Autonomous Evolutionary Algorithm

Matej Šprogar
Faculty of Electrical Engineering and Computer Science
University of Maribor
Slovenia

1. Introduction

Evolutionary algorithms (*EA*) are randomized heuristic search methods based on the principles of natural evolution (Banzhaf et al., 1998; Goldberg, 1989; Holland, 1975; Bäck, 1996; Koza, 1992). If we know how to describe the problem using the terminology of artificial evolution, the EAs are quite easy to apply. Actually, the search for solution(s) is *transformed* into a search for the best EA setup – a mixture of highly correlated settings and functions (encoding scheme, run-time parameters, fitness (objective) function, selection mechanism. . .). Finding a good EA setup is a problem because EAs are chaotic systems where small variations in initial setup produce large variations in the long-term behavior of the model. A good setup for one problem is mostly unusable for another, although similar problem.
Evolutionary algorithm that would be easy to apply in any problem domain would have to be autonomous in a sense that it would regulate its own behavior and would have no need for human intervention (except for the preparation phase, of course). This article discusses the operating principles of such an algorithm and presents its implementation. The *Autonomous EA* (AEA) is an experiment in the evolution of evolutionary algorithms. It is not much different from existing EAs and the line between the two is sometimes very blurred. Actually, AEA combines known concepts, insights and solutions from EAs, artificial life, chaos theory and complex adaptive systems theory into a new form of evolutionary algorithm.
The nomenclature used in different fields is overlapping (for example individual/ solution/object/agent). In AEA the evolving individual represents the solution: a population of individuals (solutions) is evolved in order to find a solution (individual) for the problem at hand. Population is just a limited representation of the vast search space of all possible solutions.

1.1 Controlling evolution

Evolutionary computing is an artificial world where computer-based models are directly written in terms of conditional actions and operations. These models can then be "run" in a simulator. Like any other, the EA simulation is controlled by many parameters. There are numerous studies of EA parameters giving suggestions on their "correct" values. Different types of control of algorithm parameters can be classified as either parameter tuning, deterministic parameter control, adaptive or self-adaptive parameter control (Eiben et al., 1999).

The EA implementers always try to create the system with as little parameters as possible. To reduce this already minimal number of parameters even further, two possibilities exist:
1. make the parameter fully self-adaptive; or
2. remove the need for a parameter.

Self-adaptation effectively "hides" a parameter and can be used for most, but not all parameters. Because self-adaptation is a much explored topic (Angeline, 1995; Eiben et al., 1999), we chose to investigate the second possibility more. We can try to remove a parameter either by replacing it with some non-parametric value/principle or by changing the operating principle(s) of the EA. Certain parameters, however, cannot be removed – for example the hardware limitations (amount of memory, available CPU time. . .).
EAs are complex systems with numerous algorithm parameters to set. For example, Genetic Programming (GP) in the Open Beagle evolutionary framework by default already includes over 20 different parameters that directly control the behavior of the underlying evolutionary computation (*ec.pop.size*, *ec.repro.prob*, *ec.sel.tournsize*, . . .) (Gagne & Parizeau, 2006).

1.2 Objective

The objective of Autonomous EA is to decrease the number of algorithm parameters to a bare minimum. The population size, for example, is one problematic parameter despite numerous attempts and claims of the 'optimal' value. Next problematic parameter is the one that controls the termination criterion. The most troublesome, however, is the fitness function. Although the fitness function does not seem to be a parameter, it is a rather complex mixture of criteria of unknown/changing importance. It must be "input" to all existing EAs by the human operator, therefore it is a parameter. Fitness can not be self-adapted (only the constraint weights of fitness function can be self-adapted) or self-induced as EAs will quickly learn to "cheat" by producing worthless individuals with good fitness. Fitness is fundamental and is actually a reason for many other control settings.

2. Fitness – the core of the problem

Nature is using simple atomic rules to guide the evolution. Paradoxically, canonical EAs are already too complicated to follow this simple rule. They constantly apply the same orthodox idea of fitness and associated complex mechanisms in their evolutionary loops. Small mistake in fitness and EA will fail to find an otherwise obvious solution altogether. Creation of adequate fitness functions demands significant knowledge of the environment to be evaluated – this can imply that the problem might have already been solved or that other, non evolutionary technique(s) might be more efficient (Angeline & Pollack, 1994). Inadequate fitness makes EA focus on sub-optimal solutions. To avoid local optima different "corrections" and "tweaks" are usually applied. These corrections need further corrections resulting in complex EAs. In EAs, fitness is at the core of everything.
Fitness function is the engineer's interpretation of the problem and is as such affected by the computational biases of human cognition (Stanovich, 2003). Biased EA is unable to discover the *generalists* – solutions with the capability to generalize – because generalization can only be the (emergent) property of solutions produced by the unbiased EA. Solutions produced by biased EA are on the contrary often brittle – they fail on the previously unseen/new data. Unless EA is used to find *specialists* (optimizations of a single fitness peak), this is a major problem. Solution is either to make the fitness unbiased or to "remove" it altogether.

2.1 The search space

The success of search depends mostly on the distribution of solutions within the search space and this distribution is determined/described by fitness. For the same simple problem a perfect fitness, for example, defines a gradual landscape well suited for deterministic methods like gradient descent; sub-optimal fitness defines a more complicated space with many spikes; and bad fitness creates a chaotic space where any informed search is impossible – in such space a random search is as successful as any other search (Wolpert & Macready, 1997).

Existing fitness-based EAs are efficient in beautiful fitness problem spaces. Differential Evolution (DE), for example, is an effective, simple and fast optimization algorithm (Price & Storn, 1997). However, EAs are mostly tested on clean mathematical problems where fitness *optimization* is the problem. More difficult learning problems (for example data mining or genetic programming) require searching in unknown spaces for solutions of unknown structures – there, fitness *itself* is the problem (at least the fitness in a form needed by EAs).

2.2 The essence of fitness

The idea behind fitness is to produce a single number that will (magically) create a hierarchy of candidate solutions – the individual at the top *better* than the ones underneath. The tendency to over-simplify by squeezing into one single fitness number matters that are too rich to be described by it is a typical human mistake (an example is stock market analysis (Mandelbrot & Hudson, 2004)). It results in a fragile system. In EAs the concept of fitness *looks* simple and because all other mechanisms are crafted to suit the concept of fitness, the resulting system is extremely complex and not at all simple. In fact, the complexity of EAs is reflected in the number of necessary algorithm parameters.

In the real world, there is no *ultimate hierarchization* of individuals nor are individuals living under the same conditions. Darwin's survival of the fittest should not be interpreted as "evaluate, pick the best and kill the rest", it should be a "live and let die and the best will prevail with more offspring" approach. This way the butterfly-effect the discarded (*presumably* insignificant) solutions might have had on the evolutionary process would not be lost!

