The Race, the Hurdle, and the Sweet Spot:
Lessons from Genetic Algorithms
for the Automation of Design Innovation and Creativity

David E. Goldberg
Department of General Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801

IlliGAL Report No. 98007
April 1998

Illinois Genetic Algorithms Laboratory
Department of General Engineering
University of Illinois at Urbana-Champaign
117 Transportation Building
104 South Mathews Avenue
Urbana, IL 61801

The Race, the Hurdle, and the Sweet Spot:

Lessons from Genetic Algorithms
For the Automation of Design Innovation and Creativity

David E. Goldberg
Department of General Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801
deg@uiuc.edu

Introduction

Design is a complex human activity involving a dizzying array of interconnected activities. Nonetheless, it seems natural to divide those activities in design that lend themselves to fairly rote algorithmic solution from those that don’t. In the former category, we might include activities such as the selection of a standard component (a beam, a transistor, or a carburetor, for example), almost all formal analysis activities related to design (mathematical modeling or computational modeling and simulation), and many activities of configuration, staging, and planning. In the latter category, are activities that go by a variety of names—words such as *innovation* and *creativity* come to mind—but the important characteristic to keep in mind here is that these activities are often *thought* to be beyond the reach of computation, and some might go so far as to reserve these activities for members of our or some other sentient species. In this article, I will argue rather strongly that computational innovation—at least certain important facets of the processes of innovation—has been achieved, and that computational creativity is plausibly within our sights. Specifically, I will argue that modern research in *genetic algorithms*—search procedures based on the mechanics of natural selection and genetics—is showing us the way toward computational innovation, thereby possibly paving the way toward a computational theory of the creative.

These are fairly bold claims, and I should confess that I didn't start out by trying to do anything as highfalutin as the construction of computational models of innovation and creativity. In fact, I first invoked (Goldberg, 1983) the term "innovation" as a simple metaphor to help explain how such simple mechanisms as genetic algorithms might plausibly do useful search. In this article, we retrace those steps by first reviewing the mechanics of simple genetic algorithms and then invoking the *fundamental metaphor of innovation* as an *explanation* for GA power of effect. Thereafter, I reverse the argument, by setting out to construct *competent* GAs—GAs that solve hard problems quickly, reliably, and accurately—through a combination of effective (1) design methodology, (2) design theory, and (3) design. While, we won’t have the opportunity to review the technical lessons in detail, the article does examine three crucial qualitative issues:

1. the key *race* between selection and the innovation operators,
2. the idea of a *control map* that helps us understand the genetic algorithm’s *sweet spot*, and
3. the primary *hurdle* or impediment to competent GA design, a hurdle that has been overcome by three different algorithms that obey the same principle: the need to identify important substructures before deciding among them.

Finally, I conclude with some speculation about what lies beyond the realm of the simply innovative and suggest that a computational theory of the creative will one day be developed using the ideas herein as springboard.

The One Minute Genetic Algorithmist

Elsewhere, I have written at length (Goldberg, 1989) about GA basics, and in this section we quickly review the fundamental mechanics by discussing what GAs process and how they process it.

Suppose we are seeking to find a *solution* to some *problem*. To apply a genetic algorithm to that problem, the first thing we must do is *encode* the problem as an artificial *chromosome* or chromosomes. These artificial chromosomes can be strings of 1s and 0s, parameter lists, or even complex computer codes, but the key thing to keep in mind is that the genetic machinery will manipulate a finite representation of the solutions, not the solutions themselves.
Another thing we must do in solving a problem is to have some means or procedure for discriminating good solutions from bad solutions. This can be as simple as having a human intuitively choose better solutions over worse solutions, or it can be an elaborate computer simulation or model that helps determine what good is. But the idea is that *something* must determine a solution's relative *fitness to purpose*, and whatever that is will be used by the genetic algorithm to guide the evolution of future generations.

Having encoded the problem in a chromosomal manner and having devised a means of discriminating good solutions from bad ones, we prepare to *evolve* solutions to our problem by creating an initial *population* of encoded solutions. The population can be created randomly or by using prior knowledge of possibly good solutions, but either way a key idea is that the GA will search from a population, not a single point.

With a population in place, *selection* and *genetic operators* can process the population iteratively to create a sequence of populations that hopefully will contain more and more good solutions to our problem as time goes on. There is much variety in the types of operators that are used in GAs, but quite often (1) *selection*, (2) *recombination*, and (3) *mutation* are used.

