EA Appendices

January 19, 2014

1 Implementing an Evolutionary Algorithm

Figure 1 illustrates the basic flow of data structures (typically implemented as objects) in an Evolutionary Algorithm.

The evolutionary process begins with a population of genotypes as shown in the bottom left corner of the diagram. These are normally generated randomly, although some EA applications involve strong biases during initialization due to known or predicted structure in the solutions.

Phenotypes are then derived from the genotypes via the developmental phase. The genotypes themselves are retained for later use in reproduction, just as organisms retain their DNA throughout life.

Fitness testing of the phenotypes then assigns a fitness value to each individual, as depicted by the numbers in each cloud. This value reflects the ability of the *phenotype* to solve the problem at hand, but it determines the prospects for the individual to pass elements of its *genotype* into the next generation.

As described above, adult selection may weed out inferior (i.e., low fitness) offspring, allowing admission into the adult pool to only the better performers. Parent selection then chooses individuals whose genotypes (or portions thereof) will be passed on to the next generation. In theory, all adults remain in the adult pool until the next round of adult selection, when all or some of them may get bumped out to make room for promising new adults.

Finally, the genetic material of the chosen parents is recombined and mutated to form a new pool of genotypes, each of which is incorporated into a new object. These objects then begin the next round of development, testing and selection.

The cycle halts after either a) a pre-determined number of generations, or b) one of the individuals has a fitness value that exceeds a user-defined threshold. The latter is only practical when the user has concrete knowledge of the fitness value that corresponds to an optimal solution. For example, in a difficult travelling salesman problem, the value of an optimal tour may not be known ahead of time, but in a constraint-satisfaction problem such as a Sudoku, the optimal score is easily ascertained without detailed knowledge of the actual solution.

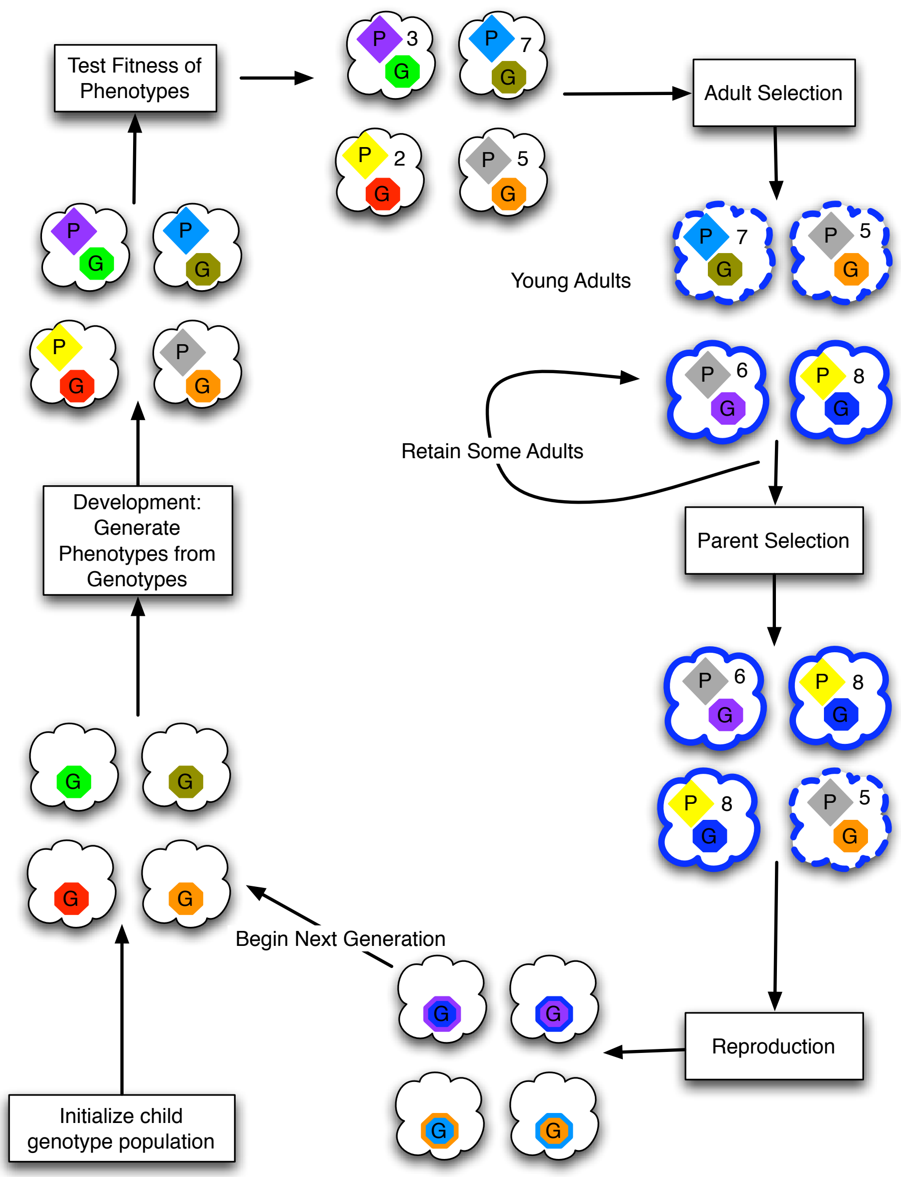

Figure 1: Flow of individual computing objects (clouds) through an evolutionary algorithm. Objects begin with only a genotype, then acquire a phenotype via development and a fitness value via performance testing. Fitness then biases selection of both adults and parents/mates.

