The Royal Road Not Taken: A Re-Examination of the Reasons for GA Failure on R1

Brian Howard[1] and John Sheppard[1,2]

[1] The Johns Hopkins University, 3400 N. Charles Street, Baltimore, MD 21218
itsbehoward@hotmail.com
[2] ARINC Engineering Services, LLC, 2551 Riva Road, Annapolis, MD 21401
jsheppar@arinc.com and jsheppa2@jhu.edu

Abstract. Previous work investigating the performance of genetic algorithms (GAs) has attempted to develop a set of fitness landscapes, called "Royal Roads" functions, which should be ideally suited for search with GAs. Surprisingly, many studies have shown that genetic algorithms actually perform worse than random mutation hill-climbing on these landscapes, and several different explanations have been offered to account for these observations. Using a detailed stochastic model of genetic search on R1, we attempt to determine a lower bound for the required number of function evaluations, and then use it to evaluate the performance of an actual genetic algorithm on R1.

1 Introduction

Many theoretical frameworks for understanding the overall performance of genetic algorithms assume implicitly that the observed evolutionary dynamics of finite populations should follow those of a hypothetical infinite population. For example, the "implicit parallelism" predicted by the schema theorem assumes that schema sampling is sufficiently unbiased to provide a reasonably accurate estimate of each observed schema's average fitness. However, in reality, sampling anomalies can profoundly influence the way that a population evolves in the short term, especially when the size of that population is small.

For example, Mitchell, *et al.* present an apparent conundrum: on a fitness landscape designed specifically to be amenable for search with a genetic algorithm, the simpler Random Mutation Hill-Climbing (RMHC) approach defeats the genetic algorithm by an order of magnitude [4]. Specifically, their paper demonstrated that the genetic algorithm requires approximately 61,000 fitness evaluations to find the optimal solution, versus only about 6,200 for the hill-climber. This observation has been widely interpreted as evidence against the relevance of the building block hypothesis. These tests used the Royal Roads fitness function R1, which is defined as follows and displayed graphically in Figure 1:

$$\mathrm{R1}(x) = \sum_i c_i \sigma_i(x), \quad \text{where } \sigma_i(x) = \begin{cases} 1 & \text{if } x \in s_i \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

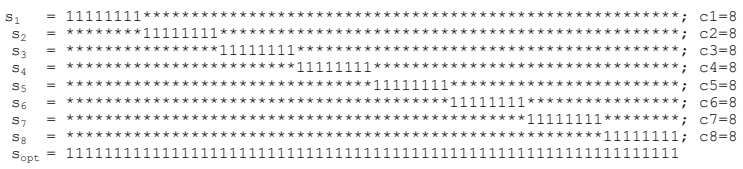

Fig. 1. Royal Roads landscape R1

Mitchell, *et al.* attributed the poor performance of the genetic algorithm primarily to the phenomenon of genetic hitchhiking: once an instance of a highly fit schema is discovered, its high fitness allows that schema to spread quickly throughout the population, with zeros in other positions in the string "hitchhiking" along with the building blocks. To reduce the influence of hitchhiking on the GA's performance, they introduced "introns" between the adjacent schemata. However, their experiments showed that, in the case of R1, introns do not improve the GA's performance significantly, highlighting the fact that the mechanism of hitchhiking in genetic algorithms is not clearly understood.

In the following section we will attempt to create a model that defines a reasonable lower bound on the expected performance of a genetic algorithm on landscape R1. We will then use this model to investigate the hypothesis that the poor performance of the genetic algorithm can be partially explained by hitchhiking, but that sampling error and population cost also contribute substantially.

