

## Chapter 2

# Artificial Evolution

**Abstract** Solving design problems in materials science requires a general framework that connects target behaviors to the microscopic parameters that produce them. Here we argue why artificial evolution is a particularly attractive means to this end. First, we present a brief survey of evolutionary computation with particular emphasis on how concepts have been refined to produce state of the art optimization schemes. We present two derivations of contemporary evolutionary optimizers: one from a heuristic, computational point of view and the other biologically motivated. Finally, we use the latter to examine new, deep connections between evolutionary optimization, game theory, and information geometry.

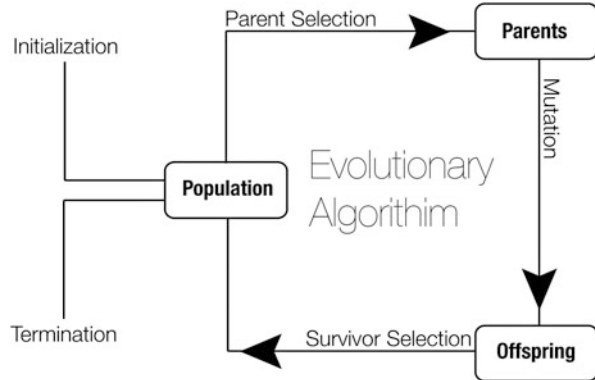
### 2.1 Introduction

Evolution is arguably the simplest picture of a design. It provides a flexible framework to create objects with desired behavior, without requiring explicit problem knowledge. One simply starts with a collection (or population) of possible solutions, and determines how well each candidate solves the problem. A potentially better population can then be generated by taking the some of the best performing objects and manipulating them to produce new candidates (Fig. 2.1).

Transforming this concept into an algorithm requires some careful choices about what is meant by mutation, best members, or even a population. In fact, picking one definition against another is what sets the boundaries between different classes of evolutionary algorithms (Eiben and Smith 2003). Yet once these terms are defined, the basic structure of diversification and selection operators acting on a population is always enforced. Furthermore, the key to a successful implementation is often maintaining a balance of power between these two conflicting forces.

To show how these concepts have been progressively refined, we present a brief, historical survey of evolutionary computation. We begin with the algorithms most directly inspired by biology: genetic algorithms (Holland 1975, 1973; de Jong 1975). As originally envisioned, candidate solutions are represented as strings of bits, meant as models for DNA. Mutation takes place by flipping bits, or permuting substrings. When fit objects are selected to propagate, they reproduce

**Fig. 2.1** Evolutionary algorithms abstracted to a flow-chart



recombinantly. That is, bit string fragments are taken from two or more parents and joined together to form an offspring. Altogether these algorithms are called canonical or simple genetic algorithms (Eiben and Smith 2003).

When performing optimizations in continuous spaces, some of these choices can become problematic (Hoffmeister and Back 1991). In particular, issues arise when parameter locality is not explicitly enforced (Rothlauf 2006). For example, the innocuous transformation of the binary string 0001 to 1000 corresponds to moving almost an order of magnitude in distance to the origin when converted directly to a base 10 number. Alternatively, joining two bit strings together in an ad hoc fashion can lead to offspring that lack resemblance to their parents. These large mutations can potentially overpower the effects of selection and undermine the algorithm’s effectiveness. In short, the mapping from bit strings to continuous parameters can obscure the relationship between the fitness of parents and the fitness of offspring.

An obvious solution to this problem is to abandon both the bit string concept and the use of recombination. In this case, mutation becomes the sole diversifying operator and it acts directly on the variables fed to the fitness function. This picture evolution takes a collection of  $\mu$  particles and subjects them each to a random modification, such as a kick of Gaussian noise. The most fit  $\lambda$  objects following the mutation are then copied to rebuild the  $\mu$  sized population, while the least fit are discarded. Developed roughly simultaneously with genetic algorithms, these approaches are termed  $(\lambda, \mu)$  evolution strategies (Schwefel 1993; Beyer and Schwefel 2002; Back et al. 1991).

The efficiency of an evolution strategy essentially reduces to how frequently the mutations generate good, new guesses. An ideal approach would learn to correlate the likelihood of mutations with increases in fitness. Early evolution strategies achieved this goal by introducing mutation variables as part of a candidate object (Beyer and Schwefel 2002). Effectively this allows evolution to take place in “phase space” rather than “configuration space”. For example, a candidate object could be represented by a mean vector of input parameters plus the variance for a Gaussian noise term. An evolution cycle would then consist of randomly perturbing the variance, using the new variance to generate noise, and then adding that noise to the mean vector. This final object is evaluated, allowing information about the effectiveness of the mutation settings to be assessed indirectly.

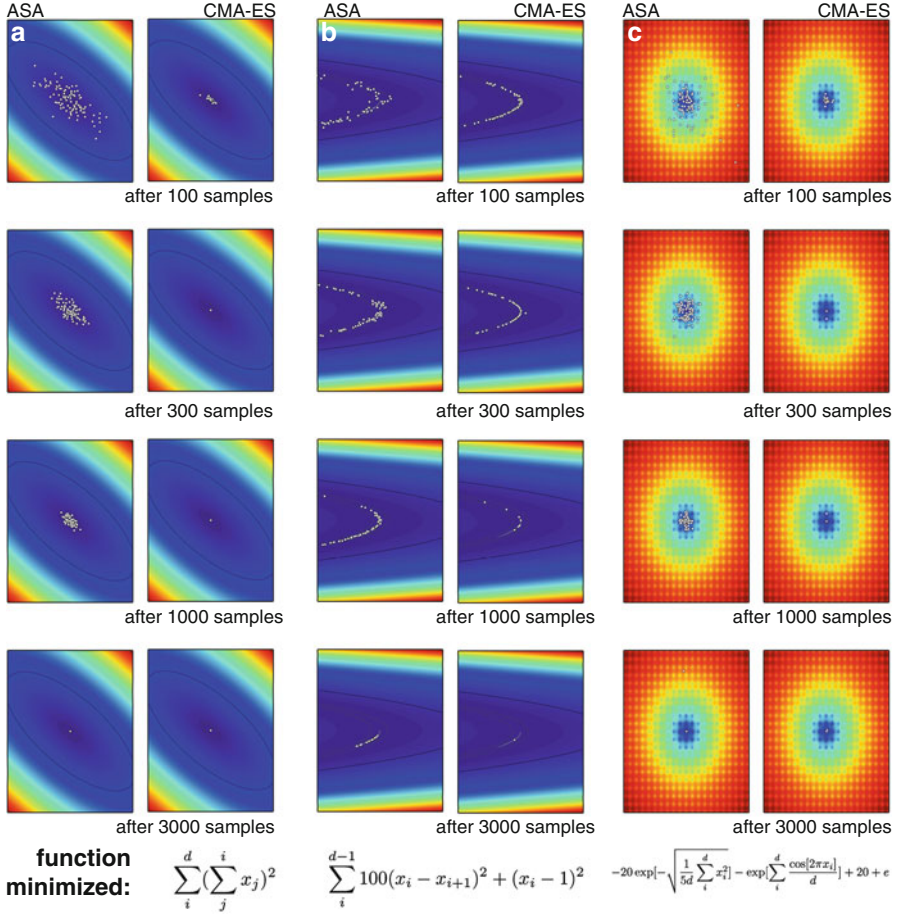
This approach has several attractive features, including proofs of global convergence (Back et al. 1993) for some variants. Moreover, evolution strategies tend to converge faster and more reliably than simple genetic algorithms on a number of continuous parameter test problems (Hoffmeister and Back 1991). However, the indirect mutation parameter updates and the particle representation mean that populations often have to be large, typically consisting of hundreds of individuals. This can make evolution strategies burdensome or impractical for optimization problems where fitness evaluations correspond to expensive computations.

