From Evolutionary Computation to the Evolution of Things

A.E. Eiben [1] and J.E. Smith [2]

1) VU University Amsterdam, de Boelelaan 1081a, 1081HV Amsterdam, The Netherlands

2) University of the West of England, Bristol, BS16 1QY, UK

Evolution has provided a source of inspiration for algorithm designers since the birth of computers. The resulting field, evolutionary computation, has been successful in solving engineering tasks ranging in outlook from the molecular to the astronomic. Today, the field is entering a new phase as evolutionary algorithms are developed that take place in hardware, opening up new avenues towards autonomous machines that can adapt to their environment. In this article we discuss how evolutionary computation compares to natural evolution, what its benefits are relative to other computing approaches and we introduce the emerging area of artificial evolution in physical systems.

The proposal that evolution could be used as a metaphor for problem-solving came with the invention of the computer[1]. In the 1970s and 1980s the principal idea was developed into different algorithmic implementations under names such as evolutionary programming[2], evolution strategies[3,4], and genetic algorithms[5], followed later by genetic programming[6]. These branches merged in the 1990s, and in the last twenty years so-called evolutionary computation or evolutionary computing (EC) has proven highly successful across a wide range of computational tasks in optimisation, design, and modelling[7,8,9]. For instance, Wang et al. addressed urgent needs in the development of low cost thin-film photovoltaic technologies, using genetic algorithms for topology optimisation. They obtained highly efficient light trapping structures that exhibited a more than 3-fold increase over a classical limit, and achieved efficiency levels far beyond the reach of intuitive designs[10]. Evolutionary approaches have also been convincingly demonstrated to be powerful methods for knowledge discovery. For example, Schmidt and Lipson[11] evolved equations based on motion-tracking data captured from various physical systems, ranging from simple harmonic oscillators to chaotic double-pendula. With very little prior knowledge about physics, kinematics, or geometry, their algorithm discovered several laws of geometric and momentum conservation, and uncovered the 'alphabet' used to describe those systems.

From the perspective of the underlying substrate, the emergence of evolutionary computation can be considered as a major transition of the evolutionary principles from 'wetware', the realm of biology, to software, the realm of computers. Today the field is at an exciting stage. New developments in robotics and rapid prototyping (3D printing) are paving the way towards a second major transition: this time from software to hardware, that is, going from digital evolutionary systems to physical ones[12,13].

In this paper we outline the working principles of evolutionary algorithms, and briefly discuss the differences between artificial and natural evolution. We illustrate the power of evolutionary problem-solving by discussing a number of successful applications, reflect on the features that make evolutionary algorithms so successful, and review the current trends of the field. Finally, we give our perspective on future developments.

Evolution and Problem Solving

The essence of an evolutionary approach to solve a problem is to equate possible solutions to individuals in a population, and to introduce a notion of fitness based on solution quality. To obtain a working evolutionary algorithm (EA) one has to go through a number of design steps. The first step is to identify a representation: that is to say a suitable data structure that can represent possible solutions to the problem. The next step is to define a way of measuring the quality of an individual based on problem specific requirements. The final step is to specify suitable selection and variation operators.

INSERT FIGURE 1 HERE

Analogous to natural evolution, an EA can be thought of as working on two levels. At the higher level (the original problem context), ***phenotypes*** (candidate solutions) have their **fitness** measured. ***Selection*** mechanisms then use this measure to choose a pool of parents for each

generation, and decide which parents and offspring go forward to the next generation. At the lower level, ***genotypes***, are objects that represent phenotypes in a form that can be manipulated to produce variations, see Box 1. The genotype-phenotype mapping bridges the two levels. At the genotypic level, variation operators generate new individuals (offspring) from selected parents. ***Mutation*** operators are based on one parent (asexual reproduction) and randomly change some values. ***Recombination*** operators create offspring by combining values from the genotypes of two (or more) parents. Finally, an ***Execution Manager*** controls the overall functioning of the algorithm. It regulates the initialisation of the first population, the execution of the selection-variation cycles, and the termination of the algorithm. It also manages the population size (typically kept constant) and other parameters affecting selection and variation. For instance, it determines the number of parents per generation, and whether mutation, recombination, or both produce the offspring for a given set of parents.

INSERT BOX 1 HERE

Evolutionary algorithms are easily transferable from one application to another because only two components, the way that the genotypes are converted to phenotypes, and the fitness function, are problem dependent. The history of EC has shown that suitable combinations of a few simple data structures can represent possible solutions to a huge variety of different problems (see Box 1). In other words, a relatively small collection of possible genotypes can accommodate many different kinds of phenotypes. Just as the genetic mechanisms underpinning natural evolution are largely species-independent, acting on DNA or RNA, so too in EC the choice of suitable variation operators depends solely on the data types present in the genotypes and not on the specific problem being tackled. Selection operators do not even depend on the chosen representation, as they only consider fitness information. This implies that for a certain problem a suitable EA can be designed easily, as long as the problem dependent phenotypes can be mapped to one of the 'standard' genotypes. From that point on, freely available EA machinery can be used.

It should be noted that just because an algorithm is formally suitable, that does not necessarily mean it will be successful. That is to say, suitability only means that the EA is capable of searching through the space of possible solutions of a problem, but gives no guarantees that this search will be either effective or efficient.

INSERT BOX 2 HERE

Positioning of Evolutionary Computation

From a historical perspective, humans have played two roles in evolution. Just like any other species, humans are the product of, and subject to, evolution. Also, for millennia – in fact for approximately twice as long we have used wheels – people have actively influenced the course of evolution in other species – by choosing which plants or animals should survive or mate. Thus humans have successfully exploited evolution to create improved food sources[14] or more useful animals[15], even though the mechanisms involved in the transmission of traits from one generation to the next were not understood.

Historically the scope of human influence in evolution was very limited, being restricted to interfering with selection for survival and reproduction. Influencing other components, such as the design of genotypes, or the mutation and recombination mechanisms, was far beyond reach. This was changed by the invention of the computer, that provided the possibility of creating digital worlds that are very flexible and much more controllable than the physical reality we live in. Together with the increased understanding of the genetic mechanisms behind evolution this brought about the opportunity to become active masters of evolutionary processes that are fully designed and executed by human experimenters 'from above'.