2 A Stochastic Model of Genetic Search on Landscape R1

In the original Royal Roads work [4]**,** Mitchell *et al.*, introduced the "Ideal Genetic Algorithm", or IGA, to determine an approximate lower bound on the number of fitness evaluations required by a genetic algorithm searching landscape R1. According to this model, the researchers assert that an ideal genetic algorithm should be able to find the optimum on R1 in approximately 696 evaluations on average. However, the following points illustrate the need for a more realistic lower bound:

1. The IGA algorithm requires perfect knowledge of all eight schemata, *a priori*. Normally, fitness functions are designed to compute the relative fitness of an entire string as a whole, without any facility to explicitly recognize partial solutions.
2. The algorithm does not use a population and, thus, the cost of maintaining a population is not considered in the model.
3. In general, the mutation and crossover mechanisms proposed do not provide a realistic analogy to what might actually be expected from a real genetic algorithm.

Perhaps the most "exact" method of modeling search using a genetic algorithm is described in Vose [9,10]. Here the genetic algorithm is described as a specific instance of the more general Random Heuristic Search process (RHS). Genetic algorithms, simulated annealing [2], PBIL [1], and hill-climbing can all be modeled under this framework. In general, RHS begins with a randomly initialized population, repre

sented as a frequency vector that describes the relative proportions of each possible member of the search space. At each step, the current population is passed to a heuristic function, $\mathcal{G}$, that returns a multinomial distribution from which the next generation can be generated via stochastic sampling. The transition from one generation to the next can be viewed equivalently as the application of a transition function, τ. Although the function τ is an induced mapping, the process is Markovian because it produces each new population contingent only on the state of the previous population.

Vose describes a precise model of a heuristic function that can be used to model a binary encoding of the simple genetic algorithm [10]. Other authors have extended this model to cover, for example, general cardinality representations [3] and alternate forms of selection [8]. In theory, a variant of Vose's simple genetic algorithm could be employed to model a GA's search of R1; however, in practice, the state space for such a model would require a transition matrix with 2^{64} states, making this approach intractable.

In an attempt to construct a more manageable model of the R1 landscape, Suzuki and Sawai reduce the search space by describing each set of eight bits in terms of the number of ones present, without tracking the exact positions of those ones within the schemata [6]. This reduces the search space somewhat, but still required Suzuki and Sawai to limit their analysis to smaller versions of Royal Roads functions having only a fraction of the total number of bits in R1. Nevertheless, using such a model, they were able to show that crossover accelerates GA search when compared to an equivalent GA without crossover.

Another attempt to model the search of R1 with a genetic algorithm was introduced by van Nimwegen, Crutchfield, and Mitchell [7]. This model groups together all strings that have the same fitness value. Using this model, van Nimwegen *et al.* were primarily interested in exploring the high-level dynamics of the search process, in particular, the occurrence of punctuated equilibria and fitness epochs. In their model, they excluded crossover from the analysis, acknowledging that such an approach may leave out many key details of search necessary to explore other problems.

In this paper, we introduce a new model of genetic search on R1. Using Vose's general RHS framework, our model groups population members by the schemata that they contain. Furthermore, when any one of the eight schemata is missing from a chromosome, our model assumes that the eight bits at the corresponding locations are true "don't cares," freshly sampling, at each generation, from the set of all binary strings of length eight. In this manner, our model engages in an "implicitly parallel search" and eliminates the possibility of bit-wise hitchhiking. By comparing the performance of such a model to the performance of an actual genetic algorithm, we should, therefore, be able to determine the approximate degree to which hitchhiking is actually a problem on R1. The following sections define this model more precisely.

2.1 Representation of the Population

In our model, each of the 256 possible combinations of the schemata s_1-s_8 is represented as a unique 8-bit string c, in which each bit, c_i, indicates the presence or ab

sence of schema s_i. A one in a particular position indicates that the corresponding schema is present, while a zero signifies that the schema is absent. For example, the bit string, 10010001, describes all strings matching the pattern:

```
11111111 ******* ******** 11111111 ******** ******** ******** 11111111
```