Simply stated, selection allocates greater survival to better individuals—this is the survival-of-the-fittest mechanism we impose on our solutions. This can be accomplished in a variety of ways. Weighted roulette wheels can be spun, local tournaments can be held, various ranking schemes can be invoked, but however we do it the main idea is to *prefer better solutions to worse ones*. Of course, if we were to only choose better solutions repeatedly from the original database of initial solutions, we would expect to do little more than fill the population with the best of the first generation. Thus, simply selecting the best is not enough, and some means of creating new, possibly better individuals must be found; this is where the genetic mechanisms recombination and mutation come into play.

Recombination is a genetic operator that *combines bits and pieces of parental solutions* to form, new, possibly better offspring. Again, there are many ways of accomplishing this, and achieving competent performance does depend on getting the recombination mechanism designed properly; but the primary idea to keep in mind is that the offspring under recombination will not be identical to any particular parent and will instead *combine parental traits in a novel manner*. By itself, recombination is not all that interesting of an operator, because a population of individuals processed under repeated recombination alone will undergo what amounts to a random shuffling of extant traits.

Where recombination creates a new individual by recombining the traits of two or more parents, mutation acts by simply modifying a single individual. There are many variations of mutation, but the main idea is that the offspring be identical to the parental individual except that *one or more changes is made to an individual's trait or traits* by the operator. By itself mutation represents a "random walk" in the neighborhood of a particular solution. If done repeatedly over a population of individuals, we might expect the resulting population to be indistinguishable from one created at random.

The Fundamental Intuition

The previous section described the mechanics of a genetic algorithm, but it gives us little idea of why these operators might promote a useful search. To the contrary, individually we saw how the operators acting alone were ineffectual, and it is something of an intellectual mystery to explain why such individually uninteresting mechanisms acting in concert might together do something useful. Starting in 1983 (Goldberg, 1983), I have developed what I call the *fundamental intuition of genetic algorithms* or the *innovation intuition* to explain this apparent mystery. Specifically, I liken the processing of selection and mutation together and that of selection and recombination taken together to *different facets of human innovation*, what I will call the *improvement* and *cross-fertilizing* types of innovation. We start first with the combination of selection and mutation and continue with the selection-recombination pair.

Selection + Mutation = Continual Improvement

When taken together, selection and mutation are a form of hillclimbing mechanism, where mutation creates variants in the neighborhood of the current solution and selection accepts those changes with high probability, thus climbing toward better and better solutions. Human beings do this quite naturally, and in the literature of total quality management this sort of thing is called *continual improvement* or as the Japanese call it, *kaizen*. When I first introduced the innovation intuition, it was largely based on introspection, but others have had similar thoughts, for example the British author and politician Bulwer-Lytton (Asimov & Shulman, 1988):

> Invention is nothing more than a fine deviation from, or enlargement on a fine model. ...Imitation, if noble and general, insures the best hope of originality.

Although this qualitative description is a far piece from an algorithmic one, we can hear the echo of mutation and selection within these words. Certainly, continuing to experiment in a local neighborhood is a powerful means of improvement, although it will have a tendency to be fairly local in scope, unless a means can be found for intelligently jumping elsewhere when a locally optimal solution is found.

Selection + Recombination = Innovation

One way of promoting this kind of intelligent jumping is through the combined effect of selection and recombination, and we can start to understand this if we liken their effect to that of the processes of human cross-fertilizing innovation. What is it that people do when they are being innovative in a cross-fertilizing sense? Usually they are grasping at a notion—a set of good solution features—in one context, and a notion in another context and juxtaposing them, thereby speculating that the combination might be better than either notion taken individually. Again, my first thoughts on the subject were introspective ones, but again others have written along similar veins, for example, the French mathematician Hadamard (1945):

> We shall see a little later that the possibility of imputing discovery to pure chance is already excluded....Indeed, it is obvious that invention or discovery, be it in mathematics or anywhere else, takes place by combining ideas.

Likewise, the French poet-philosopher Valéry had a similar observation:

> It takes two to invent anything. The one makes up combinations; the other chooses, recognizes what he wishes and what is important to him in the mass of the things which the former has imparted to him.

Once again, verbal descriptions are far from our more modern computational kind, but something like the innovation intuition has been clearly articulated by others.