2 Essential Components of an Evolutionary Algorithm

To facilitate the process of Figure 1, an EA designer must include the following basic components:

1. Genotype representation
2. Phenotype representation
3. Translator of genotypes into phenotypes
4. Genetic operators for forming new genotypes from existing ones.
5. Fitness assessment procedure
6. Selection strategy for child phenotypes
7. Selection strategy for adult phenotypes

The first four of these components are often discussed together, since a) the genotypic level is, by definition, that for which we define genetic operators, and b) the choices of genotypes and phenotypes are strongly influenced by the potential ease/difficulty of both the translation mechanisms and genetic operations. Therefore, the upcoming section on genotypes and phenotypes is interspersed with discussions of points 3 and 4 as well.

Several of these 7 components are general and hence reusable across many different problem domains. Typically, the selection strategies are extremely modular and highly problem independent. A good EA has a library of a half dozen or so (see below) strategies which can be tested to find the best fit for a given situation.

At the other extreme, the fitness assessment procedure is always very problem dependent, since it embodies the key knowledge as to *what* a solution to a particular problem should do.

Phenotypes also have a tendency to be problem dependent, since they embody much of the semantics of the domain. However, there is still possibility for code reuse at the phenotypic level. For example, a general phenotypic type such as *permutation of integers 1..N* could be used to represent solutions to both an N-city travelling salesman problem (TSP) and an N-task ordering problem. In this case, the only difference would be in the fitness assessment procedure.

Genotypes are often amendable to reuse, particularly the very low level variants such as bit vectors. For these genotypes, general genetic operators are easily written. As genotypes move to higher levels, i.e., move closer to the phenotypic representations, their generality declines and the need for special purpose genetic operators often arises.

In general, a good EA provides a solid backbone of code for running the basic evolutionary loop along with a library of reusable phenotypes, genotypes, genotype-to-phenotype translators, and selection strategies. It also includes simple hooks for adding in new representations and translators, preferably as sub-classes to existing ones.

Despite this generality, even a good EA will require that the user understand the basic evolutionary procedure in order to tune parameters such as the population size, selection strategy, mutation and crossover rate, etc. Furthermore, many complex problems will require a specialized phenotypic (and possibly genotypic) representation not included in the off-the-shelf EA. In short, it helps to enjoy programming!

3 Setting EA Parameters

The astute choice of genotype, phenotype, genetic operators and fitness function are normally the difference between problem-solving success and sending the EA off into a hopeless search space with nothing but a broken compass! Thus, at a qualitative level, these decisions are pivotal.

At a quantitative level, other choices can affect the efficiency of an EA; if not considered seriously, they can also ruin the EA's search. These parameters include the population size, the stopping criteria, and the mutation and crossover rates.

Although EA theoreticians have tried for years to produce useful guidelines for selecting these parameter values, the *No Free Lunch Theorem* [**?**] has been the most influential, and it effectively says that you **cannot** make general statements about proper parameter settings for search algorithms: they are strongly problem dependent. So, although an EC guru cannot give you the magic value of population size, mutation or crossover rate for all problems, a domain expert who has applied EAs to specific problems can tell you good parameter settings for EAs in that domain.

For example, if the representational choices have been so difficult that no recombination operator can guarantee high heritability, then a low crossover rate, of say 0.2, might be wise. This indicates that when two parents are chosen for crossover, only 20% of the time will they actually be recombined; in the remaining 80%, they will simply be copied (with possible mutations) into the next generation.

Mutation rates typically come in at least two varieties: per genome and per genome component (e.g., per bit in a bit-vector genome). Depending upon the problem, these may vary from as high as .05 per component (e.g. 5% of all genome components are modified) to .01 per individual (e.g. 1 % of all individuals are mutated in just ONE of their components).

Goldberg [**?**] and De Jong[**?**] provide some useful, general, tips for choosing EA parameters. One of the most critical, and most general, involves the well-known balance between exploration and exploitation [**?**]. To wit, De Jong emphasizes a balance of strength between the explorative forces of reproduction and the exploitative powers of selection.

Reproduction explores the search space by creating unique new genomes; as mutation and crossover rates rise, the extent of this exploration increases. Selection mechanisms control exploitation via the degree to which they favor high-fitness over less-impressive genotypes. De Jong's key message is that if reproductive exploration is high (e.g., the mutation rate is high), then selection should be highly exploitative as well. Conversely, low reproductive exploration should be complemented with weak selection.

This makes intuitive sense. If the EA is generating many unique genotypes in each generation, then in any reasonably-difficult search space, most of those genotypes will have lower fitness than the parents. Hence, it makes sense to filter them somewhat ruthlessly with a strong selection mechanism, such as tournament selection with low ϵ and high K. But if reproduction produces only a few unique individuals, then weak selection is required to give those new children a fighting chance; otherwise, evolution will stagnate. Note that in both cases, the end result of a reproductive step followed by a selective step should be approximately the same amount of *innovation* in the next generation. The bottom line is that the *absolute* mutation and crossover rates are less important than their exploratory strength *relative* to the exploitative degree of selection.

Also, the population size of an EA deserves careful consideration. Normally, a large population size of 100, 1000 or even 500,000 is desirable for hard problems. Unfortunately, computational resources generally restrict practical population sizes to the 1000-10000 range, although, again, this is highly problem dependent. Some phenotypes, such as solutions to 20-city travelling-salesman problems or 20-node map-coloring problems, are easily checked for fitness, and thus, large populations can be simulated over many generations in just a few seconds of run time. Other phenotypes, such as electronic circuits or robot controllers, require extensive simulations (if not live runs of real robots) to test fitness. This can greatly reduce the practical population size to values as low as 10 or 20. Higher values may take weeks to run 50 or 100 generations!