Throughout this paper we will refer to such 8-bit strings as *schema configurations,* or simply as *configurations*. Given this representation, a particular population can be characterized by a real vector $\vec{p}$ of length 256, which indicates the frequencies of each of the possible R1 schema configurations in that population. We will index these configurations with the integers 0–255, such that each index maps to the integer representation of the corresponding schema configuration, interpreted as a binary number. Hence, the configuration 00000000 is given index 0, while configuration 00001000 is given index 8. Using this indexing scheme, the i^{th} position in a particular population vector represents the frequency with which this schema configuration occurs in the underlying population. For example, the following population vector:

$$\vec{p} = [0.25 \quad 0.05 \quad \cdots \quad 0.01] \tag{2}$$

indicates that in 25% of the underlying population, none of the schemata s_1-s_8 are present (configuration 00000000). In 5% of the population, only schema s_8 is present (configuration 00000001), while in 1% of the population all of the schemata s_1-s_8 are present (configuration 11111111).

By grouping together all bit strings with the same arrangement of schemata, this representation reduces the state space from the set of 2^{64} 64-bit binary strings to the set of 2^8 8-bit binary strings. Furthermore, this model forces us to assume that when a schema is not present in a string, then nothing else is known about the eight bits in that schema's partition. If we build our model's operators such that the distribution of these unknown "don't care" bits is assumed to be uniform, then we have a model that implicitly removes the possibility of bit-level hitchhiking.

2.2 Mutation

If the population is represented by the vector, $\vec{p}$, then the mutation operation can be described as a 256 × 256 matrix, $\mathbf{M} = [m_{ij}]$, where each element m_{ij} represents the probability of transitioning from a population member with the schema configuration i to a population member with schema configuration j, in one generation, due to mutation. With this representation, the mutation operator can be applied to a population using matrix multiplication:

$$\vec{p}_m = \vec{p}\mathbf{M} \tag{3}$$

Given any particular eight-bit parent configuration, c, we can calculate the probability that mutation transforms this schema configuration into child configuration c' by considering each of the eight schemata independently. For each of the eight schemata, $s_i \in \{ s_1, \ldots, s_8 \}$, there are four distinct cases corresponding to whether the child string has the schema given the parent does/does not have that same schema.

The individual likelihoods of these four cases can be calculated with the application of basic probability theory, assuming that if a particular population member lacks schema s_i, then the actual bits within partition i are equally likely to be zeroes as they are ones. Starting with the child having the schema given the parent does not, we need to compute:

$$\Pr(c_i' = 1 \mid c_i = 0) \tag{4}$$

Equation (4) can be decomposed as follows. Let "$i \mapsto m$" denote the occurrence of at least one mutation in partition i.

$$\begin{aligned} \Pr(c_i' = 1 \mid c_i = 0) &= \Pr(c_i' = 1 \mid c_i = 0, i \mapsto m) \Pr(i \mapsto m \mid c_i = 0) \\ &\quad + \Pr(c_i' = 1 \mid c_i = 0, i \not\mapsto m) \Pr(i \not\mapsto m \mid c_i = 0) \\ &= \Pr(c_i' = 1 \mid c_i = 0, i \mapsto m) \Pr(i \mapsto m) \end{aligned} \tag{5}$$

Here, the probability of mutating partition i is independent of the state of the parent, so $\Pr(i \mapsto m \mid c_i = 0) = \Pr(i \mapsto m)$. Also, note that the second term in the sum drops out because $\Pr(c_i' = 1 \mid c_i = 0, i \not\mapsto m) = 0$. So, in other words, the probability of creating schema s_i in an offspring from a parent that lacks s_i is equal to the probability of creating this schema from such a parent given that at least one mutation occurs in this schema partition, times the probability that such a mutation occurs.

