A Genetic Algorithm with Self-Distancing Bits but No Overt Linkage

William A. Greene
Computer Science Department
University of New Orleans
New Orleans, LA 70148
bill@cs.uno.edu
504-280-6755

Abstract

We present a novel representation and crossover operator for genetic algorithms. Bits are not linked to one another. Instead, the current population suggests a pseudo-distance between each pair of bits; this pseudo-distance really measures the degree to which bits appear to participate in a building block. Then crossover respects the pseudo-distance: it is clumps of nearby bits that have their values copied from parent to child. Thus our new approach does directly (preservation of building blocks under crossover) what other approaches only hope to do indirectly. Our approach is tested on several problems, ranging from simple to very challenging, and the results compared to standard approaches. In these problems, the new approach is successful and usually outperforms the standard approaches.

1. INTRODUCTION

We assume there is a problem of interest to us, and we wish to use a genetic algorithm to search among the solutions to the problem. There is a plenitude of solutions to the problem, of varying quality; some are rather good solutions, and some are only fair. We assume there is a known measurement of the quality of a solution, which we term its *fitness* and which is a non-negative real number. Individual solutions are identifiable with the property values they exhibit, along a known set of properties. Solutions differ one from another by having different values for these properties.

In the standard representation used for genetic algorithms, an individual solution (which we now begin to term simply an *individual*) gets represented by representing its property values as values, which in this paper we will take to be bits, which are linked together in a linear sequence, like beads along a strand, that is, like genes along a chromosome. Under this representation, which mimics a biological model, mating with crossover continues the mimicry, in particular of haploidal reproduction. One or more crosspoints are chosen at random along the strand, parental genetic sequences are clipped at those points, and parental genetic fragments are exchanged, to form the children. Also mutation is easily mimicked, by changing an occasional bit value.

Two bits, as a pair, can assume any one of four bit-pair values, namely, (0, 0), (0, 1), (1, 0), and (1, 1). Let us say that two bits are *closely related*, provided that they can exhibit a bit-pair value that is rather beneficial to a solution, in the sense that having this pair of values significantly increases the fitness of the solution. This notion of a beneficial bit-pair value is an instance of what [Holland, 1975] terms a building block.

When we read the proof of the Schema Theorem (see [Holland, 1975] or [Goldberg, 1989]), we learn that two closely related bits can suffer from a great hazard. To lie far apart from one another along the bead strand increases the likelihood that the beneficial bit-pair value will be destroyed under crossover. One parent may exhibit the beneficial bit-pair value, but if a crosspoint is chosen between the pair of bits, it can happen that neither child exhibits the pair.

This hazard was recognized at the dawn of genetic algorithms, and since then attempts have been made to contend with it. One possibility is reordering bits, with the intention of having closely related bits wind up situated near one another. [Holland, 1975] noted that the *inversion* operator (a subsequence of bits gets its order reversed) might be useful in reordering bits dynamically. [Goldberg, 1989, pp.166-179] argues that for permutation-based representations (as might be used in the traveling salesman problem), certain permutation-based crossover operators, such as PMX [Goldberg and Lingle, 1985], combine the actions of crossover and reordering. The messy genetic algorithm mGA of [Goldberg, Korb, and Deb, 1989], along with its other distinctive features, implicitly permits reordering of bits. Bui and Moon have a sequence of papers that deal with ordering and preordering bits; of particular interest to them are graph problems, such as the graph bisection problem (see section 3.5 below). In [Bui and Moon, 1993] there is a preprocessing step which makes the sequence of bits (one for each graph vertex) reflect certain vertex adjacencies as they are evidenced in, say, breadth-first traversal. Then in [Bui and Moon, 1995] they follow a different course; this time the bits are placed, not in a one-dimensional sequence, but instead at integral points of multi-dimensional real space; they argue the latter allows room for a more faithful representation of adjacencies. In [Greene, 2000] it is shown that under reasonable assumptions, a schema theorem obtains when the structure of bits and their linkages is liberalized to be as general as a connected graph, and then in [Greene, 2001] there are experiments with such alternative bit arrangements. [Sehitoglu and Ucoluk, 2001] explicate a regime for exchanging bits with their neighbors to the left or right, based upon whether they appear to participate in a building block.

(A new school, of probabilistic modeling, takes a completely different tack, but with the same general goal of identifying and exploiting those bits which are closely related. This school is exemplified by the research in [Muehlenbein and Paass, 1996], [Pelikan, Goldberg, and Cantu-Pas, 1998], and [Harik, 1999]. In this school, simulation of biological crossover is abandoned, in favor of a generate-and-test approach. The typical regimen is to loop on two steps: use the current population to infer some probabilistic dependencies between the values of the various bits, then use those probabilities to stochastically manufacture plausible individuals that will comprise the next generation.)

