Evolutionary Algorithms in Search and Problem Solving

Keith L. Downing
The Norwegian University of Science and Technology
Trondheim, Norway
keithd@idi.ntnu.no

September 21, 2006

1 Search Problems

Imagine yourself climbing a mountain in the middle of the night while carrying only a weak flashlight and a basic GPS device (that only gives your current longitude, latitude and elevation). Your goal is the top of the mountain, but you cannot see it.

At each point during your walk, the altitude reading will give some indication of your progress, especially when compared to readings from earlier points, but at no time will you receive complete directions to the top of the mountain. You will always be guessing, but in an educated manner based on your previous longitude-latitude locations and their altitudes. For example, if, during the past 20 minutes, your latitude has remained constant while your longitude has decreased and your altitude has increased, then a good strategy is to continue along this line of decreasing longitude.

Computational search problems have a similar character. Assuming a problem, P, to be solved, educated guesses are made about the solutions to attempt. After being constructed, each solution is evaluated with respect to its performance on P. In essence, the solution is analogous to the longitude and latitude coordinates, while the evaluation corresponds to the altitude.

Evolutionary search is a form of parallel search, since, in most cases, an EA uses a population of size larger than 1. The analogy then becomes one of a team of scouts, all of whom have a weak flashlight, a GPS and a simple radio to which they can contact a helicopter. The scouts cannot communicate with one another but can send their GPS information to the helicopter pilot, who can speak to the scouts and coordinate the search.

Typical strategic choices of the pilot include the following:

1. Pick up a scout from a location that does not appear promising.
2. Command a scout to make a small move in a random direction.

3. Deposit a scout (paratrooper) on the mountain in an area that other scouts have found to be promising.

4. Deposit a scout at an intermediate location between two scouts whose locations seem promising.

These actions have clear analogies to mutation and crossover in EA's, which are its basic search operators.

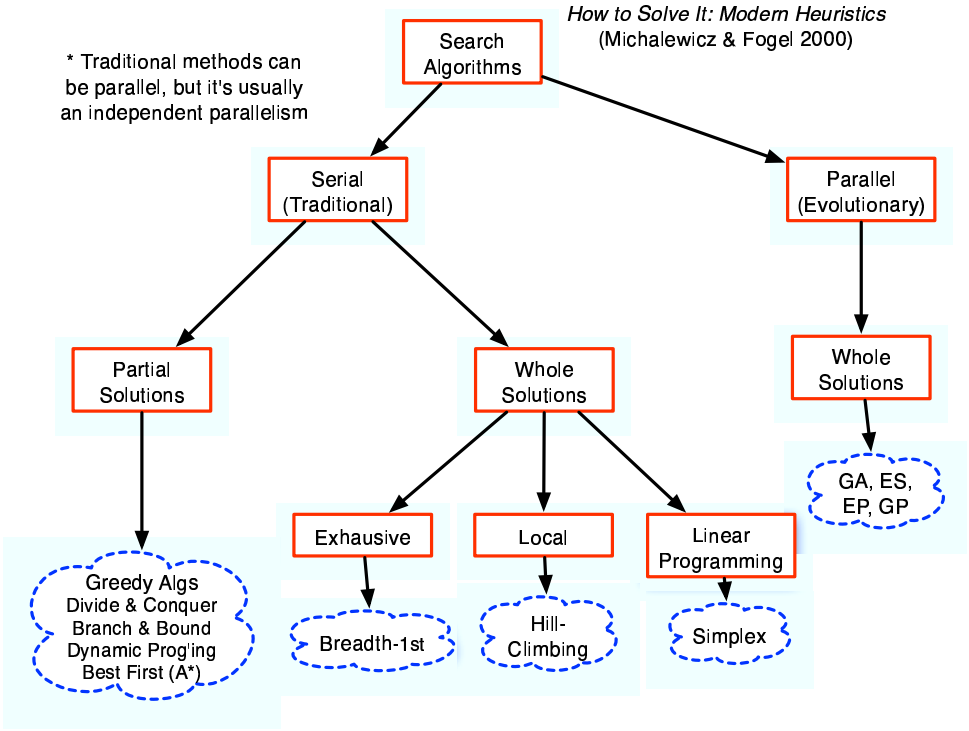

Figure 1: A classification of search algorithms, where evolutionary algorithms (GA, ES, EP and GP) appear as untraditional parallel approaches working on whole solutions.

Figure 1 illustrates many forms of computational search, with EAs as a parallel variety that uses complete solutions. The use of partial solutions means that a method evaluations pieces of solutions and uses this information to prioritize the pieces on which to build further. EAs rarely work in this manner; they rely on complete solutions and a fitness function to evaluate them.

The traditional approaches are predominantly serial, but most can be parallelized. However, these modifications entail the independent treatment of a collection of partial or complete solutions, while EAs involve a strong interaction between population members in at least two senses:

1. The probability that the area about an individual receives further consideration is directly tied to that individual's **relative** fitness.

2. Two (or more) solutions are often combined to create new ones.

In short, EAs are relatively unique in their combination of parallelism, complete solutions and stochasticity. This will be elaborated on later in the chapter.

2 Evolutionary Search

EAs perform a search process that spans three interconnected spaces: genotype, phenotype, and fitness. Essentially, the EA searches in genotype space, but the trajectory of that search is strongly determined by the phenotypes and fitness values to which they are mapped.

Consider the simple example shown in Figure 2. An EA is used to find a satisfactory assignment of M items, of various weights, to K containers (labelled 0 to K-1), each with a maximum weight limit of W, such that the final filled weights of each container are as close as possible to being equal to one another (i.e., the variance in the weights is minimized).

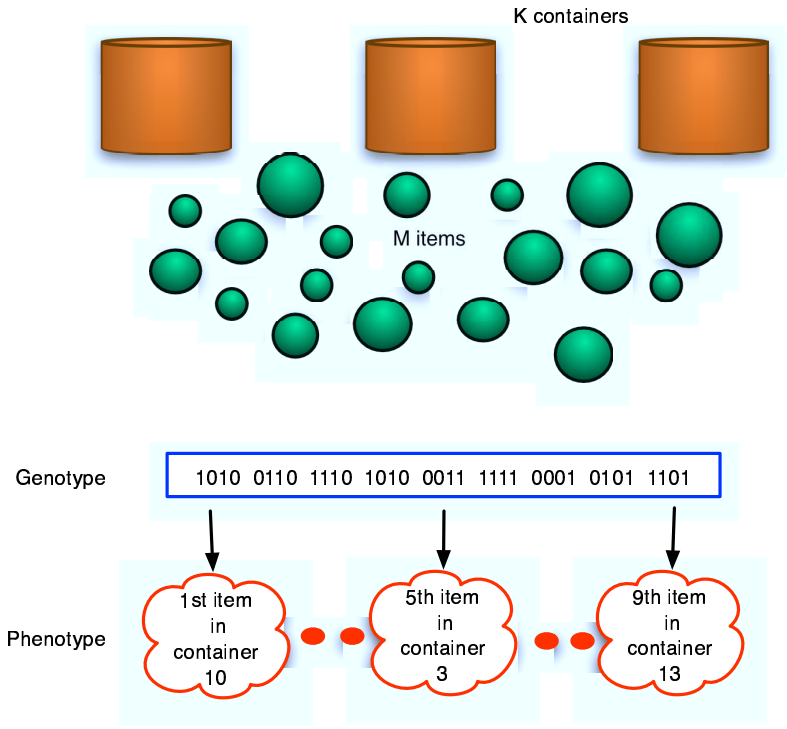

Figure 2: A simple search problem of assigning objects (balls) to containers (cylinders) so as to a) put all objects in containers without exceeding any container's weight limit, and b) minimize container weight variance.

