A Test Problem Generator for Non-Stationary Environments

Ronald W. Morrison
GRC International, Inc.
1900 Gallows Road
Vienna, VA 22182
rmorrison@grci.com

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

Abstract: There has recently been a growing interest in the study of EA performance in changing environments. Studies of EAs in dynamic environments, however, have used only a few, limited fitness functions. As has been shown in the study of EAs in static environments, comparative studies of the effectiveness of various EAs in dynamic environments will require more rigorous and standardized test functions. These functions should have a simple representation mechanism, so that a wide range of complexity and sophisticated dynamics can be unambiguously described in an elementary manner. This paper proposes an initial test function generator with these characteristics to facilitate further understanding of the effectiveness of different EA implementations in different types of dynamic environments.

1 Non-Stationary Problems

The robust capability of the various types of evolutionary algorithms (EAs) to find solutions to difficult problems has permitted them to become the optimization and search techniques of choice for many applications. But what happens if the information that has been provided to an EA changes? Despite the obviously successful application of evolutionary techniques to complex problems in many different environments, the resultant solutions are often fragile, and prone to failure when subjected to even minor changes in the problem. Since information is accumulated by an EA during successive iterations, there is an implicit assumption of consistency in the evaluation function. Unfortunately, the consistency assumption is not the case in many real problems. A variety of engineering, economic, and information technology problems require systems that adapt to changes over time. Examples of problems with non-stationary environments include: target recognition, where the sensor performance may vary based on environmental conditions; scheduling problems, where scheduling demands and available resources may vary over time; investment portfolio evaluation, where the assessment of investment risk varies over time; and data mining, where the contents of the database being mined are continuously updated

There is a recent growing interest in the performance of EAs in changing environments (Mori, Imanishi, Kita, and Nishikawa 1997), (Lewis, Hart, and Ritchie 1997), (Bäck 1998). This is not surprising in view of the fact that many of the problems that EAs are being used to solve are known to vary over time. The "successes and failures" of EAs in dynamic fitness landscapes that have been reported to date, however, have mostly just measured the speed of the adaptation of some specific EA implementation. This measurement is usually made using a simple example problem and often reported without analysis of the dynamics of the sample problem selected or the generality of any adaptation speed results. As has been shown in the study of EAs in static environments, comparative studies of the effectiveness of various EAs in dynamic environments will require rigorous and standardized test functions. Furthermore, since it is likely that techniques that are successful in improving the performance of EAs in adapting to some types of fitness landscape change are likely to be less effective in other types of landscape changes, standardized test functions will be required to cover a variety of different types of dynamic behavior.

As a first step in improving this situation, we have developed a test problem generator capable of providing instances of a wide variety of dynamic landscapes. In this paper we present the basic architecture of the generator, elaborate on its capabilities, and illustrate its use. We begin by examining some of the types of non-stationary landscapes.

2 Types of Dynamic Landscapes

In the building of a test problem generator for dynamic landscapes, an important consideration is the range of dynamic behavior that the generator should be able to produce. In the simplest non-stationary problems, the overall shape (morphology) of the fitness landscape is constant, but it drifts along one or more axes over time. It is relatively easy, for example, to take any two-dimensional static landscape $f(x, y)$ and make it vary over time in this simple sense by redefining it as $g(x, y, t) = f(x_{t-1} + \Delta x_t, y_{t-1} + \Delta y_t)$, and by providing the "motion algorithm" for computing Δx_t and Δy_t. The motion itself can have several properties including its speed (rapid/slow relative to EA time), periodic/aperiodic, etc. The focus here is typically on how well an EA can follow (track) a moving landscape.

More difficult and more interesting are landscapes whose morphology changes over time (see, for example, (Goldberg and Smith 1968) or (Grefenstette 1992)). Providing a simple way to contruct interesting test problems of this type is the goal of the work presented here. We do so by focusing on the "peaks" in a landscape and consider how peaks might independently change over time.

2.1 Changing Fitness Peak Heights

There are many real-world problems that can be modeled by simply allowing the heights of fitness peaks to change over time reflecting, for example changes in the cost of raw materials, changes in comsumer preferences, etc. resulting in global optima becoming local optima and vice-versa. Traditional EAs have no difficulty converging on initially high fitness peaks, but the loss of diversity in the population makes it difficult for them to adapt to change (Grefenstette 1992).