In the current paper, we will take a novel step towards correcting the hazard mentioned above. We will stay within the tradition that simulates crossover, but our bits will not be overtly linked together at all!

Reading the proof of the Schema Theorem suggests the following line of reasoning. When parental genetic material is inherited, it should be inherited in a certain piecemeal way. When parental bit values are copied into child bits, closely related bits should be copied in clumps. We will loop to copy parental bit values. When an uncopied parental bit is chosen to be the next one copied, we will copy not only that bit's value, but also the values of those still uncopied bits which are suitably closely related to it.

Above we alluded to reordering efforts, and non-linear linkage schemes. We now describe these as follows. Such approaches link or re-link bits one to another, with an eye to positioning closely related bits close together (generally this means a short path length between them). Then it should follow that when links are chosen for clipping during crossover, there will be a decreased likelihood that closely related bits get separated from one another.

The appeal of our approach is that it does directly what the other approaches only hope to do indirectly: closely related bits tend to clump together when parental genetic material is being copied into children. Linking bits one to another is a held-over artifact from the biological model of a chromosome (especially this is so when the linkage is into a linear sequence), and we now dispense with it.

Below we introduce a plausible measurement for close relatedness between bits. We will think of this measurement as also giving a pseudo-distance between bits. Our genetic algorithm will be generational (an entire new population $P(t+1)$ at time tick $t+1$ is created from the population $P(t)$ at time t, as opposed to a steady-state approach). Also, the measurement of close relatedness (or pseudo-distance) between bits is re-calculated for each generation. We term our approach a linkless self-distancing genetic algorithm.

2. THE LINKLESS SELF-DISTANCING GENETIC ALGORITHM

In this section we describe the details of our approach, LSDGA. Of course, our approach is mostly distinguished by a new way of representing an individual in the population, then by the algorithm for crossover that follows from it.

2.1. AN INDIVIDUAL

An individual is a set of bit-values. The bits are not linked together. Each individual in the population consists of the same number of bits, and that number is constant over all generations of the population. An individual has a fitness, which is a non-negative real number. The fitness value is problem-dependent.

2.2. CLOSELY RELATED BITS

How can we gauge when two bits are closely related? We do not purport to provide a perfect answer to that question, but our answer is plausible and persuasive. Future research may improve upon our answer.

As a pair, two bits can assume any one of four different bit-pair values, namely, (0, 0), (0, 1), (1, 0), and (1, 1). Our sentiment is that two bits are closely related, provided that they can exhibit a bit-pair value that is rather beneficial to an individual, in the sense that having this bit-pair value significantly increases the fitness of the individual. [Holland, 1975] would say that the two bits form a (small) building block.

We think of the current population as a sampling of the entire fitness landscape, and as such it offers evidence as to which bits are closely related. We consider just the fitter half of the current population (some other fractional part of the fitter individuals may be a better choice); denote this subset S. Now let two bits $b1$ and $b2$ be given. Considering just bits $b1$ and $b2$, the members of S may potentially exhibit any one of the four bit-pair values, and of course the members of S have their respective fitnesses. If among the members of S, fitness is rather disproportionately concentrated above just one of the four possible bit-pair values assumable by the bit-pair $b1$ and $b2$, then that is what we will deem close relatedness between $b1$ and $b2$. (Put another way, let us consider the opposite circumstance. If fitness is evenly distributed above the four bit-pair values assumable by $b1$ and $b2$, then we would say these bits are independent of one another, meaning that the value of one has little to do with the value of the other, as far as contributing to the fitness of an individual.)

Specifically, we do the following. For $i = 0, 1$, and $j = 0, 1$, let F_{ij} be the sum of the fitnesses of those elements of population subset S for which bit-pair ($b1$, $b2$) assumes the value-pair (i, j). Let $F = F_{00} + F_{01} + F_{10} + F_{11}$. The four fractions $r_{ij} = F_{ij} / F$ lie in the real unit interval [0, 1], and add up to 1.0. We want to detect the situation when one of these fractions is rather close to 1.0 (and the other three are nearly 0.0). More than one expression would reveal such; we use a familiar entropy calculation,

$$(1/2) \cdot \sum_{i,j} (-r_{ij} \cdot \log_2 r_{ij})$$

and denote this value by *dist*($b1$, $b2$). The normalizing factor 1/2 guarantees that this value lies in the real unit interval [0, 1]. Finally we observe that bits $b1$ and $b2$ more nearly fulfill our notion of being closely related exactly when *dist*($b1$, $b2$) more nearly approximates zero, and we will think of *dist*($b1$, $b2$) as being a pseudo-distance value.