The dictionary defines Darwinian fitness as the number of offspring or close kin that survive to reproductive age (Dictionary.com, 2006). This definition is impossible to directly implement in EAs because artificial individuals do not live a life. Fitness is a *shortcut* that allows EAs to by-pass the phase of life of individuals. Fitness effectively determines the number of surviving offspring and replaces the individual's true Darwinian fitness! This shortcut only seemingly produces a desired result because life is a chaotic process, extremely susceptible to initial conditions. Ignorance of this results in many problems associated with EAs (Toffoli, 2000).

2.3 Limitations and assumptions

The EAs are always limited by the availability of computational resources. Bremermann states that faster computers are insufficient, "we must look for quality, for refinements, for tricks, for every ingenuity that we can think of." (Bremermann, 1962). This is why the EA community regards early Friedberg's approach (Friedberg, 1958; Friedberg et al., 1959) as being immature, attributing this mostly to the fact, that he used the so-called *binary* fitness. EAs are going in the direction of establishing more elaborate fitness measures and selection schemes. The argument for this is the belief that without an accurate enough ranking, the natural dynamics of artificial evolution might be compromised (Banzhaf et al., 1998; Angeline & Pollack, 1994). If unlimited computational resources were available, EAs could operate without discarding any individuals. Because this is not the case, EAs must employ some sort of artificial selection, selection being "a name for the ability of those individuals that have outlasted the struggle for existence to bring their genetic information to the next generation" (Bäck, 1996). The EAs fail to follow this simple definition because they falsely assume that:

1. evolution is a process of fitness-based ordering and selection; and
2. the individual's fitness is measurable and is independent of the fitness of its offspring.

Firstly, individual's fitness is based on the observed performance and EAs use this score to justify the selection, although biologists argue that "fitness cannot be used as a cause but merely as a description of natural selection" (Henle, 1991). EAs mistake the measurement of ability to fit the purpose for survival. Interestingly, EA researchers are aware of the fact that biological struggle for existence has no counterpart in EAs, yet they ignore this or find it at most an interesting research field (Bäck, 1996).

Secondly, EAs calculate the fitness (f) without considering the individual's potential to have better offspring. EAs rely on false impression that only fitness for purpose is important. In fact, the *ability* to create good offspring and forward the genetic information into next generations is what survival is about, yet this can not be computed in advance. If one were trying to compute it anyway, fitness values of all of individual's offspring would be needed and for that the fitness values of all of the offspring's offspring would be needed, etc.:

$$f(x) = u(x) + f(\hat{x}), \tag{1}$$

where u is a performance/utility function (for example accuracy) measuring the success of x at solving the given problem and $\hat{x}$ is x's offspring $(f(\hat{x}) = \sum f(\hat{x}_i))$. This is a highly recursive definition. Canonical EAs assume that the individual with better fitness will produce better offspring than the competing individual with worse fitness score, or at least that the probability for the opposite is low enough:

$$P(f(\hat{y}) > f(\hat{x}) \mid u(x) > u(y)) \approx 0\,, \tag{2}$$

A basic assumption of the fitness function, namely that seemingly-better individual is assigned a better (in example higher) score, neglects the fact that the seemingly-worse individual can posses the much-needed building block of the global solution. The search space is normally so enormous that EA must not afford to lose this building block although located in a seemingly bad individual. But because of the limited computational resources, EAs use

$$u(x) > u(y) \Rightarrow f(x) > f(y)\,, \tag{3}$$

to simplify (1) into

$$f(x) \approx u(x)\,. \tag{4}$$

For fitness calculation standard EAs rely on (4) because it's easy to implement and execute. The problem is that (4) does not link the ability to survive with the ability to solve a problem in any way, whereas (1) automatically makes this link. In standard EAs, survival is

artificially directed by the *selection* of survivors. Eq. (4) makes the evolution discover local solutions because of premature convergence problems. EAs acknowledge this and try to compensate by applying special mechanisms, which again have side-effects for which additional fixes are needed. . . In EA literature, numerous statements like: "the niching methods have been developed to reduce the effect of genetic drift resulting from the selection operator in the standard GA (Sareni & Krähenbühl, 1998)" can be found.
Something similar occurs, for example, when evaluating the chess board position. The quality of a player's move is determined by evaluation of the board after the move. If perfect fitness were available, the computer would *always* beat the human champion. Even worse, it could tell the winner right from the start! However, perfect fitness would only be possible to compute if fitness values of *all* successive moves were available; for these the scores of all the successive's successive moves would be needed. . . In computer chess the effort has not gone to teaching computers about chess, but to improving the algorithms for deciding when to cut off calculations and when to calculate more deeply. Something similar occurs in EAs, where special fitness-based mechanisms (e.g. different criteria weights, selection schemes . . .) are introduced and "improved" all the time, but fitness as a concept is never questioned.

2.4 Implicit perspective

The quality and quantity of population's members are coupled properties – quality affects quantity and vice versa. Holland described this phenomena in the terms of adaptive complex systems (*cas*) (Holland, 1995). He, like many others, recognized that fitness must "depend on the context provided by the site". Unlike EAs, however, he placed particular emphasis on avoiding an overt fitness criterion. He introduced the concept of a *resource* and his agent could reproduce only after it had acquired enough resources to make a copy of itself. Holland effectively replaced traditional explicit fitness with fully implicit resource acquisition. He avoided the fitness calculation because fitness is implicitly defined by the resource acquisition of his agents, which live or die in terms of their ability to collect critical resources.
Holland's principles were never successfully applied to engineering problems (where emphasis is put on finding *best* solutions). Rather, they've been used for studying complex adaptive systems, natural systems and in Artificial Life (AL) research. Artificial evolution in engineering was always based on explicit fitness; the numerous constraints implied by engineering problems make application of Holland's ideas troublesome because they're not directly goal driven as the engineer needs it.

2.4.1 Co-evolution

Co-evolutionary search should be more successful than 'complete' static fitness evaluation because co-evolving individuals sample the problem space more efficiently (Angeline & Pollack, 1994; Pagie & Mitchell, 2002; Hillis, 1990). Paredis, however, observed that in some cases co-evolution does not lead to better results (Paredis, 1997). These cases are often characterized by the occurrence of the so-called *Red Queen dynamics*[1] (Pagie & Hogeweg, 2000; Juille & Pollack, 2000), which can be prevented from persisting by the heterogeneity in the populations (Pagie & Mitchell, 2002).

[1] Evolutionary change may be required to stay in the same place. Cessation of change may result in extinction.

Co-evolutionary search in EAs is mostly said to use "implicit" fitness, where the fitness of evolving solutions depends on the state of other, co-evolving individuals. This, however, is not implicit according to Holland's definition because co-evolution is still calculating and using fitness. Fitness must not be a calculated value but rather an observed property of an individual; this observation can only be made *after* the individual has performed actions in its world (lived a life)!

3. Autonomous Evolutionary Algorithm

To achieve autonomy from human intervention AEA employs co-evolution of two competing populations. The fight for survival between two individuals, one from each respective population, simulates life and determines survivors in the simplest and most unbiased manner possible; in the process the number of offspring and individuals to be discarded are determined (this is necessary to keep the simulation within the available memory limits). The co-evolution terminates automatically[2] after one population dominates the other.