Let μ be the bitwise mutation probability. If the parent string lacks a particular schema, s_i, and the child gains this schema after applying the mutation operator, then we know that a mutation must have occurred somewhere within the eight bit partition attributed to this schema. The probability that a mutation occurs somewhere within a particular 8-bit schema is:

$$\Pr(i \mapsto m) = 1 - (1 - \mu)^8 \tag{6}$$

Assuming a uniform probability distribution for bits in the parent that are not part of a previously discovered schema, the probability that any one of these bits is a one is 0.5, and the probability that any one of these bits is a zero is also 0.5[†]. Let μ' be the bitwise mutation rate within a schema, *given that at least one such mutation has occurred.* Then the first half of the right side of equation (5) reduces to the following:

$$\Pr(c_i' = 1 \mid c_i = 0, i \mapsto m) = (0.5(1 - \mu') + 0.5(\mu'))^8 = (0.5^8) \tag{7}$$

Substituting the above result, along with the probability that a mutation occurs in a given schema, back into equation (5), we get the following result:

$$\begin{aligned} \Pr(c_i' = 1 \mid c_i = 0) &= \Pr(c_i' = 1 \mid c_i = 0, i \mapsto m) \Pr(i \mapsto m) \\ &= (0.5)^8 (1 - (1 - \mu)^8) \end{aligned} \tag{8}$$

Note further that

[†] Actually, the probability that a bit is 0 is slightly greater than 0.5, since we *know* that the parent's bit sequence is not 11111111; that is, at least one of the bits must be zero.

$$\Pr(c_i' = 0 \mid c_i = 0) = 1 - \Pr(c_i' = 1 \mid c_i = 0) \quad (9)$$

The remaining cases are straightforward. If the parent contains a particular schema, s_i, then the probability that the child does not contain this schema after mutation is simply the probability that a mutation occurs anywhere within this schema partition:

$$\Pr(c_i' = 0 \mid c_i = 1) = \Pr(i \mapsto m) \quad (10)$$

Likewise,

$$\Pr(c_i' = 1 \mid c_i = 1) = 1 - \Pr(i \mapsto m) \quad (11)$$

Given these four cases, it is now possible, for any particular parent configuration c, and child configuration c' to calculate the probability that c' is the offspring of c after applying the mutation matrix, **M**, where:

$$m_{ij} = \prod_{k=1}^{8} \Pr(c_k' \mid c_k) \quad (12)$$

2.3 Crossover

Crossover is a bit more complex because we need an operator that takes into account the interaction between *two* parents to produce each offspring. For our model, we represent one-point crossover with a 256 × 256 × 256 hypermatrix **C**, such that when this hypermatrix is divided into planes along its third dimension, each plane with index $\breve{c}$ represents a transition matrix, where each entry $m_{\hat{c}c'}$ describes the probability that a parent with a particular schema configuration, $\hat{c}$, produces a child c', given that the second parent is $\breve{c}$. At each generation, **C** can be applied to the next generation as follows. Let $\Pr(\breve{c})$ be the probability of selecting configuration $\breve{c}$ as the second parent in crossover, given the distribution of configurations resulting from mutation, $\vec{p}_m$, and $\mathbf{C}_{\breve{c}}$ be the slice of hypermatrix **C** corresponding to the second parent $\breve{c}$. Then, the new distribution of configurations, $\vec{p}_{m\chi}$, after mutation and crossover is calculated as follows:

$$\vec{p}_{m\chi} = \sum_{\breve{c}} \Pr(\breve{c})[\vec{p}_m \mathbf{C}_{\breve{c}}] \quad (13)$$

Next we will describe a procedure for building the crossover hypermatrix, **C.** Note that crossover can occur in the middle of a particular schema partition or between schema partitions, resulting in 15 distinct crossover points. Assuming that crossover is equally likely to occur at any of the 64 bit positions in the R1 chromosome, the probability that crossover occurs within a particular partition, given that crossover occurs, is 7/63, while the probability of crossover occurring between two particular adjacent partitions is 1/63.