At this point a reader may interject that a building block may consist of more than just two bits. We respond by reasoning as follows. If fitnesses are concentrated above one of the eight possible bit-triple values assumable by a triad of bits, then even more so is fitness concentrated above the bit-pair values of any two bits of the triad. This is because the latter concentration includes the concentration above the bit-triple value. On the other hand we also caution the reader as follows. For triads, the corresponding concentration metric (measuring the spread of eight fractions) should use not 1/2 but 1/3 as its normalizing factor. The concentration metric for a triad is not necessarily either greater or less than the concentration metric for a pair of bits drawn from the triad.

As remarked earlier, we practice generational evolution. Mating with crossover among sundry pairs of members of the population $P(t)$ at time t is used to create the next population $P(t+1)$. The first step in this generational rollover is to use (the fitter half of) population $P(t)$ to calculate the pseudo-distances between all pairs of bits; these distances will be used during crossover. Note that pseudo-distances are re-computed on each generation. (If an individual consists of 100 bits, there are 100 * 99 / 2 = 4950 bit-pairs, and so also that many distances get calculated.)

During crossover, we want bits that are suitably close together in pseudo-distance to get copied in clumps. To this end we calculate a *cutoff* value for pseudo-distance. We found the following worked well in our experiments. Bit-pair pseudo-distances are grouped into a histogram of twenty 5% brackets, then the chosen cutoff is that pseudo-distance that as an upper bound contains at least 20% of the bit-pair pseudo-distances (or sometimes it is 25%).

2.3. CROSSOVER

Crossover is now easily described. Two parents produce two children. To copy parental bit values into child bits, we loop on four steps. Step 1: choose an uncopied bit b at random. Step 2: to it, group those uncopied bits b' such that $dist(b, b') \leq \text{cutoff}$. Step 3: for k = 1, 2, copy this group of bit-values from parent(k) into child(k). Step 4: with 50% probability, interchange the roles of child(1) and child(2). (Step 4 means we may expect parts of a parent's genetic material to wind up in both children.) Let us note that the copying of isolated bits (ones not suitably close to other bits) resembles uniform crossover [Syswerda, 1989]. (In step 2, we sometimes prevent copying of an entire clump that is too large by making nearby bit b' jump a 50-50 hurdle before we group it with b.)

It is conceivable that premature convergence of the population can occur. Once the (fitter half of the) population members closely resemble one another, most bit pairs will seem to have fitness concentrated above a particular bit-pair value, so most bit pairs will appear to be quite close to one another, and a child may simply duplicate a parent. In section 2.4, which concerns how the next generation of the population is constructed from the current one, we take steps to diversify the population.

2.4 GENERATIONAL CHANGE

Our initial population $P(0)$ consists of random individuals.

To construct the next generation $P(t+1)$ of the population from $P(t)$, we first practice elitism, and have the fittest two members of $P(t)$ survive intact into $P(t+1)$. Then $P(t+1)$ is filled up to the same size as $P(t)$ by mating with crossover. As an aside, population size is kept small, typically between 25 and 50. Also, we linearly scale the set of fitness values present in the current population into a real interval of the form [1, maxVF] (for maximum virtual fitness) in such a way that the smallest fitness value in the population is mapped to 1, and the largest is mapped to maxVF. (Typically maxVF is 2 or 4.) Then, individuals are selected for parenting by using a weighted roulette wheel based upon the scaled fitnesses (see [Goldberg, 1989]).

Two parents produce two children. Each child is added to $P(t+1)$, but only if it is distinct from the individuals already in $P(t+1)$. Next we sort $P(t+1)$ into decreasing order of fitness, as a prelude to mutation.

The elite survivors in $P(t+1)$ are spared any mutation. For the rest, mutation is graduated and stochastic. A single mutation step consists of flipping the value of a randomly chosen bit. Each individual is subjected to some number of mutation attempts; in our experiments, the maximum number of attempts is 40. Mutation is stochastic, in that a mutation attempt succeeds to become a completed mutation step only with a 50% probability. Mutation is graduated, in that less fit individuals are subjected to more mutation attempts. The number of attempts is proportional to the rank of the individual within the (sorted) population. Thus the least fit individual is subjected to 40 mutation attempts, and we expect about 20 of these to result in the flipping of a bit in that least fit individual.

This completes the construction of $P(t+1)$.