Standard EAs use fitness to create a hierarchy (ranking) of individuals. Position within this hierarchy defines the number of offspring. AEA, on the contrary, does not rank the individuals. Rather, it mimics a predator-prey like system, where individual survives only by outperforming another individual. In the process an individual holds and collects a virtual resource – energy – which is needed to create and shape the offspring. AEA essentially simulates the *flow* of energy between the two co-evolving populations.

3.1 Life of an individual

Standard EAs treat individuals as non-living objects. Individuals are created, evaluated and very likely also immediately destroyed without any impact whatsoever on the rest of the population. AEA, on the contrary, makes each individual alive – each living individual has to fight for survival and only surviving individuals reproduce.

AEA maintains two separate populations of individuals of the same type. Fight for survival is a simple competition between two randomly chosen individuals, one from each respective population. The competition is about solving an atomic task from the problem at hand. The better of the two competing individuals at this atomic task is the survivor. By using an atomic task we make this process as unbiased as possible[3].

AEA never tries to judge *how-much-better* one individual is compared to another. By making every individual "alive" (putting him through the fight-for-survival test) we get a list of survivors, what effectively removes the need for a selection phase.

The loser's energy reserves are transferred to the winner and then the loser is removed from the system – the allocated spatial resources (every individual occupies a certain amount of available virtual space / physical memory) are again made available for offspring. The energy determines the creation (number and genetic structure) of offspring. The main result of individual's life are fluctuations in quantity and distribution of virtual energy within the AEA system.

[2] Of course the evolution run can also be terminated artificially.

[3] Typical atomic task is one fitness case from the learning database.

3.1.1 Compared to traditional EA

The regular EAs look similar – the selection operator can give an individual zero, one or even more chances to “survive” and reproduce (this number is either random (e.g. tournament selection) or based on the fitness (e.g. roulette-wheel selection)). AEA, on the contrary, selects individuals to be removed only *after* they participated in the fight-for survival and lost their energy to the winning individuals. Effectively this is a “select all” selection strategy followed by a necessary[4] removal of some individuals.

Next important difference is how the fight for survival is made. To make the outcome as unbiased as possible, AEA employs the stochastic sampling of a *single* problem instance (atomic task) as the minimal measure of competence. Consequently also an overall bad performer can win against the almost perfect opponent, especially if the learning data contains noise. AEA lets the evolution decide which individual is better in the long run. . .

AEA goes the opposite way of traditional fitness. Instead of using all available information, AEA makes use of only the tiniest fraction of it. AEA does not give importance to how many problem instances does one solution solve nor does it make any biased presumptions whether it is better to solve one instance over another. It just says: for this atomic task, this solution is better. The worry of handling noisy or missing data is left to the evolution. Sometimes a good individual will be defeated by a weak one; good individuals, however, have more energy and more offspring and will therefore probably survive at other opportunities.

3.2 The core of the autonomous algorithm

The main resource needed for reproduction is energy, which is exchanged only between competing individuals. The winning individual simply collects the loser’s energy reserves. Second resource in AEA is the space – each individual occupies a certain amount of memory. Each gene in individual’s genotype occupies one unit of memory space – size of an individual equals to number of its genes. Both populations have limited space for holding individuals. After the population space is full, offspring production is suspended until further individuals are removed from the population.

The algorithm 1 shows AEA’s core. At start, the two co-evolving populations P_1 and P_2 are created and randomly initialized to fully occupy the assigned memory space. Individuals $x \in P_1$ and $y \in P_2$ are fully qualified solutions. The initial sizes of populations ($|P_1|$, $|P_2|$) depend on the average size of fresh individuals created by a typical initialization routine. Individuals should live simultaneously but because today’s computers employ serial CPUs we can only simulate this parallelism. First, random interaction pairs (x,y) are determined using the *random_pairs* function, which selects two random, previously unprocessed members, one from P_1 and the second from P_2, respectively. This creates a set of random pairs Q. Because P_1 and P_2 in general differ in size ($|P_1| \neq |P_2|$) only the smaller of the two populations is fully used in one cycle of the main loop ($|Q| = min(|P_1|,|P_2|)$); the remaining individuals will be processed in the next cycle(s)[5].

Next is the actual life of an individual: from each available pair $(x,y) \in Q$ the winner is determined. Winner takes the loser’s energy and waits for breeding, while the loser is discarded in order to free its memory space in its native population. Fight for survival redistributes the energy and frees the spatial resources.

[4] Limited computational resources require a limited population.
[5] This must be guaranteed by the *random_pairs* function.

Ideally, the surviving (active) individual would get one chance to breed and produce active offspring. This is unfortunately impossible to implement because the offspring would quickly overfill the population's memory space. To avoid this AEA employs a simple waiting list (*children*), where the inactive offspring waits to be included in the active population. Until the list is emptied no further breeding takes place. Of course, other workarounds are possible. Before the main cycle is repeated and only after all of population's members participated in a fight for survival, the *breed* function produces offspring for respective population (Algorithm 2). New children are produced only if all previous children were processed by the main loop. Function *create_child* produces one child using traditional variation operators (sexual crossover and mutation). The number of inactive children that are transferred into active population depends on the number of free spatial resources. This is to ensure as parallel evolution of all individuals as possible. The size of the resulting population is *not* calculated nor artificially maintained at a certain level as is common in EAs; it is only limited by the available memory space.

Algorithm 1 The core loop of AEA.

```
// two competing populations
population P1(memory/2), P2(memory/2);

while ( true ) {
    // create a set of random pairs of individuals
    // not all individuals are necessarily paired
    Q = random_pairs( P1, P2 );

    // fight-for-survival
    for-each ( pair ( x, y ) in Q ) {
        if ( x wins against y ) {
            x.energy += y.energy;
            P2.erase( y );
        } else {
            y.energy += x.energy;
            P1.erase( x );
        }
    }

    // termination criteria: empty population
    if ( P1.empty() ) return P2;
    if ( P2.empty() ) return P1;

    // only breed the fully processed population
    if ( P1 has been fully processed ) P1.breed();
    if ( P2 has been fully processed ) P2.breed();
}
```

Algorithm 2 Breeding of individuals within population.

```
population::breed()
{
    if ( children.empty() ) {
        // create offspring
        for-each ( individual x in this population ) {
            e = x.energy;
            while ( e>0 ) {
                individual y = random_member( this population );

                child = crossover( x, y );
                child.mutate( e );
                child.energy = 1;

                children.enque( child );
                e = e-1;
            }
            x.energy = x.energy / 2;
        }
    }
    // try to move children into active population
    this population.transfer( children );
}
```

The new-born individuals have by default exactly one energy point; this allows the system to grow and sustain larger individuals because energy is a vital resource in reproduction. The number of descendants equals the leading parent's energy. The higher the collected energy the more offspring the individual has. Each individual is a result of sexual reproduction of the primary parent with a random partner. The main parent produces a child by investing a certain portion (reproductive energy – *e*) of its energy into creation of the child's genotype. The amount of energy invested directs the creation phase, in particular the selection of mutation point and the probability of mutation of that point. The completion of the breeding phase also takes away 1/2 of the main parent's energy. The energy of the partner does not influence creation of main parent's child; partner only provides its genotype for copying.