Assume that each genotype is a bit string of length $M\lceil logK \rceil$: one gene of length $\lceil logK \rceil$ for each item, where the gene specifies the container in which to place the item.

A phenotype is then a simple list of container labels, of length M. For simplicity, assume that K is a power of 2, so each $logK$ substring of bits translates into a unique integer between 0 and K-1.

Many adequate fitness functions are possible, and their details are unimportant at this point, but in general, they will give higher scores to partitions in which a) none of the container weights exceeds W, and b) the variance among the container weights is low.

In terms of the connections between the three spaces, the mapping between genotype and phenotype space will be bijective (i.e., each genotype will map to a unique phenotype, and all phenotypes will be mapped to by exactly one genotype), while that from phenotype to fitness space will be many-to-one, since several phenotypes may be awarded the same fitness.

Figure 3 shows the 3 spaces and their connectivity for a container problem with K = 4. The nth pair of bits on the genome encodes the destination container for the nth item. The developmental process is a simple conversion of bit pairs to integers (between 0 and 3), as shown at the phenotypic level in the diagram. The phenotypes then map to fitness values in the upper landscape.

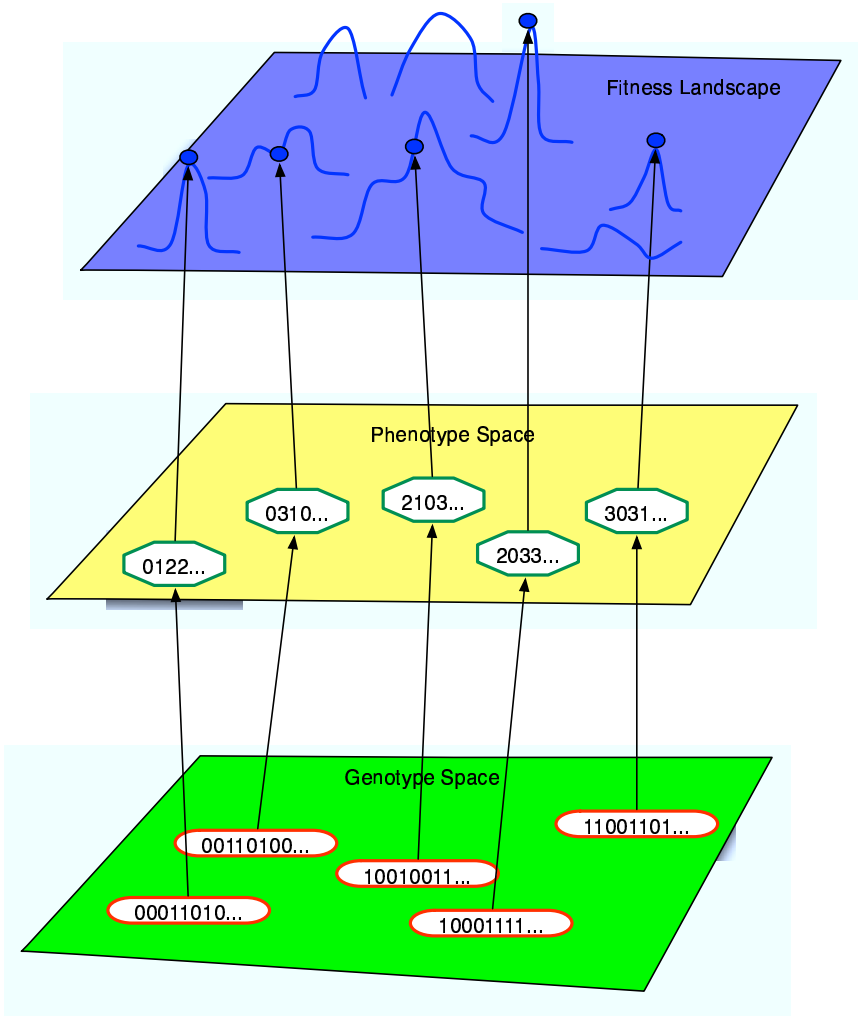

Figure 3: Mappings between genotype, phenotype and fitness spaces.

Various features of the EA determine the nature of and interactions among these three spaces,

and consequently, the difficulty of the evolutionary search. Starting on top, the fitness landscape is probably the most critical of the three. If it is relatively smooth, with gently rolling hills and gradual inclines leading to a summit corresponding to the optimal phenotype, then many search algorithms, not just EAs, will stand a good chance of finding that summit. However, if the summit sits atop a very steep peak that rises straight up from the zero plane, then search becomes a needle-in-the-haystack endeavor. Similarly, if there are many local (but not global) maxima on the landscape, then the search process is easily detoured toward and bogged down at these deceptive sites.

The fitness landscape is merely a visualization of the fitness function, so a landscape with one Mount Everest rising from the plains corresponds to a fitness function that gives no partial credit to any of the phenotypes directly *under* the plains. Hence, when the EA generates such phenotypes, it literally does not know *which way is up* and can only guess by searching out in random directions from the zero-fitness phenotypes.

A landscape with many peaks reflects a fitness function that gives a lot of partial credit to phenotypes. Unfortunately, the generous allotment of credit is not sufficient to insure a successful search. The basis for smooth landscapes with gradual ascents to promising summits is the **correlation** between phenotypes and fitness. Given a phenotype, P, and its fitness f(P), if most phenotypes in the neighborhood of P also have a fitness close to f(P), then the two spaces are well correlated. And clearly, if search moves a small distance from P, the movement in the fitness landscape will only be a small ascent or descent, if not a neutral movement along a local plateau.

Unfortunately, devising fitness functions that insure this correlation is nearly impossible in many problem domains. Even in the container example, areas of the space are almost guaranteed to have low correlation. For example, assume that the items are ordered from heaviest to lightest. Then, small changes to the last container assignment (i.e., that for the lightest item) will probably not have a great effect upon fitness, since switching the container of the lightest item will not change total weights very much. Hence, in regions of phenotype space where all container assignments are the same (across all phenotypes in the region) except the last assignment, we can expect a rather high correlation between phenotypes and fitness.