2.5. STOPPING CONDITIONS

Generations of populations P(1), P(2), P(3), …, are formed, until some maximum number of generations has been reached, or some stopping condition has been met. Since we view genetic algorithms as a heuristic approach to problem solving, in this research we often content ourselves when a very good though sub-optimal solution has been unearthed. For many of the problems used in our experiments, a maximum fitness is known for the problem, and so we may content ourselves if we reach 98% of that value.

3. EXPERIMENTS

We have explored the behavior of our approach on a number of problems. The problems range from simple ones, to very challenging ones drawn from the literature.

3.1. COUNTING WEIGHTED 1'S

In this simple experiment, there are 80 bits, and they are numbered 1 through 80. The fitness of an individual equals the sum of the bit numbers of those bits whose value is 1 (versus 0). This problem is not quite the "counting 1's" problem (see Experiment 3.3); we might term it "counting weighted 1's". The maximum fitness is 1 + 2 + 3 + … + 80 = 80 * 81 / 2 = 3240. There is one individual of maximum fitness. The fitness landscape everywhere slopes upward towards this maximum. (Why? if we hill-climb in Hamming space, then changing a 0 to a 1 in an individual produces an individual of increased fitness, with the degree of increase depending on the bit number.) We used 98% of maximum fitness, or 3175.2, as acceptable fitness. We ran 20 trials of this experiment, using 40 as our population size, and allowing up to 100 generations per trial. We would expect the bits having high bit numbers to rapidly converge to the value of 1, and indeed this is what can be observed when a trial's generations are examined sequentially. All but two trials found an individual of acceptable fitness before exhausting all 100 generations. Over the 20 trials, the average fitness of the fittest individual found on a trial was 3186.75 and the average final generation number was 63.0. So, our new approach is successful at finding rather good solutions in a reasonable amount of time. Table 1 summarizes the results of this experiment.

Table 1: Counting Weighted 1's

Bits per individual	80
Optimal fitness	3240
Acceptable fitness	3175.2
Number of trials	20
Population size	40
Max generations	100
Avg final gen. num	63.0
Avg best fitness	3186.75

3.2. WEIGHTED 8-BIT GROUPS

This experiment is somewhat similar to the preceding one. Again an individual consists of 80 bits, but they are no longer numbered. Instead, they are grouped into 10 groups of eight bits each, and the groups are numbered 1 through 10. A subgroup of bits makes its own contribution to fitness. For group number k, $1 \le k \le 10$, let $n(k)$ denote the absolute value | (number of 1's in k-th subgroup) minus 4 |. Note $n(k)$ is in the range 0..4, and has the maximum value 4 when either all (eight) of the subgroup's bits are 1's or all are 0's. The fitness of an individual is then defined to be $\Sigma_k(k \cdot n(k))$. The maximum fitness is 1*4 + 2*4 + … + 10*4 = 220. For this problem, there are 2^{10} = 1024 individuals of maximum fitness. These individuals exhibit all 0's or all 1's in each subgroup. As earlier, we used 98% of maximum fitness as acceptable fitness. We ran 20 trials of this experiment, using 40 as our population size, and allowing up to 100 generations per trial. We would expect subgroups to converge towards all 1's or all 0's, with convergence happening sooner in subgroups having a higher group number, and this can be observed when a trial's generations are examined. This time, 13 of the trials found an individual of acceptable fitness before exhausting all 100 generations. Over the 20 trials, the average fitness of the fittest individual found on a trial was 215.5 and the average final generation number was 84.3. See Table 2.

Table 2: Weighted 8-bit Groups

Bits per individual	80
Optimal fitness	220
Acceptable fitness	215.6
Number of trials	20
Population size	40
Max generations	100
Avg final gen. num	84.3
Avg best fitness	215.5

3.3. TARGETING A SPECIFIED INDIVIDUAL

In [Greene, 2001] the following experiment is described. An individual is a 2-dimensional 24 x 24 grid of bits. A particular individual is distinguished (it resembles the letter capital-A against an opposing background); this individual becomes the target. Then, an arbitrary population individual has an *error*, equal to its Hamming distance from the target, with a maximum value of 24 * 24 = 576. Thence the individual has a fitness, defined as maximum error minus own error. Thus maximum fitness also equals 576, and we use 98% of that, or 564.48, as acceptable fitness. We note that this problem is isomorphic to a "counting 1's" problem. In a counting 1's problem, the fitness of an individual is the number of bits having value 1. For the problem at hand, fitness equals the count of bits which have the target's corresponding bit value. In a counting 1's problem, each bit acts as an independent building block, there is one individual of maximum fitness, and the fitness landscape everywhere has the same slope upward towards the maximum. In the cited paper, two parents are cut by a random 2-dimensional sub-grid, for purposes of exchanging bit groups at crossover time.

