

---

# Expected Allele Coverage and the Role of Mutation in Genetic Algorithms

---

**David M. Tate**

Department of Industrial Engineering  
University of Pittsburgh  
Pittsburgh, PA 15261

**Alice E. Smith**

Department of Industrial Engineering  
University of Pittsburgh  
Pittsburgh, PA 15261

## Abstract

It is part of the traditional lore of genetic algorithms that low mutation rates lead to efficient search of the solution space, while high mutation rates result in diffusion of search effort and premature extinction of favorable schemata in the population. We argue that the optimal mutation rate depends strongly on the choice of encoding, and that problems requiring non-binary encodings may benefit from mutation rates much higher than those generally used with binary encodings. We introduce the notion of the expected allele coverage of a population, and discuss its role in guiding the choice of mutation rate and population size.

## 1 INTRODUCTION

Most genetic algorithm research to date has used mutation as a mechanism for recovering desirable genes that have been accidentally deleted from the population. In particular, most authors explicitly reject the use of mutation as an exploratory tool, seeking to identify new desirable genetic structures. This conservative strategy is based primarily on empirical findings using simple string encodings over low cardinality alphabets. Nevertheless, this conservative strategy is often extended to the design of GAs with more complex encodings and genetic operators. In this paper, we attempt to identify the particular features of simple string encodings that make high mutation undesirable, and argue that these features are absent in many more complex GA implementations. We further suggest that, in many cases, high mutation rates are not only desirable, but necessary.

We first review the traditional characterization of the role of mutation, as formulated by Goldberg. We then show that this characterization is sensitive to the complexity of the encoding. We introduce the concept of **expected allele coverage** as a measure of the degree to which exploratory mutation is or is not necessary. We then argue that certain classes of optimization problems require encodings with low expected allele coverage, and thus that low mutation implementations are not always ideal. Finally, we present a illustrative example of a hard combinatorial problem for which high mutation rates gave considerably better results than low mutation rates.

## 2 THE ROLE OF MUTATION

Goldberg, speaking of binary-string encodings in particular, characterizes the usefulness of mutation as follows (Goldberg 1989, p. 14):

“...[M]utation plays a decidedly secondary role in the operation of genetic algorithms. Mutation is needed because, even though reproduction and crossover effectively search and recombine extant notions, occasionally they may become overzealous and lose some potentially useful genetic material [...]. In artificial genetic systems, the mutation operator protects against such an irrecoverable loss. [...] By itself, mutation is a random walk through the string space. When used sparingly with reproduction and crossover, it is an insurance policy against premature loss of important notions. [...] We note that the frequency of mutation to obtain good results in empirical genetic algorithm studies is on the order of one mutation per thousand bit (position) transfers.”

In the context of simple string encodings, this is an accurate characterization. We shall argue, however, that more complex encodings may require significantly higher mutation rates and not all optimization problems can be encoded effectively using simple strings.

We begin with two general comments on mutation. First, is mutation necessarily nothing more than a random walk through string space? In a sense, yes. A generational GA in which each generation replaces the previous population with a population composed entirely of mutations of randomly-chosen members of the old population will not, in general, improve in average fitness. However, the same is true of reproductive operators. The selection of parents based on their fitness, and the elimination of less-fit members of the old population, are what drive the improvement of the population in a GA. The difference between crossover operations and mutation operations can be small compared to the cumulative effects of selection and culling. Were we to select individuals for mutation based on their relative fitness in the old population, and construct an entire new population that way, we would see continuous improvement of the average fitness of the population. The notion of mutation as random walk is valid only when mutation is applied *uniformly* over the population, without regard to fitness. Selection pressures act on individuals without regard to whether their phenotypes are the result of breeding or mutation.

A second traditional view of mutation is that it is destructive and must be used sparingly. In a generational GA implementation, this is true. Since highly-fit individuals do not persist from one generation to the next, any process which introduces random noise into offspring will tend to destroy the very characteristics for which their parents were selected. However, in a steady-state GA, in which the majority of the population persists unchanged from one generation to the next, this is not necessarily a problem. The danger of eliminating desirable schemata is diminished even further if we allow the possibility that both a mutant encoding and the individual which gave rise to it might survive into future generations, as we did in (Tate and Smith under review). This allows the possibility of mutation as a local search mechanism in the neighborhood of highly-fit solutions, without the danger of eliminating the highly-fit solutions themselves in the process.