However, in regions where only the *first* container assignment varies among neighbors, we can imagine very little correlation, since the phenotypes vary with respect to their assignment of the *heaviest* item, and there are surely many situations in which switching the container of the heaviest item can cause a transition from a reasonably good and completely legal solution to one in which a container has a total weight exceeding W. Since most fitness functions would punish such load violations severely, fitness would drop precipitously between neighboring phenotypes.

Assuming that the fitness and phenotype spaces are well correlated, the EA still has no guarantee of success. The mapping between genotypes and phenotypes must also show a smooth correspondence. Although genotypes and phenotypes are equivalent in some types of EAs, they often are not, and a non-trivial developmental process is required to convert genes into traits. Now, genetic operators such as mutation and crossover operate on genotypes, not phenotypes, so when a high-fitness parent genotype is mutated slightly (by, say, flipping a single bit), then if genotype and phenotype space are well correlated, the child should have similar traits to the parent and have a comparably high fitness. Thus, using mutation as a vehicle of change, search can move smoothly about the fitness

landscape, without frequent abrupt dips and hops. Crossover normally incurs major changes to genotypes, so even in a well-correlated landscape, its does not produce smooth search.

In EAs, mutation is critical for zeroing in on nearby maxima (whose general neighborhood may have been discovered by crossover), but it requires good genotype-phenotype and phenotype-fitness correlation in order to succeed. Once again, superior correlations are hard to guarantee. As the developmental process becomes more complex, the genotype-phenotype correlation often deteriorates dramatically.

However, even trivial developmental links can cause poor correspondence. For example, if bit strings are translated into integers in the straightforward manner of a base-2 to base-10 conversion, then the correlation always breaks down for the higher-order bits. To wit, the genotypes 0001 and 1001 are neighbors, since they differ by a single bit, but their phenotypes, 1 and 9, are quite distant. If these phenotypes represent settings for a thermostat, for example, then 1 and 9 would probably give much more divergent outcomes (and thus a larger fitness difference) than 1 versus 2. So in this case, the lack of genotype-phenotype correlation could easily cause a poor genotype-fitness correlation and make evolutionary search difficult.

Conversely, in a K=4 container problem, 1 and 9 would just represent container labels, so the difference between 1 and 9 versus 1 and 2 would not necessarily mean anything in terms of the actual problem solution and its fitness: moving an item between containers 1 and 9 need not be significantly better or worse than moving it between 1 and 2.

In general, the correlations between these spaces are rarely perfect, and when they are, the problem is usually either trivial or better solved using a traditional search method. EAs are designed to handle the tough cases: those with fitness landscapes containing scattered local maxima and steep peaks jutting out of flat planes, and those with non-trivial mappings between the syntax (i.e., genotype) of a problem solution and its semantics (i.e., the phenotype). When applied to a particular problem, the discussion above gives us concrete quantitative criteria for a) assessing approximately *how hard* these inevitably difficult searches will be, and b) designing fitness functions and genotypic and phenotypic representations (along with their mapping) so as to make evolutionary search as computationally tractable as possible.

3 Exploration versus Exploitation in Evolutionary Search

As mentioned above, the fitness landscape is a visualization of the fitness function. Unfortunately, for complex phenotype spaces, a complete visualization of the fitness values for each phenotype is computationally impossible. Hence, we cannot simply look at the fitness landscape (in a multi-dimensional space) and *eyeball* the global maxima. Also, we cannot simply solve the fitness function (the way we solve $x^2 - 3x - 18 = 0$ for x) using symbolic techniques to find the optimal phenotype.

Instead, we must perform search, meaning that partial or complete solutions must be generated, tested and modified, with changes taken in directions that appear to lead to improved solutions, i.e., directions that seem to head toward a maximal summit of the fitness landscape.

Whereas classic AI search techniques such as A* deal with partial solutions, evolutionary algorithms work with complete solutions/designs. Also, while A* tends to process these partial solutions serially and independently, EAs work in parallel with many different solutions and often combine them to produce hybrid children.

Search is a partially blind process, since the search module can never see the entire fitness landscape; it only knows the terrain near its attempted solutions. Each such attempt is therefore a probe point into the fitness landscape, and the intelligence in search comes from the judicious choice of these probes such that global optima can be discovered using as few probes as possible.

Two competing tendencies often govern the probing process: exploration and exploitation. In exploration, the search motor tries new solutions that are quite different from all previous probes, in an attempt to get some information about an uncharted area of the fitness landscape. In contrast, exploitation involves trying new solutions/probes that are only slightly different than previous promising probes. Thus, an exploitative strategy remains in an area of known potential and tries to zero in on the best solution in that region.

A good search motor strikes a proper balance between exploration and exploitation, typically by beginning search in an exploratory mode and then gradually becoming more exploitative. In biological terms, search motors begin by accentuating variation before gradually moving to a more inheritance-center strategy wherein *apples never fall far from the tree.*

In general, mutation embodies an exploitative change, whereas recombination via crossover is more explorative. Since EAs often do both, the *frequency* of each genetic operator is often a measure of the degrees of exploration and exploitation. From a very general perspective, all genetic operators can be considered explorative, since they do produce variation, whereas selection strategies embody exploitation by giving priority to the best individuals. But again, this is a very broad-brush approximation, since both genetic operators and selection strategies have tuning mechanisms to adjust their balance of exploration and exploitation. Figure 4 illustrates the broad and more-detailed views these interactions.

Essentially, selection pressure mirrors exploitation. When pressure is high, the EA strongly shifts focus to the high-fitness solutions, at the expense of the lesser solutions. This often means that one or a few highly-fit individuals produce a large fraction of the next generation. Hence, population diversity drops, which effectively reduces exploration.

Figure 5 illustrates exploration versus exploitation for evolutionary search. At time T, the population is reasonably well spread across genotype space and the fitness landscape. An exploratory strategy encourages even further spreading, with no extra emphasis on points near the middle (where a high-fitness genotype was discovered at time T). Conversely, the exploitative strategy shifts more resources (i.e. genotypes) to the areas that have already yielded high fitness values. In terms of EA genetic operators, the explorative strategy may have employed more crossover to produce new genotypes that are distant from the parents, while exploitation might have slightly mutated the two best individuals from time T to produce neighboring genotypes at T+1.

