ applied to x_1, a gene of X', will result in either an increase or decrease of the fitness function. Whether it will be an increase or a decrease depends on the sign of another gene, x_3. Thus there is still coupling between alleles on the genome. On the other hand, a small positive mutation in δ applied to (multiplied by) $< x_1, x_3 >$ will always result in an increase in fitness, regardless of the states of other genes. Therefore, $< x_1, x_3 >$ is more suited to be a gene than any single variable. Similarly, a small positive mutation δ applied to the complementary vector $< x_1, -x_3 >$, the second gene of X_0, will do nothing to the fitness function, independent of the value of the other variables. These two genes thus span the search space much more effectively because they are uncoupled.

3.2 What Can We Learn from the Eigenvalues?

The eigenvalues λ hold the scaling factors of the new space, by which any variation operators can be calibrated; in our example, these are 2 and 3. Dividing the mutation operators by these factors would allow all blocks to be explored at the same resolution. Similarly, subspaces with large eigenvalues are more dominant and should be explored first if gradual complexification is employed [15].

Degenerate eigenvalues(two or more equal eigenvalues) indicate a subspace of the landscape space, spanned by the corresponding eigenvectors, where variables are uniformly coupled (or decoupled.) Negative eigenvalues may indicate collapsed subspaces.

3.3 Using Transformations in a GA

Based on the analysis above, we conclude that

- The genome reordering is equivalent to a linear transformation of the fitness landscape, therefore,
- There exist non-discrete genome orderings, and
- The ordering that yields optimal genetic linkage at a point in the landscape is given by the eigenvectors of the Hessian at that point.
- Eigenvalues of the Hessian reveal properties of the search space.

Once a linkage-optimal ordering transformation has been determined by computing the eigenvectors of the Hessian of $F(X)$, a GA can proceed regularly by evolving individuals h_i but evaluating $F(T_0 h_i)$ instead of directly $F(h_i)$ Many varieties of GAs, such as those incorporating partial solutions, learning, and sophisticated composition operators can easily be modified to use this formulation.

4 Test Function

We tested the eigenvector reordering process on evolution of a solution to a multidimensional function of real variables, $Z(X)$, that composes n base functions $\Psi(x)$:

$$Z(X) = \frac{1}{2} \sum_{i=1}^{n} \Psi(b_i) \tag{12}$$

where

$$\Psi(u) = \frac{\cos(u) + 1}{|u| + 1} \tag{13}$$

and

$$b_i = \sum_{j=1}^{k} a_{ij}\, x_{(i+j) mod\, n} - c_i \tag{14}$$

This function is a real-valued adaptation of Kauffman's NK landscapes [6]. Kauffman defined a function with n bits, in which each bit's fitness contribution depends arbitrarily on its k neighbors. NK landscapes thus have "tunable

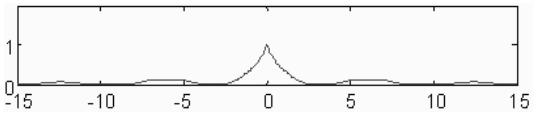

Fig. 2. The base function $\Psi(u)$ used to construct the test function. The full test function is a shuffled kth-order mixing of n base functions

ruggedness" and are often used to test GAs. Here we defined a similar problem that uses real values instead of bits. First, we define a multimodal base function $\Psi(u)$ with one global maximum at $u = 0$ and several smaller maxima, shown in Fig. 2. This base function is deceptive if the search space is larger than $\pm 2\pi$ around $u = 0$ because then in most of the space the gradients lead to local optima. The function Z is a sum of the n single dimensional base functions; however the blocks are not separable: Each base function is evaluated at position b_i, which is a mixing function of k elements of the argument vector X. The mixing is obtained through an array of coefficients a_{ij} and an offset c_i, each set arbitrarily. Because of the mixing of order k, optimizing one dimension may (and usually does) lead to adverse effect in the other $k-1$ dimensions. Finally, the genome elements $x_{i..n}$ are shuffled so that variables that contribute to the same base function are maximally apart on the genome, so as to create the worst linkage. The function is defined so that it always has a single global optimum with the value n, and it occurs at $X = A^{-1}c$ (where A is a k-diagonal matrix with the elements of a on its diagonals, and X is the unshuffled vector.) In our study, we generate the coefficients a_{ij} randomly with a uniform distribution in the range of ± 1, and the coefficients c_i with a uniform distribution in the range of ± 8. These values were selected arbitrarily, and were not tuned.