The computational demand associated with artificial evolution can be reduced by orders of magnitude using a more structured approach (Hansen et al. 2003; Wierstra et al. 2008). Rather than defining a population as collection of particles, one could use a probability distribution with a fixed form. With this simplification, adaptation is defined by changing in the distribution parameters. For example, if the population is fixed to be Gaussian, the parameters to adapt are the mean and the covariance matrix. The process of selection becomes tantamount to deciding how information from points sampled should be integrated to adapt these parameters.

Algorithms built around this philosophy represent the state-of-the-art for evolutionary computation. On both test bed and real world problems, methods from this family boat impressive performance and in many cases outperform algorithms like simulated annealing, particle swarm optimization, and the other evolutionary approaches (Hansen et al. 2009). The most impressive algorithm built around this philosophy is the covariance matrix adaptation evolution strategy or the CMA-ES (Hansen et al. 2003).

Figures 2.2 and 2.3 show direct comparisons between adaptive simulated annealing (Ingber et al. 2012) (ASA) and the CMA-ES on three different two dimensional test problems. Each test problem presents a different difficulty that a good black-box optimizer should be able to overcome: the first is non-separable, the second is non-convex and ill-conditioned, and the third is multimodal. An advantage was given to ASA in that its parameters were tuned first to produce quality performance whereas the CMA-ES was run with default parameters only. Yet in all three cases, the CMA-ES significantly outpaces adaptive simulated annealing. For example, Fig. 2.2 shows the best points found by each algorithm when run 100 times with a fixed budget of 100, 300, 1,000 and 3,000 function evaluations. Clearly both algorithms are able to solve all three problems. But what takes ASA thousands of function calls, the CMA-ES achieves in a few hundred.

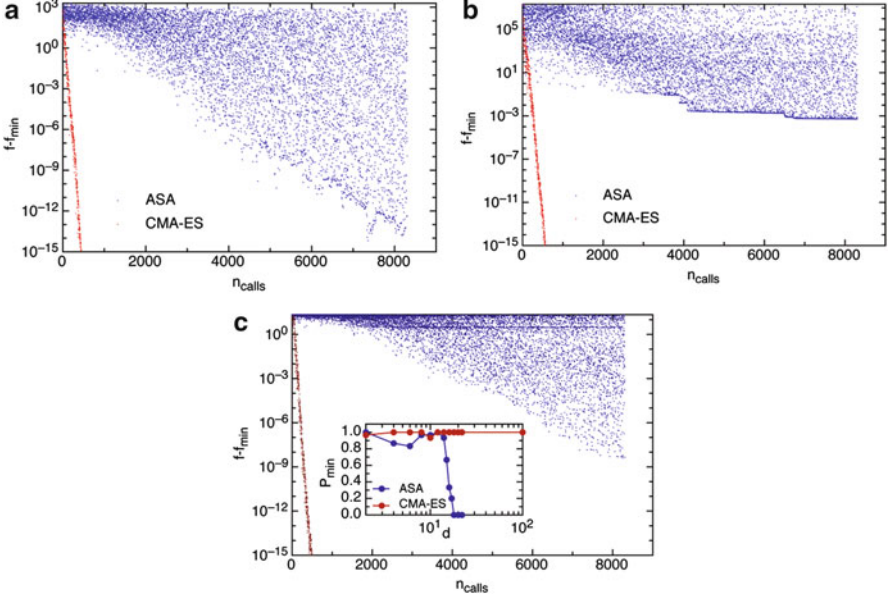
In this work, the CMA-ES is used almost exclusively. The following two subsections deal with explaining its inner workings in more complete detail. Initially, we will derive the basic update rules from an empirical point of view, emphasizing how to build up the CMA-ES from invariance principles and heuristics. We will also discuss the advantages this black-box optimizer affords over others. This practical, but unsystematic derivation for the CMA-ES may leave one wondering what exactly this algorithm has to do with evolution. Thus, the following section derives the same essential ingredients, but from the starting point of evolutionary game theory. This allows us a richer formalism to expand upon later, as well as elucidates some of the more intriguing properties connecting evolutionary computation to game theory and information geometry.



**Fig. 2.2** A comparison between the CMA-ES and adaptive simulated annealing (ASA) on a non-separable problem (a), an ill-conditioned problem (b), and a multimodal problem (c). The *white dots* in each panel correspond to the best object found during an independent run of each algorithm, after being given the number of function evaluations stated below. One can clearly see that in all three cases, both algorithms converge, but the CMA-ES does so significantly faster, typically completing the task after a few hundred evaluations

## 2.2 Deriving the CMA-ES Heuristically

Suppose that we are given a Gaussian function with mean  $\mu_i$  and covariance matrix  $\Sigma_{ij}$  and tasked with finding the point  $x$  that minimizes some unknown function  $g(x)$ . As a first step, we sample  $N$  points using these parameters. Naturally, some points have lower values of  $g(x)$  than others, and so selection should promote these points to appear more frequently in the subsequent distribution. Since both  $\mu_i$  and  $\Sigma_{ij}$  are averages, one way to reflect this is by setting the new mean and covariance



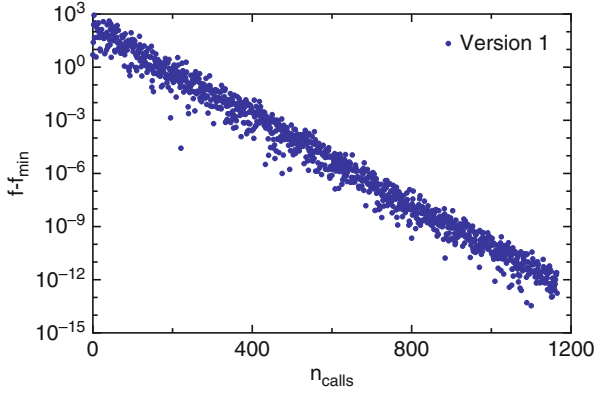
**Fig. 2.3** Function values obtained against the number of evaluations made by the CMA-ES and ASA while solving the test problems shown in Fig. 2.2. Note problems (a) (b) and (c) correspond to the same labels from Fig. 2.2. While both algorithms achieve exponential convergence in terms of the best found solutions against time, the CMA-ES converges at a faster rate. In part, this is because of the fact that one updating cycle for the CMA-ES takes only six function evaluations whereas hundreds must be used by simulated annealing. Plotted in the inset for (c) is the probability of finding the global minimum for the  $d$  dimensional generalization of problem (c):  $f = -20 \exp[-0.2 \sqrt{\frac{1}{d} \sum_i^d x_i^2}] - \exp[\frac{1}{d} \sum_i^d \cos[2\pi x_i]] + 20 + e$ . At first, both algorithms scale to solve the problem in higher dimensions, with typical probabilities near 0.9. But ASA seems to fail in dimensions higher than 20, whereas the CMA-ES continue to find the global optimum even in a 100 dimensional search space without any parameter adjustment

matrix,  $\mu'_i$  and  $\Sigma'_{ij}$ , equal to weighted averages, emphasizing the best points in the generation. In other words, we can update the parameters as:

$$\mu'_i = \sum_x w(x) x_i \quad \Sigma'_{ij} = \sum_x w(x) (x_i - \mu_i)(x_j - \mu_j) \quad (2.1)$$

$$\mu'_i = \mu_i + \sum_x w(x) (x_i - \mu_i) \quad \Sigma'_{ij} = \Sigma_{ij} + \sum_x w(x) [(x_i - \mu_i)(x_j - \mu_j) - \Sigma_{ij}] \quad (2.2)$$

where the last line follows assuming the weights sum to one. The transformation from the first to the second line both is to improve numerical stability and can be generalized to a moving average by multiplying the driving term with a small factor less than one.