To formalize the contributions of exploration and exploitation to evolutionary progress, De Jong [2] (pp. 160-162) includes an enlightening description of Price's classic equation:

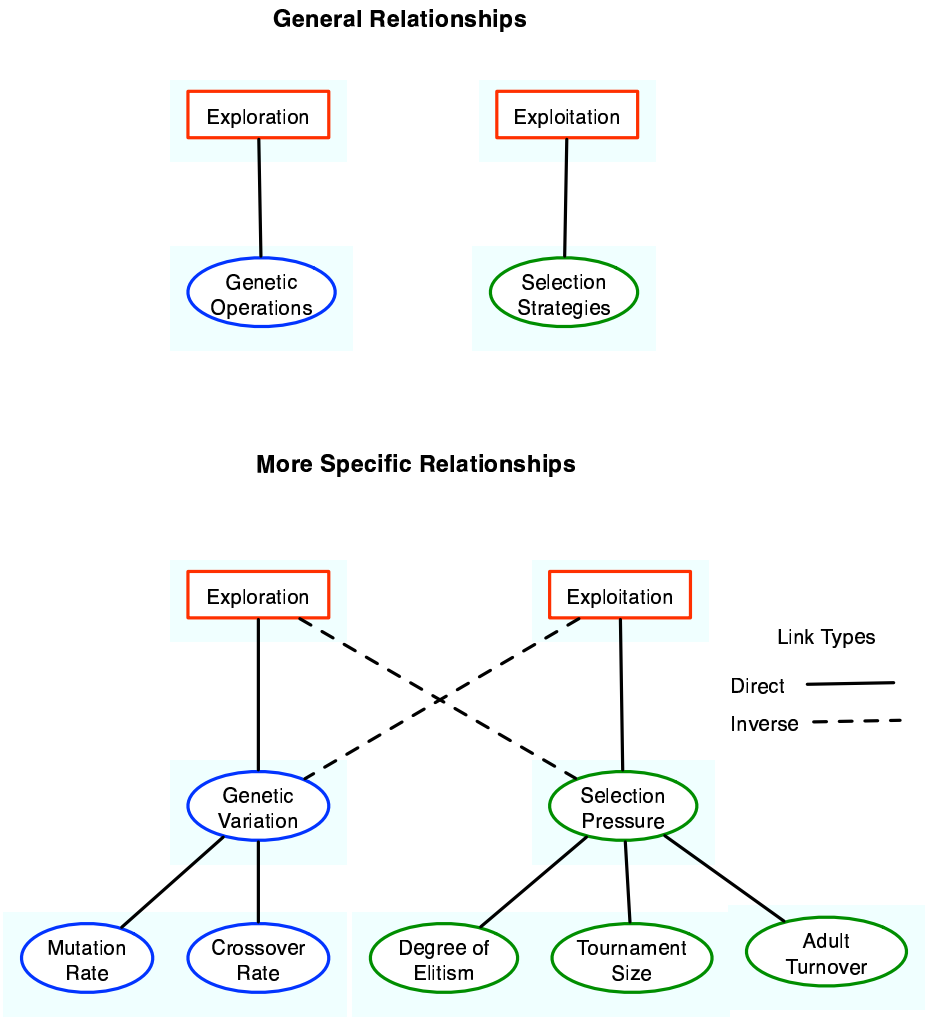

Figure 4: Broad and more detailed views of the relationships between exploration, exploitation, genetic operators and selection.

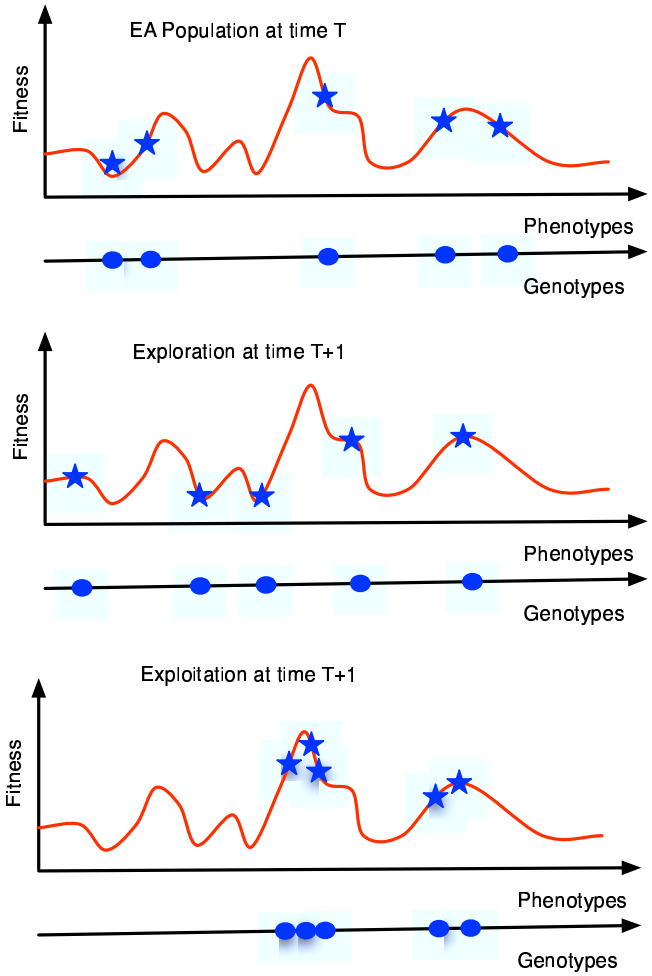

Figure 5: Exploration versus exploitation in evolutionary search. Circles denote the population of (5) genotypes, while stars represent their fitness. This assumes a perfect correlation between genotype and phenotype space such that genotypes produce phenotypes directly above them in the diagram.

$$\Delta Q = \frac{Cov(z,q)}{\overline{z}} + \frac{\sum z_i \triangle q_i}{N\overline{z}} \tag{1}$$

Here, ΔQ is the change in some trait/quality, Q, over the entire population, while q_i is the value of that quality in parent i. The trait is assumed to be graded, to some degree, so that individuals will have more or less of the trait, with q_i quantifying that amount. Also, $\triangle q_i$ denotes the difference between the average q value in parent i's children and q_i.

As mentioned earlier, fitness in evolutionary biology is normally measured in terms of reproductive success, so the fitness correlate in Price's equation is z_i, the number of children to which parent i contributes portions of its genotype; $\overline{z}$ is the population average of the z_i. Also, N is the population size, and Cov(z,q) is the covariance of z and q, as defined by:

$$Cov(z,q) = \sum_{i=1}^{N} \frac{(z_i - \overline{z})(q_i - \overline{q})}{N} \tag{2}$$

In a nutshell, the covariance will be a high positive value if there is a direct relationship between reproductive success and the degree of trait q. It will be a large negative value if the two are inversely related, i.e. many individuals with high q values produce few offspring. It will be close to zero if there is no clear relationship between q values and reproductive success.

The first term in Price's equation 1 denotes the selective pressure, since the covariance of q and z indicates the degree to which possession of the trait correlates with reproductive success. When selection pressure is low, **any** value of q could yield just as many offspring as any other value, which would be reflected in a low Cov(z,q) value. Conversely, when selection pressure is high, certain traits (in Price's model, high values of q) give a definite reproductive advantage over others, reflected in many (high-q, high-z) pairs in the population, and thus a high Cov(z,q) value.