With a basic understanding of the mechanics of genetic algorithms and an intuitive understanding of their power, we now examine some of the technical lessons of modern GA design and what those lessons can teach us about the processes of innovation.

Down and Dirty: Some of the Technical Lessons of GA Design

The primary difficulty of using innovation as an *explanation* or *design metaphor* for GAs is that the processes of innovation are themselves not very well understood. The more interesting possibility is that the design of more effective GAs should shed mechanistic light on one or more facets of innovation itself. In fact, the last decade has seen great strides in GA design theory and GA design to the point where it is reasonable to assert this claim. In the remainder of this section, we briefly review some of the key lessons of GA design. Here, I will abandon my concern above for the facet of innovation I called continual improvement and instead will focus entirely on the cross-fertilizing type of innovation by briefly discussing five key facets of selectorecombinative GA design.

What do GAs process? The primary idea of selectorecombinative GA theory is that genetic algorithms work through a mechanism of *quasi-decomposition* and *reassembly*. John Holland (1975), one of the great early pioneers of genetic and evolutionary computation (GEC), called well-adapted sets of features that were components of effective solutions *building blocks* (BBs). The basic idea is that GAs (1) implicitly identify building blocks or subassemblies of good solutions and (2) recombine different subassemblies to form very high performance solutions.

BB growth and timing. Another key idea is that BBs or notions exist in a kind of competitive *market economy of ideas*, and steps must be taken to ensure that the best ones (1) grow and take over a dominant market share of the population and (2) the growth rate can be neither too fast, nor too slow.

BB decision making. Understanding selectorecombinative GAs helps us understand that the decision making among different, competing notions is *statistical* in nature, and that as we increase the population size, we increase the likelihood of making the best possible decisions.

BB identification and exchange. Perhaps the most important lesson of current research in GAs is that the *identification and exchange of BBs is the critical path* to innovative success. First generation GAs, usually fail in their ability to promote this exchange reliably. The primary

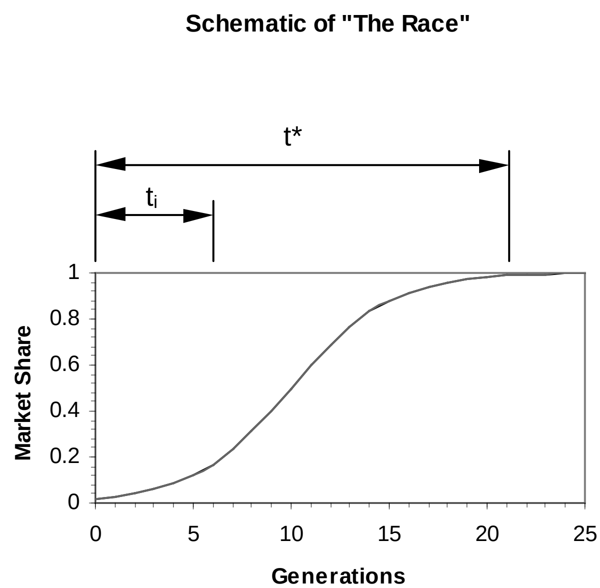

Figure 1. A schematic of the race between innovation and selection.

design challenge to achieving competence is the need to identify and promote effective BB exchange.

Hard problems are BB challenging. A final lesson of GA theory for cross-fertilizing innovation is that problems that are hard from the standpoint of innovation are problems whose BBs are hard to acquire. This may be because the BBs are deep or complex, hard to find, or because different BBs are difficult to separate, but whatever the difficulty, it may be understood in strictly mechanistic terms.

In hindsight, one wonders why it took so long to enunciate these principles, because individually they are so sensible and almost unremarkable. But before 1991 (Goldberg, 1991) there was no integrated understanding of selectorecombinative GA operation, and it wasn't until 1993 (Goldberg, Deb, & Thierens, 1993) that all the quantitative pieces of the puzzle were in place. Decomposing the facets of selectorecombinative GAs and devising quantitative design theory for each of the facets resulted immediately in two key developments: (1) the determination of the limits of first-generation selectorecombinative GAs, and (2) the design and implementation of adaptive and self-adaptive recombination mechanisms that overcome those limitations. To better understand these developments, we examine the race, the sweet spot, and the hurdle.

