REVIEW

doi:10.1038/nature14544

From evolutionary computation to the evolution of things

Agoston E. Eiben[1] & Jim Smith[2]

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 astronomical. Today, the field is entering a new phase as evolutionary algorithms that take place in hardware are developed, opening up new avenues towards autonomous machines that can adapt to their environment. We discuss how evolutionary computation compares with natural evolution and 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 past 20 years so-called evolutionary computation or evolutionary computing has proven to be highly successful across a wide range of computational tasks in optimization, design and modelling[7–9]. For instance, urgent needs in the development of low-cost thin-film photovoltaic technologies were addressed using genetic algorithms for topology optimization[10]. This led to highly efficient light-trapping structures that exhibited more than a threefold increase over a classic limit, and achieved efficiency levels far beyond the reach of intuitive designs. It has also been convincingly demonstrated that evolutionary approaches are powerful methods for knowledge discovery. For example, equations were evolved to model motion-tracking data captured from various physical systems, ranging from simple harmonic oscillators to chaotic double pendula. This approach discovered, with limited prior knowledge of physics, kinematics or geometry, several laws of geometric and momentum conservation, and uncovered the 'alphabet' used to describe those systems[11].

From the perspective of the underlying substrate in which the evolution takes place, 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: from software to hardware, going from digital evolutionary systems to physical ones[12,13].

In this Review 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, review the current trends of the field, and 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 on the basis of solution quality. To obtain a working evolutionary algorithm one has to go through a number of design steps. The first step is to identify a representation: 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 (Fig. 1).

Analogous to natural evolution, an evolutionary algorithm 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 (Box 1). 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 initialization 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.

Evolutionary algorithms are easily transferable from one application to another because only two components are problem dependent: the way that the genotypes are converted to phenotypes and the fitness function. The history of evolutionary computation has shown that suitable combinations of a few simple data structures can represent possible solutions to a huge variety of different problems (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 evolutionary computation the choice of suitable variation operators depends solely on the data structure 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 evolutionary algorithm can be designed easily, as long as the

[1]VU University Amsterdam, de Boelelaan 1081a, 1081HV Amsterdam, the Netherlands. [2]University of the West of England, Bristol BS16 1QY, UK.

problem-dependent phenotypes can be mapped to one of the 'standard' genotypes. From that point on, freely available evolutionary algorithm machinery can be used.

It should be noted that just because an algorithm is formally suitable, it does not necessarily mean it will be successful. Suitability only means that the evolutionary algorithm 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.

Positioning of evolutionary computation

From a historical perspective, humans have had two roles in evolution. Just like any other species, humans are the product of, and are subject to, evolution. But for millennia (in fact, for about twice as long as we have used wheels) people have also 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 mutation and recombination mechanisms, was far beyond our reach. This changed with the invention of the computer, which 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 (Table 1). However, they certainly are a form of evolution. As Dennett[16] said "If you have variation, heredity, and selection, then you must get evolution".

From a 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 evolutionary algorithms 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 evolutionary computation template under various names. Today, the family of evolutionary algorithms includes historical members: genetic algorithms, evolution strategies, evolutionary programming and genetic programming; and younger siblings, such as differential evolution and particle swarm optimization[17–25]. These

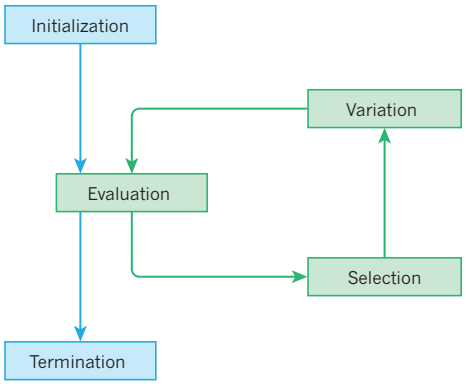

Figure 1 | The principal diagram of evolutionary algorithms. The initialization process seeds the search with a population of randomly created solutions. After this the algorithm enters a loop of evaluating the current generation of solutions, selecting some to act as the basis for the next generation, and then creating new solutions through variation (mutation or crossover). Periodically, the algorithm checks to see whether user-specified termination criteria are met — such as reaching a desired level of fitness, or undergoing a certain number of generations without improvement.

BOX 1

Representing solutions to problems as genotypes

Genotypes are objects that represent phenotypes (candidate solutions to problems) in a form that can be manipulated to produce variations.

Examples of data structures frequently used as genotypes are shown in the Figure. One suitable mutation operator is shown for each, with its action shown by red arrows. Note that the mutation operator must deliver a child of the same data type — for example, a valid mutation operator for permutations must result in a valid permutation. Complex problems might require complex genotypes with appropriate mutation operators.