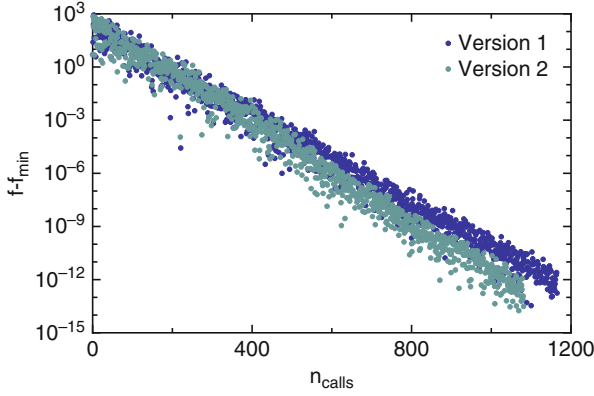
**Fig. 2.4** Using Eqs. 2.1 to minimize the function  $x_1^2 + x_2^2$ . These simple update rules achieve a steady rate of exponential convergence on this and several other test problems

In addition, its very useful to pick fixed weights that only depend on the ranked position of  $g(x)$ . In other words, we pick a set of weights for all time-steps and match points to weights ignoring the magnitude of differences in  $g(x)$  and focusing instead on their ordinal value. The logic here is two fold. First, an ideal optimizer should perform equally well on all rank preserving transformations of the original function because the positions of all the extrema are unchanged. Second, by using a rank based scheme, a steady pressure for improvement is applied, even if sampled values get arbitrarily close to the absolute function minimum.

Remarkably, these simple update rules are fairly good at solving optimization problems in and of themselves. In fact, Fig. 2.4 shows an algorithm using just 2.2 to minimize the parabolic function  $x_1^2 + x_2^2$ . The algorithm proves itself surprisingly efficient: it converges to the minima exponentially, reducing the function value roughly 1.3 orders of magnitude every 100 function evaluations.

Of course there are modifications that might improve the efficiency of this algorithm. After implementing a change, Fig. 2.2 will be revisited to illustrate the significance of each alteration. To keep track of the changes, each modification will be called by a different version number (note that these labels do not correspond to labels in the literature).

An obvious way to improve upon version 1 is to introduce a mechanism for amplification. For example, if every update to the mean points in the same direction, it makes sense to accelerate the rate the covariance matrix expands in that direction. This can be implemented by introducing a kind of damped momentum in the update



**Fig. 2.5** Implementing a momentum addition via Eqs. 2.5 leads to a minor improvement over version 1. Specifically, the rate of convergence appears slightly faster. Thus, it would be difficult to observe any difference if the function were not minimized over several orders of magnitude

equations that stretches the covariance matrix along the typical direction the mean travels. Transforming these concepts into code produces version 2:

$$\mu'_i = \mu_i + \sum_x w(x)(x_i - \mu_i) \quad (2.3)$$

$$p'_i = (1 - c_1)p_i + c_2(\mu'_i - \mu_i) \quad (2.4)$$

$$\Sigma'_{ij} = (1 - c_3 - c_4)\Sigma_{ij} + c_3 \sum_x w(x)[(x_i - \mu_i)(x_j - \mu_j) - \Sigma_{ij}] + c_4 p'_i p'_j \quad (2.5)$$

where  $c_1, c_2, c_3$  and  $c_4$  are positive constants less than 1.

Figure 2.5 compares an implementation of version 1 against version 2, again minimizing the function  $x_1^2 + x_2^2$ . The updated version of the algorithm is technically faster, but not by much. We comment that the modifications to produce version 2 are included here to make a connection with the standard algorithm in the literature. On the basis of experience, however, we believe that updates using Eq. 2.5 are negligibly better than updates with Eq. 2.2, although appreciable more complicated. While these modifications are, as far as the literature goes, components of the complete CMA-ES algorithm, they do not seem to embody essential ones.

A more significant modification can be made by addressing the fact that the adaptations to the covariance matrix simultaneously change the shape and the size. It make sense to decouple these effects, in particular to produce amplification effects in scale when the covariance matrix is generally shrinking or expanding. To do this, one can split  $\Sigma_{ij}$  into two parts so that  $\Sigma_{ij} = \sigma^2 C_{ij}$ . Now,  $\sigma$  can scale  $\Sigma_{ij}$ , while  $C_{ij}$  determines its dominant directions.

The updates for  $\sigma$  can be developed by considering how far the mean actually moves relative to the width of the current covariance matrix. If the mean consistently



moves less than one the standard deviation, it makes sense to start shrinking the width of  $\Sigma_{ij}$ . Conversely, if the mean consistently moves much further than the standard deviation, it makes sense to start scaling  $\Sigma_{ij}$  up. This heuristic can be encoded by introducing a momentum term  $q_i$  for  $\sigma$  and updating both as

$$q'_i = (1 - c_5)q_i + c_6\sqrt{\Sigma^{-1}_{ij}}(\mu'_j - \mu_j) \quad (2.6)$$

$$\sigma' = \sigma \exp[c_7(\frac{\|q'_i\|}{\langle\|N(0, I)\|\rangle} - 1)] \quad (2.7)$$

where the  $c$  terms are again small, positive constants and  $\langle\|N(0, I)\|\rangle$  denotes the average length of a normally distributed vector of the same dimension. The idea here is that the vector  $q_i$  represents a weighted average of  $\sqrt{\Sigma^{-1}_{ij}}(\mu'_j - \mu_j)$  from each step. If most of these moves fall within the standard deviation, then  $(\mu'_i - \mu_i)\Sigma^{-1}_{ij}(\mu'_j - \mu_j) \approx \langle\|N(0, I)\|^2\rangle$ . When this happens, the update rule for  $\sigma$  yields no change, while smaller deviations lead to contraction and larger deviations lead to expansion.

Separating  $\Sigma_{ij}$  into the scale and direction components also requires the removal of any bias in changes to  $C_{ij}$  from adjusting  $\sigma$  independently. This is achieved by changing rules (2.4) and (2.5) into

$$p'_i = (1 - c_1)p_i + c_2\frac{\mu'_i - \mu_i}{\sigma'} \quad (2.8)$$

$$C'_{ij} = (1 - c_3 - c_4)C_{ij} + c_3 \sum_x w(x) \left[ \frac{x_i - \mu_i}{\sigma'} \frac{x_j - \mu_j}{\sigma'} - C_{ij} \right] + c_4 p'_i p'_j \quad (2.9)$$

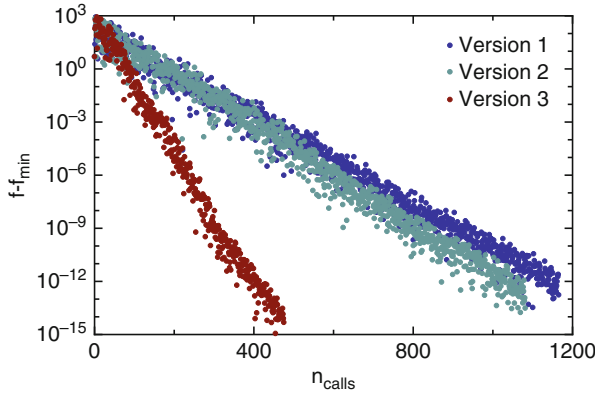
As seen in Fig. 2.6, separating the covariance matrix into shape and scale parameters yields a significant payoff in speed. Indeed, the rate of convergence has been enhanced from 1.3 orders of magnitude per hundred function evaluations to 3. Practically speaking, this means that in the amount of time it takes version 1 to finish a single optimization, version 3 can be run almost three times.