2.2 Changing Fitness Peak Shapes

Closely related to changes in peak heights are changes in their shape (e.g, tall and narrow, short and wide, etc.). Alone, shape changes don't affect the performance of traditional EAs much. However, in conjunction with other simultaneous changes, they can pose considerable difficulty.

2.3 Changing Fitness Peak Locations

Another source of difficulty for current EAs are fitness peaks that change location (Goldberg and Smith 1968). EAs with populations lacking in diversity have difficulty tracking such changes. Tracking becomes even more difficult if the peaks can move independently of one another.

2.4 The Dynamics of Change

So far we have focused on *what* has been changing over time. Equally important is *how* things are changing, i.e., the dynamics of change. The two most common dynamics studied are: a slowly drifting motion (e.g., (Angeline 1997) or (Bäck 1998)) and an oscillatory motion (e.g., (Dasgupta and McGregor 1993) or (Mori, Imanishi, Kita, and Nishikawa 1997)).

Clearly, the form of the dynamics will play an important role in the kind of modifications one might make to existing EAs. To handle slowly drifting changes it may be sufficient to incrementally increase population diversity through adaptive local search operators (Varak, Jukes, and Fogarty 1997). However, oscillating changes are likely to require more fundamental changes such as diploid representations (e.g., (Goldberg and Smith 1968), (Dasgupta and McGregor 1993) or Ryan (Ryan 1997)), or additional "memory" functions (e.g., (Neubauer 1997) or (Mori, Imanishi, Kita, and Nishikawa 1997)).

One additional dynamic of interest is that of an abrupt, unexpected, catastrophic change to the fitness landscape, capturing the kinds of effects that power failures have on complex networks, that 10-car accidents have on traffic flow, etc. Triggered mutation operators are one example of the changes made to EAs to handle such dynamics (Cobb and Grefenstette 1993).

3 Problem Generator Characteristics

Our stated goal is to develop a test problem generator for dynamic fitness landscapes that provides for easy and systematic testing and evaluation of EAs over a wide range of dynamics as discussed in the previous section. To achieve this a dynamic problem generator should have:

(1) Easily modifiable landscape complexity that is scalable to complexity levels representative of problems found in nature.

(2) Simple parametric methods for specifying the morphological characteristics and changes, including peak re-ordering, peak relocation, peak re-shaping.

(3) Simple methods to specify the type of dynamics to apply including: small steps, large steps, multiple step sizes, recurrent motion, and chaotic motion.

(4) Simple representation mechanisms, so that a complex environment with sophisticated dynamics can be unambiguously defined in an elementary manner.

(5) Reasonable computational efficiency.

4 Problem Generator DF1

In this section we describe in detail the features of DF1, our current problem generator. In designing DF1 we have adopted the view that the process of specifying a test problem as a two step process. The first step involves specifying a baseline static landscape of the desired morphological complexity. Having done this, the second step is to add the desired dynamics. We discuss each of these steps in more detail in the following sections.

4.1 Specifying the morphology

To provide a parameterized way of specifying the basic morphology of the landscape, in DF1 we use a "field of cones" of different heights and different slopes randomly scattered across the landscape. This static function is similar to the plane of Gaussian distributions used in (Grefenstette 1999). The static function used in DF1 can be specified for any number of dimensions. For simplicity and for visualization purposes we will use 2-dimensional examples throughout the remainder of the paper. In this case we have:

$$f(X,Y) = max_{i=1,N}\left[H_i - R_i * \sqrt{(X - X_i)^2 + (Y - Y_i)^2}\right]$$

where N specifies the number of cones in the environment, and each cone is independently specified by its location (X_i, Y_i), its height H_i, and its slope R_i. Each of these independently specified cones are "blended" together using the *max* function.

Each time the generator is called it produces a randomly generated morphology of this type in which random values for each cone are assigned based on user-specified ranges:

$$H_i \in [Hbase, Hbase + Hrange]$$

$$R_i \in [Rbase, Rbase + Rrange]$$

and

$$X_i \in [-1, 1]$$

$$Y_i \in [-1, 1]$$

There are several advantages to the use of this function as the static basis for the dynamic environment. They include:

(1) the ability to represent a wide range of complex landscapes.

(2) the surface contains non-differentiable regions.

(3) landscape characteristics are parametrically identified.

(4) fitness values can be easily restricted to be positive (or any other minimum value), if desired.

(5) the function is easily extendable to higher dimensional spaces.

(6) the function offers three different features that can be made dynamic (height, location, and slope).