The second term in 1 captures the degree to which children **differ** from their parents, with respect to trait q. It therefore represents population variability from generation to generation, the opposite of inheritance.

Clearly then, Price's equation shows that the rate of evolutionary change is directly proportional to the combination of selective pressure and variability. If both are high, evolutionary change should be swift, but if either is low, the rate is reduced.

Consider each of these options. If variation is high but selection low, then many new genotypes (i.e., many diverse q values) are produced, but all receive approximately equal priority in terms of reproduction. Hence the population will not deviate much from some average value of q. Conversely, if selection is high but variation is low, then an initial population (of presumably similar q values) will not diversify. Thus, selection, no matter how strong, will have very little to *favor*, since all q values will be similar.

For evolution to attain optimal, but hard to reach, areas of a search space, it needs to progress through many rounds of ruthless filtering of high-variance gene pools. Each filtering round moves

the population's center of gravity to a slightly higher point, from which many exploratory moves are taken by the genetic operators, with only the best such moves resulting in useful new points, to which evolution shifts its focus and produces new *feelers* in search space, etc.

The assumption of homogeneous initial q values normally holds in nature but not in EAs, which are often initialized with random genotypes. However, even with a diverse gene pool as starting point, evolution will stagnate under conditions of high selection and low variance. To wit, high selection will initially favor one or a few of the slightly better genotypes, causing many of the new offspring to have q values within a tight range. From there, exploration will be quite slow due to low variation. In effect, selection forces the population to converge to homogeneity so early in evolution that the favored individuals probably only represent local maxima in the search space; evolution then gets *stuck* there due to low variation.

4 Evolutionary Search as Resource Allocation

One very useful characterization of search is as a resource allocation problem. For an EA running on a laptop computer, the analogy makes perfect sense: given a limited amount of runtime, only a certain population size, M, and number of generations, G, are feasible. This amounts to MG fitness evaluations, or MG attempts to solve the search problem. Each attempt is a resource, and one hopes that their EA is using them wisely.

In his classic book, Holland [1] uses a 2-armed bandit (a hypothetical type of slot machine) to illustrate and analyze search as resource allocation. The analogy clearly illustrates the tradeoffs between information gathering (i.e., exploration) and information-biased choice (i.e. exploitation), when both actions have known costs but unpredictable outcomes.

The problem is straightforward. You walk into a casino with a small bucket of quarters. In the corner stands a bright shining slot machine with 2 arms, the 2-armed bandit. The sign above the bandit says that one of the arms has better odds of hitting winners, but which arm? Your task is to find out which arm is best while simultaneously maximizing your total profit. Your only means of information gathering is pumping quarters into the machine, pulling one arm or the other, and recording the result. For simplicity of analysis, we assume that a win entails of payoff of 1 unit (say a dollar), whereas a loss pays 0 units.

Naturally, if you knew the better arm, you would invest all N quarters on it. Unfortunately, you will have to use some, possibly many, of the N quarters to gather enough statistics to convince yourself of a significant difference between the arms. In short, you need to explore by playing both arms a reasonable number of times, k. If the statistics begin to show a marked difference between the two arms, then the remaining N-2k quarters are probably best invested in the (apparently) better arm.

To analyze this mathematically, Holland asks the following question:

> If n of the N trials (i.e.,quarters) are invested in the arm that, based on observations

alone, seems worst, then what is the total expected loss of money over all N trials?

First, let $\delta = |p_1 - p_2|$ be the amount that is lost on any trial during which the non-optimal arm is played, where p_1 and p_2 are the actual (but unknown to the player) winning odds for each arm, respectively. Second, let e(N-n,n), be an error probability that after n trials on one arm and N-n on the other, the player's observed outcomes give a false impression, indicating that the wrong arm is the most lucrative. Clearly, e(N-n,n) decreases when both n and N increase.

There are thus two sources of loss:

1. The observed outcomes give the correct impression of the relative payoffs of the 2 arms, so the only losses occur during the n trials during which the weaker arm was tried. The expected value of this loss is $(1 - e(N - n, n))\delta n$.
2. The observed outcomes give a false impression, so the N-n trials on the misleadingly *better* arm were bad bets. This loss has an expected value of $e(N - n, n)\delta(N - n)$.

Combining these two loss scenarios, the total expected loss, L(N-n,n) is:

$$L(N - n, n) = \delta \left[n(1 - e(N - n, n)) + (N - n)e(N - n, n) \right] \tag{3}$$

This equation is the basis for a very useful theorem [1] (page 77), whose main consequence is that the total loss is minimized when the relationship between N-n and n is exponential. In other words, the gambler should invest exponentially more money on the arm that appears best at any given time.

Formally, it looks like this:

$$N - n \sim \sqrt{8\pi b^4 N^2} e^{\frac{n}{2b^2}} \tag{4}$$

Here, b is function of the means and variances of the probability distributions for the two arms.

Essentially, an evolutionary algorithm is posed with a m-armed bandit problem, where m is the population size. Since Holland's result generalizes for $m > 2$, the best way to achieve well-adapted populations as quickly as possible is to allow the most fit individuals to increase at an exponential rate relative to the weaker individuals. In short, the EA should produce exponentially more child genotypes from the highly fit than from the weaker adults.

As Holland shows (and we will see below), this is exactly what a classic genetic algorithm using fitness-proportionate selection does.

4.1 Schema Theory for Genetic Algorithms

Holland [1] developed schema theory to help quantify the resource-allocation view of evolutionary search. The theory helps explain how GAs utilize information (in terms of the relative fitness among population members) to create new genotypes and push the population toward higher average fitness.

A schema is simply a subset of genotype space. In a classic GA with fixed-length bit-vector genotypes, a schema can be expressed as a bit string with some known and some unspecified bits. For example, in an 8-bit genotype space, 1******1 is a high-cardinality schema containing all genotypes that begin and end with a 1; conversely, 1010101* is a very specific schema containing only two members: 10101010 and 10101011. In standard GA terminology, the star (*) is called a *wildcard*.

The non-wildcard bits in a schema are termed *defining bits*, and their cardinality constitutes the schema's *order*. Clearly, low order schema have many more members than high-order schema; and the number of members in schema S is simply $2^{L-order(S)}$.

For an L-bit genotype space, every genotype, G, is a member of exactly 2^L schema. To determine each of these, simply take each bit of G and either leave it as is or replace it with a wildcard. Thus, for each of the L bits there are two independent choices. When a choice has been made for each bit, the result is one schema in which G is clearly a member. Since 2^L different combinations of choices are possible, G is a member of at least 2^L schema. Since any schema s_i that includes G must match G in all of s_i's defining bits, s_i must be included in the 2^L schema generated above. Hence, 2^L is an exact value.