3.2.1 Energy and reproduction

The individual's energy level determines:

1. the number of offspring,
2. the mutation probability,
3. the mutation point.

Crossover and mutations are interdependent: a new child is constructed from copies of both parents' genes. Size of the child directly depends on the selection of the random crossover point within each parent. For a duplication of 1 gene 1 reproductive energy point must be provided. When this energy is exhausted, the gene about to be copied is subject to mutation with probability $mut\_prob = \phi/(1.0 + e)$, where ϕ is a random number from interval [0,1]. For linear genomes the mutation point is therefore the e^{th} gene in the sequence. Tree-like genomes can be copied in either breadth-first or depth-first order. The remaining genes are copied without further energy consumption. The same result can also be achieved by making a mutation-free child and then mutating its e^{th} gene – the procedure used in Algorithms 2 and 3.

Algorithm 3 Mutation depends on available energy.

```
void individual::mutate( e )
{
    mut_prob = φ/(1+e);
    if ( random(mut_prob) ) {
        if (e < |genome|) {
            gene = genome[e];
            mutate( gene );
        }
    }
}
```

The parent's energy is not always used in full when making children. Rather, children are created with decreasing amounts of energy. This way a constantly changing number and position of possible mutations is introduced in offspring – the first child's reproductive energy *e* equals parent's energy, the second child's reproductive energy is one less etc. A "good" parent has enough energy to copy most of its genetic material faithfully and produce offspring with little or no mutations. The children of larger individuals without large energy reserves, however, are more likely to undergo mutations.
If $energy > |genome|$ no mutation is performed (as it is impossible to mutate a non-existing gene!). This is advantageous in creating offspring of parents with high energy status with respect to their size – higher energy suggests the parent is successful thus its genes should be preserved. Reproduction code in AEA (Alg. 2) makes pressure for many "good" copies of energy-full individuals. Parents with low energy reserves will have few and less-similar children.

3.3 Example of reproduction

For illustration of the reproduction consider Fig. 1, where a crossover of two GP-trees is shown: the first 5-node parent ("1 + (7 – 4)") holds *energy* = 8 energy points; for the creation

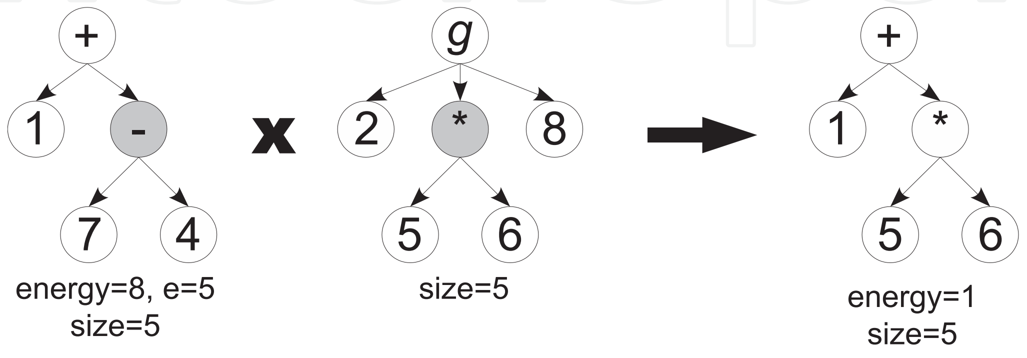

Fig. 1. Crossover without mutations.

of the fourth child $e = 5$ energy points are designated. The partner individual is "$g(2, 5 * 6, 8)$". The randomly chosen crossover points in both parents are '-' and '*'. Regular crossover results in a 5 node expression "1 + (5 ∗ 6)". Because $e \geq 5$ (size of the child), all nodes are copied/created without mutations.
If the crossover point in Fig. 2 was selected at '4' then regular EA crossover would produce the expression "1 + (7 − (5 ∗ 6))" with 7 nodes. Because there were only ($e = 5$) energy points available for creating this child, AEA is unable to reproduce all 7 nodes. Instead the first five nodes are copied ('+', '1', '-', '7' and '*') and the sixth node is mutated with probability $\phi/(1 + 5)$ from '5' $\rightarrow$ '3'; the remaining node(s) are copied without mutations.

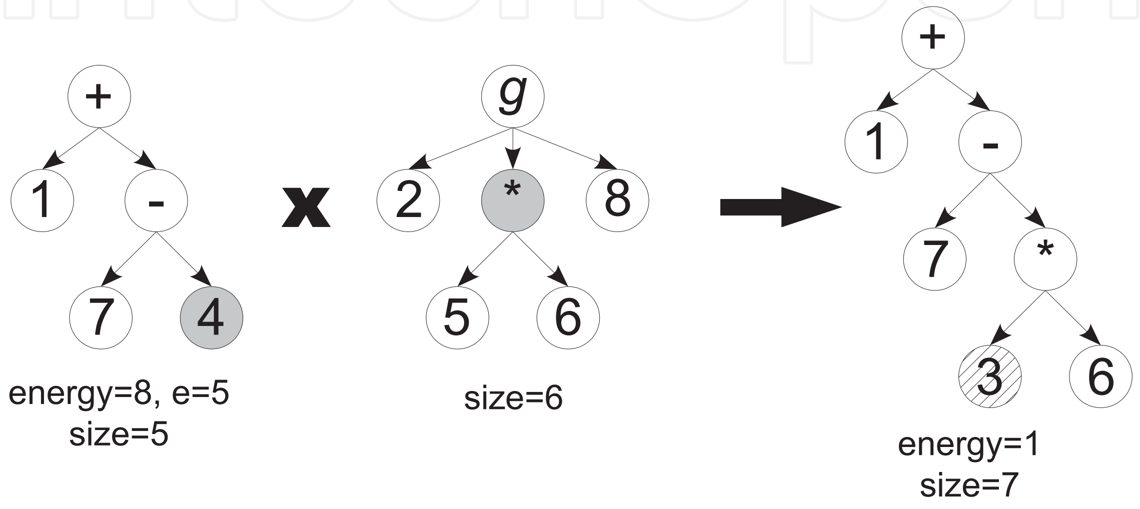

Fig. 2. Crossover with mutations.

3.4 Population size

Autonomous EA does not have any population size setting. This parameter is made obsolete by the idea of co-evolving populations in a spatially limited environment, where energy is used to direct the reproduction phase. An upper limit to population size must exist (or else population's size would explode in a matter of few generations); AEA implements this limitation through a total number of genes a population can hold. Consequently, a population can hold a large number of small individuals or a small number of very large individuals. One gene is said to occupy one unit of computer memory. Available memory is the first unavoidable setting of AEA.
The phase plot in Fig. 3 displays three possible scenarios of how the populations' sizes change during evolution. The evolution generally starts in the point S_0 and progresses through S_1 to terminate in either S_2 or S_3. The area of S_1 is an ever-changing state, where small improvements or changes in P_1 are counterbalanced by changes in P_2 and vice versa. From state S_1 the AEA can escape into either S_2 or S_3. However, if AEA is unable to break out of S_1 in "reasonable" time, it must be terminated artificially.
If one population is initialized to contain "much" better individuals the system will be unable to reach S_1. Instead, the weaker population will vanish too quickly and the system will follow dotted arrows in Fig. 3. The evolution *needs* to cycle in S_1 for some time before any significant progress can be expected. The prerequisite for this is the balanced quality of initial two populations. Only two populations of the roughly same quality level can obey the interaction principles from Fig. 4.