Bitstrings are the natural choice for problems for which solutions are composed of 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.

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.

Real-valued vectors can capture continuous optimization 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.

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.

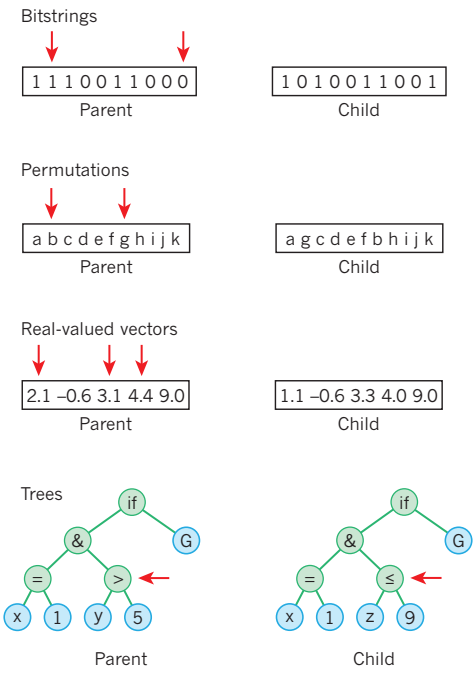

Table 1 | Main differences between natural evolution and evolutionary algorithms

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 multifactor force based on environmental conditions, other individuals of the same species and those of other species (predators). Viability is tested continually; reproducibility is tested at discrete times.	Randomized operator with selection probabilities based on given fitness values. Survivor selection and parent 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 parameterized procedure. A few systems use generative and developmental genotype–phenotype maps.
Variation	
Offspring are created from one (asexual reproduction) or two parents (sexual reproduction). 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 synchronized.	Typically centralized with synchronized birth and death.
Population	
Spatial embedding implies structured populations. Population size varies according to the relative number of birth and death events. Populations can and do go extinct.	Typically unstructured and panmictic (all individuals are potential partners). Population size is usually kept constant by synchronizing time and number of birth and death events.

differ in some details, terminology or motivational metaphor, but are, in essence, all instances of the same algorithmic template.

It is common to categorize 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 evolutionary algorithm can be achieved by an appropriate choice of representation and variation operators. The optimality is a more complex issue. Although 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 fitness) holds, an evolutionary algorithm 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 on the basis of a criterion grounded in the problem to be solved then the evolutionary algorithm will tend to find solutions that optimize the fitness values, or at least approximate them. This implies that evolutionary algorithms can be used to solve optimization problems and, consequently, any problem that can be transformed into an equivalent optimization task. This includes most problems in design, and those connected with building or learning models from data. Nevertheless, it is important to understand that evolutionary algorithms are not optimizers[26], but approximators, and they are not optimal since we might not know whether the fitness of the best evolved solution is in fact the highest value possible. Yet, 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 can 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 difficult problems in a variety of scientific and technical domains, offering several advantages over conventional optimization and design methods.

One appealing example from the design domain concerns X-band antennas for the NASA Space Technology 5 (ST5) spacecraft[27]. The normal approach to this task is very time and labour intensive, relying heavily on expert knowledge. The evolutionary-algorithm-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 evolutionary algorithms generated and tested thousands of completely new solutions, many with unusual structures that expert antenna designers would be unlikely to produce. Evolutionary algorithms have also been successful in many other aeronautical and aerospace engineering endeavours. Problems in this field typically have highly complex search spaces and multiple conflicting objectives. Population-based methods such as evolutionary algorithms have proven effective at meeting the challenges of this combination. In particular, so-called multi-objective evolutionary algorithms 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 be found in bioinformatics. For instance, by mining the ChEMBL database (which contains bioactive molecules with drug-like properties), a set of transformations of chemical structures was identified that were then used as the mutation operator in an automated drug-design application[29]. The results showed clear 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 evolutionary computing framework.

Numerical and combinatorial optimization are important application areas of evolutionary algorithms. Particularly challenging is black-box optimization, where the nature of the objective function requires numerical (rather than analytical) methods, and gradient information can only be approximated by sampling solutions. A systematic experimental study compared mathematical programming and evolutionary computation of a range of synthetic black-box optimization problems, which were 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 evolutionary algorithms 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 BIPOP-CMA-ES, is able to find the optimum of a broader class of functions, solve problems with

a higher precision and solve some problems faster. The power of evolution strategies (especially the very successful CMA-ES variants[31]) for real-life black-box optimization problems from industry has been discussed extensively[32]. Evidence gathered from years of academic research and development for industrial applications suggests that the niche for evolution strategies is formed by optimization tasks with a very limited budget for how many solutions can have their fitness evaluated. Although this finding is not in line with conventional wisdom within the field, there is ample support for this proposal.