The full algorithm is the complete CMA-ES. For the sake of clarity, it has been reproduced below in pseudocode.

### 2.2.1 Invariance Properties

The justifications used at each step to produce the CMA-ES are extremely heuristic. Thus, it begs the question, why exactly does this algorithm work so well? A popular speculation is that the CMA-ES boasts a large number of invariance properties (Hansen 2006; Ollivier et al. 2011). By invariance, we mean the algorithm's performance will not be significantly impacted by a given problem transformation. These invariances include:





**Fig. 2.6** Decoupling the width of the covariance matrix from its shape produces a significant decrease in the number of function calls needed to complete an optimization. Specifically, the rate of convergence is enhanced by almost a factor of 3

---

### Algorithm 1 the CMA-ES

---

**Require:**  $c, w, \sigma, \mu$ , population size

$p_i \leftarrow 0$

$q_i \leftarrow 0$

$C_{ij} \leftarrow \delta_{ij}$

**while**  $g(x) \geq g_{tot}$  **do**

    Get a population of samples  $x \leftarrow N(\mu, \sigma^2 C)$

    Produce the weighted list  $w(x)$  for the samples given each value of  $g(x)$

$\mu'_i \leftarrow \mu_i + \sum_x w(x)(x_i - \mu_i)$

$q'_i \leftarrow (1 - c_5)q_i + c_6 \sqrt{\Sigma^{-1}}_{ij}(\mu'_j - \mu_j)$

$p'_i \leftarrow (1 - c_1)p_i + c_2 \frac{\mu'_i - \mu_i}{\sigma'}$

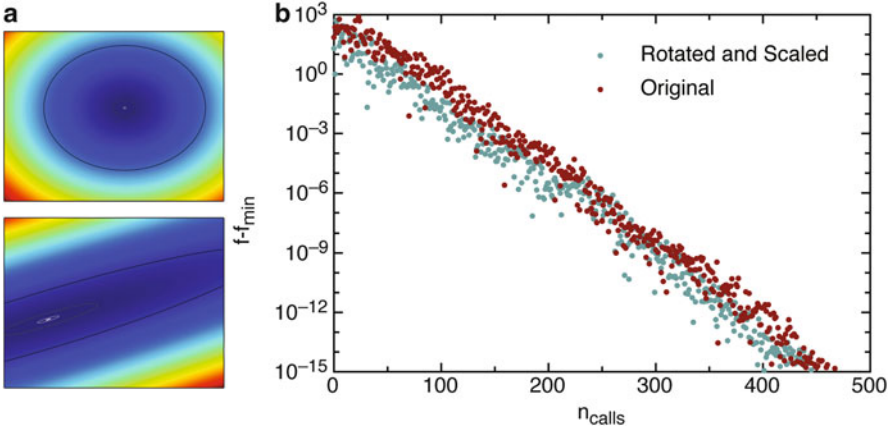
$C'_{ij} \leftarrow (1 - c_3 - c_4)C_{ij} + c_3 \sum_x w(x) \left[ \frac{x_i - \mu_i}{\sigma'} \frac{x_j - \mu_j}{\sigma'} - C_{ij} \right] + c_4 p'_i p'_j$

$\sigma' \leftarrow \sigma \exp \left[ c_7 \left( \frac{\|q'_i\|}{\langle \|N(0, I)\| \rangle} - 1 \right) \right]$

**end while**

---

- Invariance to translations: clearly any serious optimizer must have this property, but explicitly the mean  $\mu_i$  is a learned parameter. Therefore, translating the search space so that  $x_i \rightarrow x_i + a_i$ , where  $a_i$  is a constant, should not seriously impact the algorithm performance.
- Invariance to rotations: since the CMA-ES learns the full covariance matrix, it is able to adapt to rotations in the input parameter space. That is, the algorithm is invariant under  $x_i \rightarrow R_{ij}x_j$  where  $R_{ij}$  is a rotation matrix. This feature is extremely useful since makes it irrelevant as to whether or not the natural parameters for a problem are linear combinations of the parameters given to the optimizer.
- Invariance to scaling: again, owing to the fact that the CMA-ES learns the full covariance matrix automatically, the algorithm will perform equally well on problems where  $x_i \rightarrow bx_i$  where  $b$  is a positive scale factor.



**Fig. 2.7** (a) A parabolic test function and the same function after a random rotation and scaling. If the CMA-ES is run on each of these problems, the performance of the algorithm, as characterized by the fitness vs. time graph is practically unchanged

- **Invariance to monotone fitness transforms:** Because the algorithm updates itself only using ranks of the sampled points, CMA-ES behaves identically when solving any order preserving transformation of the original problem. That is, the CMA-ES is invariant under  $g(x) \rightarrow f(g(x))$  provided  $f(x)$  leaves the ranks intact. Examples of this transformation include constant scaling, offsets, or even exponentiation.

To demonstrate the utility of these invariances, Fig. 2.7 compares two independent runs of the CMA-ES started with the same initial mean and covariance matrix. One run optimizes a simple parabolic function while the other optimizes a randomly translated, rotated, and scaled version of the same problem. In spite of these distortions, the fitness values found at each step are essentially identical, asserting that if the CMA-ES is run on two versions of the same problem, differing only by an affine transformation, the performance will be indistinguishable.

Finally, it's worth noting that, while there are a number of free parameters  $c$ , in practice they rarely have a significant impact on algorithm performance. In fact, they can typically be set using empirical relations that depend only on the problem dimension (Hansen et al. 2003; Hansen 2006). The result is that, from a user perspective, the CMA-ES is parameter free. Practically speaking this is a major benefit since it means that few, if any, optimization runs have to be spent on calibration. Instead, the optimizer can be put to work immediately on problem solving.

### 2.2.2 *The Benefit of Parameterization*

There can be something dissatisfying about parameterizing the search distribution as a Gaussian. It seems it would be better to allow the algorithm to pick a distribution that best reflects the global search landscape. Indeed, simulated annealing, genetic algorithms and traditional evolution strategies are all built around the philosophy of particle based, implicit distributions. Yet all three of these algorithms are typically outperformed by the CMA-ES.

We speculate that part of the reason for this incongruity is that implicit distributions demand a tremendous amount of information to be statistically sound, and yet most of this information is inevitably discarded by the optimizer. For example, simulated annealing typically needs to take a large (say order 100) number of steps at each iteration to ensure that the sampling is in fact from a Gibbs distribution with the prescribed temperature. Thus computational effort is, in part, devoted to simply generating new sample points. Conversely, in parameterized distributions, points are sampled with the sole purpose of updating parameters. This means that, in principle, one only needs to generate as many points as there are open parameters, yielding a small and systematized demand.

To put this in perspective, consider how much cost is expended to produce any change in the average solution quality. For simulated annealing, the average fitness changes every time a new step is reached in the annealing schedule, (i.e. every time the temperature changes), and to achieve this goal, typically a few hundred function calls have to be expended. The same is true for a population based evolutionary algorithm: to go from one generation to the next, a few hundred candidate solutions need to be evaluated. Both cases stand in stark contrast to the CMA-ES where an average change in fitness only costs a few dozen function evaluations.