Given parent strings, $\hat{c}$ and $\breve{c}$, a crossover event at a particular point can be described with a bit mask. We will describe building the crossover hypermatrix, **C** with an example. Suppose $\hat{c}$ = 26, $\breve{c}$ = 52, and the crossover point is 10. Using the in

dexing scheme described in Section 3.1, $\hat{c}$ = 26 maps to chromosome 00011010 and $\check{c}$ = 52 maps to chromosome 00110100. With crossover point 10, the masks are **A** = 11111000 and **B** = 00000111 respectively. The children are determined by applying the following logical operators:

$$c' = (\hat{c} \wedge \mathbf{A}) \vee (\check{c} \wedge \mathbf{B})$$
$$c'' = (\check{c} \wedge \mathbf{A}) \vee (\hat{c} \wedge \mathbf{B})$$

where **A** is Mask A and **B** is Mask B. Given this information, there are two possible children, $c' = 00011100$ and $c'' = 00110010$.

Given this, we can determine the probability either child will result via crossover, given parents $\hat{c}$ and $\check{c}$. Let χ be the probability crossover occurs. Let ψ_i indicate crossover point i was selected (e.g., ψ_{10} corresponds to crossover point 10). Finally, let n = the number of children generated by crossover. Then

$$\Pr(c') = \Pr(c'') = \left(\frac{1}{n}\right)\Pr(\psi_i)\chi \tag{14}$$

The procedure described above works perfectly when crossover occurs between schemata. When crossover occurs within a partition, the situation is a bit more complicated. For example, at crossover point 11, **A** = 11111000 and **B** = 00000011. Because bit six in both of the masks is a zero, simply applying the masks in the logical operations would prevent either child from ever having schema six. In reality, the offspring may indeed have the schema, either because both parents had the schema, or because the schema is created in the child after mixing bits from both parents. Thus in the event a schema partition is disrupted by crossover, we need to explicitly calculate the probability that the child inherits the disrupted schema and modify our list of children and their probabilities accordingly. There are 3 distinct possibilities.

If both parents have the schema, then the probability that the child will have the schema after crossover, even if it occurs in the middle of this schema, is 1. However, because the crossover masks turn off the schema bit in the offspring, we must modify the children produced by the procedure outlined above by turning this bit back on. The probabilities are calculated as shown in equation (14), using the appropriate $\Pr(\psi_i)$.

If exactly one of the parents has the schema that is disrupted, then there are actually *four* possible children. First, it is possible that both children lose the schema. At the same time, based on the actual configuration of the parents, it is possible that either child could "regain" the schema. Thus, the probability that the schema in question is "preserved" in the offspring is calculated as follows:

$$\Pr(s_i \mid \psi_{2i-1}) = \frac{1}{7}\sum_{j=1}^{7}(0.5)^{8-j}(1-(0.5)^j) \tag{15}$$

where i is the index of the schema partition considered, and the sum is taken over the seven possible crossover positions within the schema partition. That is, if that crossover occurs in some schema, it is equally likely to occur in each of seven distinct places, between each of the eight bits comprising the partition.

Thus the probability of producing children c''' and c'''', in which schema *i* is preserved after crossover is calculated as follows:

$$\Pr(c''') = \Pr(c'''') = \left(\frac{1}{4}\right)\Pr(s_i \mid \psi_{2i-1})\Pr(\psi_i)\chi \tag{16}$$

While for children c' and c'', in which schema *i* is lost, it is:

$$\Pr(c') = \Pr(c'') = 1 - \Pr(c''') \tag{17}$$

If neither parent has the schema that is disrupted, it is still possible that the schema could be created in the child simply due to the chance mixing of bits from the two parents. Once again, there are four possible children, which can be determined in exactly the same manner as shown above. In this case, we calculate the probability of generating a new schema in the child, assuming that both parents lack the schema, and that the bits in both parents are uniformly distributed. This can be computed as:

$$\Pr(s_i \mid \psi_{2i-1}) = \frac{(0.5)^8}{7}\sum_{j=1}^{7}(1-(0.5)^j)(1-(0.5)^{8-j}) \tag{18}$$

Again, there are seven distinct places where crossover can occur. If crossover occurs at bit *j*, then parent $\hat{c}$ must have only ones in positions 1 through *j*, and also must *not* have all ones in positions *j* + 1 through 8 (since we know that the other parent does not have the schema). Similarly, parent $\breve{c}$ must have all ones in positions *j* + 1 through 8, and also must not have all ones in positions 1 through *j*.