The different branches of Evolutionary Computation (discussed below) have varying philosophies on population size. In general, Evolutionary Strategies (ES) and Evolutionary Programming (EP) researchers tend to use very small populations of 20 or less; some use a single individual! Genetic Algorithm (GA) and Genetic Programming (GP) aficianados frequently prefer large populations of hundreds or thousands. In general, to achieve the full power of parallel stochastic search in tough solution spaces, large populations are necessary. But to find a satisfactory combination of 30 parameters, for example, a small population of only 10 or 20 may be sufficient.

Finally, the stopping criteria for an EA must be determined. One can either a) set in a known (or estimated) maximum fitness as a threshold and stop simulation when a genotype achieves it, or b) simply set a predetermined generation limit, G, and run until then. Both are trivial aspects of the EA, and the optimal choice is easily determined via experience in the problem domain. In general, if multiple runs of the same EA (on the same problem) are being performed in order to accurately assess the EA's problem-solving efficiency, then considerable run-time can be saved by cutting a run when fitness reaches a threshold value.

The tuning of EA parameters can often use up a significant fraction of your total project time. It is not unusual to hack together an EA in a few days (or hours) but to then spend an order of magnitude more time to actually get it to find good solutions. Start early! Be patient!

4 Fitness Assessment

In population biology, *fitness* generally refers to an individuals ability to produce offspring [**?**]. In evolutionary computation, such a definition would appear circular, since EAs use fitness values to **determine** reproductive success. EAs, unlike biologists, cannot watch a population reproduce and then afterwards assign fitness to the productive individuals. EAs must be more proactive and *play God* by stepping in and restricting access to the next generation. The fitness evaluation is the first (and main) step in that process.

As discussed earlier, an EA generally has little knowledge about *how* a good solution/hypothesis should be designed, but solid information about *what* a good solution should be. Most of that information is implicit in the fitness function. Whereas the phenotypic representation and the genotype-to-phenotype mapping embody knowledge of how to generate a *legal* individual, it is the fitness function that assesses the individual's *goodness* with respect to the problem. So the extent of *how-to-construct* knowledge in the EA is normally restricted to legal phenotypes, not superior ones.

Fitness assessment involves two steps: performance testing and fitness assignment. In the former, the phenotype is applied to the problem to be solved by the EA. The results of this are then converted into a quantitative fitness value, which then follows the individual around like a college grade-point average to open (and close) doors to the future. That is, the selection mechanisms use the fitness value to prioritize and filter hypotheses.

In the curve-fitting GP example, a typical performance test goes through each of the n x-y pairs, (x_i, y_i). The value x_i is used as input to F_j, the complete function defined by phenotype j, producing output $\tilde{y}_i$, which is then compared to y_i to compute a contribution to the total error, E_j. The sum-of-squares is a typical error function for this purpose:

$$E_j = \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2 \tag{1}$$

The fitness of phenotype j could then be as simple as:

$$Fitness(F_j) = \frac{1}{1 + E_j} \tag{2}$$

In the robot example above, the performance test would involve a simulated 2d block world with red and blue objects. Each phenotype (i.e., set of behavioral rules) would be downloaded into a simulated robot that would then move around this world under the control of these rules. Initially, the red and blue objects would be randomly spread about the plane. The robot would then run for a number of timesteps, say 1000. Upon completion, the state of the world would be analyzed for certain factors of relevance to a solution. These factors are, ideally, necessary and sufficient conditions for problem success, but such conditions are not always easy to define. But in this case, our definition of the task is fairly straightforward, so a good final world state should have a) all red objects surrounded by many blue objects b) few blue objects standing alone in the plane.

To quantify this, for each red object, count the number of blue objects that are within a short distance d of its center, a.k.a., *warning blues*. The initialization procedure might insure that no blue objects are within d of any red object. By itself, this warning-blue count could be a fairly effective fitness value. In addition, the EA could take into account the isolated blue objects and, say, subtract their count from that of the warning-blues. In most cases, it is useful to avoid negative fitness values, so the isolated-blues count might be divided by a scaling constant, or all negative results could simply be mapped to zero fitness.

For a travelling-salesman problem (TSP), the phenotype would probably encode the proper sequence of cities to visit: first Dallas, then Denver, then Portland, etc. The performance test would then *walk through* that route and tally up the total distance. A standard fitness value would then be the inverse of this distance.

If an EA were used to design a good classifier for a machine-learning system, then the performance test would involve testing each phenotype classifier on a training set of (input, desired-output) pairs. The classification error, E, of a phenotype - i.e., number of the classifiers outputs that differed from the desired outputs - could then be the basis of its fitness, F. For example, $F = \frac{1}{1+E}$.

Error is also a natural fitness metric for scheduling problems. Assuming that the EA must design a good assignment of college exams to discrete time slots and classrooms, then the phenotype would consist of times and rooms for each exam. Then, given a list of all students, professors and the courses that each takes/gives, the performance test would measure error as the number of violations of hard (e.g., no student or professor should have two exams at the same time) and soft (e.g., no student should have two exams on the same day) constraints.

In general, the exact fitness value matters very little, but the *relative* fitnesses of hypotheses must reflect the relative utility of their solutions. If one individual receives a fitness of 10, while another receives 2, then in the eyes of the system designer, the former should be approximately 5 times *better*. Failure to achieve appropriate fitness spacing among individuals can cause evolutionary search to sputter. However, as discussed in the section on selection mechanisms, there are certain phases of the search when it is desirable to artificially widen or narrow the effective fitness gaps.