Because randomized algorithms are developed by considering how expectation values or distributions behave, it is easy to ignore the demands and difficulties associated with constructing these objects. If the sampling method is elaborate, needs to converge, or if the population is implicit, a large number of function evaluations can be needed only to make the logic behind the algorithm statistically sound. Conversely, if the samples are generated exactly from a prescribed distribution, several sources of error are eliminated a priori, allowing statistically similar improvements to be achieved at a lower computational cost.

The potential disadvantage of parameterizing the search distribution is that it can only examine one specific region, thereby excluding the possibility of simultaneously exploring multiple, distant minima. But given this concern, its worth asking how many function evaluations are needed before a particle based method gains the advantage. For example, a simulated annealing algorithm trapped in a local well should always find its way to the global minima, provided it is given infinitely many function evaluations (Kirkpatrick et al. 1983). But in practice, it can become the case that the algorithm is not given enough time to leave the well, thereby reducing it to an effectively local search method. More generally, both simulated annealing and types of evolution strategies are guaranteed to find global optima on

search problems given an infinitely large number of evaluations per iteration (Back et al. 1993; Kirkpatrick et al. 1983). But in practice, realizing the conditions for these proofs on complicated problems can require demands beyond either the expectations or inclinations of a user. When this happens, it becomes more practical to converge quickly with a local search algorithm and restart than to try and distinguish the global optimum in one fell swoop.

Altogether these results emphasize the tradeoff between algorithm generality and algorithm speed. At a theoretical level, population based methods favor the former while parameterized methods, like the CMA-ES, favor the later. Yet in practice, speed is typically the deciding factor for whether an algorithm is useful. To the extent that this is true, parameterized algorithms gain a measurable advantage over particle based algorithms, as reflected in empirical benchmarking (Hansen et al. 2009).

## 2.3 Deriving Evolutionary Algorithms from Evolutionary Game Theory

At first glance its hard to see what, if anything, the CMA-ES has to do with evolution in the biological sense of the word. The heuristic derivation plus the parameterized distribution seem to have little to do with the concepts of selection or survival of the fittest. Yet in this section we will show that algorithms in the same class as the CMA-ES share a deep connection with evolutionary game theory. In fact, using evolutionary game theory as a jumping off point one can reproduce several of the main ideas that make the CMA-ES so powerful.

The objective here is not to completely reproduce the CMA-ES, but rather to find a framework that allows its key features to be generalized. To the extent that the CMA-ES is successful because of invariance properties, the core of the algorithm is defined by the original weighted mean and covariance matrix updates in Eq. 2.1, and the rank-based weighting scheme. Thus the goal becomes to derive these features, which can be nicely met by starting with the fundamental equation in evolutionary game theory (Hofbauer and Sigmund 2003).

### 2.3.1 The Replicator Equation

Consider the simple model for a biological system where objects reproduce by replication (Hofbauer and Sigmund 2003). That is, each object makes identical copies of itself. Further, the rate at which objects replicate is set by the quality or fitness of an object  $x_i$ , denoted by  $f(x_i)$ :

$$\dot{n}_i = n_i f(x_i) \tag{2.10}$$

where  $n_i$  is the number of  $x_i$  objects. Moreover, the probability that a randomly selected object is of the type  $x_i$  is  $P_i = n_i / \sum_i n_i$  and, given (2.10) this probability evolves as

$$\dot{P}_i = \frac{\dot{n}_i}{\sum_i n_i} - \frac{n_i}{\sum_i n_i} \frac{\sum_i \dot{n}_i}{\sum_i n_i} \quad (2.11)$$

$$\dot{P}_i = \frac{n_i f(x_i)}{\sum_i n_i} - \frac{n_i}{\sum_i n_i} \frac{\sum_i n_i f(x_i)}{\sum_i n_i} \quad (2.12)$$

$$\dot{P}_i = P_i(f(x_i) - \langle f \rangle) \quad (2.13)$$

where the angle brackets denote averaging over all possible objects  $x_i$ . The system can be generalized to a continuous framework by replacing  $n_i$  with the number density  $n(x)$  and the probability  $P_i$  with the probability density  $\rho(x)$ , or

$$\dot{\rho}(x) = \rho(x)[f(x) - \langle f \rangle] \quad (2.14)$$

This equation describes some of the basic concept of evolution, but its not complete. Clearly, objects fitter than average proliferate while the probability of selecting objects that are less fit than the average decays to zero. However, as it stands objects don't compete, they only reproduce. It would be more appropriate if increasing the number of fit objects directly caused the reproduction rate  $f(x)$  for unfit objects to suffer as well. This can be accomplished by introducing a payoff function  $\Pi(x, x')$  and setting  $f(x) = \int dx' \Pi(x, x') \rho(x')$ . This payoff function specifies the amount of reward an object at  $x$  gets when competing against an object at  $x'$ . The fitness is then defined as the average payoff when competing against members of the population drawn at random. For example, if we choose  $\Pi(x, x') = \mathbb{1}[g(x') > g(x)]$  so that an object at point  $x$  is awarded one unit of fitness every time it encounters an object with a larger  $g$ , then averaged over the whole population, the fitness  $f(x)$  equals the probability of having a lower value of  $g$  than another object drawn randomly from the population. Written explicitly, a payoff function changes Eq. 2.14 into

$$\dot{\rho}(x) = \rho(x) \left[ \int dx' \Pi(x, x') \rho(x') - \int \int dx dx' \Pi(x, x') \rho(x') \rho(x) \right] \quad (2.15)$$

although in what follows, we will often keep  $f(x)$  as shorthand.

Equation 2.15 is known as the continuous replicator equation (Oechssler and Riedel 2002, 2001; Cressman 2005) and appears in diverse fields ranging from game theory to ecology to economics (Hofbauer and Sigmund 2003). In the following sections, we will show how some special manipulations can also connect it with the CMA-ES.

### 2.3.2 Parameterizing the Replicator Equation

At a glance, its hard to see how exactly the replicator equation can be used as an algorithm. To evaluate the replicator equation as is, one would need to calculate the fitness at every point. This makes the equation irrelevant for design and optimization since evaluating every point is tantamount to brute forcing the problem.

However, the continuous replicator equation can be turned into something more practical by constraining the distribution to have a fixed structure. For example, the distribution  $\rho$  could be fixed to always be a Gaussian or a Cauchy functional form. Doing so makes  $\rho$  a function of both  $x$  and a finite number of parameters  $\lambda_i$ . Continuing with the Gaussian example, the parameters  $\lambda$  could be the mean  $\mu$  and the covariance matrix  $\Sigma$  (though other choices are possible). With the form of the distribution fixed, these parameters become the sole source of time dependance. This transforms Eq. 2.14 into

$$\dot{\rho}(x|\lambda) = \rho(x|\lambda)[f(x) - \langle f \rangle] \quad (2.16)$$

$$\frac{\partial \log(\rho)}{\partial \lambda_i} \dot{\lambda}_i = [f(x) - \langle f(x) \rangle] \quad (2.17)$$

Now, the trick is to isolate a set of update rules for each  $\lambda_i$  that depend only on averages, rather than point-wise values. Once this is done, averages can be approximated by sampling from the distribution, thereby avoiding the issue of analyzing every point. This goal is met with a bit of algebra. First we multiply both sides by  $\rho \frac{\partial \log(\rho)}{\partial \lambda_j}$  and integrate to get

$$\left\langle \frac{\partial \log(\rho)}{\partial \lambda_i} \frac{\partial \log(\rho)}{\partial \lambda_j} \right\rangle \dot{\lambda}_i = \left\langle \frac{\partial \log(\rho)}{\partial \lambda_j} [f(x) - \langle f(x) \rangle] \right\rangle \quad (2.18)$$

The matrix on the right hand side can then be inverted to produce

$$\dot{\lambda}_i = \left\langle \frac{\partial \log(\rho)}{\partial \lambda_i} \frac{\partial \log(\rho)}{\partial \lambda_j} \right\rangle^{-1} \left\langle \frac{\partial \log(\rho)}{\partial \lambda_j} [f(x) - \langle f(x) \rangle] \right\rangle \quad (2.19)$$

This equation is also has a name: the information geometric optimization flow (Wierstra et al. 2008; Ollivier et al. 2011). In recent work, it has been proposed as a unifying framework for black-box optimizers in the same class as the CMA-ES (Akimoto et al. 2010; Ollivier et al. 2011), although prior work has not yet connected it to the replicator equation. Instead, it has been derived abstractly with concepts from information theory and differential geometry. But, before examining the ramifications of connecting these two disciplines, we turn to examining in greater detail the content behind Eq. 2.19.