The Race

This synopsis of the hard fought design lessons for selectorecombinative GAs hardly does them justice, and the first key to greater insight is to understand the critical race that goes on in a competitive innovating system. This can be seen in figure 1, a schematic that illustrates the main idea. In an evolving system acted upon by selection alone, we would expect to see an S-shaped time history of the market share of the best individuals in the population. Such curves can be computed exactly for selection schemes of various types (Goldberg & Deb, 1991), but here qualitative understanding suffices. If we understand the growth of market share under selection alone, it is a straightforward matter to calculate a characteristic *takeover time* or t^* as the time it takes to go from a single good individual (a proportion $P = 1/n$, where n is the population size) to a population all but filled with good individuals (containing $n–1$ superior individuals or a market share $P = (n–1)/n$ in proportion). This takeover time is shown approximately on the schematic. This observation may seem somewhat pedestrian, and by itself it is none too useful, because real genetic algorithms use operators other than selection. What difference could it possibly make to understand the behavior of a competitive system under selection alone?

The answer to this question comes quickly and convincingly if we *imagine another characteristic time*, call it the *innovation time* t_i, which we shall define as the mean time for recombination or innovation operator to achieve a solution better than any achieved to this point. With such a characteristic time in mind there are two basic situations that we must be concerned with: the situation where the takeover time is greater than or equal to the innovation time, $t^*\geq t_i$, and that where the innovation time exceeds the takeover time $t^*<t_i$.

In thinking about these two situations, we immediately wonder which is the more advantageous for a selectorecombinative GA, and the answer is apparent with some straightforward reasoning as follows. The condition where innovation time leads (is less than or equal to) the takeover time is most advantageous for continuing innovation, because prior to the best individual dominating the population, recombination *creates a better individual*. Thereafter this better individual starts to dominate the population, and in essence, the innovation clock is reset. This cycle of partial takeover and continued innovation is repeated over and over again, resulting in the happy condition I have dubbed *steady-state innovation*.

Contrast this virtuous setting with the condition where innovation time lags (is greater than) takeover time. In such a situation, the current best guy continually increases in market share without serious competition and ultimately takes the population to substantial convergence. At the point where recombination and associated mechanism might inject something good, it is now too late, because *diversity is a necessary condition of selectorecombinative success*. To understand this, we simply need to recognize that if we, for example, have a population full of strings converged to 11111 that 11111 crossed with 11111 is 11111 almost regardless of how and where we cross. As a result, when innovation lags takeover, the population is stalled from making progress on the basis of selection and recombination alone. This situation was called *premature convergence* (DeJong, 1975) fairly early in the GA literature, but until the introduction of the above *time scales* argument (Goldberg, Deb, & Thierens, 1993), there was no means of analyzing the boundary between innovative success and failure. With the understanding of the crucial role of time scales and the race, rational analysis and design of competitive innovating GAs has advanced quite rapidly.

The Sweet Spot

One of the tools critical to these rapid advances is the so-called *control map*, which helps us delimit a genetic algorithm's *sweet spot*. Technical details of these developments are in the original papers (Goldberg, Deb, & Thierens, 1993; Thierens, 1995; Thierens & Goldberg, 1993), but here we strive for qualitative understanding of the key points. These can best be obtained by focusing on the schematic of the sweet spot of a genetic algorithm operating on an easy problem as shown in figure 2. In this map, we plot the feasible settings of the GA's control parameters, s, the selection pressure, and p_c, the probability of crossover. The selection pressure is simply the

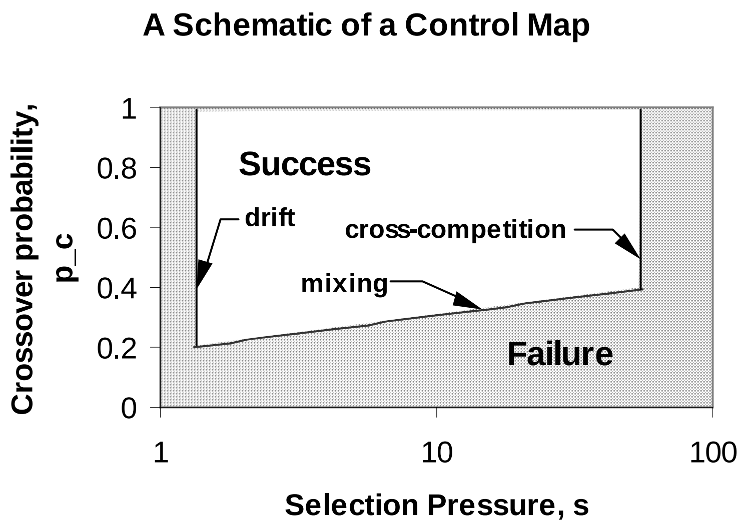

Figure 2. A control map shows the GA's sweet spot or zone of success for setting GA control parameters.

number of copies that are given to the best individual in the population under selection alone. The crossover probability is the frequency with which mated chromosomes actually undergo the exchange of crossover.