The idea that mutation is primarily useful for retrieving valuable schemata would imply that a mutation rate ought to be set so that the rate at which highly-fit schemata are accidentally deleted from the population is roughly equal to the rate at which mutations introduce

new desirable schemata. Inherent in this view is the assumption that there is no need to find highly-fit schemata that are not present in the initial population, or that cannot be generated from that population by repeated breeding operations. In other words, this view assumes that nearly all useful gene alleles are present in the initial population. We examine this assumption in detail in the next section.

### 3 EXPECTED ALLELE COVERAGE AS A FUNCTION OF ENCODING

In the quote discussed above, Goldberg assumed simple binary string encodings of length L. The order-one schemata, or possible alleles, for this encoding are the 2L possible single-bit specifications, such as "bit 23 is a zero" or "bit 18 is a one". Of these 2L possible alleles, exactly L will occur in every string. Given a random population of N encodings, the probability that a given allele is absent from the population is thus  $2^{-N}$ , and the expected number of uninstantiated alleles is  $L/2^{N-1}$ . The expected proportion of all possible alleles which occur in a random population of size N is thus  $1 - (1/2)^N$ . We term this the expected allele coverage for this encoding and population size. For binary encodings with simple crossover and mutation, even with a population as small as N=10 the expected allele coverage is greater than 99.9%. Thus, for binary string encodings, it is perfectly reasonable to assume that all useful alleles are present in a random initial population, and that we need only guard against their loss.

However, the high expected coverage of simple binary encodings does not generalize to higher complexity encoding structures. For example, suppose that instead of using binary strings, we encode the same solution space using equivalent octal strings. In this case, we require only about L/3 digits per string, but each position in a given string can take any one of 8 different values. The number of possible alleles is thus  $8L/3$ .

We can analyze the expected allele coverage for an encoding as follows. Let K denote the number of distinct symbols used in the encoding (i.e. the alphabet size), N the population size, and  $D = L/\log_2(K)$  the number of digits in each encoded string. There are thus KD possible alleles for this encoding. The probability that exactly m distinct symbols are used in the  $j^{\text{th}}$  digit position, over the whole population, is given (Feller 1957, p. 60) by Equation 1:

$$p_m(N, K) = \frac{1}{K^N} \cdot \frac{K!}{m!} \sum_{v=0}^{K-m-1} (-1)^v \cdot \frac{K-m}{K-v} \cdot \frac{K-m-v}{K}^N$$

The expected number of distinct symbols used in the  $j$ th digit position is thus Equation 2:

$$E_m(N, K) = \sum_{m=1}^{K-1} m \cdot p_m(N, K)$$

and, since all digit positions are disjoint with respect to alleles, the expected proportion of all possible alleles which will occur in the given random population is given by Equation 3:

$$E[ac] = \frac{D}{KD} \cdot \sum_{m=1}^{K-1} m \cdot p_m(N, K) = \frac{1}{K} \sum_{m=1}^{K-1} m \cdot p_m(n, K)$$

where  $ac$  is expected allele coverage.

Note that this expected allele coverage does not depend on  $D$ . Using this formula, we find that the expected allele coverage for a random population of  $N$  octal strings is somewhat lower than for binary strings. For  $N=10$ , we only expect to see 73.7% of all possible alleles in the initial population. To ensure an expected coverage of at least 99%, we would need a population size of  $N=35$ . For hexadecimal strings, 99% expected allele coverage would require a population of  $N=72$ . (See Table 1).

Table 1. Required Population to Achieve 99% Expected Allele Coverage.