These calculations lead directly to Equation (18) for computing the probability that two parents that lack a particular schema create the schema in an offspring, given that a crossover occurs within this schema's bit locations. Given these three distinct cases, it is now possible to compute the crossover hypermatrix, **C**, by enumerating all possible combinations of parents $\hat{c}$ and $\breve{c}$ to produce various children.

2.4 Sigma Truncation Selection

Before applying crossover and mutation, the population is first subject to sigma truncation selection (Mitchell, 1998) and frequencies are updated accordingly, where

$$\text{Expected Offspring for } \vec{p}_i = \begin{cases} 1 + \dfrac{f(\vec{p}_i) - \bar{f}}{2\sigma}, & \text{if } \sigma \neq 0 \\ 1.0 & \text{, if } \sigma = 0 \end{cases} \tag{19}$$

3 Experiments

3.1 Quantifying Hitchhiking

To simulate a search using our model, we begin by creating an initial population probability distribution that describes the relative likelihood of each possible configu

ration of schemata, given a uniform distribution of bits in the underlying population. Then we sample stochastically from this population probability distribution vector to create a new population vector for some particular finite population size. Next, the new (sampled) population vector is used as the input to the model outlined in Section 2 to generate the sampling distribution for the subsequent generation. These two steps are then repeated until a population member with the optimum fitness is selected during the sampling phase of the search process, at which point the search terminates and the number of generations required to find the optimum is recorded.

By performing 100 replications of search using this model, we found that, on average, the hitch-hiking free genetic algorithm required approximately 18,432 fitness evaluations to find the optimum on landscape R1, using a population size of 128, a bitwise mutation rate of .005, and a crossover rate of 0.7. In contrast, when we tested an implementation of a standard simple genetic algorithm using the same parameters as our model, the average number of evaluations required to achieve the optimum was approximately 64,490. This result agrees with the original Royal Roads research [5]. Thus, removing the effect of bitwise hitchhiking in our model has a major impact on the performance of the GA. However, given that RMHC requires only 6,200 evaluations, there must be factors in addition to hitchhiking that influence the GA's performance relative to the hill-climbing algorithm such as sampling errors.

3.2 Quantifying Sampling Errors

Genetic drift and sampling errors are artifacts of a finite population size. As population size increases, these effects should diminish. For Random Heuristic Search, it can be shown that in the limit of an infinite population, the heuristic function, $\mathcal{G}$, converges to the transition function, τ. In other words, for an infinite population, the expected distribution at time t, $\vec{p}_t$, can be calculated by iterating the heuristic function on the initial population vector, bypassing the sampling phase altogether.

$$\vec{p}_t = \vec{p}_0(\mathcal{G}^t) \tag{20}$$

To assess the performance of the infinite population model, we need to calculate the expected number of steps required for an arbitrary string selected from an initially random population to achieve the optimum state, via the transition function $\mathcal{G}$. One way to compute this is to iterate $\mathcal{G}$ on an initially uniformly distributed population and record the frequency with which strings enter the optimum state after each time step:

$$\text{Avg \# Steps to Optimum} = \sum_{t=0}^{t \le \text{maxsteps}} \eta_t * t \tag{21}$$

where η_t represents the percentage of strings that first visit the optimum after t steps, and "maxsteps" is chosen large enough such that $\Sigma \eta_i > 0.99999$.

To ensure that we don't recount strings that visit the optimum state more than once during the time interval, we need to make a minor modification to our model. By eliminating all rows and columns that directly transition into or out of the optimum state from the population vector, $\vec{p}$, the mutation matrix, **M**, and the crossover ma

trix, **C**, we transform the optimum state into an absorbing state. At each generation, the proportion of population members in the optimum state can be calculated by subtracting from 1.0 the total proportion of members remaining in the abbreviated population.