Figure 4 depicts the influences the two populations' quality and size have on each other. Solid arrows represent positive influence and dashed arrows represent the negative impact of one entity on another. The "quality" is impossible to define (or else we'd have a perfect fitness function!) but is a property that should be maximized. This goal is achieved by the positive reinforcing loop: higher quality population will probably remove "low-quality" members from the lower quality population resulting in improvement of the average "quality" and reduction in size of the weaker population. The influence of increase in size on the opposing population's size and quality is sometimes positive and sometimes negative – many trivial solutions can be beneficial for the opposing population, large number of good solutions, however, can be catastrophic.

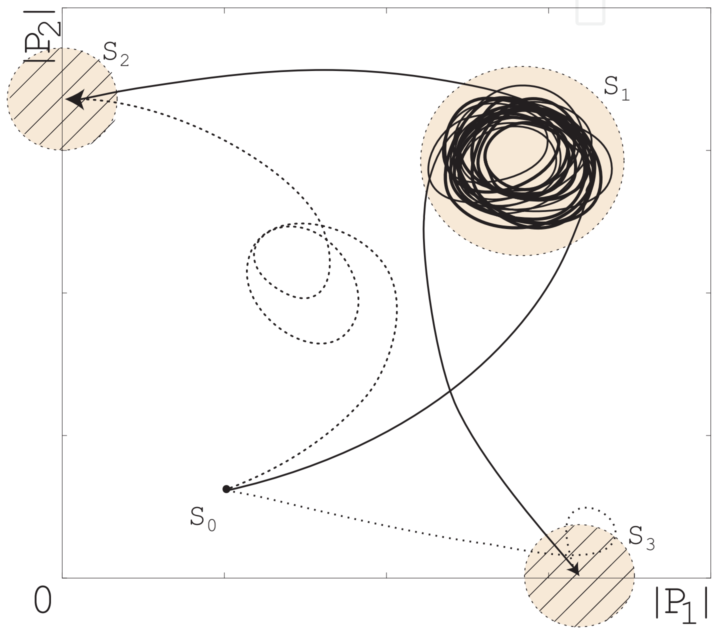

Fig. 3. Typical evolutionary scenarios.

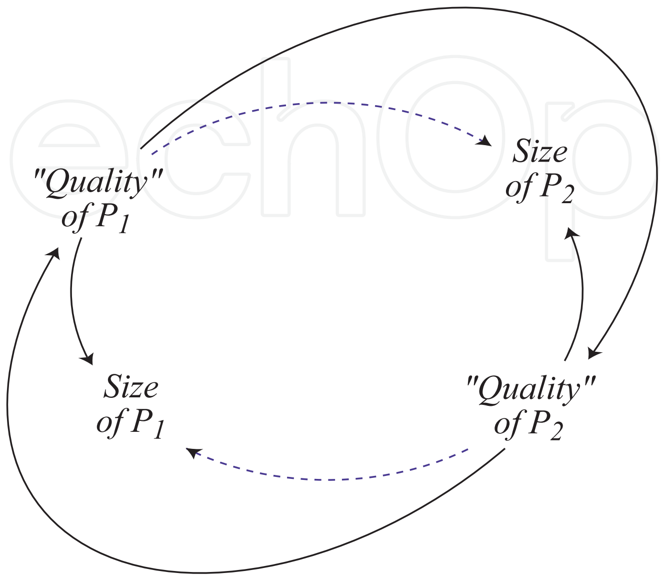

Fig. 4. Relations between quality and number of individuals.

This picture displays the complex interactions that drive the evolutionary search. The link between size and quality is dynamic. In the much studied predator-prey model (Blanchard et al., 1997), only the size of both populations is relevant while the quality by definition remains constant. In AEA the quality is expected to vary making this model more complex, though it should exhibit behavior similar to that of the predator-prey model.

3.5 Run termination in AEA

In standard EAs, run termination seems a non-problematic parameter: the EA run simply terminates if fitness hits a pre-set threshold or after a certain amount of processing has been done. The problem is again fitness. As discussed before, fitness is not an objective measure of success and can thus not be used as a termination criterion (see also section 4). The processing time, unfortunately, must remain a mandatory parameter for AEA, too.

Natural populations live in an ever-changing environment (S_1 area), where they're constantly challenged to improve their qualities. In history, the species became extinct for various reasons (for example the meteorite (supposedly) wiped out the dinosaurs, humans killed the dodo birds. . .). AEA treats one population as a species fighting for supremacy over another species. Because the two populations (P_1 and P_2) are genetically fully isolated, they can physically represent the same but "logically" different species. The main auto-termination criterion for the main loop is therefore the moment of absolute victory – the $|P_i|\rightarrow 0$ moment.

If there's no such event for a predefined amount of time, the evolution run can be interrupted artificially (just like in standard EAs). The problem remains, however, how to recognize/select the "best" solution from the remaining individuals. In traditional EAs, fitness-best individuals are proclaimed general solutions, yet there is no fitness criterion integrated within the AEA. One option is to use the classical fitness function just to select the resulting individual from the final population. This fitness is *not* used to guide the evolution in any way; it is rather calculated only after the evolution has already (been) stopped!

4. Case study

Symbolic function identification is often used as an illustrative example for evolutionary methods, especially genetic programming (Koza, 1992). Although simple regression problems are quite quickly solved by most GP implementations, more complicated or noisy problems remain a challenge. The presented case study focuses on the robustness of the evolved symbolic functions.

AEA can easily be used to evolve genetic programs. The standard GP representation of an individual – a tree-like structure – is convenient also for AEA. The tree consist of a number of nodes, each node representing one instruction to be executed. Size of the individual corresponds to the number of nodes in the tree.

The *success predicate* introduced by Koza requires perfect knowledge whether the solution is correct. The symbolic regression with *noisy* data set does not have a perfect fitness function nor perfect termination criterion. In order to determine the probability of satisfying the problem's success predicate, Koza measured the number of processed individuals. Here, we'll use the number of "function executions" as a measure of processing done by both algorithms.