Alphabet Size	Required Population Size	Expected Coverage(%)
2	7	99.22
4	17	99.25
8	35	99.07
16	72	99.04
32	146	99.03

Of course, it is not difficult to construct a nonrandom population of simple strings such that all possible alleles are instantiated (Holland 1975). The allele coverage of the initial population only becomes a problem for encodings in which there are positional dependencies. Consider the Traveling Salesman Problem (TSP) implementation which encodes tours as permutations. For a  $(D+1)$ -city problem, the encoding space consists of all permutations of the integers 1, 2, ...,  $D$ . There are  $D!$  such encodings, on an alphabet of size  $D$ . The possible alleles for this encoding space are the  $D^2$  possible specifications of a particular city at a particular point in the tour. Exactly  $D$  of these alleles are instantiated in any given permutation. The expected

allele coverage for a random initial population of  $N$  is given by evaluating Equation 3 when  $K=D$ , since expected coverage is additive over gene positions even when the allele values of different genes are not independent. Thus, permutation encodings lose expected coverage as a function of problem size, since they use what is in effect an encoding alphabet that grows linearly with problem size. Furthermore, since there are  $D^2$  possible alleles, but only  $D$  of them are instantiated in any individual, even a carefully constructed nonrandom population would require a population of at least  $D$  individuals to cover the possible allele set.

The binary adjacency matrix TSP encoding proposed by Whitley, Starkweather, and Shaner (Whitley et al. 1991) can also be analyzed. They encode tours as binary strings whose individual genes represent the possible unordered adjacencies among cities in the tour. For a  $(D+1)$ -city problem, this requires  $D(D+1)/2$  bits per string, of which exactly  $D+1$  entries are nonzeros in any given string. This is equivalent (for the purpose of this analysis) to an ordinary binary encoding in which each bit position has probability  $2/D$  of being a "1". The probability that a "1" never occurs in a given bit position, in a random population of size  $N$ , is thus  $(2/D)^N$ , and the probability that a "0" never occurs in that position is  $((D-2)/D)^N$ . The overall expected allele coverage in a random population of  $N$  is given by Equation 4:

$$E[ac] = 1 - [2^N + (D-2)^N] / 2D^N$$

The smallest nonrandom population which covers the possible allele set has  $D/2$  members, corresponding to  $D/2$  edge-disjoint Hamiltonian cycles in the TSP graph.

Figure 1 shows the population size required to achieve at least 90% expected allele coverage for the various encodings described in this section. For simple enumerative encodings, such as binary or octal integers, this number does not grow with problem size, but is an approximately linear function of the cardinality of the alphabet used. For the Whitley et al. encoding, the required population increases at a roughly one-for-one rate as a function of problem size. For permutation encodings, the required population increases at a roughly two-for-one rate. Clearly, the choice of encoding in a GA implementation can have a significant effect on the expected allele coverage of a randomly-generated population, and thus on the amount of exploratory mutation that may be desirable during that population's evolution. In the next section we discuss why higher complexity encodings may be advantageous despite their lower expected allele coverage.

## 4 THE CASE FOR NONBINARY ENCODINGS

Goldberg (Goldberg 1991, Theorem 2) notes that low-cardinality encodings maximize the number of distinct schemata per bit, and that the number of schemata that a given allele is an instance of is strictly decreasing in the cardinality of the encoding alphabet (for a fixed information content). However, there exist classes of optimization problems for which no attractive enumerative encoding exists.

In particular, when applying GA techniques to combinatorial optimization, the size of the solution space can be a problem. For any combinatorial problem class, the most compact binary encoding is a one-to-one correspondence between binary integers and feasible solutions: essentially an enumeration of the feasible set. For example, for a TSP on  $D+1$  cities, the feasible tours can be mapped one-to-one onto the permutations of the numbers  $1, 2, \dots, D$  and the integers  $1, 2, \dots, D!$ . To implement a binary encoding on this search space would require  $\log_2(D!)$  bits for each string. (In fact, if implemented as an array of boolean variables in Pascal or C, it would require  $\log_2(D!)$  bytes per string.) As shown in Figure 2, this number grows by  $D \log_2(D)$ , so that a 50-city TSP requires more than 200 elements per array. In contrast, a permutation-based encoding of a 50-city problem might involve operations on an array of 49 1-byte integers. This encoding involves only 1/4 as many genes to be manipulated. The adjacency matrix encoding of Whitley et al., on the other hand, grows quadratically in required array length as a function of problem size, needing 1225 elements per array for a 50-city problem.