To generate a wide range of static problems of varying complexity, one need only specify the parameters: N (the number of peaks), Rbase (the minimum value of the slope control variable), Rrange (the allowed range of for the slope control), Hbase (the minimum cone height), and Hrange (the range of allowed cone heights). Two examples of the variety of fitness landscapes generated by varying these five parameters are illustrated in Figures 1 and 2.

In Figure 1 we see one of the randomly generated landscapes specified to have 5 peaks of heights ranging

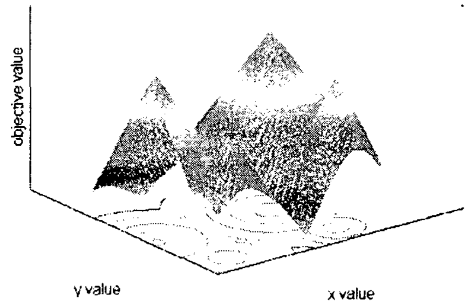

Figure 1: DF1 with N=5, Hbase=3, Hrange=3, Rbase=2, Rrange=5.

from 3 to 6, with slopes ranging from 2 to 7. In Figure 2 we see a much more complex randomly generated landscape specified to have 500 peaks of heights ranging from 50 to 75, and slopes ranging from 100 to 120.

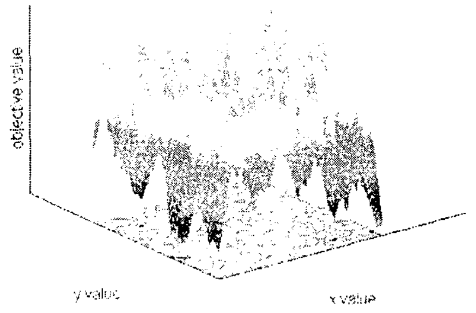

Figure 2: DF1 with N=500, Hbase=50, Hrange=25, Rbase=100, Rrange=20.

4.2 Specifying the dynamics

As discussed earlier, it is desirable to provide a simple mechanism for describing a wide range of dynamic performance. Since we change the dynamic features of the environment by discrete step sizes, we would like a simple method for controlling the generation of a variety of different step sizes. One method for generating a variety of dynamics with the change of a single parameter is to use a one-dimensional, non-linear function that has simple bifurcation transitions to progressively more complex behavior. One of the most common and well-studied functions with this behavior is the logistics function given by:

$$Y_i = A * Y_{(i-1)} * (1 - Y_{(i-1)})$$

where A is a constant, and Y_i is the value at iteration i.

A bifurcation map of the logistics function is provided in Figure 3. This figure shows the values of Y that can be generated on each iteration of the logistics function for values of A between 1 and 4. For example, if a value of 2.2 is chosen, the logistics function will generate a constant Y value of .5455 on each iteration. For a larger A of 2.9, a larger constant Y value of .6552 is generated.

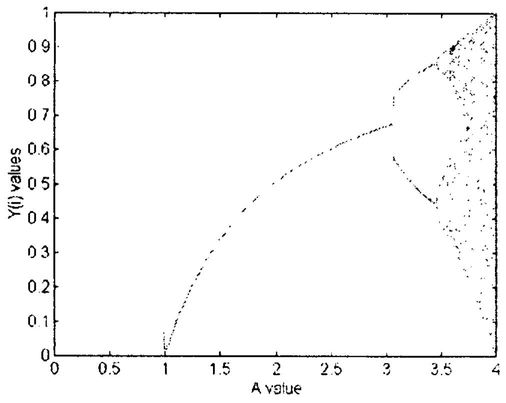

Figure 3: The Logistics Function.

As A is increased, more complicated dynamic behavior emerges. Values slightly larger than the first bifurcation point generate two different values of Y for alternate iterations. For example, values of 3.3 and 3.4 generate pairs of Y values {.8236, .4794} and {.8420, .4520} respectively. As larger A values are chosen the function bifurcates again and provides more complex behavior. At $A = 3.5$, the Y values generated are {.3828, .5009, .8269, .8750}. Still larger values of A generate chaotic sequences of Y values within the range shown in Figure 3.

In DF1 we use the Y values produced on each iteration to select our step sizes for the dynamic portions of our environment. More specifically, we attach a logistics function with a specified A value to each dynamic entity. The result is a simple procedure for users to obtain complex dynamics by specifying:

(1) The number of peaks in the landscape that will move.