### 2.3.3 Evolution Equations for the Exponential Family

To see how exactly Eq. 2.19 works, one can pick a family of distributions and examine the consequences. A natural choice is the exponential family in which distributions are of the form  $\exp[T_i(x)\lambda_i]/\int dx \exp[T_i(x)\lambda_i]$ . Because exponential family distributions have the property that  $\partial_{\lambda_i} \log[\rho] = T_i(x) - \langle T_i(x) \rangle$ , the resulting update equations simplify significantly:

$$\dot{\lambda}_i = \langle \partial_{\lambda_i} \log[\rho] \partial_{\lambda_j} \log[\rho] \rangle^{-1} \langle \partial_{\lambda_j} \log[\rho] f(x) \rangle \quad (2.20)$$

$$\dot{\lambda}_i = \langle (T_i - \langle T_i \rangle)(T_j - \langle T_j \rangle) \rangle^{-1} \langle (T_j - \langle T_j \rangle) f(x) \rangle \quad (2.21)$$

$$\dot{\lambda}_i = \text{Cov}[T_i, T_j]^{-1} \text{Cov}[T_j, f(x)] \quad (2.22)$$

Apparently, the parameters  $\lambda_i$  adapt themselves to follow the correlations between  $T_i$  and  $f(x)$ , although the exact process is obscured by the matrix inversion. A clearer perspective can be obtained by examining how the averages  $\langle T_i \rangle$  evolve

$$\frac{d}{dt} \langle T_i \rangle = \partial_{\lambda_k} \langle T_i \rangle \dot{\lambda}_k = \langle T_i \partial_{\lambda_k} \log[\rho] \rangle \langle \partial_{\lambda_k} \log[\rho] \partial_{\lambda_j} \log[\rho] \rangle^{-1} \langle \partial_{\lambda_j} \log[\rho] (f - \langle f \rangle) \rangle \quad (2.23)$$

Replacing all of the  $\partial_{\lambda_i} \log[\rho]$  terms by  $T_i(x) - \langle T_i(x) \rangle$  gives

$$\frac{d}{dt} \langle T_i \rangle = \langle T_i (T_k - \langle T_k \rangle) \rangle \langle (T_k - \langle T_k \rangle)(T_j - \langle T_j \rangle) \rangle^{-1} \langle (T_j(x) - \langle T_j(x) \rangle)(f - \langle f \rangle) \rangle \quad (2.24)$$

$$\frac{d}{dt} \langle T_i \rangle = \text{Cov}[T_i, T_k] \text{Cov}[T_k, T_j]^{-1} \text{Cov}[T_j, f] \quad (2.25)$$

$$\frac{d}{dt} \langle T_i \rangle = \text{Cov}[T_i, f] \quad (2.26)$$

Very simply, we find that the averages conjugate to each  $\lambda_i$  evolve to directly follow correlations in fitness. This provides a clearcut interpretation. For example, if we take  $T_i = x_i$  then updating with Eq. 2.19 creates a current of probability flux pointing in the direction most correlated with improvements in solution quality:  $\frac{d}{dt} \langle x_i \rangle = \text{Cov}[x_i, f]$ .

### 2.3.4 Evolution for a Gaussian Distribution

If the search distribution is taken to be a multivariate Gaussian, the results in the prior section can be used to derive its update rules. Specifically, define  $\lambda = [\Sigma_{ij}^{-1} \mu_j, -\frac{1}{2} \Sigma_{ij}^{-1}]$  so that  $T = [x_i, x_i x_j]$ . Inserting this into the update equation for the moments, (2.26), gives



$$\frac{d}{dt}\langle x_i \rangle = \text{Cov}[x_i, f] \quad \frac{d}{dt}\langle x_i x_j \rangle = \text{Cov}[x_i x_j, f] \quad (2.27)$$

This result can be rewritten in terms of the mean and covariance matrix only:

$$\frac{d}{dt}\mu_i = \text{Cov}[x_i, f] \quad \frac{d}{dt}\Sigma_{ij} = \text{Cov}[(x_i - \mu_i)(x_j - \mu_j), f] \quad (2.28)$$

Structurally, Eq. 2.28 is the differential equation version of the CMA-ES Eq. 2.1.

A potential difference is the presence of a rank-based weighting function in Eq. 2.1 versus the fitness function,  $f(x)$ , in Eq. 2.28. But recall that, for a replicator equation with selection,  $f(x) = \int dx' \Pi(x, x') \rho(x')$ . By inserting a payoff function, the correspondence can be completed. For example, the previous choice  $\Pi(x, x') = \mathbb{1}[g(x') > g(x)]$  produces weights that depend only on the ranked values of each object. Specifically, this payoff function is equivalent to the weighting scheme  $w(x) \propto P(x)$  where  $P$  is the probability that the fitness at  $x$  beats another object drawn randomly from the distribution. Generally, weighting schemes can be matched to payoff functions by the relation  $w(x) \propto \int dx' \Pi(x, x') \rho(x')$ .

At this point, the goal of generalizing all the key invariances boasted by the CMA-ES has been fulfilled. Moreover, the presence of a clear correspondence between biological evolution and the CMA-ES suggest a connection with natural evolution is less tenuous than it first seems: the core concepts that the CMA-ES is built around can be linked to fundamental elements in biology. Explicitly, the CMA-ES as originally defined by Eqs. 2.1 can be thought of a parameterized model for replicating and competing species, in which selection pressure drives changes to the distribution.

### 2.3.5 Solving the Replicator Equation with Exponential Families

Putting optimization aside for the moment, the correspondence between evolutionary algorithms and the replicator equation can also be used for more theoretical ends. As it turns out, exponential families are also a good ansatz for creating exact solutions to the full replicator equation 2.15. As a general example, consider an infinite exponential family of the form  $\exp[\Sigma_n \lambda_n x^n]$  (for ease of notation, the  $x^n$  is being used symbolically and the sum is carried out over all order  $n$  combinations of the  $x$  variables:  $x^n = x_i x_j x_k \dots$ ). In this case, the evolution equations for the average values  $\langle x^n \rangle$  are

$$\frac{d}{dt}\langle x^n \rangle = \text{Cov}[x^n, f] \quad (2.29)$$

But calculating the evolution equation for the moments from the full replicator equation produces

$$\frac{d}{dt}\langle x^n \rangle = \int dx \dot{\rho} x^n = \langle \rho x^n (f - \langle f \rangle) \rangle = \text{Cov}[x^n, f] \quad (2.30)$$

Evidently, all the moments evolve identically in the parameterized infinite exponential family model and the original replicator equation, implying the two distributions are indistinguishable.