The explosion of the feasible set is even worse for more complex combinatorial problems. Figure 3 illustrates the required number of binary array elements for several different combinatorial search spaces, under an enumerative encoding scheme. Partitioned permutations correspond to the “flexible bay” solution set used by Tong (Tong 1991), and later by (Tate and Smith under review) for unequal-area facility layout. Slicing trees are a standard encoding for VLSI component placement and 2-dimensional stock-cutting (Cohon 1987, Cohoon et al. 1991, Tam 1992, Wong and Liu 1986). Arbitrary subgraphs arise in certain connectivity and multi-assignment problems, when the solution space consists of the set of all subgraphs of the complete graph on  $D$  nodes. There are two such subgraphs, so that a complete binary encoding would require bits per string.

Another reason besides encoding size why simple binary encodings may not be desirable for a given GA implementation has to do with the relationship between genotype and phenotype for a given encoding. For example, suppose we have a combinatorial problem for which there are  $M$  feasible solutions. From the point of view of allele coverage and encoding size, any mapping of these  $M$  solutions onto binary integers of length  $\log_2(M)$  is equally good. On the other hand, if we impose a purely random numbering on the set of feasible solutions, and encode these solutions by their corresponding number (in binary), we get a GA in which there is no relationship between individual schemata in encoding-space and structural features in solution-space. As a result, all schemata of the same order will tend to have roughly identical average fitness, and the evolutionary mechanism of the GA cannot distinguish among them. This GA will have a very slow improvement rate.

To summarize, it may be that there is no attractive simple string encoding for a given combinatorial problem. Even if a one-to-one mapping from the solution set is found, it may be too computationally burdensome to encode and decode individual solutions, or the encoding may not preserve enough structural information from solution space to guarantee good convergence. As a result, we may be forced to use complex, position-dependent encodings for certain optimization problems.

## 5 COPING WITH LOW COVERAGE

In order to implement a higher complexity encoding successfully, it is important to recognize which aspects of the GA will be affected by the choice of encoding. In particular, if an encoding is chosen which has poor expected allele coverage, the implementation population size and/or mutation rate must be adjusted to compensate. Increasing the population size is a mixed blessing, however. Complex encodings can require a huge population size to achieve the desired expected coverage level, while computational overhead per generation increases dramatically. In addition, increasing the population size increases the population’s “fitness inertia”, and thus slows the overall population improvement rate (Goldberg 1989, p. 111).

The alternative is to increase the relative mutation rate. To this end, it is both necessary and desirable to separate mutation and breeding into independent, parallel operations on the current population. Much of the criticism of high-mutation GA implementations is

based on the assumption that mutation can only occur as a noise process within breeding, and that increased mutation rates thus necessarily destroy the ability of the GA to perform conservative recombination of extant schemata. Thus, Goldberg tells us (Goldberg 1989, p. 111) that “As the mutation rate is increased to a value  $p_m = 0.1$ , the off-line performance...more and more starts to resemble a random search. (Of course, a mutation probability of  $p_m = 0.5$  is a random search, regardless of [crossover rate and population size]).” This is true only if mutation is constrained to interfere with breeding. By allowing the next generation to contain both unmutated offspring of parents from the current generation and mutated clones of members of the current generation, we can simultaneously achieve the conservative recombination essential to GA convergence and the exploration of encoding-space necessary to overcome low allele coverage in the initial population.

We have demonstrated the efficacy of this strategy on a classical constrained combinatorial optimization problem, the Unequal Area Facility Layout problem. UAFL is defined as follows: a rectangular region  $R$ , with dimensions  $L$  by  $W$ , is to be partitioned into  $n$  connected subregions (or “departments”) with specified areas  $a_i$ ,  $i = 1, \dots, n$ . The individual department areas satisfy  $\sum a_i = LW$ . The cost of a given partition (or “layout”) is a weighted sum of the pairwise distances between department centroids in the layout. The problem is NP-complete (Garey and Johnson 1979), and there are uncountably many feasible layouts for a given problem instance.

To date, solution methods for UAFL have concentrated on some well-defined subset of the solution space. For example, by dividing the region  $R$  into small rectangular “unit regions”, and attempting to assign each department to some collection of contiguous unit regions, the problem can be transformed into a large quadratic assignment problem (QAP) (Tate and Smith, forthcoming). We applied GA techniques to UAFL, using an encoding based on the idea of flexible bays. A flexible bay layout is obtained by first partitioning the region  $R$  into  $k$  “bays”,  $1 \leq k \leq n$ , and then assigning an ordered sequence of departments to each bay. The width of a given bay is determined by the total area of the departments assigned to it, and by the dimensions of the region  $R$ .