It could be argued that evolutionary algorithms are not faithful models of natural evolution. However, they certainly are a form of evolution. As phrased by Dennett[16]: "*If you have variation, heredity, and selection, then you must get evolution*". In Table 1 we compare natural evolution and artificial evolution as used in contemporary evolutionary algorithms.

INSERT TABLE 1 HERE

From the computer science perspective, evolutionary algorithms are randomized heuristic search methods based on the generate-and-test principle: producing an offspring amounts to

generating a new point in the search space, and testing is done through fitness evaluation. What distinguishes EAs from other algorithms in computer science is the unique combination of being stochastic and maintaining their working memory in the form of a population of candidate solutions. It should be noted that there are many variations of the generic EC template under various names. Today, the family of EAs includes historical members: genetic algorithms, evolution strategies, evolutionary programming, and genetic programming; and younger siblings, such as differential evolution and particle swarm optimization [17,18,19,20,21,22,23,24,25]. These differ in some details, terminology, or motivational metaphor, but are all in essence instances of the same algorithmic template.

It is common to categorise algorithms according to completeness (can they generate every possible solution), optimality (are they guaranteed to find the best solution, and identify it as such), and efficiency. The completeness of an EA can be achieved by an appropriate choice of representation and variation operators. The optimality is a more complex issue. While optimal methods exist for many problems, their run-time scales so poorly that they are impractical to use in most non-trivial cases – hence the interest in "heuristic" methods. As long as the heredity principle (similar individuals have similar fitnesses) holds, an EA will have a 'basic instinct' to improve the population's fitness over time, because the selection operators are biased towards choosing fitter individuals for reproduction and survival. Thus if we can define artificial fitness based on a criterion grounded in the problem to be solved then the EA will *tend* to find solutions that optimise the fitness values, or at least approximate them. This implies that EAs can be used to solve optimisation problems and, consequently, any problem that can be transformed into an equivalent optimisation task. This includes most problems in design, and those connected with building or learning models from data. Nevertheless, it is important to understand that EAs are not optimizers[26], but 'approximizers', and they are not optimal since we might not know whether a good evolved solution is the best possible. Nevertheless, they become very interesting when approximate solutions are acceptable, for instance, if the global optimum is not known or not required.

Applications of Evolutionary Computation

The hypothesis that embedding the principles of evolution within computer algorithms would create powerful mechanisms for solving difficult, poorly understood problems is now supported by a huge body of evidence. Evolutionary problem solvers have proven capable of delivering high quality solutions to hard problems in a variety of scientific and technical domains, offering several advantages over traditional optimisation and design methods.

One appealing example from the design domain concerns X-band antennas for NASA's ST5 spacecraft[27]. The normal approach to this task is very time and labour intensive, relying heavily on expert knowledge. The EA-based approach not only discovered effective antenna designs, but could also adjust designs quickly when requirements changed. One of these antennas was actually constructed and deployed on the ST5 spacecraft, thus becoming the first computer-evolved hardware in space. This project also demonstrates a specific advantage of evolutionary over manual design. The EAs generated and tested thousands of completely new solutions, many exhibiting unusual structures that expert antenna designers would be unlikely to produce. Evolutionary algorithms have been also successful in many other cases in aeronautical and aerospace engineering. Problems in this field typically suffer from highly complex search spaces and multiple conflicting objectives. Population-based methods such as EAs have proven effective at meeting the challenges of this combination. In particular, so-called multi-objective EAs, change the selection function to explicitly reward diversity, so that they discover and maintain high-quality solutions representing different trade-offs between objectives – technically they approximate diverse segments of the Pareto front[28]. Many examples can also been found in bio-informatics. For instance, Besnard et al. mined the ChEMBL database to discover a set of transformations of chemical structures that they then used as the mutation operator in an automated drug design application[29]. The results clearly showed benefits, particularly in accommodating multiple target profiles such as desired polypharmacology. This nicely illustrates how other approaches, or existing knowledge, can be easily co-opted or accommodated within an EC framework.

Numerical and combinatorial optimisation form important application areas of EAs. A particularly challenging class is black-box optimisation, where the nature of the objective function requires

numerical (rather than analytical) methods, and gradient information can only be approximated by sampling solutions. A recent systematic experimental study compared mathematical programming and evolutionary computation on a range of synthetic black-box optimisation problems when allowed different amounts of computing time and resources[30]. The results showed that mathematical programming algorithms –that were designed to provide quick progress in the initial stages of the search– outperform EAs if the maximum number of evaluations is low, but this picture changes if the computational budget is increased. Ultimately, the study concludes that an evolutionary algorithm, in particular the BIPOP-CMA-ES, is able to (1) find the optimum of a broader class of functions, (2) solve problems with a higher precision, and (3) solve some problems faster. The power of evolution strategies (especially the very successful CMA-ES variant[31]) for real-life black-box optimisation problems from industry is treated extensively in the recent book by Bäck et al.[32]. Their evidence, gathered from many years experience of both academic research, and development for industrial applications, suggests that the niche for evolution strategies is formed by optimisation tasks with a very limited budget for how many solutions can have their fitness evaluated. Although this finding is not in line with the 'common wisdom' within the field, the book, and the references therein, provide ample support for this proposal.

Machine learning and modelling is another prominent area where evolutionary algorithms have proved their power, especially as many contemporary approaches otherwise rely on (often crude) greedy or local search to refine and optimise models. For example, neuroevolutionary approaches use EAs to optimise the structure, the parameters, or both simultaneously, of artificial neural networks[33,34]. In other branches of machine learning, using EC to design algorithms has been shown to be very effective as an alternative to hand-crafting them, for instance, for inducing decision-trees[35]. Furthermore, EAs have been applied to prediction problems. For instance, Widera et al. tackled the problem of predicting the tertiary structure of a protein by evolving a key component of automated predictors, the function used to estimate a structure's energy[36]. State of the art methods in protein structure prediction are limited by assuming linear combination of energy terms here, whereas a genetic programming (GP) method easily accommodates expressions based on a much richer syntax. The best energy function found by the GP algorithm provided significantly better prediction guidance than traditional functions. Their algorithm was able to automatically discover the most and least useful energy terms, without having any knowledge of how these terms alone are correlated to the prediction error.