5 Experimental Results

We carried out a series of tests, with different setting of n and k. All problems have bad shuffling (poor linkage.) In each experiment, the problem coefficients were generated randomly and were not tuned. We then carried out several runs to collect average performance and standard deviation. We also ran a parallel hillclimber as a baseline control. Individuals were initialized randomly in $X = \pm 10$ with uniform distribution. Single-point crossover and mutations of maximum size ± 0.1 were used for all offspring. In all experiments we used a GA with a diversity maintenance technique known as "Deterministic Crowding" [9] in which a more successful offspring replaces the parent it most closely resembles after pairwise matching. Diversity maintenance is important to avoid premature convergence, in which case crossover operators would do little and the effect we wish to study would vanish.

First, let us look closely at the effect of using a Hessian Eigenvector transformation. Figure 3 shows average performance of an Eigenvector GA on a problem with $n = 16$ and $k = 4$. Population size is 100, and average and standard devia-

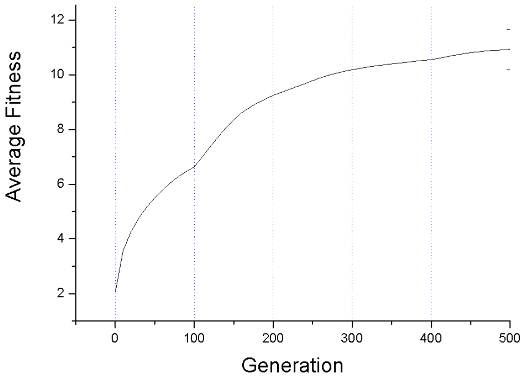

Fig. 3. Performance of a GA on a shuffled real-valued NK landscape with $n = 16$ and $k = 4$. Reordering transformation is recomputed every 100 generations (at vertical lines,) and performance boosts are visible at those instances. Statistics are of 20 runs. Eigenvector GA uses 5% more evaluations than regular GA

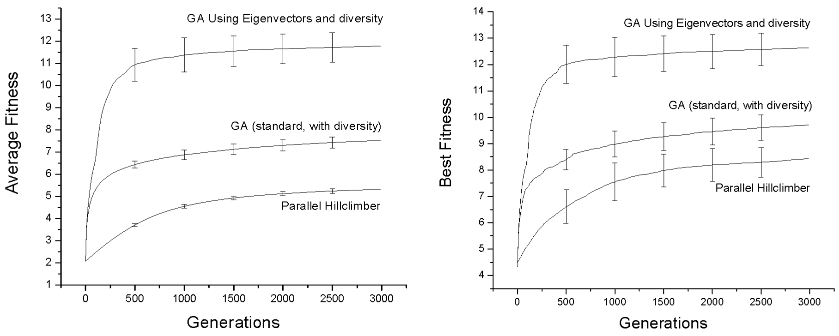

Fig. 4. Performance comparison on a shuffled real-valued NK landscape with n=16 and k=4. Reordering transformation is recomputed every 100 generations. Statistics are of 20 runs

tion is based on 20 runs. The transformation is re-computed numerically every 100 generations, around the currently best solution, adding 5% evaluations. We will discuss the cost of this computation in the next section. The points of re-computation are marked with a vertical solid line; note how the curve "boosts" its optimization rate at those intervals. Figure 4 shows the long-term behavior of the solver over 3000 generations using the same parameter settings. Both average fitness of the population and best fitness in the population are shown. We see that the eigenvector GA has significantly outperformed the regular GA. The contribution of the eigentransformation has been exhausted after 500 generations or so. After that period, both algorithms progress at roughly the same rate. We hypothesize this point is where contribution of first-order (linear) linkage has been exhausted.

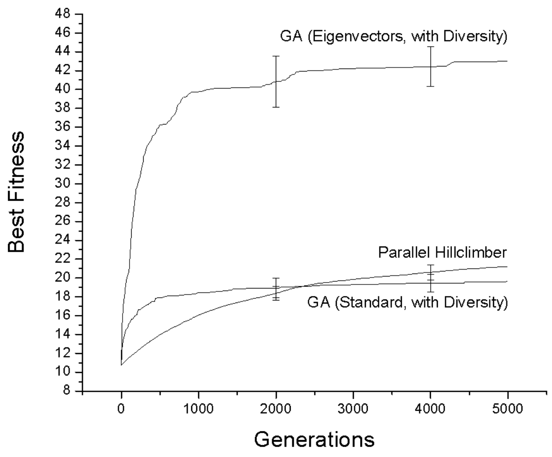

Fig. 5. Performance comparison on a shuffled real-valued NK landscape with $n = 64$ and $k = 8$. Reordering transformation is recomputed every 100 generations. Statistics are of 10 runs

The performance boost provided by the eigentransformation becomes more significant as the problem becomes harder both in the number of parameters (n) and in the amount of coupling between the parameters (k). Figure 5 shows average performance of an Eigenvector GA on a problem with n=64 and k=8. Population size is 100, and average and standard deviation is based on 10 runs. At n=256 and k=16, we observed similar performance boosts.