In viewing evolutionary search as resource allocation, it is difficult to chart the progress of any single genotype, since crossover and mutation tend to reduce pure cloning to a minimum. Hence, even a top-fitness individual may have no any exact copies in the next generation. However, progress in terms of evolving schema membership can also be charted; in this case, the size of a schema is simply the number of individuals in the current population who belong to it. This focus on changing schema sizes gives a very clear indicator of evolutionary progress, wherein schema with above (below) average fitness (among existing members) tend to increase (decrease), and at a rate that is, in fact, exponential.

The formal Schema Theorem provides a lower bound for the expected size of a schema in generation t+1, given its size in generation t. Following the notation of De Jong [2] (pg. 192), let $m_i(t)$ be the size of schema i at time t. Then, $E(m_i(t))$ is the expected value of that size. The Schema Theorem is then:

$$E(m_i(t+1)) \geq m_i(t) * \frac{\overline{f_i}(t)}{\overline{f}(t)} * (1 - \epsilon_i) \tag{5}$$

where $\overline{f}(t)$ is the average fitness of the entire population and $\overline{f_i}(t)$ is the average fitness of all schema i members that appear in the current population. The ϵ_i term represents the probability that a member of schema i is mutated or crossed over in such a way that the child genotype is not in i.

The first two terms of the theorem presuppose the use of fitness-proportionate selection. This connection becomes clear when we consider a schema, s_i, with size m_i. Using fitness-proportionate selection, the total area on the roulette wheel that s_i receives at the end of generation t is:

$$\frac{m_i(t) * \overline{f_i}(t))}{\sum_{indiv \in P} f_{indiv}(t)} \tag{6}$$

where P is the population, which has size M. The numerator is equivalent to the sum of the fitness values of all members of s_i.

Assuming full generational replacement (i.e., all M parents are replaced by M children in each new generation), the roulette wheel will be spun exactly M times. Hence, the expected number of times that a parent will be chosen from s_i is:

$$\frac{M * m_i(t) * \overline{f_i}(t))}{\sum_{indiv \in P} f_{indiv}(t)} = \frac{m_i(t) * \overline{f_i}(t))}{\frac{1}{M} \sum_{indiv \in P} f_{indiv}(t)} = m_i(t) * \frac{\overline{f_i}(t)}{\overline{f}(t)} \tag{7}$$

In the absence of all genetic operations, such as mutation and crossover, parents are simply cloned during reproduction, and the schema sizes evolve according to the following equation:

$$E(m_i(t+1)) = m_i(t) * \frac{\overline{f_i}(t)}{\overline{f}(t)} \tag{8}$$

Notice that this is an equality, not an inequality as in the Schema Theorem. Without genetic operations, there is no source of variation, hence no exploration of genotype space. Evolution is completely governed by the exploitative power of selection, and the population will converge (at an exponential rate) to a homogeneous group in which all members are clones of the most fit genotype in the initial population.

4.1.1 Evolution as an m-Armed Bandit

The following simple derivation shows that schema with above-average fitness will increase in size at an exponential rate, while below-average schema will decrease exponentially. Thus, a genetic algorithm using fitness-proportionate selection operates in a manner recommended by Holland's m-armed bandit analysis, where m is the population size.

First, in equation 8, substitute $m_i(t+1)$ for it's expected value $E(m_i(t+1))$ and then subtract $m_i(t)$ from both sides of the equation to produce:

$$m_i(t+1) - m_i(t) = m_i(t) * \frac{\overline{f_i}(t)}{\overline{f}(t)} - m_i(t) = m_i(t)[\frac{\overline{f_i}(t)}{\overline{f}(t)} - 1] \tag{9}$$

Assuming that $m_i(t+1) - m_i(t)$ approximates the derivative of $m_i(t)$ with respect to time (where each generation represents a time step) and then dividing each side by $m_i(t)$ yields:

$$\frac{dm_i(t)/dt}{m_i(t)} = [\frac{\overline{f_i}(t)}{\overline{f}(t)} - 1] \tag{10}$$

Next, integrate both sides and use the basic calculus relationship $\int \frac{dx/dt}{x} = \ln x$:

$$\ln m_i(t) = \int [\frac{\overline{f_i}(t)}{\overline{f}(t)} - 1]dt = [\frac{\overline{f_i}(t)}{\overline{f}(t)} - 1]t + c \tag{11}$$

where c is the constant $\ln m_i(0)$, the natural log of the initial size of schema i.

Finally, take the exponent of each side to attain:

$$m_i(t) = e^{[\frac{\overline{f_i}(t)}{\overline{f}(t)} - 1]t + c} \tag{12}$$

Thus, it is clear that $m_i(t)$ changes as an exponential function of t. Most importantly, the coefficient of t is positive only if the average fitness of the schema members exceeds the average fitness of the population as a whole. Conversely, the coefficient is negative when the schema's average is below that of the population. A positive coefficient entails exponential growth, while a negative coefficient means exponential decay. Thus, a schema's size will either increase or decrease exponentially, depending upon the average fitness of its members relative to that of the entire population. Again, this is precisely what the k-armed bandit theory suggests as the optimal strategy for the allocation of resources (i.e., genotypes) during evolution.

4.1.2 The Disruptive Effects of Genetic Operators

The inequality in the Schema Theorem stems from the introduction of genetic operators and the simple fact that, in Holland's original work, the disruptive properties of genetic operators were included, but the constructive potential was omitted. The schema disrupting factors of mutation and crossover are bundled up in the ϵ_i term:

$$\epsilon_i \leq (1 - (1 - p_{mut})^{k_i}) + (p_{cross} * \frac{dl_i}{L-1}) \tag{13}$$

where:

p_{mut} = the probability (per bit) of mutation.

p_{cross} = the probability (per mating pair of parents) of performing single-point crossover.

L = the length of the genotype, i.e., the total number of bits.

k_i = the order (i.e., number of non-wildcard bits) of s_i.

dl_i = the *defining length* of s_i, which is the distance between the first and last defining bits of s_i.

For example, in schema s_0 = *1**0*, these variables are: L = 6, $k_0 = 2$, and $dl_0 = 3$.

Now, the first term in equation 13 represents the probability that an individual's membership in s_i is ruined due to mutation. The term $(1 - p_{mut})^{k_i}$ denotes the probability that none of the k_i defining bits gets mutated. Then, $(1 - (1 - p_{mut})^{k_i})$ is the probability that at least one of the defining bits does get flipped.

The second term in equation 13 reflects the chance of schema membership being destroyed by crossover. Here, the defining length plays a key role, since any crossover point that is chosen between the first and last defining bits of s_i will clearly separate the defining bits from one another in the children. Assuming that the crossover point is chosen randomly from the L-1 interior points of the genotype, the probability that it is chosen between the first and last defining bits is simply $\frac{dl_i}{L-1}$.