5 Natural and Artificial Selection

Whereas fitness assessment should provide a very objective, unbiased measure of an individual's quality, selection strategies often introduce stochasticity into the processes of survival and mating. Just as in nature, where the strongest and fastest may accidentally fall off a cliff, meet an oncoming train, or simply have a couple *off nights* during mating season, there is often no guarantee with EAs that either a) the most fit individuals will pass on the most genes, or b) the worst fit individuals will not reproduce at all. This is generally an advantage, since high-fitness individuals may have hidden weaknesses or represent merely local optima, while low-fitness genotypes may encode useful traits that have not yet been complemented with enough other traits to show their true utility. In short, an EA normally profits by keeping some options open: by maintaining a good balance between exploration and exploitation.

5.1 Fitness Variance

In theoretical evolutionary biology, classic results by Sewall Wright and Ronald Fisher led to the *Fundamental Theorem of Natural Selection* [**?**, **?**], which states that the rate of evolution (measured as the rate of change of the composition of the gene pool) is directly proportional to the **variance** of individual fitness. So evolution requires fitness variation, and the greater the variation, the faster the population evolves (and adapts to its environment).

Noting that fitness in biology refers to reproductive efficacy, the fundamental theorem implies that evolution can only occur when individuals have different reproductive rates. If each individual produces one offspring (or each mating pair produces 2), then the gene pool will stagnate.

The same principle applies to evolutionary algorithms: If all individuals have the same fitness, then they will all have approximately the same likelihood of passing on their genes, and the population will not change much from one generation to the next. Hence, for evolution to proceed, a sizable fitness variance should be maintained.

Unfortunately, EAs often suffer from a fitness variance that is either too high or too low. In the early generations of an EA run, most of the individuals tend to have (very) low fitness, while a few will have more respectable values simply by virtue of doing a few things correctly (often by accident). For example, a robot that always picks up blue blocks and then deposits them anytime it sees other blocks, whether blue or red, will score better than one that never picks up anything. This results in a very high fitness variance, since most of the individuals will have zero or near-zero fitness, while a select few will have values reflecting *decent* performance, but nothing exceptional.

The problem is that the low fitness individuals will normally have almost no chance of reproducing, while the respectable genotypes will quickly dominate the population. Hence, the EA will *prematurely converge* to a homogeneous population of mediocre individuals, a local optima. This homogeneity implies very low fitness variance, so evolution will grind to a halt and the EA's best-found solution will be far from optimal. To remedy this situation, the fitness variance should be artificially reduced during the early stages of an EA run.

Conversely, near the end of a run, the population will often converge on an area of the fitness landscape that, indeed, does hold an optimum or near-optimum solution, but since all individuals are quite good, there will be little fitness variance and thus no driving force for continued improvement. The population will essentially run out of useful hints regarding the goal. In these cases of *late stagnation*, the fitness variance should be artificially increased so that *the cream of the cream* gets a slight, but significant, reproductive edge.

These artificial modifications to fitness variance are achieved by selection mechanisms, which often scale fitness values before picking and pruning parents. In effect, this alters the fitness landscape, as shown in Figure 2.

5.2 Selection Pressure

The concept of *selection pressure* ties directly to the scenario of Figure 2. In effect, it is the degree to which the real fitness variance translates into differences in reproductive success. A high selection pressure will give a strong reproductive advantage to individuals that are only slightly better than their peers, while a low selection pressure will treat individuals more evenly, despite fitness differences. So the example of Figure 2 reflects an initially low selection pressure, with a high selection pressure toward the end. This strategy of selection-pressure change is often ideal for an EA.

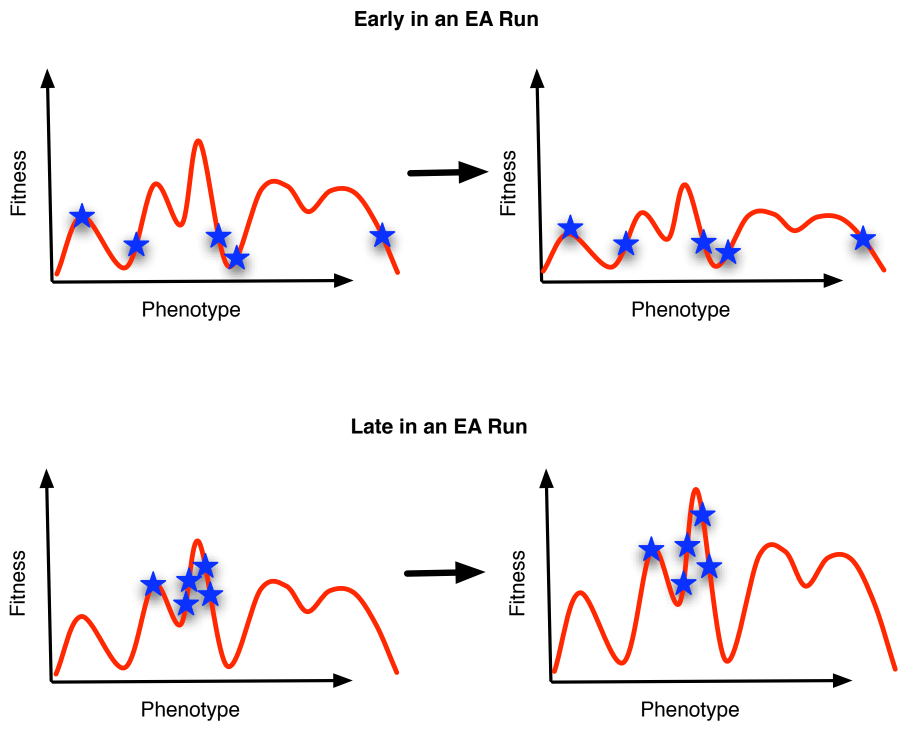

Figure 2: Virtual modifications to the fitness landscape performed by the fitness scaling often incorporated into parent-selection mechanisms. Stars represent individuals. In early stages of evolution, fitness variance is artificially decreased, which implicitly compresses the fitness landscape. In late stages, variance is artificially increased, leading to an implicit stretch of the landscape.