The design of controllers for physical entities – such as machinery or robots – has proved another fruitful area. For example, Filipic et al. evolved control strategies for operating container cranes using a physical crane to determine fitness values[37]. The evolution of controllers is also possible *in situ* – for example, in a population of robots during, and not just before, their operational period[38,39]. Evolutionary robotics[40] is an especially challenging application area because of two additional issues that other branches of EC do not face: the very weak and noisy link between controllable design details and the target feature(s); and the great variety of conditions under which a solution should perform well. Normally in EC there is a 3-step evaluation chain: genotype ⇒ phenotype ⇒ fitness; for robots the chain is 4-fold: genotype ⇒ phenotype ⇒ behaviour ⇒ fitness[a]. Furthermore, the behaviour depends on many external factors, creating an unpredictable environment in which the robot is expected to perform. Nevertheless, since the manual design of an autonomous and adaptive mobile robot is extremely difficult, evolutionary approaches offer large potential benefits. These include the possibility of continuous and automated design, manufacture, and deployment of robots of very different morphologies and control systems[41]. Several studies have demonstrated such benefits, where robot control systems that were automatically generated by artificial evolution were comparatively simpler or more efficient than those engineered using other design methods[42]. In all cases, robots initially exhibited uncoordinated behaviour, but a few hundreds of generations were sufficient to achieve efficient behaviours in a wide range of experimental conditions.

[a] In this 4-fold chain the robots morphology and controller form the phenotype. However, it could be argued that the behaviour should be considered as phenotype, because it is the entity that is being evaluated.

Several state-of-the-art algorithms for applications across a great variety of problem domains are based on hybridising evolutionary search with existing algorithms, especially local search methods. This kind of hybridization can be thought of as adding 'lifetime learning' to the evolutionary process. Freed from the restrictions of natural evolution (such as learned traits not being written back immediately to the genotype), and able to experiment with novel types of individual and social learning, the theory and practice of so-called memetic algorithms has become an important topic in the field[43,44,45]. Such hybrid algorithms can often find good (or better) solutions faster than a pure EA when the additional method searches systematically in the vicinity of good solutions, rather than relying on the more randomised search carried out by mutation[46,47]. For example, Smith et al. solved the Cell Suppression Problem – deciding which data cells to suppress in published statistical tables in order to protect respondent's confidentiality[48]. Using a combination of graph-partitioning, linear programming, and evolutionary optimisation of the sequence in which vulnerable cells were considered, they produced methods that could protect published statistical tables several orders of magnitude greater than had previously been possible. Memetic algorithms have obtained an eminent place among the best approaches to solving really hard problems.

State of the Art

Evolutionary algorithms have overcome considerable initial scepticism, and over the last twenty years evolutionary computation has grown to become a major field in computational intelligence[7,8,9]. As well as solving hard problems in various application areas, EAs have demonstrated that their emphasis on randomness as a source of variation has special advantages: the lack of problem-specific preconceptions and biases of the algorithm designer opens up the way to unexpected, 'original', solutions that can even have artistic value[49,50,51,52]. The perception of evolution as a problem solver has broadened from seeing evolution as a heuristic algorithm for (parametric) optimisation to considering it as a powerful approach to (structural) design[53,54].

In general, EAs have proven competitive in solving hard problems in the face of problem characteristics like non-differentiability, discontinuities, multiple local optima, noise, and nonlinear interactions among the variables, especially if the computational budgets are sufficiently high. Evolution is a slow learner, but the steady increase in computing power, and the fact that the algorithm is inherently suited to parallelisation, mean that more and more generations can be executed within practically acceptable wall-clock time.

The performance of evolutionary algorithms has also been compared to that of human experts, and there is now substantial and well-documented evidence of EAs producing measurably human-competitive results[55]. The annual competition for the so-called Humies Awards for human-competitive results highlight a great variety of hard problems for which EAs delivered excellent solutions[56].

The success and popularity of EAs can be attributed to a number of algorithmic features. In general, EAs are attractive because they are:

- *Assumption-free*. Applying an EA consists of specifying the representation for candidate solutions and providing an external function that first transforms the genotype into a candidate solution and then provides an evaluation. Internally EAs make no explicit assumptions about the problem, hence they are widely applicable and easily transferable at low cost.
- *Flexible*. EAs can easily be used in collaboration with existing methods such as local search. They can be incorporated within, or make use of, existing toolsets. Combinations with domain-specific methods often lead to superior solvers because they can exploit the best features of different approaches.
- *Robust*. The use of a population and the randomized choices mean EAs are less likely to get trapped in sub-optimal solutions than other search methods. They also are less sensitive to noise or infidelity in the models of the system used to evaluate solutions, and can cope with changes in the problem.
- *Not focussed on a single solution.* Having a population means that an EA terminates with a number of solutions. Thus, users do not have to pre-specify their preferences and weightings in advance, but can make decisions after they see what is possible to achieve. This is a great advantage for problems with many local optima, or with a number of conflicting objectives.

- *Capable of producing the unexpected.* EAs have often been shown to find effective, but non-intuitive solutions, as they are blind to human preconceptions – often valuable in design domains.

The theoretical underpinning of EAs remains a hard nut to crack. Mathematical analysis can illuminate some properties, but even digital evolutionary processes exhibit very complex dynamics that allows only limited theory forming, despite the diverse set of tools and methods ranging from quantitative genetics to statistical physics[57]. One important theoretical result is the No Free Lunch Theorem. This states that EAs are not generic super solvers – but neither is any other method, because there is no such thing[58]. Instead, "an EA is the second best solver for any problem", meaning that in many cases a carefully hand-crafted solver that exploits problem characteristics is superior for the problem at hand, but that it might take years to create that solver. A traditional issue for theorists is algorithm convergence. Early results were based on Markov-chain analysis addressing convergence in general[59], but more recent work found specific relationships between algorithmic setup and expected runtimes[60]. Despite all the difficulties, the field is making progress in theory[61,62].