For crossover to destroy membership, two events must happen: paired parents must actually be crossed over, and the crossover point must split the defining bits. The product $p_{cross} * \frac{dl_i}{L-1}$ represents the probability of the conjunction of those two events.

Since membership in s_i can be lost by either mutation **or** crossover, the two terms are added in equation 13 to produce an upper bound on the probability of schema disruption. As mentioned above, this is an upper bound, not an exact value, since the schema-constructing properties of mutation and crossover are ignored.

Remember that ϵ_i is the probability that a child will **not** be in s_i even though one of its parents is. To omit construction means to ignore the possibilities that new members of a schema may be **created** by genetic operators. Hence, even though crossover may split up the s_i *defining bits* of an individual - i.e., those that qualify it for membership in s_i - these bits are occasionally reassembled in the child due to pertinent matches between the bits of the two parents.

As a simple example, consider again schema s_1 = 10**01. Assume that the following two parents, both members of s_1, are crossed over, with a single crossover point after the 3rd bit:

P1 100001

P2 101101

This yields two children, both of which are also in s_1.

C1 100101

C2 101001

The Schema Theorem (correctly) predicts that the defining bits are disrupted during the split, but it fails to account for the fact that they are properly reassembled in both children. This explains why equation 13 provides only an upper bound for the disruptive probability. In most cases, the actual probability will be lower due to the constructive operations that are also at work.

In addition, the Schema Theorem ignores cases such as the following crossover of two non-s_1 members (also after the 3rd bit):

P3 100000

P4 111101

which produces one member of s_1, C3.

C3 100101

C4 111000

In short, the Schema Theorem only accounts for increases in a schema's size due to the exploitation of selection, not the exploration inherent in the genetic operators. These operators are assumed to be purely disruptive of the schema. Holland was perfectly aware of this shortcoming and therefore used an inequality in equation 13, not an equality.

Since the inverse probability, $(1 - \epsilon_i)$, that of **not** disrupting the schema, is used in equation 5, the opposite inequality is used, and the result represents the lower bound of the expected value of the schema size.

4.1.3 Schema Theorem in Action

As a simple example of the Schema Theorem at work, consider a small population of 8-bit vectors. The fitness function is one-max, meaning that fitness is simply the number of 1's in the genotype. In this case, the genotype and phenotype are the same; there is no need for development.

We begin with a randomly generated initial population (with fitness values in parentheses):

P1: 00000001 (1)

P2: 11110110 (6)

P3: 11000001 (3)

P4: 00110011 (4)

P5: 10111011 (6)

P6: 10010000 (2)

P7: 01000001 (2)

P8: 01100001 (3)

The population average fitness at time 0 is $\overline{f}(0) = 3.375$. Now consider two schema:

S0: *00*****

S1: *11*****

Each has two members in the population: P1 and P6 for S0, and P2 and P8 for S1. Thus, each schema has the following average fitness:

S0: $\overline{f}_0(0) = \frac{1+2}{2} = 1.5$

S1: $\overline{f}_1(0) = \frac{6+3}{2} = 4.5$

Clearly, $\overline{f}_0(0)$ is well below the population average, $\overline{f}(0)$, while $\overline{f}_1(0)$ is well above it. Hence, the Schema Theorem predicts that the former schema will decrease in size, while the latter will increase.

Assuming full generational replacement and no mutation but a 100% single-point crossover rate, 8 fitness-biased choices will be made among the parents to produce 4 mating pairs. A typical result would be (where the number in parentheses denotes the index of the crossover point):

1. P3 × P4 (2)
2. P2 × P8 (7)
3. P5 × P6 (1)
4. P2 × P7 (5)

Notice that the higher fitness individuals were selected slightly more often. The 4 crossovers produces 8 children:

C1: 11110011 (6)

C2: 00000001 (1)

C3: 11110111 (7)

C4: 01100000 (2)

C5: 10010000 (2)

C6: 10111011 (6)

C7: 11110001 (5)

C8: 01000110 (3)

The population's average fitness has now increased to $\overline{f}(1) = 4.0$ and the size of S1 has doubled to 4 members: C1, C3, C4 and C7. S1's average fitness, $\overline{f}_1(1)$ has also increased from 4.5 to 5. Schema S0 remains stable in size and average fitness. However, we can expect its size to eventually decline as the S1 members take over the population due to their high average fitness and hence their dominance of the roulette wheel.

To wit, another round of fitness-biased reproduction produces:

C1: 11110011 (6)

C2: 11110111 (7)

C3: 11110101 (6)

C4: 11110011 (6)

C5: 01101011 (5)

C6: 10110000 (3)

C7: 11000110 (4)

C8: 01110011 (5)

Now, S0 has no members while S1 has 6, with $\overline{f}_1(2) = 5.83$. Just as the Schema Theorem predicts, the highly fit schema increase in size. Clearly, the increased investment in S1 pays off, since the population average fitness continues to rise: $\overline{f}(2) = 5.25$.

In addition, it is clear that membership in other high-fitness schema, such as 111****, ******11 and 11****11 has increased over the two generations. Schema Theory encourages us to a view population not as a small collection of individuals but as a large set of competing schema. This brings out one key element of EA parallelism: by determining the fitness of just a few individuals, an EA **estimates** the average fitness of an exponential number of schema, and those with high averages will tend to proliferate while those without will decay.

The highly fit schema are the building blocks of good solutions, so as they take over the population, the chances of good components coming together during crossover will increase, at least in theory. This is known as the *Building Block Hypothesis* for genetic algorithms. It holds true in many cases but fails in *deceptive* fitness landscapes where the optimal individuals are members of many schema with low average fitness.

A simple deceptive landscape is generated by a modified one-max function that gives a fitness value equal to the number of 1's in all cases except two: a) fitness(111111) = 0, and b) fitness(00000000) = 8. However, such pathological cases are mainly of interest to EA theoreticians. When using EAs on many practical problems, evidence of the BBH is common.

Of course, if the defining length of a schema is too high, then crossover can disrupt it so often that it never gets a good foothold in the population. Thus, it behoves highly fit schema to have compact representations (i.e., short defining lengths). As a GA designer, it is important to be aware of this and construct representations such that interacting genes are close to one another on the chromosome. As a simple example, if three genes interact such that the combination 101 of their 3 alleles is optimal, then placing the genes side-by-side on the chromosome insures that once 101 arises in an individual, it will tend to survive crossover intact. Restated in terms of schema, **101*** is more stable than *1*0*1**, which is more stable than 1***0**1.

4.1.4 Schema and Representational Choices

A very cursory analysis of schema combinatorics, continued from earlier, has clear implications for the choice of genotypic representations. Given the implicit parallelism of GA's, i.e., that each genotype acts as a representative for many schema, it is easy to calculate degrees of this parallelism for different vector formats.

In general, if genotypes consist of L genes, with each gene taking on one of K values, then as shown before:

- There are K^L strings in genotype space.
- There are $(K+1)^L$ schema in genotype space.
- Each string is a member of 2^L schema.

