Algorithm 38 *Differential Evolution (DE)*

$\alpha \leftarrow$ mutation rate ▷ Commonly between 0.5 and 1.0, higher is more explorative
popsize $\leftarrow$ desired population size

$P \leftarrow \langle\ \rangle$ ▷ Empty population (it's convenient here to treat it as a vector), of length *popsize*
$Q \leftarrow \square$ ▷ The parents. Each parent Q_i was responsible for creating the child P_i
for i from 1 to *popsize* **do**
    $P_i \leftarrow$ new random individual
Best $\leftarrow \square$
repeat
    for each individual $P_i \in P$ **do**
        AssessFitness(P_i)
        if $Q \neq \square$ and Fitness(Q_i) $>$ Fitness(P_i) **then**
            $P_i \leftarrow Q_i$ ▷ Retain the parent, throw away the kid
        if *Best* $= \square$ or Fitness(P_i) $>$ Fitness(*Best*) **then**
            Best $\leftarrow P_i$
    $Q \leftarrow P$
    for each individual $Q_i \in Q$ **do** ▷ We treat individuals as vectors below
        $\vec{a} \leftarrow$ a copy of an individual other than Q_i, chosen at random with replacement from Q
        $\vec{b} \leftarrow$ a copy of an individual other than Q_i or $\vec{a}$, chosen at random with replacement from Q
        $\vec{c} \leftarrow$ a copy of an individual other than Q_i, $\vec{a}$, or $\vec{b}$, chosen at random with replacement from Q
        $\vec{d} \leftarrow \vec{a} + \alpha(\vec{b} - \vec{c})$ ▷ Mutation is just vector arithmetic
        $P_i \leftarrow$ one child from Crossover($\vec{d}$, Copy(Q_i))
until *Best* is the ideal solution or we ran out of time
return *Best*

Crossover can be anything: but one common approach is to do a uniform crossover (Algorithm 25), but guarantee that at least one gene from Q_i (the gene is chosen at random) survives in P_i.

3.5 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a stochastic optimization technique somewhat similar to evolutionary algorithms but different in an important way. It's modeled not after evolution per se, but after swarming and flocking behaviors in animals. Unlike other population-based methods, PSO does not resample populations to produce new ones: it has no selection *of any kind*. Instead, PSO maintains a single static population whose members are Tweaked in response to new discoveries about the space. The method is essentially a form of **directed mutation**. The technique was developed by James Kennedy and Russell Eberhart in the mid-1990s.[46]

[46] Among the earliest papers on PSO is James Kennedy and Russell Eberhart, 1995, Particle swarm optimization, in *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948. Eberhart, Kennedy, and Yuhui Shi later wrote a book on the topic: James Kennedy, Russell Eberhart, and Yuhui Shi, 2001, *Swarm Intelligence*, Morgan Kaufmann.

Like Differential Evolution, PSO operates almost exclusively in multidimensional metric, and usually real-valued, spaces. This is because PSO's candidate solutions are Mutated *towards* the best discovered solutions so far, which really necessitates a metric space (it's nontrivial to Mutate, say, a tree "towards" another tree in a formal, rigorous fashion).

Because of its use in real-valued spaces, and because PSO is inspired by flocks and swarms, PSO practitioners tend to refer to candidate solutions not as a population of individuals but as a **swarm** of **particles**. These particles never die (there is no selection). Instead, the directed mutation moves the particles about in the space. A particle consists of two parts:

- The particle's location in space, $\vec{x} = \langle x_1, x_2, ... \rangle$. This is the equivalent, in evolutionary algorithms, of the individual's genotype.
- The particle's velocity, $\vec{v} = \langle v_1, v_2, ... \rangle$. This is the speed and direction at which the particle is traveling each timestep. Put another way, if $\vec{x}^{(t-1)}$ and $\vec{x}^{(t)}$ are the locations in space of the particle at times $t-1$ and t respectively, then at time t, $\vec{v} = \vec{x}^{(t)} - \vec{x}^{(t-1)}$.