(2) Whether the motion will be applied to the height of the peaks, the slope of the peaks, the x-location of the peaks, the y-location of the peaks, or any combination thereof.

The particular value of A chosen for each of the moving features specifies whether the movement will be small same-sized steps, large same-sized steps, steps of few different sizes, or chaotically changing step sizes. As suggested by Figure 3, we require that the values chosen for A must be greater than one and less than four.

What remains then is to map the range of Y values produced into appropriately scaled step sizes for the particular dynamic feature. This is accomplished by scaling the Y values to keep the step sizes less than 0.5 (50%) of the user-specified range of values, and then using the step size as a percentage of the range to add to or subtract from the current parameter value. For example, for an individual cone that is increasing dynamically in height, first the current height is computed as a percentage of maximum height:

$$Hpct = H/(Hbase + Hrange)$$

The current Y value is then scaled by a user-supplied height scaling factor and added to the $Hpct$:

$$Hpct = Hpct + Y * Hscale.$$

If this is less than the 100% of the valid range, then the new height value is computed from the percentage value. If it is greater than 100%, then the step change sign is reversed, and remains reversed for each iteration until the minimum value of the range is reached, at which point it is reversed again.

Hence, the dynamics of the landscape can be simply and precisely specified by providing:

- Nummove - the number of peaks in motion
- Ah - the A value for peak height dynamics, if height dynamics are chosen
- Ar - the A value for cone slope dynamics, if slope dynamics are chosen
- Ax - the A value for x-axis movement dynamics, if x movement is chosen
- Ay - the A value for y-axis movement dynamics, if y movement is chosen

and a scaling factor for each A value specified.

5 Examples

In this section we illustrate a small set of the different types of dynamics that are possible and easily described with this test function. It is possible to generate dynamic functions similar to many of those already studied in the EA literature using this generator. For ease of illustration here, all examples use relatively small uniform incremental steps (small A values). It is possible, but harder to visualize dynamics involving larges step sizes, several different fixed step sizes, or chaotic step size selection.

The landscape changes can also be made recurrent (similar to the recurrent knapsack problem commonly studied in the EA literature) through selection of the initial configuration and the step-size parameter A.

5.1 Linear motion

With DF1 even simple linear motion produces rather interesting dynamics. Consider a simple landscape consisting of several peaks two of which are moving using simple linear motion. As they move, they can hide/expose other peaks as a result of *max* blending. When they reach user-specified boundaries, they "rebound" like a billiard ball. It is difficult to visualize this without providing a complete movie of the dynamics produced. Figure 4 attempts to do so by showing frames 2, 4, and 6 of a movie of two cones (from a larger group of cones) moving in opposite directions in the x-y plane.

If you look carefully, you should see them start out close together in the middle of the landscape, and then move in opposite directions, one to the left and one to the right. As they move, a previously hidden peak in the center emerges.

5.2 Changing cone shapes

Figure 5 shows the changes in the landscape caused by the slope of one cone starting out quite small (resulting in a broad peak that overshadows may others) and increases over time. As the cone's extent narrows, other hidden peaks become visible.

This illustrates a nice property of the generator, namely, that even though a fixed number of cones are initially specified, the dynamic complexity of the landscape can change rather dramatically via the masking effect due to the use of *max* blending.

5.3 Changing peak heights

As a final example, Figure 6 illustrates an environment where the peaks are stationary, but their relative height changes over time.

Although we have kept things deliberately simple for visualization purposes, even more complex dynamics can be obtained by having more than one of these types of changes happening simultaneously, and by using higher values of A to dynamically vary step sizes.

6 Summary and Future Work

We have presented our initial efforts at designing a dynamic test function generator with a simple representation mechanism that can unambiguously describe a wide range of complexity and dynamics in an elementary manner.