6 Computational Cost

Additional computational cost is incurred by the linear transformation and eigenstructure calculation, as well as by additional evaluations. The transformation cost adds $\mathcal{O}(n^2)$ arithmetic operations per evaluation and $\mathcal{O}(n^2)$ arithmetic operations per Hessian calculation for computing derivatives from the samples, and computing the eigenvectors. Both of these are typically negligible compared to the cost of a single evaluation of a hard practical problem.

6.1 Evaluation Cost

The evaluation cost is especially critical because direct numerical calculation of the Hessian matrix involves approximately $2n^2$ samplings of the search space. If each individual is at a different location of the landscape, this may amount to $2n^2p$ extra evaluations per generation, where p is the population size. This cost is prohibitive, and so more efficient schemes must be found. If linkage properties of the landscape are consistent over time, significantly fewer than n^2p samplings are necessary. For example, the results shown in Fig. 4 required only one Hessian calculation per 100 generations. Since for that problem n=16, this amounts to only 5% additional evaluations, and the advantage gained is well worth it. The number of evaluations grows as $\mathcal{O}(n^2)$ but the interval at which Hessian-recalculation is required increases too since propagation of building blocks is slower. If the linkage properties of the landscape are consistent over space, the Hessian does not need to be tracked for every individual. For the results shown in Fig. 4 we recalculated the Hessian only around the currently best individual, but use it for all individuals in the diverse population. A harder class of problems exists where linkage properties of the landscape change over time and over space (nonlinear linkage.) Prior work dealing with various forms of linkage learning and composition typically assumed that the linkage properties are fixed for the given problem. For such linkage-variable problems we might either spend more evaluations to track separate Hessian matrices for clusters of individuals, and also update them more frequently. Alternatively, we could extract Hessian information indirectly from the samplings already being done by the GA anyway.

6.2 Indirect Computation of the Hessian

The Hessian can be computed indirectly from the GA samples using cross-correlation statistics. As described by the "Two-armed Bandit" problem [3], there is a trade off between exploration and exploitation. However, with a linkage-learning algorithm as proposed here, it is possible to use all samples to probe linkage properties and thus enhance exploration without incurring additional function evaluations. The key to indirect extraction of linkage information is the understanding that linkage can be learned just as efficiently even from individuals with low fitness. Hessian approximation techniques used in quasi-Newton methods [13][14] could be adapted to make use of this information. Another way to gather linkage properties from arbitrary, unstructured samplings of the search space is through least-squares fitting to a linkage modeling function. For example, assume a two dimensional landscape $F(x_1, x_2)$ can be described very roughly by the conic section equation

$$F(x_1, x_2) = ax_1^2 + 2bx_1x_2 + cx_2^2 + dx_1 + ex_2 + f \tag{15}$$

The linkage coefficient we are interested in is the magnitude of parameter b with respect to parameters a and c. These parameters reveal how the landscape is influenced by the combination (x_1x_2). Note that the function is not used to model the landscape for direct optimization; that is, we do not proceed to compute the optimum of F, because there is no guarantee whatsoever that the landscape is of degree 2 (if it was then much more efficient optimization techniques could be used.) Instead, we only use it to probe how variables are dependent on each other, by fitting a conic surface locally to a small patch of the landscape. A landscape with n parameters will have $(n+1)(n+2)/2$ coefficients, and so $\mathcal{O}(n^2)$ samplings will be needed to determine the coefficients through least squares modeling. Direct computation of the Hessian also requires $\mathcal{O}(n^2)$ samplings, and so the new formulation is not a improvement in terms of the number of evaluations. However, the direct computation method required structured samplings on a grid, whereas the new formulation can use an unstructured pattern of samples and can therefore use samples performed anyway by the GA in course of its normal operation, provided they are close enough. The quality of the linkage approximation degrades as the sampling radius δ increases, just like direct numerical computation of the Hessian degrades with $\mathcal{O}(\delta^2)$ according to Taylor expansion around the center point ($\delta \simeq 0.1$ for the test function described in this paper.) It is therefore necessary to use samples that are within a small region with respect to nonlinearities in the function. In a diverse set of samples this can be done by clustering samples into groups, at the cost of cluster management.