4.1 Symbolic regression

Objective of this study is the discovery of a symbolic expression that satisfies a set of data points. Target function is the well known $t(x) = x^4 + x^3 + x^2 + x$ (Koza, 1992). In a perfect

learning environment GP excels in finding an exact solution because the data set includes no noise and an obvious fitness function is available.
To test the robustness of standard GP (SGP) and AEA-GP produced solutions, three data sets (L_0, L_1 and L_2) were created. Each data set included 41 points $\{(x_i,y_i)$ with $x_0 = -1$, $x_{i+1} = x_i + 0.05\}$. In L_0 the dependent values y_i equaled $t(x_i)$. In L_1 and L_2, however, Gaussian noise $N(\mu,\sigma)$ was added. For L_1 and L_2 the dependent variables were $y_i = t(x_i) + N(0,0.02)$ and $y_i = t(x_i) + N(0,0.5)$, respectively. Figure 5 shows the data points of all three learning data sets.

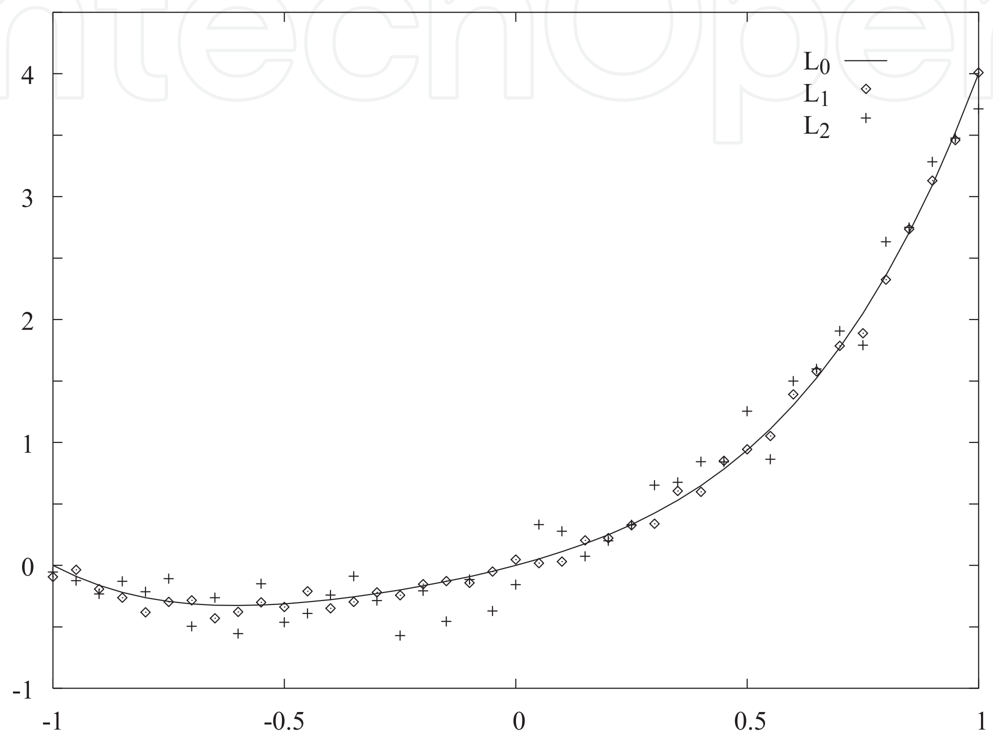

Fig. 5. The three data sets L_0, L_1 and L_2.

4.1.1 Standard GP

All SGP runs in this section were done by the OpenBeagle library (Gagne & Parizeau, 2006). When not explicitly stated otherwise, the default settings were population size 50, tournament selection 2, mutations 0.05 (including shrink-mutation), swap-sub-tree mutation 0.5, standard random-node mutation 0.05, crossover 0.9 and crossover point inside a tree 0.7. OpenBeagle used an adjusted fitness measure based on the accumulated error. The fitness value F of a symbolic function f on data L was calculated by

$$F(f,L) = \frac{1}{1+\sqrt{\sum \frac{(f(x_i) - L[x_i])^2}{|L|}}} \tag{5}$$

Another important criteria for a solution is size. In general, however, it is impossible to say whether a function comprising, for example 21 nodes, is always superior to the one comprising 25 nodes. Another problem are the introns. The effort to count function's introns would require additional processing and would slow down the evolution considerably. Because of this we chose to use the simple adjusted fitness measure only.

The described fitness measure needs to execute the function *f* exactly $|L|$-times, in our case 41-times. With the non-corrupted data set (L_0), the target function $t(x)$ had a fitness of 1. With noise the situation changed. Properties of the three data sets (L_0, L_1 and L_2) and the fitness values of target function $t(x)$ are shown in table 1.

data set	y	$F(y, L_i)$
L_0	$t(x)$	1
L_1	$t(x) + N(0, 0.05)$	0.9588
L_2	$t(x) + N(0, 0.2)$	0.8470

Table 1. Properties of the three data sets.

Both, OpenBeagle and AEA-GP, were equipped with the same function (FS) and terminal (TS) set:

```
FS = {+, -, *, \%, SIN, COS, EXP, RLOG},
TS = {x}
```

where '%' and 'RLOG' are protected division and protected logarithm, respectively. In order to choose the final solution from the evolved population, and to allow for a comparison with OpenBeagle, AEA-GP was equipped with the fitness measure (5).

4.1.2 OpenBeagle results

OpenBeagle offers a large number of available run-time options. The tests were performed using four different population sizes (P=50, 100, 200, 500). The evolution was interrupted if fitness hit the maximum or if the total number of fitness evaluations exceeded 1 million. Each configuration setup was used for 10 independent runs and the best-of-run individuals were recorded. Table 2 shows statistics for the hall-of-fame individuals of all 10 runs in each

data set	min	max	*mean* ± σ	$t(x)$
P=50				
L_0	0.9990	1.0000	0.9999 ± 0.0003	9/10
L_1	0.9704	0.9797	0.9744 ± 0.0034	0/10
L_2	0.8732	0.9295	0.9110 ± 0.0173	0/10
P=100				
L_0	0.9977	1.0000	0.9998 ± 0.0007	9/10
L_1	0.9540	0.9802	0.9706 ± 0.0081	0/10
L_2	0.9060	0.9513	0.9228 ± 0.0146	0/10
P=200				
L_0	0.9959	1.0000	0.9993 ± 0.0015	8/10
L_1	0.9628	0.9801	0.9742 ± 0.0053	0/10
L_2	0.9024	0.9441	0.9233 ± 0.0141	0/10
P=500				
L_0	1.0000	1.0000	1.0000 ± 0.0000	10/10
L_1	0.9597	0.9767	0.9707 ± 0.0054	0/10
L_2	0.8937	0.9310	0.9146 ± 0.0124	0/10

Table 2. OpenBeagle-GP scores for best-of-run individual's fitness values for respective data sets using different populations sizes, averaged over 10 independent runs.

category. It shows results for respective data sets (column 1), the minimum achieved fitness (column 2), maximum fitness (column 3), mean fitness with standard deviation (column 4) and count of runs producing the target $t(x)$ (column 5).
The clean learning data set L_0 was not problematic – OpenBeagle found the target $t(x)$ in 36 out of 40 runs. If the population size was set to 500 it even scored 10/10!
Due to the noise in L_1 and L_2, the fitness function was unsuccessful in recognizing the target $t(x)$ and headed straight towards functions that were over-fitted to the learning data. In 80 runs it never recognized $t(x)$ as the final solution and only once evolved a solution with a fitness score below $F(t(x))$. For both, L_1 and L_2, GP almost always encountered the function $t(x)$ *during* the run, but discarded it and proceeded towards greater fitness. This was because the chosen (or any other) fitness function could not compensate for the noise in the data.