Important Research Trends

The development of evolutionary computation continues along a number of research threads.

Automated design and tuning of EAs. Experience has shown that there are several design choices behind an EA that greatly influence its performance. To reduce the number of design decisions to be made, and the impact of poor choices, the community is working on automated design aids. These can customize an initial algorithm setup for a given problem off-line (before the run), or on-line (during the run)[63]. Techniques such as automated parameter tuning[64,65,66,67] and adaptive parameter control continue to make advances in this area[68,69,70,71].

Using surrogate models. Increasingly EAs are being used for problems where evaluating each population member over many generations would take too long to permit effective evolution given the resources available. A range of approaches, collectively known as surrogate models, are being developed that use computationally cheaper models in place of full fitness evaluations, and refine those models via occasional full evaluations of targeted individuals[72,73,74,75].

Handling many objectives. Having proven highly successful for finding solutions to problems with multiple objectives (typically up to ten)[76], the community is now making rapid advances in the field of many objectives – moving way beyond the capabilities of other algorithms[77,78,79]. In tandem with algorithmic advances, this has spurred renewed interest in Interactive Evolutionary Algorithms, which have been successfully applied to elicit user preferences and knowledge in many areas from design to art[51,52]. Recent results suggest a useful synergy, with periodic user interaction to incorporate preferences helping to focus search down to a more manageable set of dimensions[80]. Importantly, this involves eliciting user preferences in response to what is discovered to be possible, rather than *a priori*.

Generative and developmental representations. Further to the traditionally simple genotype-phenotype mappings, the use of indirect encodings is gaining traction. Such generative and developmental representations allow the reuse of code which helps scale up the complexity of artificially-evolved phenotypes, for instance, in evolutionary robotics, artificial life, and morphogenetic engineering[81,82,83,84,85].

Outlook

The range of problems to which EAs have been successfully applied has grown year-on-year, and there is every reason to expect this to continue. In the future we expect to see increasing interest in applying EAs to embodied/embedded systems: that is, employing evolution in populations where the candidate solutions are controllers or drivers that implement the operational strategy for some situated entities, and are evaluated within the context of some rich dynamic environment; not for what they *are*, but what they *do*. Examples include policies for Web-agents, information retrieval strategies, software for machinery and smart devices, and controllers for autonomous robots[86,87]. In such cases the evolved solutions are embedded in entities that exist and act in a 'habitat', the Internet or the physical world, that is too complex and dynamic to be modelled perfectly. Enhancing the system with the ability to evolve and adapt after deployment can complement the

offline optimisation approach employed during the design stage. The novelty of such systems is that evolutionary changes take place within the operational period. These systems will be different because they replace the traditional design-and-deploy approach by a design-deploy-adapt loop where the evolutionary component is a principal part of the system.

This approach is gaining traction in two areas already. In the field of Search Based Software Engineering, evolutionary algorithms are gaining prominence in response to the mismatch between the availability of expert software engineers, and the explosion of interconnected devices requiring new and/or updated software[88]. Meanwhile, recent developments in rapid fabrication technologies (3D printing) and ever smaller and more powerful robotic platforms mean that EC is now starting to make the next major transition to the automated creation of physical artefacts and "smart" objects[89]. On the long term this can lead to a disruptive robotic technology where design and production are replaced by selection and reproduction without the involvement of human designers and human operated facilities.

INSERT FIGURE 2 HERE

Last but not least, we foresee a fruitful cross-fertilisation with biology in the coming decade based on a bi-directional flow of inspiration, understanding, and know-how. On the one hand, the advancing insights in molecular and evolutionary biology can be used to make more sophisticated evolutionary algorithms and may help solve previously intractable problems. The opportunities and challenges of this avenue have been outlined in a research agenda to transform artificial evolution to computational evolution[90]. On the other hand, a new kind of artificial evolution –the Evolution of Things– opens new horizons for biologists. As early as 1992 John Maynard Smith commented: "*So far, we have been able to study only one evolving system and we cannot wait for interstellar flight to provide us with a second. If we want to discover generalizations about evolving systems, we have to look at artificial ones*"[91]. Artificial evolution implemented on real hardware, as in evolutionary robotics, offers a new research instrument to this end[42,92,93,94,95,96,97]. The use of real hardware overcomes the principal deficiency of software models, which lack the richness of matter that is a source of challenges and opportunities not yet matched in artificial algorithms[98]. Hence, they can provide new insights into fundamental issues such as the factors influencing evolvability, resilience, the rate of progress under various circumstances, or the co-evolution of mind and body. Using a non-biochemical substrate for such research is becoming technologically ever more feasible, and it increases the generalizability of the findings. In particular, using a different medium for evolutionary studies can separate generic principles and ground truth from effects that are specific for carbon-based life as we know it.

REFERENCES

1. Turing, A. Intelligent Machinery, (1948) report for National Physical Laboratory, in Machine Intelligence 7, B. Meltzer and D. Michie (eds.) 1969; also in Collected Works (Volume 1).
2. Fogel, L. Owens, A.J., and Walsh., M.J. *Artificial Intelligence Through Simulated Evolution.* (Wiley, New York, 1966).
3. Rechenberg, I. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. (Fromman-Hozlboog Verlag, Stuttgart, 1973).
4. Schwefel, H-P. *Numerical Optimization of Computer Models.* (Birkhäuser Verlag, Basel, 1977).
5. Holland, J.H. *Adaption in Natural and Artificial Systems.* (The University of Michigan Press, Ann Arbor, 1975).
6. Koza, J.R. *Genetic Programming*. (MIT Press, Cambridge, MA, 1992).
7. Eiben, A.E. and Smith, J.E. *Introduction to Evolutionary Computing* (Springer, 2003).
8. Ashlock, D. *Evolutionary Computation for Modeling and Optimization* (Springer, 2006).
9. De Jong, K. *Evolutionary Computation: A Unified Approach* (MIT Press, 2006).
10. Wang, C., Yu, S., Chen, W., and Sun, C. Highly Efficient Light-Trapping Structure Design Inspired By Natural Evolution. *Scientific Reports* 3, Article number: 1025 doi:10.1038/srep01025 (2013).
11. **Schmidt, M., and Lipson H. Distilling Free-Form Natural Laws from Experimental Data, *Science* 324, 81-85 (2009). Provides a forceful demonstration of the power of**