The methods described herein can be easily extended in several ways, the simplest of which is to start from

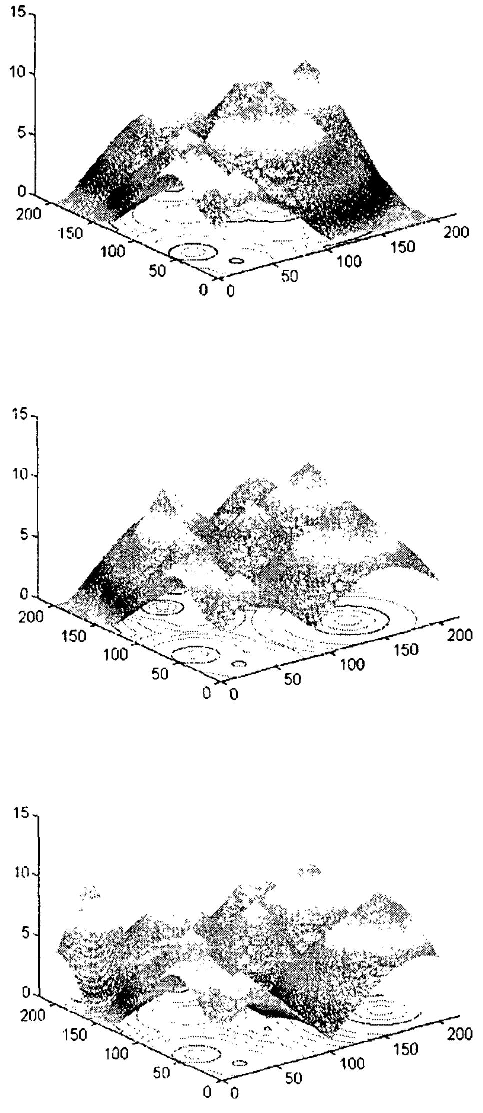

Figure 4: X-Y Movement of 2 Cones, $Ax = 2.0$, $Ay = 2.0$, $Xstepscale = .15$, $Ystepscale = .25$.

different static functions. For example, the static function chosen for DF1 has sharp, cone-like peaks. If more rounded peaks are preferred, a DF2 generator is easily derived from DF1 that uses the same base function except that no square root is taken. An example is of the effects of this simple change is given in Figure 7.

It is clear that other static functions will be required to adequately represent additional classes of problems. For example, dynamically changing discrete optimization problems are of considerable interest, but not gen-

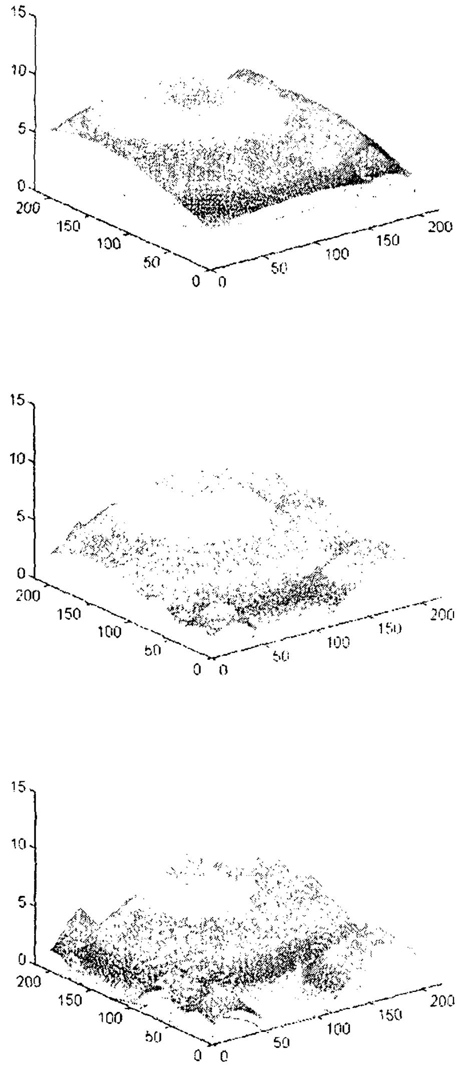

Figure 5: One Cone Changing Slope, $Ar = 2.2$, $Rstepscale = .3$.