4.1.3 Autonomous EA

The interaction between the two opposing AEA-GP individuals was based on the comparison of the absolute error both candidate functions made on one random learning instance. AEA-GP was set to terminate artificially if the number of function executions reached 41 million executions (one fitness evaluation in OpenBeagle was a calculation of 41 function values, thus the SGP run executed at most 41*1M=41M functions). Of course, AEA GP also auto-terminated if any population lost all of its members ($P \to 0$). At the end, the population was inspected and, according to the SGP's fitness measure, "best" solutions were recorded.
Table 3 shows statistics for best-fitness individuals averaged over 10 independent runs for three different memory settings per respective data set. The minimum, maximum, mean and standard deviation of highest fitness values at the end of each AEA run are presented. Additionally, the average number of interactions $\overline{I}$ pro run is displayed. Column 5 shows the count of perfect solutions t(x) produced in auto terminated runs ($P \to 0$). Last column counts the number of runs producing target function t(x) regardless of the termination criteria.

data set	min	max	$mean \pm \sigma$	$\overline{I}$	$t_{P\to 0}/P \to 0$	$t(x)/n$
M = 50000, I_{max}=41M						
L_0	0.8171	1.0000	0.9353 ± 0.0840	≈ 8.8M	4/6	6/10
L_1	0.8195	0.9588	0.8907 ± 0.0597	20.5M	0	4/10
L_2	0.7819	0.8553	0.8081 ± 0.0223	20.5M	0	0/10
M = 250000, I_{max}=41M						
L_0	0.8355	1.0000	0.9550 ± 0.0728	≈ 16.2M	3/3	7/10
L_1	0.8562	0.9588	0.9434 ± 0.0324	20.5M	0	7/10
L_2	0.7785	0.8573	0.8215 ± 0.0302	20.5M	0	0/10
M = 500000, I_{max}=41M						
L_0	0.8476	1.0000	0.9706 ± 0.0620	≈ 15.0M	4/4	8/10
L_1	0.9077	0.9600	0.9403 ± 0.0215	20.5M	0	4/10
L_2	0.7913	0.8676	0.8419 ± 0.0239	20.5M	0	0/10

Table 3. AEA statistics for best-of-run solutions with respective memory settings M.

Like SGP, AEA-GP was also successful in finding the target function $t(x)$ in problem set L_0. The more memory was available, the better were the results. The first row of table 3 shows that AEA-GP produced target function $t(x)$ 6 times in 10 runs; in 4 out of 6 cases, the perfect solution $t(x)$ was found after the evolution auto-terminated; in two cases the $t(x)$ was present in

the population P after the time-out of 20.5M interactions (2 evaluations per interaction make 41M evaluations). Interestingly, 2 auto-terminated runs did not produce $t(x)$. This could be attributed to smaller populations; M, set too low, increased the probability of the type $S_1 - S_3$ scenario (Fig. 3). Average AEA run on L_0 with M = 50000 nodes took 8.8M interactions.
When describing L_1 and L_2 with 100% accuracy, no straightforward function exists. The termination criterion $P \rightarrow 0$ was even less likely to be satisfied (as was the case with L_0) as the noise disturbed each partially winning function. Consequently, AEA-GP always terminated only after 20.5M interactions and most[6] fit individual was the pronounced solution of the run.
With L_1, AEA-GP was able to find target function $t(x)$ in 15 out of 30 runs. SGP, on the contrary, almost always ended up with an excessively over-fitted solution; only in one run out of 40 did it evolve a population with a maximal fitness *lower* than 0.9588. AEA-GP managed just the opposite: it evolved a solution with a fitness *higher* than 0.9588 in one run only.
With L_2, AEA-GP did not find target $t(x)$ in any of the 30 runs, but neither did SGP. SGP encountered several $t(x)$ quite early in the run but then discarded them in favor of other excessively over-fitted solutions. AEA, on the contrary, evolved towards $t(x)$ but failed to produce the desired target at the termination time[7].
Interestingly, with L_2, SGP's terminal population included the $t(x)$ once, yet SGP failed to *recognize* it. AEA-GP also saw $t(x)$ during the evolution. Due to the noise in L_2, however, the target vanished and was not present at termination time. If present, it would have been mostly recognized because the best individuals' mean fitness (0.8419) was always lower than $F(t(x))$ = 0.8470. SGP's achieved minimum fitness was always over-fitted well above that (0.8732).

4.2 Remarks

When comparing AEA with other evolutionary computing techniques, e.g. GP, special attention should be paid to the interpretation of the inherent time-line. GP terminates immediately upon encountering the first solution with the perfect fitness score (e.g. f = 1). Autonomous EA, on the contrary, auto-terminates only when P gets exhausted – this may have been long after the first 100% solution (f = 1) is found. It can be said that AEA is more time consuming than EA even though the AEA's interaction operator is mostly much faster than the full fitness calculation. On the other hand, the presented termination criterion is more problem independent than the common generational and/or success-oriented predicates, especially when without a perfect measure of quality – the case of most real-world problems.
Therefore, the traditional fitness-based EAs should be preferred over AEA if the search space is free of noise and if the learning set includes *all* possible instances and if it remains of manageable size (e.g. the n-multiplexer problem etc.). Namely, EAs converge faster if the solution can be described perfectly by a fitness function. In such cases their tendency to over-fit the training data is not problematic.

5. Conclusions

The presented AutonomousEA exploits and simplifies existing EA philosophy. It is based on a simulated interaction between two populations competing in a tournament-like manner.

[6] Again, fitness F is not optimal, but is most convenient.
[7] In some of the unofficial runs set to terminate at 40M interactions, AEA eventually produced $t(x)$ even in L_2.

The main loop guarantees each individual a momentarily action (life!) in a quasi-parallel style. The core algorithm continuously creates interactions of two random individuals from two opposing populations and takes care, that every individual gets its turn as soon as possible. The dynamic sizing of both populations is implicitly governed by the self-regulating principles (as in the predator-prey system), which determine the number of offspring and amount of mutations. The population size is therefore a direct consequence of individuals' ability to survive. This is how the AutonomousEA creates a link between the immeasurable qualitative and measurable quantitative properties.

Both initial random populations always include incompetent individuals, which are terminated during the evolution yet are necessary to shape the content and size of the population. Most of the processing time is used to create a population with precisely the right density of quality solutions. Higher density increases the probability of successful crossover and the creation of even better offspring, which in turn eliminates all individuals from the competitive environment. Empty environment is the termination criterion and signals a successful completion of the evolutionary search.

The AEA is based on the co-evolution of two populations of the individuals of the same type. The smaller (endangered) population evolves faster because all of its members are always active thus they have additional breeding chances compared to the larger population.