6 Selection Strategies for Evolutionary Algorithms

In the classic EA reference literature [?, ?, ?], the terminology of selection varies a bit, but the basic concept of a selection mechanism or strategy is normally the same. They involve the filtering and biasing of individuals jockeying to *mature* and *reproduce*.

Referring back to the evolutionary algorithm cycle (in evolalgs.pdf), a selection strategy must be defined for both *adult selection* and *mate selection* (a.k.a. *parent selection*). The former typically involves the immediate removal of a (possibly empty) subset of child genotypes, along with the filtering of some (or all) parents from the previous generation. The remaining children and previous adults then constitute the new adult population. Conversely, mate selection constitutes repeated choices of adult genotypes, which are then used to produce the next generation of genotypes. Mate selection often involves the scaling of fitness values as the basis for the stochastic choice of parents.

6.1 Adult Selection

As shown in Figure 3, there are three main protocols for adult selection:

1. *A-I: Full Generational Replacement* - All adults from the previous generation are removed (i.e., *die*), and all children gain free entrance to the adult pool. Thus, selection pressure on juveniles is completely absent.
2. *A-II: Over-production* - All previous adults die, but m (the maximum size of the adult pool) is smaller than n (the number of children). Hence, the children must compete among themselves for the m adult spots, so selection pressure is significant. This is also known as (μ, λ) selection, where μ and λ are sizes of the adult and child pools, respectively.
3. *A-III: Generational Mixing* - The m adults from the previous generation do not die, so they and the n children compete in a free-for-all for the m adult spots in the next generation. Here, selection pressure on juveniles is extremely high, since they are competing with some of the best individuals that have evolved so far, regardless of their age. This is also known as $(\mu + \lambda)$ selection, where the plus indicates the mixing of adults and children during competition.

6.2 Parent Selection

For parent/mate selection, a host of mechanisms are available. These vary in the degree to which individuals compete locally or globally and by the type of fitness scaling that may occur prior to filtering. With *local* selection mechanisms, such as tournament selection, individuals participate in a competition with only a small subset of the population, with the winner moving immediately to the mating pool. With *global* selection mechanisms, an individual implicitly competes with every member of the adult population.

6.2.1 Global Selection Mechanisms

Many of the global selection mechanisms involve a *roulette wheel* on which each adult is alloted a sector whose size is proportional to the adult's fitness. In asexual reproductive modes, if n children are to be produced, the wheel is spun n times, with the winning parent on each spin sending a (possibly mutated) copy of its genotype into the next generation. In sexual reproductive schemes, pairs of wheel spins are employed, with the two winner parents passing on their genotypes, which may be recombined and mutated, normally yielding 2 children. Repeating this process n/2 times yields the complete child population.

Naturally, fitness values must be normalized in order to divy up the area of the roulette wheel. In addition, almost all global selection mechanisms scale the fitness values prior to normalization. These scaling techniques are typically the defining feature of the selection mechanism.

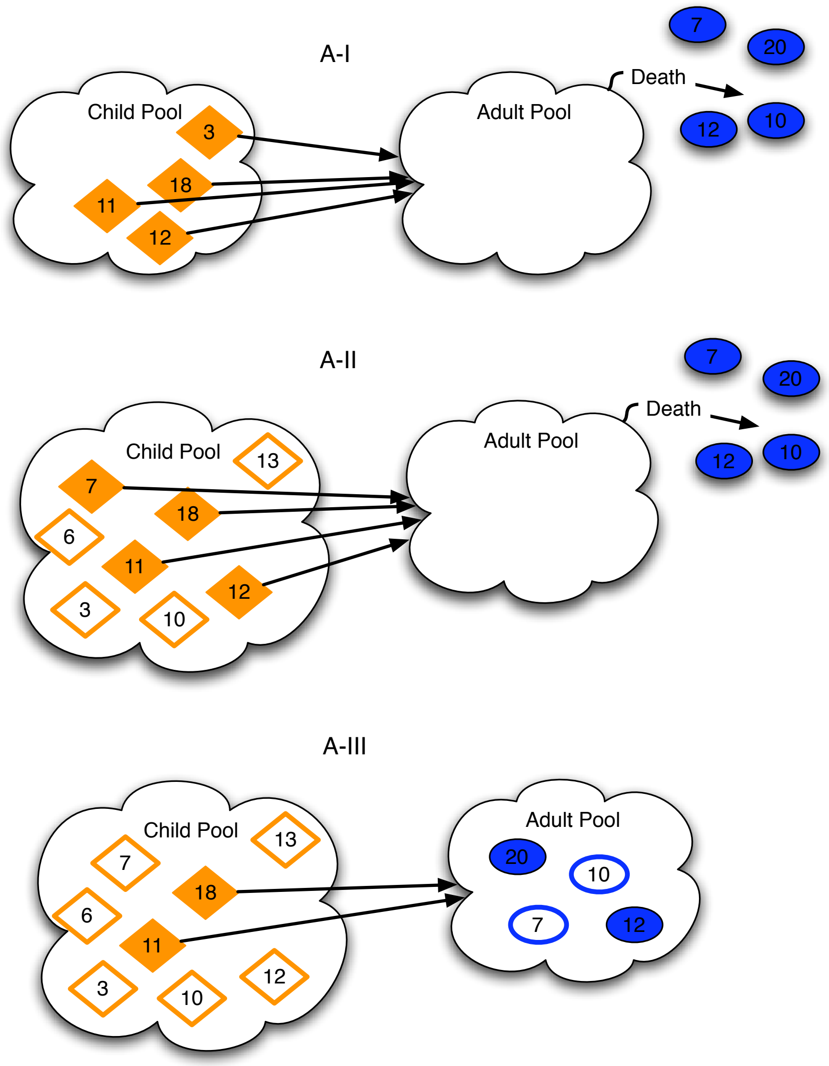

Figure 3: Three basic protocols for adult selection. Top: Complete turnover of adults with m = n. Middle: Complete turnover of adults with n > m. Bottom: Adults and children compete equally for the m adult spots. In each diagram, numbers denote fitness, while unfilled diamonds (children) and circles (adults) indicate individuals who lose out in the competition. Arrows emanating from diamonds indicate children who win acceptance into the adult pool.