This technique was used to compute η_t, the percent of the population newly arriving at the goal state for each generation, *t*. Given this information, the average number of steps required for an arbitrary population member to reach the goal state, assuming the transition function of an infinite population, can be calculated as described in equation (21). In this manner we calculated that the approximate number of generations required for an initially random string to first visit the optimum state should be about 38 generations, assuming an infinite population.

The preceding calculation quantifies the number of generations required for a *single* string to arrive at the optimum. To determine the role sampling error plays in limiting search efficiency, we want to compare the empirical, finite population model of Section 3.1 to a theoretical, finite population model that has the *dynamics* of an infinite population. For such a model we need to calculate, at each generation, the probability that *at least one population member* out of a population of some particular size has reached the optimum. At each generation this is:

$$\Pr(\text{Optimum visited by at least 1 member by time } t) = 1 - \left((1 - \sum_{i=0}^{i \le t} \eta_i)^{PopSize} \right) \quad \textbf{(22)}$$

Using this derived distribution we again applied equation (21) to calculate the average number of generations for a "drift-less" population of size 128 to discover the optimum on R1. The resulting calculation yields an estimate of approximately 30.44 generations, or about 3,896 function evaluations.

3.3 Population Cost

In Section 3.2, we calculated that if a population of size 128 could evolve free from the effects of sampling error and hitchhiking, then the number of generations required to discover the optimum would be approximately 30.4, a performance level that actually surpasses that of RMHC. Unfortunately, the limited population size ensures that drift and hitchhiking will remain a problem. If our model is accurate, however, we should observe that, as the population size is increased, the performance of a real genetic algorithm should approximate our finite population model from Section 3.2.

Fig. 2 shows the relationship between population size and search efficiency for an implementation of a genetic algorithm using the same parameters as in the original Royal Roads research. The performance predicted by the model described in Section 3.2 is also displayed as a function of population size. When population size increases, the number of generations required by the actual genetic algorithm converges to the number predicted by the model. In fact, when the population size increases to about 8,000, the performance of the model and the actual GA are approximately the same.

Fig. 3 shows the relationship between population size and the total number of evaluations required to find the optimum, for the actual genetic algorithm. Note that increasing the population size to 8,000 drastically impacts the performance of the

genetic algorithm in terms of the total number of fitness evaluations. For the settings used in the original Royal Roads work, the optimum performance occurs at a population size of around 500, at which the average number of evaluations required was 33,041. Thus, it appears the primary shortcomings of the genetic algorithm are related to the cost of maintaining a population, given a serial implementation, rather than hitchhiking, which is a consequence of utilizing a population size that is too small.

4 Conclusions

In this paper we have re-examined the performance of genetic algorithms on Royal Roads fitness landscapes in comparison to the performance of simpler hill-climbing algorithms. By building a model of genetic search on landscape R1, we have attempted to show that the poor performance of the standard GA can be explained in terms of the trade-off between the cost required to maintain a large population and the consequences due to hitchhiking and drift when utilizing a population that is too small.