evolutionary methods for tasks that are considered to require highly educated scientist to perform.
12. Eiben, A.E., Kernbach, S., and Haasdijk, E. Embodied artificial evolution: Artificial evolutionary systems in the 21st Century, *Evolutionary Intelligence* **5(4)**, 261-272, (2012).
13. Eiben, A.E. In Vivo Veritas: Towards the Evolution of Things, in B. Filipic, T. Bartz-Beielstein, J. Branke, and J. Smith (Eds.), *Proceedings of the 13th International Conference on Parallel Problem Solving from Nature*, 24-39 (Springer, 2014).
14. Piperno, D. R.; Ranere, A. J.; Holst, I.; Iriarte, J.; Dickau, R. Starch grain and phytolith evidence for early ninth millennium B.P. maize from the Central Balsas River Valley, Mexico. *PNAS* **106 (13),** 4957–4958 (2009).
15. Akey, J. M.; Ruhe, A.L.; Akey, D.T.; Wong, A. K.; Connelly, C.F.; Madeoy, J., Nicholas, T.J.; Neff, M.W. Tracking footprints of artificial selection in the dog genome. *PNAS* **107 (3),** 1160-1165 (2010).
16. Dennett, D. *Darwin's Dangerous Idea.* (Penguin, London, 1995).
17. Goldberg, D. *Genetic Algorithms in Search, Optimization, and Machine Learning.* (Addison-Wesley, 1989).
18. Fogel, D.B. *Evolutionary Computation*. (IEEE Press, 1995).
19. Schwefel, H.-P. *Evolution and Optimum Seeking*. (Wiley, New York, 1995).
20. Bäck, T. *Evolutionary Algorithms in Theory and Practice*. (Oxford University Press, Oxford, UK, 1996).
21. Banzhaf, W., Nordin, P., Keller, R.E. and Francone, F.D. *Genetic Programming: An Introduction*. (Morgan Kaufmann, San Francisco, 1998).
22. Storn, R. and Price, K. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *J. Global Optimization* **11,** 341–359 (1997).
23. Price, K.V., Storn, R.N. and Lampinen, J.A. *Differential Evolution: A Practical Approach to Global Optimization*. Natural Computing Series. (Springer, 2005).
24. Kennedy, J.; Eberhart, R.C. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, 1942–1948 (IEEE Press, 1995).
25. Kennedy, J.; Eberhart, R.C. *Swarm Intelligence*. (Morgan Kaufmann, 2001).
26. De Jong, K.A., Are genetic algorithms function optimizers?, in Manner, R. and Manderick, B. (eds.), *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature*, 3-13 (North-Holland, Amsterdam, 1992).
27. Hornby, G.S., Lohn, J.D., Linden, D.S. Computer-Automated Evolution of an X-Band Antenna for NASA's Space Technology 5 Mission. *Evolutionary Computation* **19(1)**, 1–23 (2011).
28. Arias-Montano, A., Coello, C.A.C. and Mezura-Montes, E. Multiobjective Evolutionary Algorithms in Aeronautical and Aerospace Engineering, *IEEE Transactions on Evolutionary Computation* **16(5)**, 662–694 (2012).
29. Besnard, J. *et al.* Automated design of ligands to polypharmacological profiles. *Nature* **492** (7428), 215-220. (2012).
30. Posik, P. Huyer, W, and Pal, L. A Comparison of Global Search Algorithms for Continuous Black Box Optimization. *Evolutionary Computation* **20(4)**, 509–541 (2012).
31. **Hansen, N., and Ostermeier (2001). Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2), pp. 159-195 (2001). *Introduced the CMA-ES algorithm widely regarded as the state of the art in numerical optimisation.***
32. Bäck, T., Foussette, C., and Krause, P. *Contemporary Evolution Strategies.* (Springer, 2013).
33. **Yao,X. Evolving artificial neural networks,Proceedings of the IEEE, 87(9):1423-1447, 1999. Winner of the 2001 IEEE Donald G. Fink Prize Paper Award, this landmark paper brought together different strands of research and drew attention to the potential benefits of combining these two forms of learning.**
34. Floreano, D., Dürr, P., and Mattiussi, C. Neuroevolution: from architectures to learning. *Evolutionary Intelligence* **1(1)**, 47-62, (2008).
35. Barros, R.C., Basgalupp, M.P., de Carvalho, A.C.P.L.F. and Freitas, A.A. A Survey of Evolutionary Algorithms for Decision-Tree Induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C* **42(3)**, 291–312 (2012).