Machine learning and modelling is another prominent area in which evolutionary algorithms have proved their power, especially as many contemporary approaches would otherwise rely on (often crude) greedy or local search algorithms to refine and optimize models. For example, neuroevolutionary approaches use evolutionary algorithms to optimize the structure, parameters, or both simultaneously, of artificial neural networks[33,34]. In other branches of machine learning, using evolutionary computing to design algorithms has been shown to be very effective as an alternative to handcrafting them, for instance, for inducing decision trees[35]. Furthermore, evolutionary algorithms have been applied to prediction problems. For instance, to tackle the problem of predicting the tertiary structure of a protein, an algorithm was designed to evolve 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 a linear combination of energy terms, whereas a genetic programming method easily accommodates expressions based on a much richer syntax. The best energy function found by the genetic programming algorithm provided significantly better prediction guidance than conventional functions. The 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 to be another fruitful area. For example, control strategies for operating container cranes were evolved 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 evolutionary computing do not face: the very weak and noisy link between controllable design details and the target feature or features; and the great variety of conditions under which a solution should perform well. Normally in evolutionary computing there is a three-step evaluation chain: genotype to phenotype to fitness. For robots the chain is four-step: genotype to phenotype to behaviour to fitness. In this four-step 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. 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, in which 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.

Several state-of-the-art algorithms for applications across a great variety of problem domains are based on hybridizing 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 being 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[43–45]. Such hybrid algorithms can often find good (or better) solutions faster than a pure evolutionary algorithm when the additional method searches systematically in the vicinity of good solutions, rather than relying on the more randomized search carried out by mutation[46,47]. For example, the cell suppression problem (deciding which data cells to disclose in published statistical tables in order to protect respondents' confidentiality)[48] was solved using a combination of graph partitioning, linear programming and evolutionary optimization of the sequence in which vulnerable cells were considered. This produced methods that could protect published statistical tables at a size that was 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.

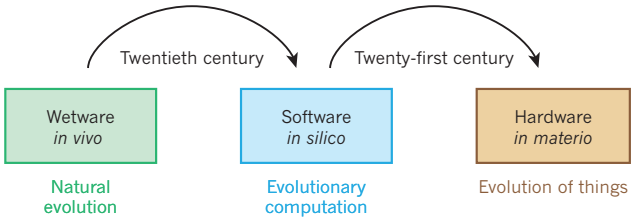

Figure 2 | Two major transitions in the history of artificial evolution. In the twentieth century computer technology enabled artificial Darwinian processes *in silico* — the evolution of digital entities. In the twenty-first century, developments in robotics, materials science and 3D printing will enable the evolution of physical artefacts or machines.

State of the art

Although initially considerable scepticism surrounded evolutionary algorithms, over the past 20 years evolutionary computation has grown to become a major field in computational intelligence[7–9]. As well as solving hard problems in various application areas, the emphasis of evolutionary algorithms on randomness as a source of variation has been shown to have particular 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[49–51] (http://endlessforms.com/). The perception of evolution as a problem solver has broadened from seeing evolution as a heuristic algorithm for (parametric) optimization to considering it to be a powerful approach for (structural) design[52,53].

In general, evolutionary algorithms have proven competitive in solving hard problems in the face of challenging 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 parallelization, mean that more and more generations can be executed within practically acceptable timescales.

The performance of evolutionary algorithms has also been compared with that of human experts, and there is now substantial and well-documented evidence of evolutionary algorithms producing measurably human-competitive results[54]. The annual Humies competition (http://www.genetic-programming.org/combined.php), which rewards human-competitive results from evolutionary computation, highlights the great variety of hard problems for which evolutionary algorithms have delivered excellent solutions.

The success and popularity of evolutionary algorithms can be attributed to a number of algorithmic features, which makes them attractive. First, they are assumption free because applying an evolutionary algorithm 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,

evolutionary algorithms make no explicit assumptions about the problem, hence they are widely applicable and easily transferable at low cost. Second, they are flexible because they can be easily used in collaboration with existing methods, such as local search; they can be incorporated within, or make use of, existing toolsets; and combinations with domain-specific methods often lead to superior solvers because they can exploit the best features of different approaches. Third, they are robust, owing to the use of a population and randomized choices, which mean evolutionary algorithms are less likely to get trapped in suboptimal solutions than other search methods. They are also less sensitive to noise or infidelity in the models of the system used to evaluate solutions, and can cope with changes in the problem. Fourth, they are not focussed on a single solution because having a population means that an algorithm terminates with a number of solutions. Thus, users do not have to pre-specify their preferences and weighting 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. Finally, they are capable of producing unexpected solutions because they are blind to human preconceptions and so can find effective, but non-intuitive solutions, which are often valuable in design domains.