For comparison's sake, first we repeated this experiment. Then secondly we re-worked this problem as a 1-dimensional problem, by re-representing each grid as a 1-dimensional bit string, under row-major representation, and practicing 1-point crossover among parents. Finally, we also used the same problem to test our new linkless self-distancing approach to representation and crossover. We ran 20 trials, allowing up to 2000 generations upon each trial. Table 3 compares the results of these three approaches. For the 1-dimensional and 2-dimensional approaches, all 20 trials exhausted all 2000 generations without unearthing an individual of acceptable fitness. Contrast that with our new approach. It invariably found an individual of acceptable fitness, with average last generation number equal to 496.0. Moreover, this was achieved while using a smaller population size. For this problem, our new approach is unmistakably better and faster.

Table 3: Targeting a Specific Individual

For each representation, an individual is made up of 576 bits, maximum fitness is 576, acceptable fitness is 564.48.

	1-diml	2-diml	LSDGA
Number of trials	20	20	20
Population size	50	50	32
Max generations	2000	2000	2000
Avg final gen. num	2000.0	2000.0	496.0
Avg best fitness	544.05	550.9	565.05

3.4. THE 20 QUEENS PROBLEM

We reprise a second problem from [Greene, 2001]. This is the 20 Queens problem, which is the analogue of the classic 8 Queens problem from chess. There is a 20 x 20 chessboard, and the goal is to have 20 queens placed into board squares so that no queen is attacking the others. In the cited paper, the chessboard surfaces as a 20 x 20 grid of bits, with bit value 1 meaning the board square is occupied by a queen, whereas value 0 means the square is empty. Fitness is addressed as follows. Errors are added up. One or more errors are occurring when any of the following holds: a row or a column contains $n \neq 1$ queens, or a diagonal or counter-diagonal contains $n > 1$ queens. The worst situation occurs when every square (not merely 20 squares) has a queen on it, in which case the total number of errors equals 2*(E - 1)*(2* E - 1), where E = edge-size. Here E = 20 and maximum error = 1482. Then the fitness of an individual is defined as maximum error minus own error. Maximum fitness is then 1482. In our experiments we take 98% of that figure, or 1452.36, to be acceptable fitness.

This is a hard problem, with many constraints to satisfy and many epistatic interactions between bits. The fitness landscape for this problem is hard to analyze, but probably it is rather jagged. Chess players know there are many individuals of maximum fitness. For instance, given one solution to this classic puzzle, the 8 symmetries of a square (obtained by rotations and reflections) provide more solutions.

The cited paper worked this problem 2-dimensionally. In doing so, two parents were cut by a random subgrid, for purposes of crossover. We first re-worked the cited research. Then we also worked this as a 1-dimensional problem, by re-representing a board in row-major form as a 1-dimensional array, and practicing 1-point crossover. Finally we worked this problem using our new approach.

Table 4 summarizes the results. The 1-dimensional and 2-dimensional approaches generally, but not always, found an acceptable individual before exhausting all allowed generations in a given trial. The average best individuals found by the three approaches have nearly equal fitnesses, but our new approach when applied to this problem finds an acceptable individual in *circa* 6 times fewer generations, and does so using a smaller population size, as well. For this problem, our new approach is very much the superior one.

Table 4: The 20 Queens Problem

For each representation, an individual consists of 400 bits, maximum fitness is 1482, acceptable fitness is 1452.36.

	1-diml	2-diml	LSDGA
Number of trials	20	20	20
Population size	100	100	40
Max generations	2000	2000	1000
Avg final gen. num	1863.1	1565.5	286.5
Avg best fitness	1450.6	1451.55	1453.8

3.5. RE: GRAPH BISECTION

The next two experiments concern graph bisection, so in this section we discuss that topic. Let a graph G, having *n* vertices and some number of edges, be given. For simplicity, we will assume *n* is even. A *bisection* of G means a partitioning of G's vertices into two subsets of the same size, *n*/2. The *cut-size* of the bisection is defined to be the number of edges which have an endpoint in each of the two vertex subsets. The *graph bisection problem* is to identify a bisection with the lowest possible cut-size.

Graph bisection has been studied by researchers such as [Kernighan and Lin, 1970], [Johnson *et al*., 1989], and [Laszewski, 1991]. It has also been examined in a series of papers by Bui and Moon and their colleagues, for instance [Bui and Moon, 1993], [Bui and Moon, 1995], and [Bui and Moon, 1996]. Graph bisection has practical applications, and is also known to be a hard problem [Bui and Jones, 1992]. Space does not allow a full explication of the issues of this problem. Our work follows the general development of the past researchers. We refer the interested reader to the cited papers.