36. Widera, P., Garibaldi, J.M., and Krasnogor N. GP challenge: evolving energy function for protein structure prediction, *Genetic Programming and Evolvable Machines* **11**, 61–88 (2010).
37. Filipic, B., Urbancic, T., and Krizman, V., A combined machine learning and genetic algorithm approach to controller design. *Engineering Applications of Artificial Intelligence* **12**, 401-409 (1999).
38. Watson, R.A., Ficici, S.G. and Pollack, J.B. Embodied Evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems* **39(1)**, 1-18 (2002).
39. Bredeche, N., Montanier, J.M., Liu, W., and Winfield, A.F.T. Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. *Mathematical and Computer Modelling of Dynamical Systems* **18(1)**, 101-129, (2012).
40. S. Nolfi and D. Floreano. Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. (MIT Press, Cambridge, MA, 2000).
41. Bongard, J. Evolutionary robotics. *Communications of the ACM*, **56(8)**, 74–85 (2013).
42. Floreano, D. and Keller, L. Evolution of adaptive behavior in robots by means of Darwinian selection. *PLOS Biology*, 8(1):e1000292. doi:10.1371/journal.pbio.1000292 (2010).
43. **Hinton, G.E. and Nowlan, S.J. How learning can guide evolution. *Complex Systems* 1(3), 495-502 (1987). *Seminal paper that showed that learning can guide evolution even though characteristics acquired by the phenotype are not communicated to the genotype.***
44. Borenstein, E., Meilijson, I., and Ruppin, E. The effect of phenotypic plasticity on evolution in multipeaked fitness landscapes. *Journal of Evolutionary Biology* **19(5)**, 1555-1570 (2006).
45. Paenke, I., Jin, Y., and Branke, J. Balancing population and individual level of adaptation in changing environments. *Adaptive Behavior*, **17(2),** 153-174 2009.
46. Chen, X.S., Ong, Y. S., Lim, M.H., and Tan, K.C. A Multi-Facet Survey on Memetic Computation. *IEEE Transactions on Evolutionary Computation,* **15(5)**, 591–607 (2011). doi:10.1109/tevc.2011.2132725
47. Krasnogor N. and Smith J.E. A tutorial for competent memetic algorithms: Model, taxonomy and design issues. *IEEE Transactions on Evolutionary Computation*, **9(5)**, 474–488 (2005).
48. Smith, J.E., Clark, A.R., Staggemeier, A.T. and Serpell, M.C. A genetic approach to statistical disclosure control. *IEEE Transactions on Evolutionary Computation* **16(3)**, 431–441 (2012).
49. Bentley, P. and Corne, D. *Creative Evolutionary Systems* (Morgan Kaufmann, 2002).
50. Romero, J.J. and Machado P., (Eds.) The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music, (Springer 2008).
51. Secretan, J., Beato, N., D'Ambrosio, D.B., Rodriguez, A., Campbell, A., Folsom-Kovarik, J.T. and Stanley, K.O. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation* **19(3)**, 373-403 (2011).
52. EndlessForms, http://endlessforms.com/ (visited on December 12, 2014)
53. Bentley, P., Evolutionary Design by Computers, (Morgan Kaufmann, 1999).
54. Hingston, P.F., Barone, L.C. and Michalewicz, Z. (Eds.), Advances in Evolutionary Design, (Springer 2008).
55. **Koza, J.R. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3), 251–284 (2010). *Offers quantifiable definitions for human-competitiveness and a well-documented overview of success stories, including the first patents thought to be granted to inventions created by AI.***
56. The Annual "Humies" Awards – 2004-2014, http://www.genetic-programming.org/combined.php (visited on December 12, 2014)
57. Eiben, A.E. and Rudolph, G. Theory of evolutionary algorithms: a bird's eye view, *Theoretical Computer Science*, **229(1-2),** 3-9 (1999).
58. **Wolpert, D.H. and Macready, W.G. No Free Lunch theorems for optimisation. *IEEE Transactions on Evolutionary Computation* 1(1), 67–82 (1997). *Game-changing results that supported the change in focus in EC and other fields away from the search for a "super solver", and inspired insightful discussions that are still ongoing.***

59. Rudolph, G. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, **5(1)**, 96–101 (1994).
60. Lehre, P.R. and Yao, X. On the Impact of Mutation-Selection Balance on the Runtime of Evolutionary Algorithms, *IEEE Transactions on Evolutionary Computation* **16(2)**, 225-241 (2012).
61. Jansen, T. *Analyzing Evolutionary Algorithms: The Computer Science Perspective*. Natural Computing Series. (Springer, 2005).
62. **Borenstein, Y. and Moraglio, A. (eds.)** ***Theory and Principled Methods for Designing Metaheuristics*****. (Springer, 2014).** ***Provides good coverage of a range of the recent approaches and results in the theory of EAs.***
63. **Eiben, A.E., Hinterding, R. and Michalewicz, Z. Parameter control in evolutionary algorithms,** ***IEEE Transactions on Evolutionary Computation*****, 3(2), 124-141, (1999).** ***Had a long lasting effect by putting the issue of parameter calibration on the research agenda and establishing the corresponding conceptual framework; reference 71 is its recent follow-up.***
64. Bartz-Beielstein, T. T. *Experimental research in evolutionary computation: the new experimentalism*, (Springer, 2006).
65. Hutter, F., Hoos, H.H., Leyton-Brown, K., and Stützle, T. ParamILS: An Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research*, **36**, 267-306, (2009).
66. Eiben, A.E. and Smit, S.K. Parameter Tuning for Configuring and Analyzing Evolutionary Algorithms, *Swarm and Evolutionary Computation*, **1(1)**, 19-31, (2011).
67. Bartz-Beielstein, T. and Preuß, M. Experimental Analysis of Optimization Algorithms: Tuning and Beyond. In: Borenstein, Y. and Moraglio, A. (eds.) *Theory and Principled Methods for Designing Metaheuristics*. Chapter 10. 205–245, (Springer, 2014).
68. Lobo, F.J., Lima, Cláudio F., Michalewicz, Zbigniew (Eds.) *Parameter Setting in Evolutionary Algorithms,* (Springer, 2007).
69. M. Serpell and J.E. Smith. Self-Adaption of Mutation Operator and Probability for Permutation Representations in Genetic Algorithms. Evolutionary Computation, **18(3)**, 491–514, 2010.
70. Fialho, A., Da Costa, L., Schoenauer, M., Sebag, M. Analyzing bandit-based adaptive operator selection mechanisms, *Annals of Mathematics and Artificial Intelligence* **60(1-2)**, 25-64 (2010).
71. Karafotias, G., Hoogendoorn, M. and Eiben, A.E. Parameter Control in Evolutionary Algorithms: Trends and Challenges. *IEEE Transactions on Evolutionary Computation.* doi: 10.1109/TEVC.2014.2308294 (2014).
72. Jin, Y. A comprehensive survey of fitness approximation in evolutionary computation, *Soft Computing* **9(1)**, 3–12, 2005.
73. Jin, Y. Surrogate-assisted evolutionary computation: Recent advances and future challenges, *Swarm and Evolutionary Computation*, **1(2)**, 61–70, 2011.
74. Loshchilov, I., Schoenauer, M., and Sebag, M. Self-Adaptive Surrogate-Assisted Covariance Matrix Adaptation Evolution Strategy, in Soule, T. and Moore, J.H. (eds.) *Proceedings of the 2012 Conference on Genetic and Evolutionary Computation*, 321-328, (ACM Press, 2012).
75. Zaefferer, M., Stork, J., Friese, M., Fischbach, A., Naujoks, B., and Bartz-Beielstein, T. Efficient global optimization for combinatorial problems, in Igel, C. and Arnold, D.V. (eds.), *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, 871-878, (ACM Press, 2014)
76. Deb, K. *Multi-objective optimization using evolutionary algorithms*. (Wiley, 2001).
77. Zhang, Q. and Li, H. MOEA/D: A Multi-objective Evolutionary Algorithm Based on Decomposition, *IEEE Transactions on Evolutionary Computation*, **11(6),** 712-731 (2007).
78. Deb, K. and Jain, H. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. *IEEE Transactions on Evolutionary Computation*, **18(4),** 577-601 (2014).
79. Jain, H. and Deb, K. An Evolutionary Many-Objective Optimization Algorithm Using Reference-point Based Non-dominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach. *IEEE Transactions on Evolutionary Computation*, **18(4),** 602-622 (2014).