Whats more, the power series solution implied by an infinite family can be reduced if the payoff function is of a special form. Specifically, if the payoff function  $\Pi(x, x')$  can be factored into  $\Pi(x, x') = \sum_n \Psi_n(x) \Phi_n(x')$ , then a distribution of the form  $\exp[\sum_n \Psi_n(x) \lambda_n(t)]$  is left invariant under the replicator equation, provided  $\dot{\lambda}_n(t) = \langle \Phi_n \rangle$ . Furthermore, the parameterized solution for the corresponding evolutionary algorithm yields exactly these dynamics. The proof is given by direct substitution into (2.15):

$$\dot{\rho}(x) = \rho(x) \left[ \int dx' \Pi(x, x') \rho(x') - \int \int dx dx' \Pi(x, x') \rho(x') \rho(x) \right] \quad (2.31)$$

$$\frac{d}{dt} \log[\rho(x)] = \left[ \int dx' \Pi(x, x') \rho(x') - \int \int dx dx' \Pi(x, x') \rho(x') \rho(x) \right] \quad (2.32)$$

inserting  $\Pi(x, x') = \Phi_n(x') \Psi_n(x)$  and  $\rho = \exp[\sum_n \Psi_n(x) \lambda_n(t)] / \int dx \exp[\sum_n \Psi_n(x) \lambda_n(t)]$  we get

$$\sum_n [\Psi_n(x) \dot{\lambda}_n(t) - \langle \Psi_n \rangle \dot{\lambda}_n(t)] = \sum_n [\Psi_n(x) \langle \Phi_n \rangle - \langle \Psi_n \rangle \langle \Phi_n \rangle] \quad (2.33)$$

Equality is obtained when  $\dot{\lambda}_n = \langle \Phi_n \rangle$ .

This equality is produced by employing the corresponding evolutionary algorithm. In this case, the coupling constants  $\lambda_n$  are governed by Eq. 2.22. Inserting  $\Psi_n(x)$  as the conjugate function to each  $\lambda_n$  gives

$$\dot{\lambda}_n(t) = \text{Cov}[\Psi_n, \Psi_m]^{-1} \text{Cov}[\Psi_m, f(x)] \quad (2.34)$$

But by inserting the assumed for the payoff function we find

$$\text{Cov}[\Psi_m, f] = \text{Cov}[\Psi_m, \Psi_n \langle \Phi_n \rangle] \quad (2.35)$$

producing the final equation for the coupling constants

$$\dot{\lambda}_n(t) = \text{Cov}[\Psi_n, \Psi_m]^{-1} \text{Cov}[\Psi_m, f] \quad (2.36)$$

$$\dot{\lambda}_n(t) = \text{Cov}[\Psi_n, \Psi_m]^{-1} \text{Cov}[\Psi_m, \Psi_k] \langle \Phi_k \rangle \quad (2.37)$$

$$\dot{\lambda}_n(t) = \langle \Phi_n \rangle \quad (2.38)$$

which is the requirement to solve the full replicator equation. Altogether we find that if the payoff function can be factored, there is an evolutionary algorithm that creates exact solutions to the full replicator equation.

On the one hand, this discussion suggests that, at some level, evolutionary algorithms are a formalism for solving classes of replicator equations. On the other hand, it is an active subject of research in evolutionary computation on how to best choose weighting functions (Wierstra et al. 2008). This analysis suggests that the payoff functions, and by extension weighting schemes, might best be selected by minimizing the difference between the true replicator equation's evolution and the parameterized one. Evidently, putting evolutionary algorithms in contact with theoretical biology invites new perspectives for natural and artificial evolution alike.

### 2.3.6 Global Convergence for Replicator Equations

At present, we have a systematic framework to derive new algorithms with the parameterized replicator equation 2.19. But so far little has been said to about any practical consequences of starting from the replicator equation. One potential benefit is that prior work has yielded conditions under which replicator equations converge to the global optimum (Oechssler and Riedel 2002; Cressman 2005). Thus it is hopeful that evolutionary algorithms built from the same concept might inherit this feature.

By way of this point, consider the following sketch of convergence. We start by considering what happens to the probability of not drawing a sample in a given region,  $K = 1 - P_A$  where  $P_A$  is the probability of drawing a sample that lives in region A. For the time being, suppose A is a small sphere around an arbitrary test point. If the density evolves according to the replicator equation, then we find that  $K$  changes via

$$\frac{d}{dt}K = -\frac{d}{dt}P_A = -\langle(f - \langle f \rangle)\mathbb{1}_A\rangle \quad (2.39)$$

where  $\mathbb{1}_A$  is 1 whenever the sampled point is in the region defined by A and 0 otherwise. Assume that  $f$  has a special point  $x_{opt}$  where the fitness is always greater than or equal to the population average. Then, by picking A as the region infinitesimally surrounding  $x_{opt}$ , we find  $\frac{d}{dt}K < 0$  when  $P_A < 1$ . Further,  $K > 0$  as long as  $P_A < 1$ . Combined, these two facts mean that  $K$  is a Lyapunov function and  $P_A \rightarrow 1$  as the system is left to evolve.

The result here may not be rigorous, but its easy to rationalize: the point that maximizes the fitness always increases in density, so in the long time limit, all the density must wind up here. The skeptical reader is directed to the complicated but exact proofs of convergence in the literature (Oechssler and Riedel 2002; Cressman 2005).

In contrast to the replicator equation, rigorous proofs explaining the convergence of evolutionary algorithms like the CMA-ES have remained elusive for over 10 years. One route to this end might be to compare the dissimilarity between an evolutionary optimizer's probability distribution and the exact replication equation solution it is approximating. If one can demonstrate that the approximation converges to the exact equation in the long time limit, then it must also inherit the global convergence properties. To our knowledge this approach is unexplored and seems like a promising route for explaining and expanding evolutionary computation's current state of the art.

### 2.3.7 Change in Entropy

By definition evolution is a process that exchanges diversity and quality. To make an optimizer, we want to transform the former to the later, but in a balanced way. If we trade the range of potential solutions for improvements too quickly, then its likely the optimizer will get stuck in a local optimum. Conversely, in order to converge, the optimizer must give up some diversity in each iteration.

To analyze this exchange, we define the change of diversity of a population in terms of its change in relative entropy (Ollivier et al. 2011). For two distributions,  $\rho(t)$  and  $\rho(t + \delta t)$  that are infinitesimally close to one another, one measure for the loss of diversity is the relative change in entropy or  $\int \rho(t) \log(\rho(t)) - \int \rho(t) \log(\rho(t + \delta t))$ . Taylor expanding to second order in  $\delta t$  gives

$$\delta S = \int \rho(t) \log[\rho(t)] - \int \rho(t) \log[\rho(t + \delta t)] = - \int \partial_i \rho \delta t - \int \rho \partial_{ii} \log[\rho] \delta t^2 \quad (2.40)$$

$$\delta S = -\delta t^2 \int \rho \ddot{\lambda}_i \partial_{\lambda_i} \log[\rho] - \delta t^2 \int \rho \dot{\lambda}_i \dot{\lambda}_j \partial_{\lambda_i \lambda_j} \log[\rho] \quad (2.41)$$

$$\delta S = \delta t^2 \dot{\lambda}_i \dot{\lambda}_j \int \rho \partial_{\lambda_i} \log[\rho] \partial_{\lambda_j} \log[\rho] \quad (2.42)$$

$$\delta S = \delta t^2 \dot{\lambda}_i \dot{\lambda}_j \langle \partial_{\lambda_i} \log[\rho] \partial_{\lambda_j} \log[\rho] \rangle \quad (2.43)$$