A bisection of the graph puts its vertices into two "halves", call them the A and B halves. To search for a bisection with minimal cut-size, we proceed as follows. Represent a bisection by using *n* bits, one for each vertex. Bit values of 0 versus 1 signify whether the associated vertex is in the A or B subset. Then we can easily compute cut-size: loop through the list of edges, tallying each whose two endpoints are in different subsets. Note that cut-size can be as low as 1, and can be no greater than the number of edges.

Our attack on a graph bisection problem will use the approach to representation and crossover which we explicated in section 2. An individual in the population consists of *n* bits (*n* = the number of vertices), moreover, we will see to it that *n*/2 of these bits have the value 0 and the other *n*/2 have the value 1. We define the fitness of an individual to be: the number of edges in G, minus the cut-size of the bisection which corresponds to the individual. Maximum fitness equals number of edges, minus 1. When the least possible cut-size of a graph is unknown, as is usually the case, in our trials we let acceptable fitness equal maximum fitness. Mating with crossover can proceed as we described it in section 2, but we need to end it with two additional steps. The first step is a repair step, and the second step is an improvement step.

(There is also another consideration made, at the start of crossover. Our representation admits an isomorphy. The result of flipping every bit of an individual is what we may term its *complement*. An individual and its complement really represent the same partition of the vertex set into two subsets. We note that mating an individual with its (near) complement is likely to produce a chaotically different child, whereas the child of two (near) identical partitions should be a (near) duplicate of its parents. Hence we make it a practice at crossover time that we mate parent-1 with whichever of parent-2 or its complement is the closer to parent-1 in Hamming distance.)

Crossover and mutation can produce an individual (child) for which the A and B subsets are not the same size. This obliges us to repair the individual, by flipping enough of the bits with the value (0 or 1) which occurs in excess.

Our repair work is heuristic. Note that to move a vertex to the other subset (A or B), that is, to flip a single bit, implies a changed bisection and hence a change in cut-size. A negative change to cut-size is favorable, for it makes cut-size become lower, as is our desire. We make a list of the bits which exhibit the excess value (0 or 1), and with each such bit we pair the change in cut-size that would result from flipping it. Repair then takes the form of looping on 2 steps until the required number of bits have flipped: (1) identify the listed bit which has the most favorable change in cut-size, remove it from the list, and flip this bit; (2) update the change-to-cut-size for those listed vertices which are adjacent to the flipped one (this is the only updating which is necessary here, the correct update is to subtract 2).

(The repair work in [Bui & Moon, 1996] is different. Their bits are stored in an array which, for the repair step, is treated as circular. They pick a random index and from there move forward, flipping those bits which exhibit the excess value, until enough have been flipped.)

Next we describe the improvement step. The improvement step, which is done to a partition which has already been balanced into two subsets of the same size, is also heuristic. The idea goes back to [Kernighan & Lin, 1970]. Similar to the earlier observation, we note that to have two vertices, one from each subset, exchange sides also implies a change in cut-size. In the improvement step, a well-chosen subset of the A-elements, together with a well-chosen subset, of the same size, of the B-elements, are identified, then these groups exchange sides. The procedure is summarized in the 5 steps given next. (1) Make a list, *AList*, of the A-elements, pairing each with the change in cut-size that would result if that vertex were to change sides; sort *AList* into increasing order. Similarly form *BList* out of the B-elements. Also create a list *BestPairs*, initially empty.

(2) Now look at a window of the best *w* elements from the *AList* and likewise the *w* best elements from *BList*. Constant *w* is window size; for it we used 10. There are *w***w* pairs of vertices, one from A and one from B, formable from our windows. Identify the pair for which there is the most favorable change to cut-size if the two vertices were to exchange sides; append that pair to the end of list *BestPairs*. (3) Remove these two vertices from *AList* and *BList*. Update the change-to-cut-size of those vertices adjacent to the two vertices. Re-sort *AList* and *BList*. (4) Loop back to step 2 until enough pairs have been put into the sequence *BestPairs*. We follow the advice of Bui and Moon and let *BestPairs* grow to length *n*/6 - 1 where recall *n* is the number of vertices in G. (Kernighan and Lin used the longer length *n* - 1.) (5) Finally, for *k* = 1, 2, 3, etc., consider the partial sums

$$\sum_{i=1}^{k} (\text{change-to-cut-size for i-th pair in BestPairs})$$

The most favorable such partial sum then identifies the *k* elements from A and *k* from B which are made to exchange sides in the improvement step.