80. Branke, J., Greco, S., Slowinski, R. and Zielniewicz, P. Learning Value Functions in Interactive Evolutionary Multiobjective Optimization, *IEEE Transactions on Evolutionary Computation*, **19(1),** 88–102 (2014).
81. Stanley, K.O. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, **8(2),** 131–162 (2007).
82. O'Reilly, U.-M. and Hemberg, H. Integrating generative growth and evolutionary computation for form exploration. *Genetic Programming and Evolvable Machines,* **8(2),** 163-186 (2007).
83. Clune, J., Stanley, K.O., Pennock, R. and Ofria, C. On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation* **15(3),** 346–367 (2011).
84. Jin Y. and Meng Y. Morphogenetic robotics: An emerging new field in developmental robotics. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, **41(2),** 145-160 (2011).
85. Doursat, R., Sayama, H. and Michel, O. (eds.) *Morphogenetic Engineering: Toward Programmable Complex Systems* (Springer, 2013).
86. Doncieux, S., Bredeche, N., and Mouret, J.-B., (eds.) *New Horizons in Evolutionary Robotics* (Springer, 2011).
87. Vargas, P.A., Di Paolo, E.A., Harvey, I. and Husbands, P., (eds.) *The Horizons of Evolutionary Robotics,* Intelligent Robotics and Autonomous Agents series (MIT Press, 2014).
88. Harman, M. and McMinn, P. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *IEEE Transactions on Software Engineering* **36(2),** 226-247 (2010).
89. Preen, R. & Bull, L. Towards the Coevolution of Novel Vertical-Axis Wind Turbines. *IEEE Transactions on Evolutionary Computation*. doi: 10.1109/TEVC.2014.2316199 (2014).
90. Banzhaf W., Beslon, G., Christensen, S., Foster, J.A., Képès, F., Lefort, V., Miller, J.F., Radman, M., and Ramsden, J.J. From artificial evolution to computational evolution: a research agenda. *Nature Reviews Genetics* **7**, 729-735 (2006).
91. Maynard-Smith, J. Byte-sized evolution, *Nature* **355** (6363), 772-773. (1992).
92. Waibel, M., Floreano, D., and Keller, L. A quantitative test of Hamilton's rule for the evolution of altruism. *PLOS Biology* 9(5):e1000615. doi:10.1371/journal.pbio.1000615 (2011).
93. Long, J. *Darwin's Devices: What Evolving Robots Can Teach Us About the History of Life and the Future of Technology* (Basic Books, New York, 2012).
94. Virgo, N., Fernando, C., Bigge, B., and Husbands, P. Evolvable Physical Self-Replicators. *Artificial Life* **18(2)**, 129-142 (2012).
95. Bongard, J. and Lipson, H. Evolved machines shed light on robustness and resilience. *Proceedings of the IEEE* **102(5)**, 899-914 (2014).
96. Bongard, J. Morphological change in machines accelerates the evolution of robust behavior. *PNAS* 108(4), 1234-1239 (2011). *Demonstrated a hitherto unknown relationship between development, evolution, morphology, and the neural control of behavior as phrased by the title.*
97. Eiben, A.E. Grand challenges for evolutionary robotics, *Frontiers in Robotics and AI* **1(4)**, doi:10.3389/frobt.2014.00004, (2014).
98. Fernando, C., Kampis, G., and Szathmáry, E. Evolvability of natural and artificial systems. *Procedia Comput. Sci.* **7**, 73–76. doi:10.1016/j.procs.2011.12.023, (2011).