Our overview of basic selection mechanisms uses Holland's [?] concept of *expected value*: the expected number of times that the parent will reproduce. The original fitness values are scaled into these expected values prior to roulette-wheel normalization. We also closely follow the description of selection mechanisms given by Mitchell [?].

The roulette-wheel metaphor is best explained by a simple example. Assume a population of 4 individuals with the following fitness values: 1) 4 2) 3 3) 2 4)1. To convert these to space on the roulette wheel, simply divide each by the sum total of fitness, 10. By *stacking* the resulting fractions, the individuals each get a portion of the [0, 1) number line. The sub-ranges for each are: 1) [0 0.4), 2) [0.4, 0.7), 3) [0.7, 0.9) 4) [0.9, 1.0). These are equivalent to sectors on a roulette wheel. Selecting a parent becomes a simple weighted stochastic process wherein a random fraction, F, in the range [0, 1) is generated. The sub-range within which F falls determines the chosen parent. Clearly, individuals with higher fitness have a greater chance of selection.

The classic selection mechanism is *fitness proportionate*, in which fitness values are scaled by the average population fitness. Of course, dividing m fitness values by their average and then normalizing is equivalent to simply normalizing the original m values. Hence, this mechanism merely scales fitnesses so that they sum to 1 and thus properly fill up the roulette wheel; it does not modify their relationships to one another. In other words, it does not alter the selective advantages/disadvantages inherent in the original values and thus does not implicitly change the fitness landscape.

Mitchell and others often use the roulette-wheel metaphor as a unique feature of fitness proportionate selection, but it applies equally well to many global selection mechanisms, since most work by normalizing all expected values and then using randomly-generated fractions to simulate the *spinning* of the wheel and choice of a parent.

Sigma scaling selection successfully modifies the selection pressure inherent in the raw fitness values by using the population's fitness variance as a scaling factor. Hence, unless this variance is 0 (in which case all fitnesses scale to expected values of 1.0), the conversion is:

$$ExpVal(i,g) = 1 + \frac{f(i) - f\bar{(g)}}{2\sigma(g)} \tag{3}$$

where g is the generation number of the EA, f(i) is the fitness of individual i, $f\bar{(g)}$ is the population's fitness average in generation g, and $\sigma(g)$ is the standard deviation of population fitness.

This has the dual effects of a) damping the selection pressure when a few individuals are much better (or worse) than the rest of the population, since such cases have a high $\sigma(g)$, and b) increasing the selection pressure when the population has homogeneous fitness (thus low $\sigma(g)$). This helps avoid the problems of early and late stages of EA runs, as discussed above. Figures 5 and 6 illustrate these effects, showing the clear advantage of sigma-scaling over fitness-proportionate scaling in effectively modifying the selection pressure inherent in the original fitness values.

Boltzmann selection is based on simulated annealing, in which the degree of randomness in decision making is directly proportional to a temperature variable. This is based on the physical property that molecules in a heated mixture exhibit more random movement (especially when it transitions from solid to liquid or liquid to gas) than under cooler conditions. With Boltzmann selection, higher (lower) heat entails a more (less) random choice of the next parent and hence less (more) selection pressure, since superior individuals have less (more) of a guarantee of passing on their genes.

The scaling equation for the Boltzmann selector is:

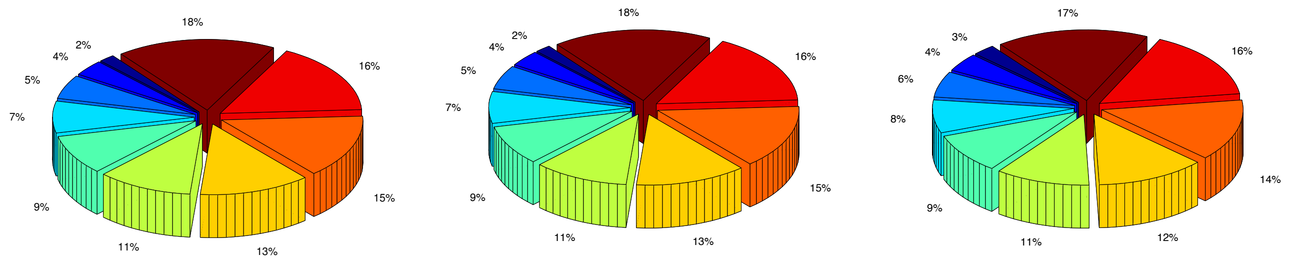

Figure 4: Comparison of selection wheels using (left) unscaled, only normalized, fitness values, (middle) fitness-proportionate scaling and normalization, and (right) sigma-scaling and normalization. The original fitness values in all 3 cases are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