Since the first order change in relative entropy as defined above always vanishes, this term represents the leading order increment along any path the parameters might take. If we follow the specific path of (2.19), then replacing one of the  $\lambda_i$  terms gives

$$\delta S = \delta t^2 \dot{\lambda}_i \langle \partial_{\lambda_j} \log[\rho] \partial_{\lambda_k} \log[\rho] \rangle^{-1} \langle \partial_{\lambda_i} \log[\rho] \partial_{\lambda_j} \log[\rho] \rangle \langle \partial_{\lambda_k} \log[\rho] (f - \langle f \rangle) \rangle \quad (2.44)$$

$$\delta S = \delta t^2 \dot{\lambda}_i \langle \partial_{\lambda_i} \log[\rho] (f - \langle f \rangle) \rangle \quad (2.45)$$

$$\delta S = \delta t^2 \langle \partial_i \log[\rho] (f - \langle f \rangle) \rangle \quad (2.46)$$

$$\delta S = \delta t^2 \partial_t \langle f - \langle f \rangle \rangle - \langle \partial_t (f - \langle f \rangle) \rangle \quad (2.47)$$

$$\delta S = -\delta t^2 \langle \partial_t (f - \langle f \rangle) \rangle \quad (2.48)$$

Eq. (2.48) has a nice physical interpretation associated with it. By rearranging the terms slightly it states

$$\frac{\delta S}{\delta t} = -\langle \delta (f - \langle f \rangle) \rangle \quad (2.49)$$

In other words, the rate that entropy (i.e. diversity) changes over a small, finite step is proportional to the population averaged change in fitness. In this sense, (2.19) makes an explicit statement about how diversity loss and fitness improvement are intertwined. Selecting a particular fitness function  $f$  is equivalent to deciding how fast diversity should be lost when the fitness increases by a fixed amount. Colloquially, if one is able to produce an average increase in fitness of  $q$  then one is willing to loose diversity at a rate of  $q$ .

## 2.4 Computation Time and Evolution

In an ideal world, the ability of an optimizer to solve a given problem would only depend upon the how rugged, discontinuous, or misshapen the minimization task is. In the real world, the dominant constraint is sufficient computer power. Even if the input-output relationship used by the optimizer is simple, optimizers can still be ineffective if the time to calculate an output is excessively long.

In light of this, we estimate that evolutionary optimization only recently became a general method for materials design. Between 1986 and 2007, Hilbert et al. (Hilbert and Lopez 2011) estimate that general purpose computing power doubled roughly every 18 months. This implies that a currently feasible simulation based optimization method for materials would have been impossibly slow 5 years ago, and will be extremely fast if implemented 5 years from now. As a specific example, consider the simulations in the following chapter. When run on a computer purchased in 2008, it took roughly 600 s to complete one simulation while the full optimization took roughly half a month to complete. Consistent with the rate of increase, the same optimization can be completed in a few days using hardware purchased in 2013, with simulations finishing every 250 s. Likewise, without specialized hardware, the same optimization would have taken roughly of 3 years to complete in 2000.

Taken optimistically, this result also implies that an automated materials design framework could transform exponential growth in computer power into exponential growth in optimized materials. To the extent that materials properties are collective phenomena, there is a lower bound of compute power needed to be near the thermodynamic limit for a given simulation model. Furthermore, adding elements

beyond this boundary in the same model may not substantially alter predictions about macroscopic properties. To the extent that this holds true, an increase in computer power every 18 months would implies that the time to go from concept to prototype halves every 2 years in automated materials design, while the scope of designable materials doubles at the same rate.

## References

- Akimoto, Y., Nagata, Y., Ono, I., & Kobayashi, S. (2010). Bidirectional relation between CMA evolution strategies and natural evolution strategies. In *Parallel problem solving from nature, PPSN XI* (pp. 154–163). Berlin/Heidelberg: Springer.
- Back, T., Hoffmeister, F., & Schwefel, H. P. (1991). A survey of evolution strategies. In *Proceedings of the 4th International Conference on Genetic Algorithms*, San Diego, San Francisco (pp. 2–9).
- Back, T., Rudolph, G., & Schwefel, H.-P. (1993). Evolutionary programming and evolution strategies: Similarities and differences. In *Proceedings of the Second Annual Conference on Evolutionary Programming*, San Diego, La Jolla (pp. 11–22).
- Beyer, H. G., & Schwefel, H. P. (2002). Evolution strategies – A comprehensive introduction. *Natural Computing*, 1(1), 3–52.
- Cressman, R. (2005). Stability of the replicator equation with continuous strategy space. *Mathematical Social Sciences*, 50(2), 127–147.
- de Jong, K. (1975). An analysis of the behavior of a class of genetic adaptive systems. Ph. D. thesis, University of Michigan.
- Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing*. New York: Springer.
- Hansen, N. (2006). The CMA evolution strategy: A comparing review. In *Towards a new evolutionary computation* (pp. 75–102). Berlin/Heidelberg: Springer.
- Hansen, N., Auger, A., Ros, R., Finck, S., & Posik, P. (2009). Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation* (pp. 1689–1696). New York: ACM.
- Hansen, N., Muller, S. D., & Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMAES). *Evolutionary Computation*, 11(1), 1–18.
- Hilbert, M., & Lopez, P. (2011). The world's technological capacity to store, communicate, and compute information. *Science*, 332(6025), 60–65.
- Hofbauer, J., & Sigmund, K. (2003). Evolutionary game dynamics. *Bulletin of the American Mathematical Society*, 40(4), 479–519.
- Hoffmeister, F., & Back, T. (1991). Genetic algorithms and evolution strategies: Similarities and differences. In *Parallel problem solving from nature*. Berlin/Heidelberg: Springer.
- Holland, J. H. (1973). Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2), 88–105.
- Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. Ann Arbor: University of Michigan Press.
- Ingber, L., Petraglia, A., Petraglia, M. R., & Machado, M. A. S., et al. (2012). Adaptive simulated annealing. In *Stochastic global optimization and its applications with fuzzy adaptive simulated annealing* (pp. 33–62). Berlin/Heidelberg: Springer.
- Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P., et al. (1983). Optimization by simulated annealing. *Science*, 220(4598), 671–680.

- Oechssler, J., & Riedel, F. (2001). Evolutionary dynamics on infinite strategy spaces. *Journal of Economic Theory*, 17(1), 141–162.
- Oechssler, J., & Riedel, F. (2002). On the dynamic foundation of evolutionary stability in continuous models. *Journal of Economic Theory*, 107(2), 223–252.
- Ollivier, Y., Arnold, L., Auger, A., & Hansen, N. (2011). Information-geometric optimization algorithms: A unifying picture via invariance principles. arXiv preprint arXiv:1106.3708.
- Rothlauf, F. (2006). *Representations for genetic and evolutionary algorithms*. Berlin/Heidelberg: Springer.
- Schwefel, H.-P. (1993). *Evolution and optimum seeking: The sixth generation*. New York: John Wiley and Sons, Inc.
- Wierstra, D., Schaul, T., Peters, J., & Schmidhuber, J. (2008). Natural evolution strategies. In *IEEE Congress on Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence)*, Hongkong (pp. 3381–3387). IEEE.



<http://www.springer.com/978-3-319-24619-2>

The Automated Design of Materials Far From  
Equilibrium

Miskin, M.Z.

2016, XIX, 89 p. 39 illus., 7 illus. in color., Hardcover

ISBN: 978-3-319-24619-2