The first assumption we make in creating an s-p_c control map is that we are using a population size n somewhat in excess of that needed to make good decisions on those building blocks necessary toward an effective solution of our problem. In the original work, we used population sizes that grew linearly as a function of the number of decision variables in the problem (Goldberg, Deb, & Clark, 1992), although more recent results (Harik, Cantu-Paz, Goldberg, & Miller, 1997) suggest that the population sizes need only grow as quickly as the square root of the number of decisions variables. Interested readers should consult the original papers for detailed sizing formulas, which can be remarkably accurate guides in sizing populations. In any event, the key thing to keep in mind is that we must size the population to ensure high quality statistical decisions. If the population is sized in this way, then success or failure is determined primarily by the outcome of the race, but secondarily by whether we have allowed conditions called drift or cross-competition to occur. We discuss each of these quite briefly.

In the previous section, we discussed the race between innovation and selection, and here we quantify that race somewhat better, although the most technical of details must be left to the original literature. Most of the schemes in common usage have takeover times (in numbers of generations) that are proportional to the log of the population size n and inversely proportional to the log of the selection pressure s. Innovation time for crossover is inversely proportional to the product of the population size n and the crossover probability p_c. Recognizing that the boundary between premature convergence and steady state innovation—we will call this the *innovation boundary*—occurs mathematically when the takeover time and innovation or mixing time are of the same order of magnitude, we conclude that the crossover probability should grow as the log of the selection pressure times some function of the population size. Moreover, values of the crossover probability larger than this value are the ones for which we should expect steady state innovation (success) to occur, and values less than this are those where we should expect premature convergence to occur, and this is the state of affairs shown in the schematic. Specifically, for the boundary labeled "innovation," we see that the critical value of crossover probability grows as the log of the selection pressure and the region of success is *above* the innovation boundary.

To the left we see the region of success bounded by the so-called *drift boundary*. Biologists have long known about the possibility of convergence in small populations as a result of the vagaries of stochastic fluctuations, and genetic algorithmists have been aware of these results for some time (De Jong, 1975; Goldberg & Segrest, 1987). Formal analysis is beyond our mission here, but intuitively when the selection pressure is small enough, stochastic variations overpower selective pressure for or against an individual's increase or decrease in population market share. In such circumstances, a trait or an individual can be expected to dominate a finite population on the basis of these stochastic fluctuations alone.

To the right, we see the region of failure dominated by what we call *cross-competition*. When selection pressures are low, we expect the competition for trait or building-block market share to be held relatively independently across the genotype. In other words, conflicting traits will compete against one another, but not against non-conflicting traits. For example, we would expect different eye-color traits to compete against one another, but we would not expect an eye-color gene combination to compete with a hair-color building block. When the selection pressure is raised to very high values (values near the population size) semantically independent traits do compete with one another. To understand this intuitively, imagine what would happen if we had a selection pressure of 99 in a population size of 100. The action of selection would make 99 copies of the best individual in the population leaving only one slot for a lesser individual, and this action would reduce diversity so severely as to cripple even the most effective recombination mechanisms. A coarse analysis is given elsewhere (Goldberg, Deb, & Thierens, 1993), but here it is sufficient to recognize that when selection pressure values are raised to values near the population size there is a threshold value above which failure should be expected because of this kind of cross-competitive interference.

The Hurdle

The previous section paints a somewhat rosy picture of selectorecombinative GA life. After all, in our figure the boundaries are well spaced, and it is a fairly easy thing for the GA user to empirically or theoretically choose a set of GA parameter values to give us good results. On the other hand, we need to question whether this state of affairs will always be the case, and unfortunately it depends on the difficulty of the problem we are trying to solve. As demonstrated theoretically and empirically elsewhere (Goldberg, Deb, & Thierens, 1993), problems that may be solved through bitwise exchanges, have large sweet spots, and almost any selectorecombinative GA with any reasonable choice of crossover operator can be expected to do well with almost any serious choice of s and p_c. We can have no objections to this, and in fact the size of the sweet spot may be taken as a measure of the robustness of the algorithm.