The theoretical underpinning of evolutionary algorithms remains a hard nut to crack. Mathematical analysis can illuminate some properties, but even digital evolutionary processes exhibit very complex dynamics that allow only limited theory forming, despite the diverse set of tools and methods ranging from quantitative genetics to statistical physics[55]. One important theoretical result is the no free lunch theorem. This states that evolutionary algorithms are not generic super solvers — but neither is any other method, because there is no such thing[56]. Instead, "an evolutionary algorithm 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 long-standing issue for theorists is algorithm convergence. Early results were based on Markov-chain analysis and addressed convergence in general[57], but more recent work found specific relationships between algorithmic set-up and expected run times[58]. Despite all the difficulties, the field is making progress in theory[59,60].

Important research trends

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

Automated design and tuning of evolutionary algorithms

Experience has shown that there are several design choices behind an evolutionary algorithm 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 set-up for a given problem offline (before the run) or online (during the run)[61]. Techniques such as automated parameter tuning[62–65] and adaptive parameter control continue to make advances in this area[66–69].

Using surrogate models

Increasingly, evolutionary algorithms are being used for problems in which 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 that refine those models through occasional full evaluations of targeted individuals[70–73].

Handling many objectives

Having proven highly successful for finding solutions to problems with multiple objectives (typically up to ten)[74], the community is now making rapid advances in the field of many objectives — moving way beyond the capabilities of other algorithms[75–77]. 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] (http://picbreeder.org). Results suggest a useful synergy with periodic user interaction to incorporate preferences that help to focus the search down to a more manageable set of dimensions[78]. 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 conventionally simple genotype–phenotype mappings, the use of indirect encodings is gaining traction. Such generative and developmental representations allow the reuse of code, which helps to scale up the complexity of artificially evolved phenotypes, for instance, in evolutionary robotics, artificial life and morphogenetic engineering[79–83].

Outlook

The range of problems to which evolutionary algorithms 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 evolutionary algorithms to embodied or embedded systems; that is, employing evolution in populations for which 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 for what they do. Examples include policies for Web-crawlers, information retrieval strategies, software for machinery and smart devices, and controllers for autonomous robots[84,85]. 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 optimization 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 conventional design-and-deploy approach by a design–deploy–adapt loop in which the evolutionary component is a principal part of the system.

This approach is already gaining traction in two areas. 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[86]. Meanwhile, recent developments in rapid fabrication technologies (3D printing) and ever smaller and more powerful robotic platforms mean that evolutionary computing is now starting to make the next major transition to the automated creation of physical artefacts and 'smart' objects[87] (Fig. 2). In the long term this could lead to a disruptive robotic technology in which design and production are replaced by selection and reproduction without the involvement of human designers and human-operated facilities.

Last but not least, we foresee a fruitful cross-fertilization with biology in the coming decade based on a bidirectional flow of inspiration, understanding and knowledge. On the one hand, the advancing insights in molecular and evolutionary biology can be used to make more sophisticated evolutionary algorithms and may help to 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[88]. On the other hand, a new kind of artificial evolution — the evolution of things — opens new horizons for biologists. In 1992, the evolutionary biologist 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 will have to look at artificial ones"[89]. Artificial evolution implemented on real hardware, as in evolutionary robotics, offers a new research instrument to this end[42,90–95]. 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[96]. 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. ■

Received 18 December 2014; accepted 24 March 2015.