3.6. BISECTING THE GRAPH U500.10

The graph named U500.10 is a test case devised by [Johnson *et al*., 1989]. It is a so-called *random geometric graph*, and the authors constructed it as follows. First, 500 points are generated, whose coordinates are random values in the real unit interval [0, 1]. (The 500 points are randomly situated in the unit square.) Then a distance value is calculated, with the property that: when all pairs of points within that distance of one another are connected by an edge, then the expected (that is, average) degree of a vertex is 10.

Bui and Moon have used this same test case. In particular we have in mind [Bui and Moon, 1996]. In that paper the authors provide a detailed comparison of several algorithms, some of their own devising and some from earlier researchers; the algorithms are tested on a multitude of graphs, including U500.10. It is fair to say that the overall best performing algorithm from this paper is the authors' genetic algorithm BFS-GBA (Breadth-First Search Graph-Bisection Algorithm). Another compared algorithm is simulated annealing as practiced in [Johnson *et al*., 1989], denoted SA.

We tested our graph-bisecting form of LSDGA on the graph U500.10. The graph's vertex and edge sets are available on-line at *dimacs.rutgers.edu/pub/dsj/partition*. This graph has 2355 edges, so acceptable fitness is 2354. In Table 5 we summarize how our own algorithm LSDGA measured up against BFS-GBA and SA. To date no one knows the least possible cut-size for U500.10; the least one known is 26. The entries of Table 5 give the least and the average cut-sizes that surfaced over trials. The column for LSDGA comes from our own experiments; the other two columns are copied from [Bui and Moon, 1996]. Our own algorithm found the same least cut-size (26) that the other two algorithms did. Also, on four of our trials our algorithm found a cut-size of 29, which is very close to the least known cut-size. Our algorithm's average cut-size (44.77) falls in between those of the other two algorithms.

Table 5: Bisecting Graph U500.10

Omitted entries in the columns occur when the values are unknown, inappropriate, or are incomparable to LSDGA.

	SA	BFS-GBA	LSDGA
Number of trials	-	-	20
Population size	-	-	32
Max generations	-	-	500
Avg final gen. num	-	-	500
Best cut-size	26	26	26
Avg cut-size	65.8	32.68	44.77

Further comparisons between these three algorithms are difficult to make. The time-costs of SA and BFS-GBA are compared by Bui and Moon, but the costs are given in terms of CPU seconds on particular processors. Algorithm BFS-GBA is a genetic algorithm, but it is a steady-state algorithm whereas ours is generational. Also, the stopping condition used is entirely different; that algorithm stops evolving "when 80% of the population is occupied by solutions with the same quality" (p. 846). Finally, we remain unclear about certain terminology in the paper.

3.7. BISECTING CATERPILLAR GRAPH CAT352

Figure 1 suggests the structure of a so-called *caterpillar graph*. A caterpillar graph is made up of identical star-like clusters, with the star centers connected one to another in

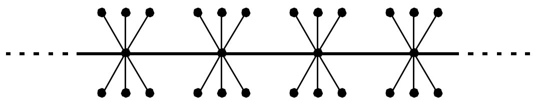

Figure 1: A Caterpillar Graph Segment

a linear sequence. [Bui and Moon, 1996] tell us that caterpillar graphs are especially difficult for certain graph bisection algorithms, such as that of [Kernighan and Lin, 1970], and simulated annealing as practiced in [Johnson *et al*., 1989]. Bui and Moon experimented with several sizes of caterpillar graph. The one named cat352 is made up of stars of 7 vertices, just like in Figure 1. In cat352 there are 50 such stars (giving 350 vertices), plus (we presume, anyway) two terminating vertices of degree 1 at the far left and far right. Plainly the minimal cut-size for graph cat352 is a mere 1, and it corresponds to splitting the graph between the two central stars.

We tested our algorithm LSDGA on graph cat352. The number of edges is 351, so our choice of maximum fitness is one less than that, or 350. This is the fitness that is achievable by the optimal bisection. Our stopping condition on each of 20 trials was to either unearth the optimal bisection or stop after 500 generations. On 4 of our 20 trials the optimal bisection was discovered. On the other 16 trials the best bisections we found had cut-sizes of 3 (6 times), 5 (4 times), 7 (3 times), and 9 (3 times). Table 6 summarizes the results, with comparisons to algorithm BFS-GBA of [Bui and Moon, 1996]. The table does not offer a comparison to [Johnson *et al.*, 1989] as they did not use caterpillar graphs. Our best cut-size is the equal of Bui and Moon, though their average cut-size is better than ours.

Table 6: Bisecting Graph cat352

Omitted entries in column two occur when the values are unknown, inappropriate, or are incomparable to LSDGA.