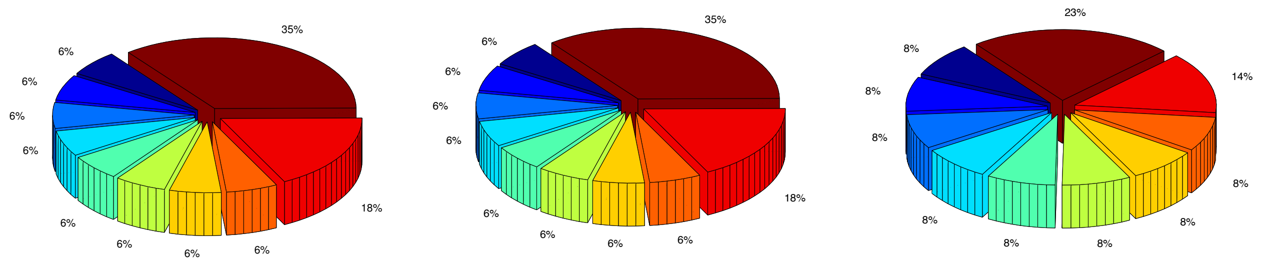

Figure 5: Comparison of selection wheels using (left) unscaled, only normalized, fitness values, (middle) fitness-proportionate scaling and normalization, and (right) sigma-scaling and normalization. The original fitness values in all 3 cases are 1, 1, 1, 1, 1, 1, 1, 1, 3, 6, which are typical of an early generation of an EA run. Note the more even distribution (i.e., lower selection pressure) for sigma scaling.

$$ExpVal(i,g) = \frac{e^{f(i)/T}}{\langle e^{f(i)/T} \rangle_g} \tag{4}$$

where g is the generation, T is temperature, f(i) is the original fitness of individual i, and $\langle e^{f(i)/T} \rangle_g$ is the population average of the fitness exponential during g.

As shown in Figure 7, as temperature falls, the odds of choosing the best-fit individual increase dramatically as it garners more and more area on the roulette wheel. Ideally, a Boltzmann selector uses a temperature that gradually decreases throughout the EA run, such that selection pressure gradually increases. Again, this ameliorates premature convergence and late stagnation.

Rank selection ignores the absolute differences between fitness values and scales them according to their relative ordering. Hence, if the best individual in the population has a fitness of 10, while second place is a 3, then the 10 will achieve no more roulette-wheel area than would a 3.1. This type of scaling also helps adjust selection pressure. It decreases the advantage of *lucky starters* during early stages of a run, and it tends to add some spacing between individuals when population fitness becomes very homogeneous. This helps combat the problems of premature convergence and late stagnation discussed above.

The basic scaling equation for rank selection is:

$$ExpVal(i,g) = Min + (Max - Min)\frac{rank(i,g) - 1}{N - 1} \tag{5}$$

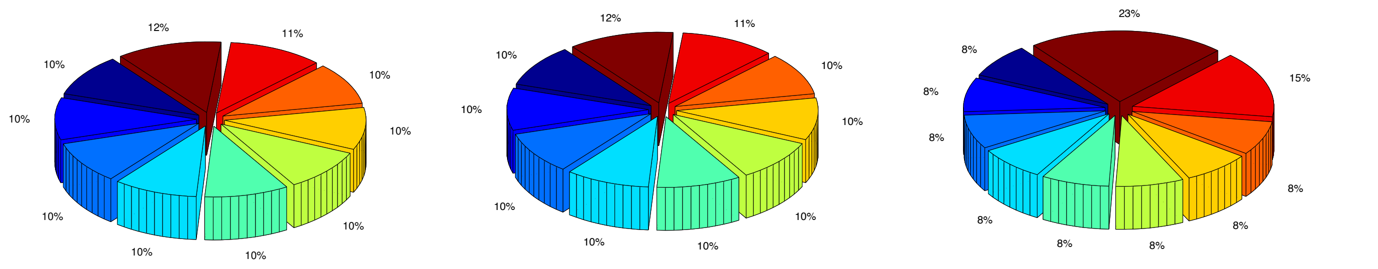

Figure 6: Comparison of selection wheels using (left) unscaled, only normalized, fitness values, (middle) fitness-proportionate scaling and normalization, and (right) sigma-scaling and normalization. The original fitness values in all 3 cases are 8, 8, 8, 8, 8, 8, 8, 8, 9, 10, which are typical of the later stages of an EA run. Note the more skewed distribution (i.e., higher selection pressure) for sigma scaling.

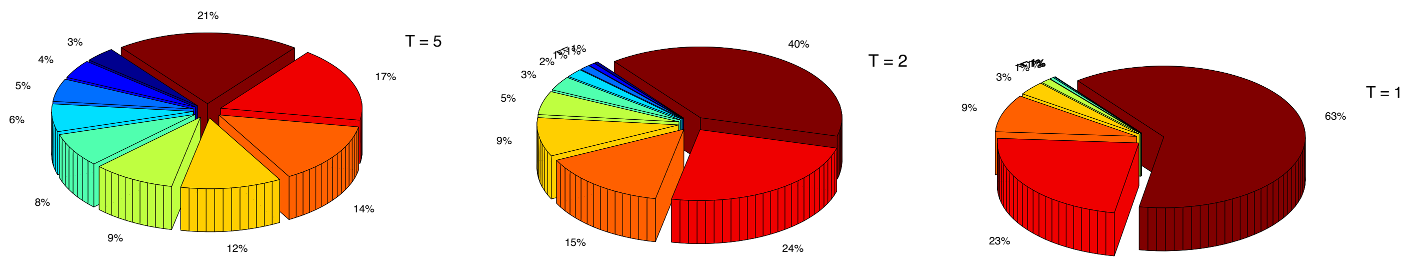

Figure 7: Increasing selection pressure by decreasing temperature in the Boltzmann selector from (left) 5° to (middle) 2° to (right) 1°. The original 10 fitness values are 1, 2, ... 10.

where N is the popuation size, g is the generation, and Min and Max are the expected values of the least and best fit individuals, respectively. rank(i,g) is the rank of the ith individual during generation g, with the least-fit individual having rank 1 and the best having rank N.