An important measure of implicit parallelism is the total fraction of the schema space, F_{ss} that any one string represents:

$$F_{ss} = \frac{2^L}{(K+1)^L} = \left(\frac{2}{K+1}\right)^L \tag{14}$$

Clearly, this decreases with increasing K, and thus the degree of implicit parallelism is higher in binary strings than in k-ary strings of fixed length L. As the size of a problem space increases,

there is a corresponding need to increase K and/or L for the genotype space, with unfortunate consequences for implicit parallelism.

But what about cases where the problem space is fixed, but the GA designer must choose K and L? Holland [1] (pg. 71) provides a simple comparison between two choices for K and L:

1. K = 2, L = 20
2. K = 10, L = 6

These give approximately equal-sized genotype spaces: 1048576 and 1000000, respectively. The key difference is in implicit parallelism. In case 1, each individual is a member of $2^{20} = 1048576$ schema, while in case two, they belong to only $2^6 = 64$. Hence, each individual in case 1 is doing 4 orders of magnitude more *work* during evolutionary search in the sense that it helps provide evaluations for nearly 50,000 more schemas than does an individual in case 2.

Although the difference is less dramatic, the F_{ss} values for cases 1 and 2 are $3.007x10^{-4}$ $3.61x10^{-5}$, respectively. So the binary representation has an order of magnitude better coverage of the total genotype space as well.

Obviously, in terms of implicit parallelism, the choice of representation has a major impact.

5 Evolutionary Problem Solving

The evolutionary approach to solving problems differs quite dramatically from traditional artificial intelligence and machine learning (ML). The difference is most easily summarized by Figure 6. Problem solving is often characterized as a search through solution space in which solutions/hypotheses are generated and then tested for feasibility, optimality, etc. Traditionally, an efficient problem solver has been viewed as one that exploits as much intelligence as possible to generate reasonably good solutions and thereby avoids wasting time testing bad hypotheses. In fact, a measure of problem-solving improvement (i.e., learning) in knowledge-based systems was the degree to which test knowledge or constraints could be re-expressed (or *operationalized*) in the generator [3].

In general, most AI systems can take a hypothesis and its test results as inputs and produce a new hypothesis that is almost guaranteed to be an improvement. For example, if a logic-based system uses disjunctions of primitive terms to produce classifications of input examples, and if the test results indicate that a hypothesis is too specific (in that it classifies many positive examples as negatives), then the system will normally add an extra disjunction in order to generalize the hypothesis. This *intelligence* that the generator contains is simply a knowledge of basic logic. Alternatively, if the representation form is an artificial neural network, then the knowledge of the backpropagation algorithm leads it to increase and decrease connection weights in ways that decrease the network's output error. In this case, the *intelligence* lies in a very basic understanding of the effects of change in a system of interlinked equations. In either case, the intelligence is representation dependent: the backpropagation algorithm could not handle a logical disjunction

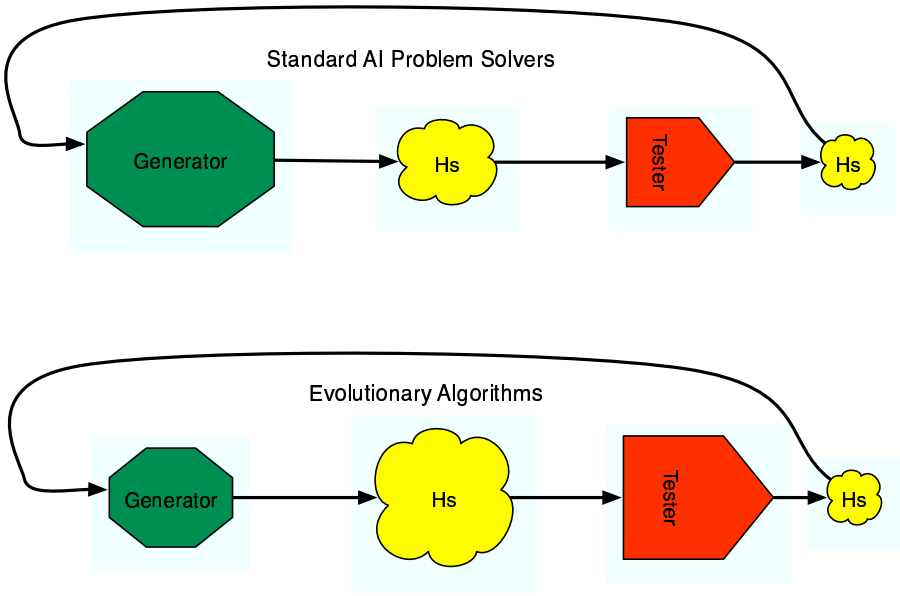

Figure 6: The generate-and-test view of problem solving with respect to both evolutionary computation and more traditional AI approaches. The size of each octagon and pentagon indicates the amount of problem-relevant knowledge that it employs, while the size of the Hs cloud denotes the number of active hypotheses under consideration.

written in standard propositional or predicate calculus, and a logical engine would be lost if given a neural network.

Evolutionary Algorithms have very little intelligence in the generator, although this varies among the EA types. In general, the greater the effort required to convert genotypes to phenotypes, the less intelligent the EA will appear in terms of its ability to generate good hypotheses.

Interestingly enough, the fact that EAs *permit* a large representational gap between genotypes and phenotypes allows the EA generator to operate relatively independently of the high-level phenotypic representation. For example, the same EA can mutate and recombine bit-string genotypes that encode everything from logical expressions to neural networks to bayesian probability tables at the phenotypic level. In this sense, EAs can be extremely representation independent. But, again, the manipulations made to the genotypes have no guarantee of producing improved phenotypes. Representation independence has a high price, but one that many are willing to pay when problems become so complex that the biases imposed by representaton-dependence prevent AI systems from finding a satisfactory solution.

From a different perspective, standard AI problem solvers include a good deal of information about *how* a good solution should be created and may even possess meta-knowledge about *why* it performs certain hypothesis manipulations. This bias helps avoid the generation of bad hypotheses; and the *why* knowledge may even enable the system to explain its choices to a human user. In contrast, an EA has little *how* or *why* information but a good deal of knowledge about *what* a good solution is. This is exactly how nature works: many random variations are tried, and the

environment determines who survives based solely on *what* they can do. Neither creationists nor classical engineers are comfortable with this approach to problem solving, but as biologists and evolutionary computationalists can document, it works!

References

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*, The MIT Press, Cambridge, MA, 2 ed., 1992.

[2] K. D. Jong, *Evolutionary Computation: A Unified Approach*, MIT Press, Cambridge, MA, 2006.

[3] D. Mostow, *Machine transformation of advice into a heuristic search procedure*, in Machine Learning: An Artificial Intelligence Approach, Morgan Kaufmann, 1983, pp. 367–403.