	BFS-GBA	LSDGA
Number of trials	-	20
Population size	-	32
Max generations	-	500
Avg final gen. num	-	431.95
Best cut-size	1	1
Avg cut-size	2.25	4.5

4. CONCLUSIONS

We have presented a new representation and crossover algorithm for genetic algorithms. Bits are not linked together at all. On each generation, the current population is used to calculate a pseudo-distance between bits. Bits which are close under the pseudo-distance are ones which appear to belong to a same building block. Under crossover, nearby bits get copied in clumps into the children. Thus our new linkless self-distancing approach does directly what other approaches to learning linkage and preserving building blocks have only hoped to do indirectly. Experiments were performed on six problems. These included simple problems, problems with numerous equally fit optima, and very challenging problems in graph bisection. Our new approach worked very successfully on these problems, equaling and often outperforming other more familiar approaches.

5. FUTURE WORK

Our approach is brand new. There are many questions that suggest themselves, which we have not yet had time to pursue. Are there better choices for our system parameters than the ones we have cited? Is there a better measure of fitness concentration than the entropy calculation we have used? Is there a Schema Theorem for our approach? In what ways can premature convergence occur, and how can it be combatted? Can careful bookkeeping allow us to adapt the approach to steady-state GA's? The approach needs to be extended to the case that the granularity of a gene is bigger than a single bit. In that vein, there would be many more than 4 ways that a pair of genes can assume a pair of values. Will our entropy calculation for fitness concentration, properly adapted, still succeed? In short, there are many issues inviting exploration.

References

Bui, T. N., and C. Jones (1992). "Finding Good Approximate Vertex and Edge Partitions is NP-Hard," *Information Processing Letters*, vol. 42, pp. 153-159.

Bui, T, N., and Moon, B.-R. (1993). "Hyperplane Synthesis for Genetic Algorithms," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 102-109. Morgan Kaufmann Publishing, San Mateo, CA.

Bui, T. N., and B.-R. Moon (1995). "On Multi-Dimensional Encoding/Crossover," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 49-56. Morgan Kaufmann Publishing, San Francisco, CA.

Bui, T. N., and B.-R. Moon (1996). "Genetic Algorithm and Graph Partitioning", *IEEE Transaction on Computers*, vol. 45, no. 7, pp. 841-855.

Cohoon, J. P., W. N. Martin, and D. S. Richards (1991). "A Multi-Population Genetic Algorithm for Solving the k-Partition Problem on Hypercubes," *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 244-248.

Goldberg, D., and Lingle, R. (1985). "Alleles, loci, and the traveling salesman problem", in *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pp. 154-159.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley Publishing.

Goldberg, D. E., Korb, B., and Deb, K. (1989). "Messy Genetic Algorithms: Motivation, Analysis, and First Results," in *Complex Systems*, vol. 3, pp. 493-530.

Greene, W. A. (2000). "A Non-Linear Schema Theorem for Genetic Algorithms," in the *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO 2000), pp. 189-194.

Greene, W. A. (2001). "Non-Linear Bit Arrangements in genetic Algorithms," in *2001 Genetic and Evolutionary Computation Conference Late-Breaking Papers*, pp. 138-144.

Harik, G. (1999). *Linkage Learning via Probabilistic Modeling in the ECGA*. IlliGAL Report No. 99010; Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana. IL.

Holland, John (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

Johnson, D. S., C. Aragon, L. McGeoch, and C. Schevon (1989). "Optimization by Simulated Annealing: an Experimental Evaluation, Part 1: Graph Partitioning", *Operations Research*, vol. 37, pp. 865-892.

Kernighan, B., and S. Lin (1970). "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell Systems Technical Journal*, vol. 49, pp. 291-307.

Laszewski, G. (1991). "Intelligent Structural Operators for the k-Way Graph Partitioning Problem," *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 45-52.

Muehlenbein, H., and Paass, G. (1996). "From Recombination of Genes to the Estimation of Distributions, I: Binary Parameters," in *Parallel Problem Solving from Nature IV*, pp. 178-187. Springer-Verlag, Berlin.

Pelikan, M., Goldberg, D. E., and Cantu-Pas, E. (1998). *Linkage Problem, Distribution Estimation, and Bayesian Networks*. IlliGAL Report No. 98013; Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana. IL.

Sehitoglu, O. T., and G. Ucoluk (2001). "A Building Block Favoring Reordering Method for Gene Positions in Genetic Algorithms," in the *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO 2001), pp. 571-575.

Syswerda, G. (1989). "Uniform Crossover in Genetic Algorithms", in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 2-9. Morgan Kaufmann Publishing, San Mateo, CA.