On the other hand, as a problem becomes more difficult—that is as a problem has building blocks larger than single bits—the size of the sweet spot shrinks even as the population size is increased nominally to account for the increased noise of the more difficult problem instance. This is problematic and results ultimately in the sweet spot vanishing. Another way to view the same problem is to ask the question what size population is required to solve problems of increasing difficulty and length. Both theoretically and empirically it has been shown (Thierens & Goldberg, 1993; Thierens, 1995) that population sizes must grow exponentially to accommodate increased difficulty and length.

This leaves us with a split decision regarding the efficacy of simple GAs. If a problem is bitwise solvable, modest population sizes may be used and accurate, reliable solutions may be expected in small numbers of function evaluations, and we should expect those numbers to scale well, growing no more quickly than a subquadratic function of the number of decision variables or bits. On the other hand, with a more difficult problem instance, simple recombination operators scale badly, requiring a superexponential number of function evaluations to get reliable answers to even boundedly difficult problems, and therein lies the rub, the hurdle to the design of competent GAs. Is there some way to design crossover mechanisms that allow solutions to hard problems to scale more like those of easy problems?

Competent GA Design

Remarkably, the answer to this question is a resounding "yes," and the trick of the endeavor goes back to our discussion of the fundamental intuition. Recall that earlier we likened the processing of selection and crossover to that of human cross-fertilizing innovation. As loose metaphor this seemed reasonable enough, but if we think of our own episodes of innovation the action of simple recombination operators seems *much less directed* than the human kind. Oftentimes when human beings innovate the building blocks being exchanged are chunked whole, consciously or subconsciously, and the human recombinative process is far from the willy nilly exchange of the simple crossover operators discussed earlier. This observation makes us wonder whether there is some way of getting building a crossover operator that can *identify* and *exchange* clusters of genes appropriate to solving a problem without the need for human intervention or advice.

A line of work dating back to 1989 (Goldberg, Korb, & Deb, 1989) has set this as its goal, and succeeded in achieving this goal for the first time in 1993 (Goldberg, Deb, Kargupta, & Harik, 1993) with the creation of the *fast messy genetic algorithm* (fmGA). Two other mechanisms have also achieved competence in the sense of solving boundedly hard problems quickly, reliably, and accurately, and these are Kargupta's (1996) *gene expression messy genetic algorithm* (gemGA) and Harik's linkage learning GA (LLGA) (Harik & Goldberg, 1997; Harik, 1997). Detailed descriptions are beyond the scope of this treatment, and the mechanisms are surprisingly different from one another in their details of operation. Despite these differences, the need to tame the race between selection and crossover pushes the GA designer to do one thing above all else: *identify building blocks before deciding among them*. Stated this way, this condition sounds straightforward enough, but achieving it is a difficult design challenge, because the natural tendency of most naive design choices is to promote decision prior to identification, thereby delaying or even prohibiting the identification of the best groupings of variables. If this happens, and the problem is even partially misleading, the tendency will be for genes to fix on the wrong values, whereupon the algorithm will be led to the wrong solutions.

Figure 3 shows the results from the first competent GA, the fast messy GA (Goldberg, Deb, Kargupta & Harik, 1993). In a problem with order-5 difficult building blocks, the fast messy GA is able to find global solutions in times that grow has a subquadratic function of the number of decision variables as expected. By contrast, the original messy GA, and mutation-oriented hillclimbing (Mühlenbein, 1992), have numbers of function evaluations that grow as a quintic function of the number of decision variables. Of course, quintic growth is superior to the expected exponential growth required of a simple GA using a recombination operator of fixed