Mitchell [?] points out the basic constraints that $Max \geq 0$ and $\sum_i ExpVal(i,g) = N$, from which it is straightforward to prove that $1 \leq Max \leq 2$ and $Min = 2 - Max$:

$$\sum_i ExpVal(i,g) = \sum_{i=1}^{N} Min + (Max - Min)\frac{rank(i,g) - 1}{N-1} = N \cdot Min + (Max - Min)\sum_{i=1}^{N}\frac{i}{N-1} \tag{6}$$

Then,

$$\sum_{i=1}^{N}\frac{i}{N-1} = \frac{1}{N-1}\sum_{i=1}^{N} i = \frac{1}{N-1}\cdot\frac{(N-1)N}{2} = \frac{N}{2} \tag{7}$$

Hence:

$$N \cdot Min + (Max - Min)\cdot\frac{N}{2} = N \tag{8}$$

Solving the above equation for Max, yields $Max = 2 - Min$. Since Min is the expected value of the worst

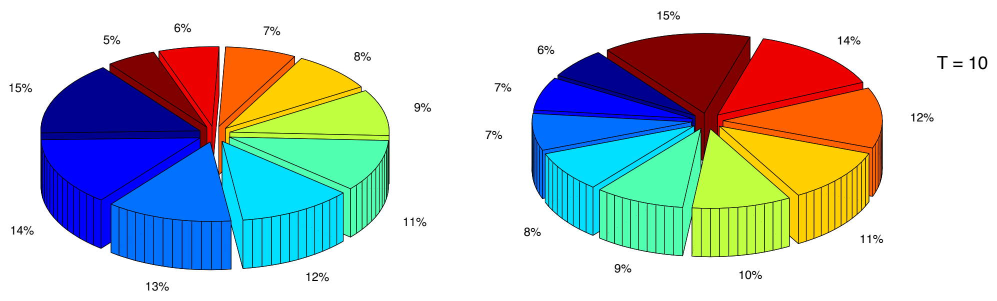

Figure 8: (Left) Normalized, rank-scaled fitness values for 10 phenotypes with original fitnesses 1,2...10. The scaling range is [0.5, 1.5] . (Right) The same fitnesses scaled by a Boltzmann selector using a temperature of 10. Notice that both give a fairly even partitioning of the roulette wheel, despite the high fitness variance. Hence, both exhibit low selection pressure for this wide fitness distribution.

individual, it should lie within the range [0, 1]: at best, the worst individual should get one copy of its genome in the next generation. Thus, $1 \leq Max \leq 2$.

As a typical setting, Min = 0.5 and Max = 1.5, or, for the maximum selection pressure possible with rank selection: Min = 0 and Max = 2.0.

Figure 8 shows that a rank selector with scaling range [0.5, 1.5] behaves similarly to a Boltzmann selector with temperature = 10. Although the color schemes are reversed in the two roulette wheels, the corresponding proportions match up well, and both exhibit a clear reduction of selection pressure (i.e. smoothing of selective advantages) for the heterogeneous fitness scenario.

For any roulette-wheel selectors, a *universal* variant is possible, wherein N pointers are evenly spaced about the wheel. Then, with just one spin, the N parents for $\frac{N}{2}$ pairings are chosen. This approach removes the random possibility of good individuals dominating the mating to a much greater degree than sanctioned by the scaled fitness distribution. The next generation thus *holds true* to that distribution.

Finally, the term *uniform selection* refers to situations where, in effect, there is no selection pressure: all individuals have exactly the same chance of being chosen. In *deterministic uniform selection*, each parent gets to produce the same number of children, while *stochastic uniform selection* involves a roulette wheel on which all adults have equal area. EAs that use uniform selection at one level, such as during mate/parent selection, must typically use some form of fitness-based selection elsewhere.

6.2.2 Local Selection Mechanisms

The classic local mechanism is *tournament selection*, wherein random groups of K adults are chosen and their fitnesses compared. With a probability of $1 - \epsilon$, the best fit of the K is chosen for the next mating, while the choice is random with a probability of ϵ. The parameter ϵ is a user-defined value. N tournaments are thus required to determine all mating pairs.

Notice that this procedure allows poor genotypes to slip through to the next generation if either a) they get grouped with even worse individuals in a small tournament, or b) they lose a tournament but sneak through via the ϵ-ruling. In general, selection holds truer to the original (global) fitness distribution when both K is large and ϵ is small. In other words, selection pressure is a function of K and ϵ, with low K and high ϵ exhibiting low pressure, and high K and low ϵ giving much stricter selection.

6.3 Supplements to Selection

Although traditionally defined as a selection mechanism in its own right, *truncation* involves the immediate removal from mating consideration of an often sizeable fraction, F, of the adult population. In the classic case, the remaining adults simply produce an equal number of offspring. However, nothing prevents the subsequent application of any global (or local) selection mechanism to choose among these adults. Truncation is quite useful in situations where the genetic operators have a hard time maintaining substantial heritability, and thus, many children from good parents are, nonetheless, very poor performers and should be culled from the population.

Elitism is simply the retainment of the best E individuals in a generation. These are simply copied directly, without mutation or recombination, to the next generation. This prevents EAs from *losing* a superior performer before a better one comes along. This can be very important when population diversity is high, and thus the odds of recreating a similar good individual on each round are lower than when the population has converged upon a good region of the search space. Again, elitism can easily be added to any local or global selection mechanism. A typical value for E is 1 or 2 individuals, or a small fraction (1-5%) of the total population. Higher fractions can often lead to premature convergence.