7 Higher-Level Compositions

Direct eigenstructure transformation resolves first-order linkage, but nonlinear transformations may resolve even higher-order linkage. We say a linkage is first order when the linkage between two variables does not depend on any other, third variable or on time. If it does, we would have a second-order linkage. Higher-order linkage effects can be identified using Kronecker tensor products of partial derivatives of arbitrary orders. Large eigenvalues of these tensors indicates a strong high-order dependency that can be resolved through a nonlinear transformation. These higher-order transformations are again provided by (nonlinear) eigenstructure analysis of tensors [8]. The approach proposed here is akin to support vector machine (SVM) methods [2] that transform the problem space so that data classes that were originally convoluted are now linearly separable. In Eq. (15) we used second-order polynomial kernels that transform the search space so that first-order linkage is optimal; other kernels, including higher-order polynomials, could be used to probe and compensate for higher-order linkage.

8 Conclusions

Performance of genetic algorithms is critically based on both diversity maintenance and genetic linkage. Here we propose that transformations can effectively be used to reorder the genome, and that the first-order linkage-optimal transformation can be found through Eigenstructure analysis of the landscape's Hessian matrix. In a series of experiments using a highly coupled, nonlinear, deceptive and shuffled function, we show how the presented algorithm produces significantly superior performance, at relatively low additional computational cost. We also propose that when high-order linkage is systematic (unlike random or "needle-in-a-haystack" landscapes) it can be resolved dynamically using high-order statistical probes that use existing evaluations. We further suggest that kernel methods that are traditionally used in machine learning to transform convoluted problems into linearly separable problems can be brought to bear on evolutionary computation to decompose high-order linkage problems into linkage optimal space, as a principled form of adaptation of the genomic representation.

Acknowledgments. This work has been supported in part by the US Department of Energy (DOE), grant DE-FG02-01ER45902, and by the US National Science Foundation Integrative Graduate Education and Research Traineeship (IGERT) program at Cornell University.

References

[1] Chen, Y.-P., Goldberg, D.E. *Introducing start expression genes to the linkage learning genetic algorithm.* Proceedings of the Parallel Problem Solving from Nature Conference (PPSN VII). Berlin, Germany:Springer, 351–360, 2002.

[2] Cristianini N., Shawe-Taylor J. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000.

[3] De Jong, K. *An analysis of the behaviour of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[4] Goldberg, D., Korb, B., and Deb, K. *Messy Genetic Algorithms: Motivation, Analysis, and First Results.* Complex Systems, 4:415–444, 1989.

[5] Holland, J.H. *Adaptation in Natural and Artificial Systems* University of Michigan Press, Ann Arbor, 1975.

[6] Kauffman, S. *The Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press, 1993.

[7] Koza J. *Hierarchical genetic algorithms operating on populations of computer programs.* 11th Int. joint conference on genetic algorithms, 768–774, 1989.

[8] Lipson H., Siegelmann H.T. *High Order Eigentensors as Symbolic Rules in Competitive Learning.* in Hybrid Neural Systems, S. Wermter, R. Sun (Eds.) Springer, LNCS 1778, 286–297, 2002.

[9] Mahfoud S. *Niching Methods for Genetic Algorithms.* IlliGAL Report No. 95001, University of Illinois at Urbana-Champaign, 77–80, 1995.

[10] Watson, R.A., Pollack, J.B. *A Computational Model of Symbiotic Composition in Evolutionary Transitions Biosystems.* to appear in 2003.

[11] Harik, G., and Goldberg, D.E. *Learning linkage.* Foundations of Genetic Algorithms 4:247–262, 1997.

[12] Kargupta, H. *The gene expression messy genetic algorithm.* Proceedings of the 1996 IEEE International Conference on Evolutionary Computation. Nagoya University, Japan, 631–636, 1996.

[13] Bertsekas, D.P. *Non-linear Programming.* Athena Scientific, Belmont, MA, 134–141, 1995.

[14] Shanno, D.F. *Conditioning of quasi-Newton methods for function minimization.* Mathematics of Computation 24:647–656, 1970.

[15] Stanley, K.O., Miikkulainen, R. *Achieving High-Level Functionality Through Complexification.* Proceedings of the AAAI 2003 Spring Symposium on Computational Synthesis. Stanford, CA: AAAI Press, 2003.