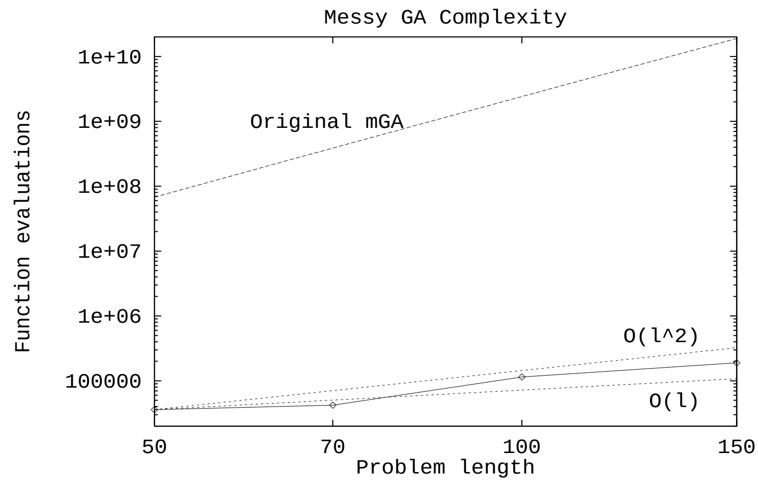

Figure 3. The fast messy GA results reported in Goldberg et al. (1993) demonstrate subquadratic growth in the computation time compared to the original messy GA and simple mutation-based hillclimbers, which grow as the fifth power of the number of variables and are much slower across the board than the fmGA.

structure. But the subquadratic growth expected on all problems of fixed difficulty is exactly the kind of robust solution genetic algorithmists have been seeking for so long.

The problems that can be solved via this mechanism are quasi-decomposable ones, but they are by no means easy. Straightforward hillclimbing mechanisms will be defeated on relatively easy instances of such problems, and it is not an exaggeration to claim that a competent selectorecombinative GA is *innovating* in a cross-fertilizing sense to quickly find global or very near global solutions. Although most GA implementations do not use such recombination operators yet, the research reported herein has asked and answered the right questions, and it is only a matter of time before such mechanisms are incorporated into everyday GA practice. While this integration may yet take some time, and the exact combination of mechanisms to use in practice is still open to refinement, I will assume in the remainder that the problem of computational cross-fertilizing innovation is solved or at least well on its way to a solution. In what remains, I speculate on the possibility of a computational theory of the creative.

From Competence to Creativity

The progress made in developing competent selectorecombinative GA whets our appetites for more. One way to consider what lies beyond competent GAs is to appeal to the innovation intuition one last time to see if it might help us imagine mechanisms that go beyond the "merely innovative."

Thinking informally and linguistically, one word that we often use to describe thinking processes that go beyond the innovative is "creative." What does it mean to be creative? Well, it probably means more than one thing[1]—just as innovation has meant at least two different things in this paper—but it is helpful to examine what we mean more closely when we say that a piece of art, for example, is creative. Oftentimes, creativity in art (or literature or science or engineering) can be understood as a certain *richness of association* with different, perhaps seemingly unrelated disciplines. Much great art alludes to prior art, literature, culture, history, and many more sources besides, and all of this can understood as a form of *knowledge transfer* from those disciplines to the subject at hand.

In the same way, a more *creative* algorithm would act to go beyond *innovation* by transferring useful information from other domains. Seen in this way, innovative mechanisms involve exchange or discovery from *within* a discipline, whereas creative computation requires a transfer from *without.* There are a number of mechanical ways of imagining how to do this, either through the transfer of deep, whole building blocks from other fields, or through the transfer of knowledge about better knowledge representation. At this juncture, the key to keep in mind is that whatever mechanisms we devise should borrow the *implicitly parallel* style of selectorecombinative GAs and avoid the one-at-a-time explicit calculation of symbolic artificial intelligence or similar disciplines. Doing so will maximize the exploitation of local processing, minimize the utilization of global mappings, data structures, and processing at the same time it gives us competent creativity that solves even harder problems, more quickly, more reliably, and more accurately. Designing and implementing such mechanisms is a tall order, but the methodological, theoretical, and implementation example of selectorecombinative GAs gives us more than a little hope that a vision of computational procedures that surprise and delight us is nearer than we thought.

Conclusions

This paper started by distinguishing between tasks in design that lend themselves to rote computation and those that don't. It has ended by undermining that very categorization and by suggesting that the processes surrounded in the most mystery—processes that go by the names "innovative" and "creative"—are yielding to computational modeling and mechanistic understanding. Specifically, work in genetic algorithms has been reviewed in the light of the so-called innovation intuition, and this has led to a more detailed consideration of five of the key technical lessons of selectorecombinative GA design. In turn these lessons have led us to consider the race between innovation and selection, the sweet spot of GA control parameters, and the primary hurdle to effective GA design, which have both exposed the weakness of first-generation GAs and shown us how to make them competent.