Table 1: Main differences between natural evolution and evolutionary algorithms

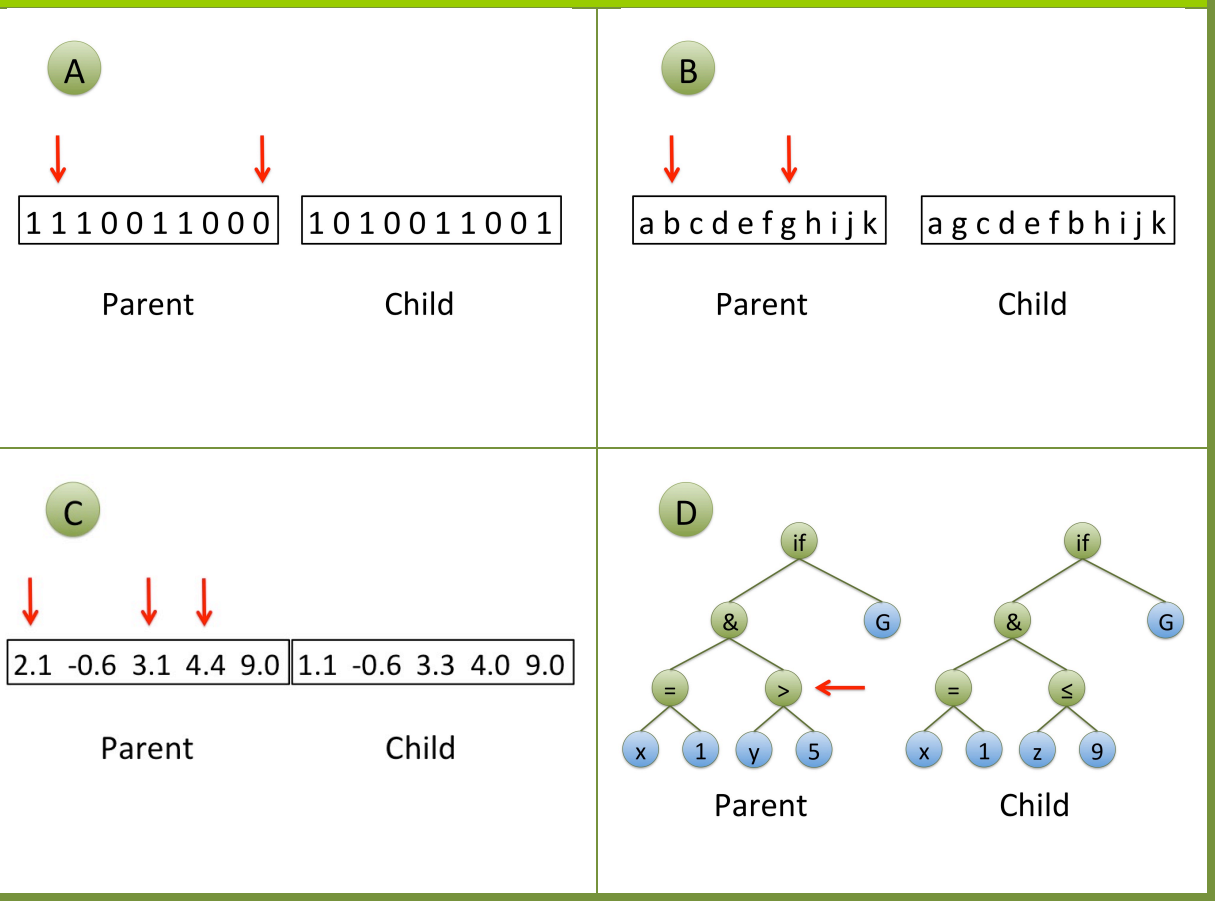
	Natural evolution	Evolutionary algorithms
Fitness	Observed quantity: *a posteriori* effect of selection and reproduction ("in the eye of the observer").	Predefined *a priori* quantity that drives selection and reproduction.
Selection	Complex multi-factor force based on environmental conditions, other individuals of the same species and other species (e.g., predators). Viability is tested continually; reproducibility is tested at discrete times.	Randomized operator with selection probabilities based on given fitness values. Parent selection and survivor selection both happen at discrete times.
Genotype-phenotype mapping	Highly complex biochemical and developmental process influenced by the environment.	Typically a simple mathematical transformation or parameterised procedure. A few systems use generative and developmental genotype-phenotype maps.
Variation	Offspring are created from one (asexual reproduction) or two parents (sexual). Horizontal gene transfer can accumulate genes from more individuals.	Unconstrained vertical gene transfer. Offspring may be generated from any number of parents, one, two, or many.
Execution	Parallel, decentralized execution; birth and death events are not synchronised.	Typically centralized with synchronised birth and death.
Population	Spatial embedding implies structured populations. Population size varies according to the relative number of death and birth events. Populations can and do go extinct.	Typically unstructured and panmictic (all individuals are potential partners). Population size is usually kept constant by synchronising time and number of birth and death events.

Figure 1: The principal diagram of evolutionary algorithms.

Figure 2: Two major transitions in the history of artificial evolution. In the 20th century computer technology enabled artificial Darwinian processes *in silico* - the evolution of digital entities. In the 21st century, developments in robotics, material science, and 3D printing enable the evolution of physical artefacts or machines.

Box 1: Examples of data structures frequently used as genotypes, and one suitable mutation operator for each, with its action shown by red arrows. Note that the mutation operator must deliver a child of the same data type – e.g. a valid mutation operator for permutations must result in a valid permutation. Complex problems might require complex genotypes with appropriate mutation operators.

A. Bitstrings are the natural choice for problems where solutions are composed from on/off or true/false decisions. The most commonly used mutation operator makes an independent choice in each position whether to invert the bit value.
B. Permutations can be used when the problem involves ordering a set of entities, such as in routing or scheduling. One simple mutation operator swaps the values in two randomly selected locations.
C. Real-valued vectors can capture continuous optimisation problems- for example where the variables represent quantities such as dimensions, or mass. Typically the mutation operator perturbs each value by adding a (small) random number.
D. Trees are branching data structures suitable for representing equations, logical expressions or program code. A common mutation operator selects a node at random, and replaces the subtree below with a new, randomly generated one.



BOX 2: A specific evolutionary algorithm for model discovery and fitting

Model fitting is a problem that frequently occurs in scientific modelling, knowledge discovery, and data mining. The task is to find a predictive model for a given data set, that is, a mapping from the independent variables describing a data point (input) onto the corresponding dependent variable (output). A working EA for this problem could be built as follows.

Representation. The phenotypes are models, whose complexity may be unknown a-priori. Variable-sized trees are one suitable data structure to use as genotypes to represent these models. The set of possible symbols used in the trees reflects what is known about the problem to be solved.

Fitness. The error of a given model (phenotype) is defined as the sum of its prediction errors over all data points. Depending on the application, this sum might be weighted – for example to penalise false negatives. The fitness of a tree (genotype) is defined as the inverse of the error of the corresponding phenotype. Thus, higher fitness corresponds to lower error.

Variation. A suitable mutation operator for trees is random replacement of a single sub-tree in a parent as illustrated in Box 1. A suitable recombination operator is the exchange of randomly selected subtrees from two parents to produce two children.

Selection. Parents can be selected by choosing *n* individuals with a probability that is proportional to their fitness. After creating *n* offspring individuals through variation operators and establishing their fitness, survivor selection removes *n* old individuals to be replaced by the offspring.

Execution manager. Let us use a population of 100 models, and select 50 as parents that are randomly paired. For each pair we produce 2 offspring by recombination, resulting in 50 intermediary offspring. These are then individually mutated to obtain 50 children. The next generation is then composed from these 50 children and the 50 parents. This results in a new generation, where half of the individuals are new.