Each particle starts at a random location and with a random velocity vector, often computed by choosing two random points in the space and using half the vector from one to the other (other options are a small random vector or a zero vector). We must also keep track of a few other things:

- The fittest known location $\vec{x}^*$ that $\vec{x}$ has discovered so far.
- The fittest known location $\vec{x}^+$ that any of the **informants** of $\vec{x}$ have discovered so far. In early versions of the algorithm, particles were assigned "grid neighbors" which would inform them about known best-so-far locations. Nowadays the informants of $\vec{x}$ are commonly a small set of particles chosen randomly each iteration. $\vec{x}$ is always one of its own informants.
- The fittest known location $\vec{x}^!$ that has been discovered by *anyone* so far.

Each timestep we perform the following operations:

1. Assess the fitness of each particle and update the best-discovered locations if necessary.
2. Determine how to Mutate. For each particle $\vec{x}$, we update its velocity vector $\vec{v}$ by adding in, to some degree, a vector pointing towards $\vec{x}^*$, a vector pointing towards $\vec{x}^+$, and a vector pointing towards $\vec{x}^!$. These are augmented by a bit of random noise (different random values for each dimension).
3. Mutate each particle by moving it along its velocity vector.

The algorithm looks like this:

Algorithm 39 *Particle Swarm Optimization (PSO)*

$swarmsize \leftarrow$ desired swarm size
$\alpha \leftarrow$ proportion of velocity to be retained
$\beta \leftarrow$ proportion of personal best to be retained
$\gamma \leftarrow$ proportion of the informants' best to be retained
$\delta \leftarrow$ proportion of global best to be retained
$\epsilon \leftarrow$ jump size of a particle

$P \leftarrow \{\}$
for $swarmsize$ times **do**
    $P \leftarrow P \cup \{$new random particle $\vec{x}$ with a random initial velocity $\vec{v}\}$
$\overrightarrow{Best} \leftarrow \square$
repeat
    for each particle $\vec{x} \in P$ with velocity $\vec{v}$ **do**
        AssessFitness($\vec{x}$)
        if $\overrightarrow{Best} = \square$ or Fitness($\vec{x}$) $>$ Fitness($\overrightarrow{Best}$) **then**
            $\overrightarrow{Best} \leftarrow \vec{x}$
    for each particle $\vec{x} \in P$ with velocity $\vec{v}$ **do** ▷ Determine how to Mutate
        $\vec{x}^* \leftarrow$ previous fittest location of $\vec{x}$
        $\vec{x}^+ \leftarrow$ previous fittest location of informants of $\vec{x}$ ▷ (including $\vec{x}$ itself)
        $\vec{x}^! \leftarrow$ previous fittest location any particle
        for each dimension i **do**
            $b \leftarrow$ random number from 0.0 to β inclusive
            $c \leftarrow$ random number from 0.0 to γ inclusive
            $d \leftarrow$ random number from 0.0 to δ inclusive
            $v_i \leftarrow \alpha v_i + b(x_i^* - x_i) + c(x_i^+ - x_i) + d(x_i^! - x_i)$
    for each particle $\vec{x} \in P$ with velocity $\vec{v}$ **do** ▷ Mutate
        $\vec{x} \leftarrow \vec{x} + \epsilon\vec{v}$
until $\overrightarrow{Best}$ is the ideal solution or we have run out of time
return $\overrightarrow{Best}$

This implementation of the algorithm relies on five parameters:

- α: how much of the original velocity is retained.
- β: how much of the personal best is mixed in. If β is large, particles tend to move more towards their own personal bests rather than towards global bests. This breaks the swarm into a lot of separate hill-climbers rather than a joint searcher.
- γ: how much of the informants' best is mixed in. The effect here may be a mid-ground between β and δ. The *number* of informants is also a factor (assuming they're picked at random): more informants is more like the global best and less like the particle's local best.
- δ: how much of the global best is mixed in. If δ is large, particles tend to move more towards the best known region. This converts the algorithm into one large hill-climber rather than

separate hill-climbers. Perhaps because this threatens to make the system highly exploitative, δ is often set to 0 in modern implementations.

- ϵ: how fast the particle moves. If ϵ is large, the particles make big jumps towards the better areas — and can jump over them by accident. Thus a big ϵ allows the system to move quickly to best-known regions, but makes it hard to do fine-grained optimization. Just like in hill-climbing. Most commonly, ϵ is set to 1.