There are thus  $2^{n \cdot n!}$  possible flexible bay layouts for a given problem, of which 1/4 are not simply rotations of other flexible bay layouts. As shown in Figure 3, the number of bits per string required to

implement an enumerative encoding of the space of flexible bay layouts grows rapidly. Furthermore, there is no convenient lexicographic ordering of flexible bay layouts which seems likely to preserve solution structures in low-order schemata. As a result, we chose to encode solutions in a complex 2-chromosome structure, with one chromosome containing sequence information and the other containing bay allocation information. The expected allele coverage for this encoding is considerably worse even than that of the permutation-based TSP.

The detailed results of this investigation are reported in (Tate and Smith under review). The pertinent results to this paper are that the best results in terms of convergence speed and solution quality were obtained for small populations (e.g.  $N=10$ ) and extremely high mutation rates (e.g. a 2:1 ratio of mutants to offspring), using parallel, independent breeding and mutation. This is an example of overcoming the low expected allele coverage of a high complexity encoding while maintaining a small population size.

## 6 CONCLUSIONS

For simple string encoded GAs, low mutation rates are sufficient. This is a direct result of the high expected allele coverage exhibited by random populations of binary strings. More complex encodings exhibit lower expected coverage. This can be adjusted for, to some extent, using nonrandom initial populations, but a more efficient way of coping with low coverage is to use higher mutation rates.

In order to achieve these high mutation rates without degenerating into a simple random search, it is beneficial to segregate mutation and breeding into separate, parallel processes. The relative frequency of breeding and mutation can be controlled exactly in this way, allowing the researcher to search “randomly” for highly-fit but previously unseen allele values while simultaneously recombining extant alleles through the breeding of unmutated offspring. It is also useful, in such cases, to use a steady-state GA implementation in which highly-fit solutions can persist from one generation to the next, even while generating mutants, and poor solutions are culled from the population. The authors report highly encouraging results from their preliminary investigations in the Unequal Area Facility Layout problem.

## References

Cohon, J. P., 1987, Genetic placement, *IEEE Transactions on CAD* 6, 956-964.

Cohon, J. P., S. U. Hegde, W. N. Martin and D. S. Richards, 1991, Distributed genetic algorithms for the floorplan design problem, *IEEE Transactions on CAD* 10, 483-492.

Feller, W., 1957, *An Introduction to Probability Theory and its Applications*. John Wiley and Sons, New York.

Garey, M. R. and D. S. Johnson, 1979, *Computers and Intractability*. Freeman, San Francisco.

Goldberg, D. E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading.

Goldberg, D. E., 1991, Real-coded genetic algorithms, virtual alphabets, and blocking, *Complex Systems* 5, 139-167.

Holland, J., 1975, *Adaptation in Natural and Artificial Systems*. U. of Michigan Press, Ann Arbor.

Tam, K. Y., 1992, A simulated annealing algorithm for allocating space to manufacturing cells, *International Journal of Production Research* 30, 63-87

Tate, D. M. and A. E. Smith, forthcoming, A genetic approach to the quadratic assignment problem, *Computers and Operations Research*.

Tate, D. M. and A. E. Smith., under review, Unequal area facility layout using genetic search, *IIE Transactions*.

Tong, X., 1991, *SECOT: A Sequential Construction Technique for Facility Design*. Doctoral dissertation, University of Pittsburgh.

Whitley, D., T. Starkweather and D. Shaner, 1991, The traveling salesman and sequence scheduling: quality solutions using genetic edge recombination. In *Handbook of Genetic Algorithms*, ed. L. Davis. Van Nostrand Reinhold, New York.

Wong, D. F. and C. L. Liu, 1986, A new algorithm for floorplan design, *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, IEEE Computer Society Press.