It is yet unclear how this work will impact the practice of GAs on the one hand and the state of knowledge on the other, but it seems reasonable and unassailable to assume that fast, accurate solutions to very hard search and optimization problems are useful and desirable across the spectrum of human endeavor. It also seems reasonable to assume that a more mechanistic understanding of the categories "innovative" and "creative" is something that the humanities, the social sciences, and physical sciences could all benefit from. But the assertion that genetic algorithms offer useful models for the understanding of such things will no doubt remain controversial for some time to come. Nonetheless, even if these models only illuminate small nooks and crannies of intelligent innovation and creativity they will have performed a large service. Certainly having at least one line of consistent methodology, theory, and design practice

[1] In fact, the definition herein differs from those discussed elsewhere in the volume in the section entitled "Creative Evolutionary Design." None of these perspectives should be viewed as "correct," and if we were to have convergence in the population of definitions at this early stage in our quest for innovating and creating machines, it would almost certainly be premature.

that has already made important strides through such difficult territory should give others reason for reflection, imitation, and hypertrophy to see how far these ideas can go.

Acknowledgments

This paper was written mainly because Peter Bentley is a persistent guy. I thank him for urging me to write this paper until I actually did so. It was written while I was on sabbatical at the Section on Medical Informatics at Stanford University. I am grateful to Mark Musen, Russ Altman, and John Koza for inviting me to lovely, albeit somewhat soggy, Palo Alto.

My contribution to this study was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-94-1-0103, F49620-95-1-0338, and F49620-97-1-0050. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are my own and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U. S. Government.

References

Asimov, I., & Shulman, J. (Eds.) (1988). *Isaac Asimov's book of science and nature quotations.* New York: Weidenfeld & Nicolson.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Doctoral dissertation, University of Michigan, Ann Arbor. (University Microfilms No. 76-9381)

Goldberg, D. E. (1983). *Computer-aided pipeline operation using genetic algorithms and rule learning*. Doctoral dissertation, University of Michigan, Ann Arbor.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley.

Goldberg, D. E. (1991). *A tutorial on genetic algorithm theory.* A presentation given at the Fourth International Conference on Genetic Algorithms, July 1991, University of California at San Diego, La Jolla, California.

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms, 1*, 69–93. (First published in 1990 as IlliGAL Report No. 90007)

Goldberg, D. E., Deb, K., & Clark, J. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems, 6,* 333–362. (First published in 1991 as IlliGAL Report No. 91010)

Goldberg, D. E., Deb, K., Kargupta, H., & Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 56–64. (First published in 1993 as IlliGAL Report No. 93004)

Goldberg, D. E., Deb, K., & Thierens, D. (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of the Society of Instrument and Control Engineers, 32*(1), 10–16. (First published in 1992 as IlliGAL Report No. 92009)

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems, 3*(5). 493–530. (First published in 1989 as TCGA Report No. 89003)

Goldberg, D. E., & Segrest, P. (1985). Finite Markov chain analysis of genetic algorithms. *Proceedings of the Second International Conference on Genetic Algorithms*, 1–8.

Hadamard, J. (1945). *The psychology of invention in the mathematical field.* Princeton University Press: Princeton, NJ.

Harik, G. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms.* Doctoral dissertation, University of Michigan, Ann Arbor. (Also published as IlliGAL Report No. 97005)

Harik, G., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation,* 7–12. (First published in 1996 as IlliGAL Report No. 96004)

Harik, G., & Goldberg, D. E. (1997) Learning linkage. *Foundations of Genetic Algorithms, 4*, 247–262. (First published in 1996 as IlliGAL Report No. 96006)

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press.

Kargupta, H. (1996). *SEARCH, evolution, and the gene expression messy genetic algorithm* (Unclassified Report LA-UR 96-60). Los Alamos, NM: Los Alamos National Laboratory.

Mühlenbein, H. (1992). How genetic algorithms really work: I. Mutation and hillclimbing. *Parallel Problem Solving from Nature*, 15–25.

Thierens, D. (1995) *Analysis and design of genetic algorithms*. Doctoral dissertation, Katholieke Universiteit Leuven, Leuven, Belgium.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 38–45.