The concept of autonomous evolutionary algorithm needs only three run-time parameters:

- Initialization setting is a parameter needed in the initialization phase of the algorithm. It can be used to tailor the first evolvable objects (for example the GP tree initial depth setting).
- Memory space limits the population's size. The larger the value, the larger the two populations. This setting should be set high to fully exploit the available hardware, because large population sizes do not result in over-fitting problems as in standard EAs.
- Processing time is the artificial termination criterion because certain results must be delivered in due time.

AutonomousEA is very simple to *run* – the set-up phase is very similar to that of an EA yet for the run-phase only the memory space must be specified, all other details are self regulated. Traditional EAs, on the contrary, require much effort to determine just-the-right values for numerous parameters of the run phase.

Natural evolution is not under pressure to discover or optimize something. Rather, it goes different ways and something always pops out. The engineer, on the contrary, must hold artificial evolution in one direction. Use of fitness to specify this direction is problematic unless we're unable to create perfect fitness, because evolution will find a solution with best fitness but of small value.

The last section documented the AEA's performance in evolving genetic programs for the noisy symbolic regression problem. Comparison with standard GP gave an insight into the power of the AEA's principles regarding generalization capabilities of the produced solutions. Main problem of EAs was their tendency to over-fit the training data – the longer they were allowed to run or the larger the population size, the larger the discrepancy. AEA, on the contrary, kept close to the global generalization level. It found better solutions when more processing power (time) or memory was available.

5.1 Future work

Interesting sub-project was to use the adjusted fitness *F* instead of atomic tasks to decide the winner of an interaction. Although not in line with the AEA philosophy, it proved beneficial

for the symbreg example. On more difficult problems, however, it showed typical fitness-related problems (over-fitting. . .).
Also, different types of populations could compete under the AEA – for example a population of solutions could compete against a population of problems, what would allow for very elegant interaction implementation. This option, however, is problematic for many problem domains as it leads to premature convergence problem with instant AEA auto-termination. Fact is that both population must evolve in parallel. We cannot create a random initial population and expect it to solve difficult problems in the first try. A mechanism that would allow for "evolution" of problems is needed.

5.2 The code

Autonomous evolutionary algorithm relies on the best-ideas-are-simple philosophy – better EAs are more simple, not more complex. The AEA library and example projects in C++ are available from the author per email request (matej.sprogar@uni-mb.si).

6. References

Angeline, P. (1995). Adaptive and self-adaptive evolutionary computations, *in* M. Palaniswami & Y. Attikiouzel (eds), *Computational Intelligence: A Dynamic Systems Perspective*, IEEE Press, pp. 152–163.

Angeline, P. & Pollack, J. (1994). Competitive environments evolve better solutions for complex tasks, *Proceedings of the 5th International Conference on Genetic Algorithms (GA-93)*, pp. 264–270.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice*, Oxford University Press.

Banzhaf, W., Nordin, P., Keller, R. & Francone, F. (1998). *Genetic Programming – An Introduction*, Morgan Kaufmann, San Francisco.

Blanchard, P., Devaney, R. & Hall, G. (1997). *Differential Equations*, Brooks/Cole Publishing Company.

Bremermann, H. (1962). Optimization through evolution and recombination., *Self–Organizing Systems* pp. 93–106.

Dictionary.com (2006). fitness, *Based on the Random House Unabridged Dictionary* . URL: *http://dictionary.reference.com/browse/fitness*

Eiben, A., Hinterding, R. & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms, *IEEE–EC* 3(2): 124.

Friedberg, R. M. (1958). A learning machine, part I, *IBMJ. of Research and Development* 2: 2–13.

Friedberg, R.M., Dunham, B. & North, J. (1959). A learning machine, part II, *IBMJ. of Research and Development* 3: 282–287.

Gagne, C. & Parizeau, M. (2006). Genericity in evolutionary computation software tools: Principles and case-study, *International Journal on Artificial Intelligence Tools* 15(2): 173–194. URL: *http://www.worldscinet.com/109/15/1502/S021821300600262X.html*

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison– Wesley.

Henle, K. (1991). Some reflections on evolutionary theories, with a classification of fitness, *Acta Biotheoretica* 39(2): 91–106.

Hillis, D. (1990). Co–evolving parasites improve simulated evolution as an optimization procedure, *Physica D* 42: 228–234.

Holland, J. (1975). *Adaptation in natural and artificial systems*, The University of Michigan Press, Ann Arbor, MI.

Holland, J. (1995). *Hidden Order – How Adaptation Builds Complexity*, Addison–Wesley Publishing Company.

Juille, H. & Pollack, J. (2000). Coevolutionary learning and the design of complex systems, *Advances in Complex Systems* 2(4): 371–394.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Natural Selection*, MIT Press, Cambridge,MA.

Mandelbrot, B. & Hudson, R. (2004). *The (Mis)Behavior of Markets: A Fractal View of Risk, Ruin and Reward*, Basic Books.

Pagie, L. & Hogeweg, P. (2000). Information integration and red queen dynamics in coevolutionary optimization, *Proceedings CEC 2000*, pp. 1260–1267.

Pagie, L. & Mitchell, M. (2002). A comparison of evolutionary and coevolutionary search, *International Journal of Computational Intelligence and Applications* 2(1): 53–69.

Paredis, J. (1997). Coevolving cellular automata: be aware of the red queen!, *in* T. Bäck (ed.), *Proceedings of the 7th Int. Conference on Genetic Algorithms (ICGA 97)*, pp. 393–400.

Price, K. & Storn, R. (1997). Differential evolution: A simple evolution strategy for fast optimization, *Dr. Dobbs Journal of Software Tools* 22(4): 18–24.

Sareni, B. & Krähenbühl, L. (1998). Fitness sharing and niching methods revisited, *IEEE Transaction on Evolutionary Computation* 2(3): 97–106.

Stanovich, K. E. (2003). The fundamental computational biases of human cognition: Heuristics that (sometimes) impair reasoning and decision making, *in* J. E. Davidson & R. J. Sternberg (eds), *The psychology of problem solving*, Cambridge University Press, pp. 291–342.

Toffoli, T. (2000). What you always wanted to know about genetic algorithms but were afraid to hear. URL: *http://www.citebase.org/abstract?id=oai:arXiv.org:nlin/0007013*

Wolpert, D. &Macready,W. (1997). No free lunch theorems for optimization, *IEEE Transactions on Evolutionary Computation* 1(1): 67–82.

Products and Services; from R&D to Final Solutions
Edited by Igor Fuerstner

ISBN 978-953-307-211-1
Hard cover, 422 pages
Publisher Sciyo
Published online 02, November, 2010
Published in print edition November, 2010

Todayâ€™s global economy offers more opportunities, but is also more complex and competitive than ever before. This fact leads to a wide range of research activity in different fields of interest, especially in the so-called high-tech sectors. This book is a result of widespread research and development activity from many researchers worldwide, covering the aspects of development activities in general, as well as various aspects of the practical application of knowledge.