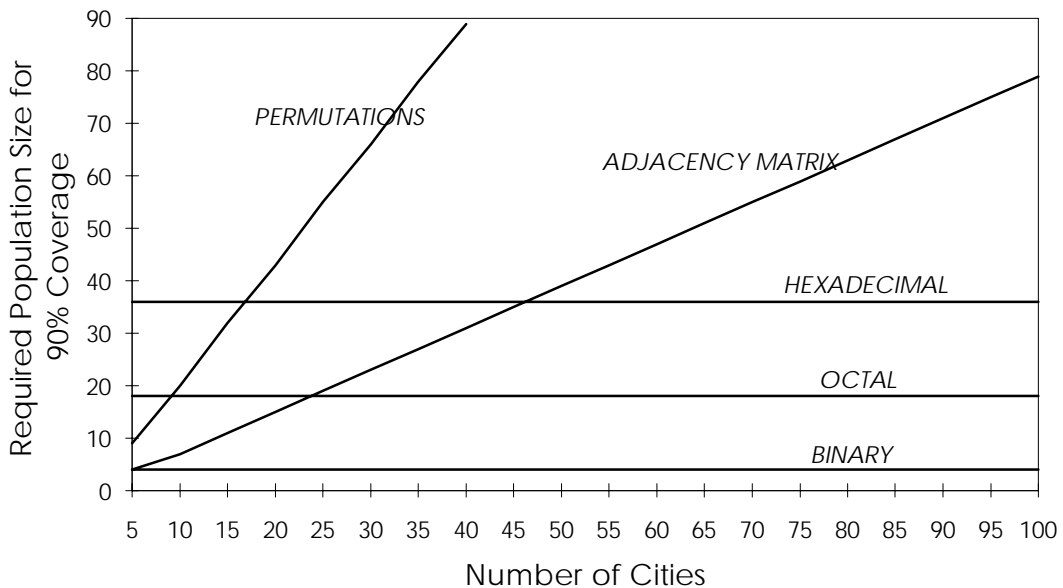


Figure 1. Required Population Size Required for 90% Expected Allele Coverage for Different Complexity Encodings.

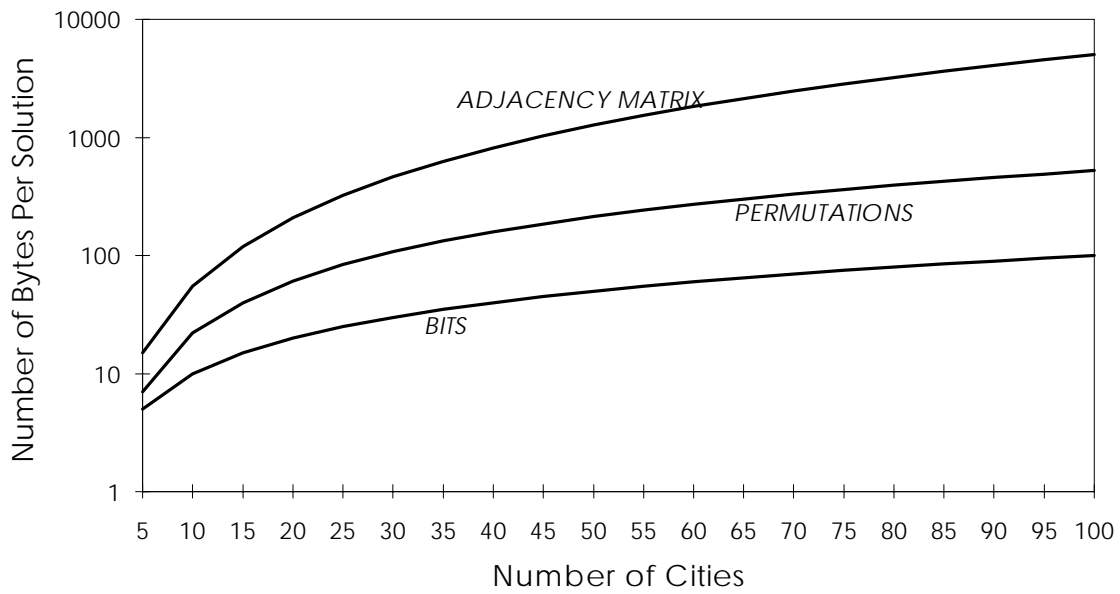


Figure 2. Number of Bytes per Solution Using Different Complexity Encodings (Log Scale).

Figure 3. Required Bits per Solution as a Function of the Problem Size for Different Encodings.