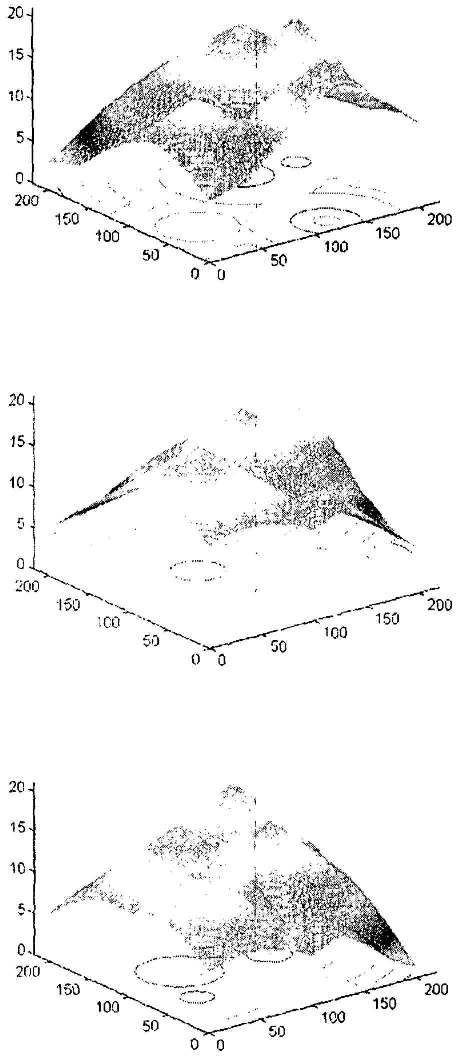

Figure 6: 14 Cones Changing Height, $Ah = 1.2$, $Hstepscale = .35$.

erable using DF1.

However, we are quite pleased with the capabilities of DF1 as it stands, and are presently using it to evaluate in a more systematic way changes one might make to EAs, such as diploid representations, tag bits, etc., to handle dynamic environments. To encourage others to do so as well, we have made DF1 available for downloading from the NCARAI website www.aic.nrl.navy.mil/galist.

Bibliography

Angeline, P. (1997). Tracking extrema in dynamic environments. In *Proceedings of the Sixth International Conference on Evolutionary Programming*, pp. 335–345. Springer.

Bäck, T. (1998). On the behavior of evolutionary algorithms in dynamic fitness landscapes. In *Proceeding of the IEEE International Conference on Evolutionary Computation*, pp. 446–451. IEEE.

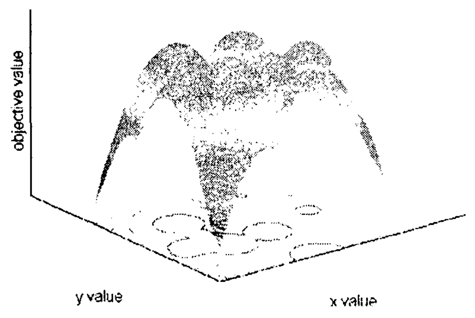

Figure 7: DF2 with N=25, Hbase=50, Hrange=25, Rbase=50, Rrange=20

Cobb, H. G. and J. J. Grefenstette (1993). Genetic algorithms for tracking changing environments. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 523–530. Morgan Kaufmann.

Dasgupta, D. and D. R. McGregor (1993). Nonstationary function optimization using the structured genetic algorithm. In R. Männer and B. Manderick (Eds.), *Proceedings of the Second International Conference on Parallel Problem Solving from Nature*. Elsevier Science.

Goldberg, D. E. and R. E. Smith (1968). Nonstationary function optimization using genetic algorithms with dominance and diploidy. In *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 59–68. Lawrence Erlbaum Associates.

Grefenstette, J. J. (1992). Genetic algorithms for changing environments. In R. Männer and B. Manderick (Eds.), *The Proceedings of the Second International Conference on Parallel Problem Solving from Nature*, pp. 137–144. North-Holland.

Grefenstette, J. J. (1999). Evolvability in dynamic fitness landscapes, a genetic algorithm approach. In *Proceedings of the Congress on Evolutionary Computation*.

Lewis, J., E. Hart, and G. Ritchie (1997). A comparison of dominance mechanisms and simple mutation in non-stationary problems. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 139–148. Morgan Kaufmann.

Mori, N., S. Imanishi, H. Kita, and Y. Nishikawa (1997). Adaptation to changing environments by means of the memory based thermodynamic genetic algorithm. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 299–306. Morgan Kaufmann.

Neubauer, A. (1997). Prediction of nonlinear and nonstationary time-series using self-adaptive evolution strategies with individual memory. In ***Proceedings of the Seventh International Conference on Genetic Algorithms***, pp. 727–734. Morgan Kaufmann.

Ryan, C. (1997). Diploidy without dominance. In ***Proceedings of the Third Nordic Workshop on Genetic Algorithms, Helsinki.*** Finnish Artificial Intelligence Society.

Varak, F., K. Jukes, and T. Fogarty (1997). Adaptive combustion balancing in multiple burner boiler using a variable range of local search. In ***Proceedings of the Seventh International Conference on Genetic Algorithms***, pp. 719–726. Morgan Kaufmann.