References

1. **Baluja, S. & Caruana, R**. (1995). "Removing the Genetics from the Standard Genetic Algorithm." In Prieditis, A., & Russell, S. (Eds.), *The Proceedings of the 12th Annual Conference on Machine Learning*, (pp.38-46). Morgan Kauffman.
2. **Kirkpatrick, S., Gelatt Jr., C. D., & Vecchi, M. P.** (1983). "Optimization by Simulated Annealing." *Science*, vol. 220, pp 671-680.
3. **Koehler, G., Bhattacharyya, S., & Vose, M. D.** (1997). "General Cardinality Genetic Algorithms." *Evolutionary Computation*, vol. 5, no. 4, pp. 439-459.
4. **Mitchell, M., Holland, J. H., & Forrest, S.** (1994). "When Will a Genetic Algorithm Outperform Hill Climbing?" In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances In Neural Information Processing Systems 6*, San Mateo, CA: Morgan Kaufmann.
5. **Mitchell, M.** (1998). *Introduction To Genetic Algorithms*. Cambridge, MA: MIT Press.
6. **Suzuki, H., & Sawai, H**. (2001). "Crossover Accelerates Evolution In GA with a Royal Road Function." *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pp. 401-412.
7. **van Nimwegen, E., Crutchfield, J. P., & Mitchell, M.** (1999). "Statistical Dynamics of the Royal Road Genetic Algorithm." *Theoretical Computer Science,* vol. 229, pp. 41-102.
8. **Vose, M. D.** (1995). "Modeling Alternate Selection Schemes For Genetic Algorithms." In Koppel, M. & Shamir, E. (Eds.), *Proceedings of BISFAI '95*, (pp. 166-178). Ramat Gan and Jerusalem, Israel: AAAI Press.
9. **Vose, M. D.** (1999a). "Random Heuristic Search." *Theoretical Computer Science,* vol 229, pp. 103-142.
10. **Vose, M. D.** (1999b). *The Simple Genetic Algorithm: Foundations and Theory.* Cambridge, MA: MIT Press.

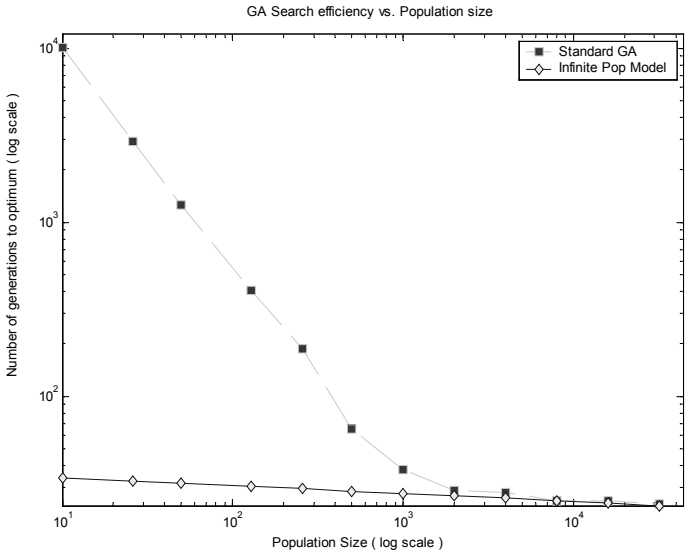

Fig. 2. GA search efficiency vs. population size

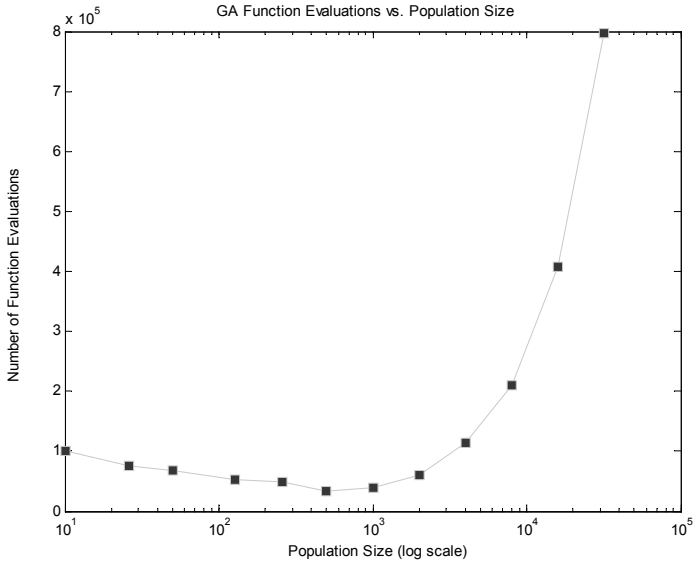

Fig. 3. GA function evaluations vs. population size