1. Turing, A. M. in *Machine Intelligence 5* (eds Meltzer, B. & Michie, D.) (Edinburgh Univ. Press, 1969).
2. Fogel, L. Owens, A. J. & Walsh. M. J. *Artificial Intelligence Through Simulated Evolution* (Wiley, 1966).
3. Rechenberg, I. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution* [in German] (Fromman-Hozlboog, 1973).
4. Schwefel, H.-P. *Numerical Optimization of Computer Models* (Birkhäuser, 1977).
5. Holland, J. H. *Adaption in Natural and Artificial Systems* (Univ. Michigan Press, 1975).
6. Koza, J. R. *Genetic Programming* (MIT Press, 1992).
7. Eiben, A. E. & 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. & Sun, C. Highly efficient light-trapping structure design inspired by natural evolution. *Sci. Rep.* **3,** 1025 (2013).
11. Schmidt, M. & Lipson, H. Distilling free-form natural laws from experimental data. *Science* **324,** 81–85 (2009).
    This paper provides a forceful demonstration of the power of evolutionary methods for tasks that are thought to require highly educated scientists to perform.
12. Eiben, A. E., Kernbach, S. & Haasdijk, E. Embodied artificial evolution: artificial evolutionary systems in the 21st Century. *Evol. Intel.* **5,** 261–272 (2012).
13. Eiben, A. E. in *Parallel Problem Solving from Nature – PPSNXII* (eds Filipic, B., Bartz-Beielstein, T. Branke, J. & Smith, J.) 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. *Proc. Natl Acad. Sci. USA* **106,** 5019–5024 (2009).
15. Akey, J. M. *et al*. Tracking footprints of artificial selection in the dog genome. *Proc. Natl Acad. Sci. USA* **107,** 1160–1165 (2010).
16. Dennett, D. *Darwin's Dangerous Idea* (Penguin, 1995).
17. Goldberg, D. *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley, 1989).
18. Fogel, D.B. *Evolutionary Computation* (IEEE, 1995).
19. Schwefel, H.-P. *Evolution and Optimum Seeking* (Wiley, 1995).
20. Bäck, T. *Evolutionary Algorithms in Theory and Practice* (Oxford Univ. Press, 1996).
21. Banzhaf, W., Nordin, P., Keller, R. E. & Francone, F. D. *Genetic Programming: an Introduction* (Morgan Kaufmann, 1998).
22. Storn, R. & Price, K. Differential evolution — a simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.* **11,** 341–359 (1997).
23. Price, K. V., Storn, R. N. & Lampinen, J. A. *Differential Evolution: a Practical Approach to Global Optimization* (Springer, 2005).
24. Kennedy, J. & Eberhart, R. C. Particle swarm optimization. In *Proc. IEEE International Conference on Neural Networks* 1942–1948 (IEEE, 1995).
25. Kennedy, J. & Eberhart, R.C. *Swarm Intelligence* (Morgan Kaufmann, 2001).
26. De Jong, K. A. Are genetic algorithms function optimizers? In *Proc. 2nd Conference on Parallel Problem Solving from Nature* (eds Manner, R. & Manderick, B.) 3–13 (North-Holland, 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. *Evol. Comput.* **19,** 1–23 (2011).
28. Arias-Montano, A., Coello, C. A. C. & Mezura-Montes, E. Multiobjective evolutionary algorithms in aeronautical and aerospace engineering. *IEEE Trans. Evol. Comput.* **16,** 662–694 (2012).
29. Besnard, J. *et al*. Automated design of ligands to polypharmacological profiles. *Nature* **492,** 215–220 (2012).
30. Posìk, P. Huyer, W. & Pal, L. A comparison of global search algorithms for continuous black box optimization. *Evol. Comput.* **20,** 509–541 (2012).
31. Hansen, N. & Ostermeier, A. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.* **9,** 159–195 (2001).
    This article introduced the CMA-ES algorithm, widely regarded as the state of the art in numerical optimization.
32. Bäck, T., Foussette, C. & Krause, P. *Contemporary Evolution Strategies* (Springer, 2013).
33. Yao, X. Evolving artificial neural networks. *Proc. IEEE* **87,** 1423–1447 (1999).
    This landmark paper, which was the winner of the 2001 Institute of Electrical and Electronics Engineers Donald G. Fink Prize Paper Award, 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. & Mattiussi, C. Neuroevolution: from architectures to learning. *Evol. Intel.* **1,** 47–62 (2008).
35. Barros, R. C., Basgalupp, M. P., de Carvalho, A. C. P. L. F. & Freitas, A. A. A survey of evolutionary algorithms for decision-tree induction. *IEEE Trans. Syst. Man Cybern. C* **42,** 291–312 (2012).
36. Widera, P., Garibaldi, J. M. & Krasnogor, N. GP challenge: evolving energy function for protein structure prediction. *Genet. Program. Evolvable Mach.* **11,** 61–88 (2010).
37. Filipič, B., Urbančič, T. & Križman, V. A combined machine learning and genetic algorithm approach to controller design. *Eng. Appl. Artif. Intell.* **12,** 401–409 (1999).
38. Watson, R. A., Ficici, S. G. & Pollack, J. B. Embodied evolution: distributing an evolutionary algorithm in a population of robots. *Robot. Auton. Syst.* **39,** 1–18 (2002).
39. Bredeche, N., Montanier, J. M., Liu, W. & Winfield, A. F. T. Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. *Math. Comput. Model. Dyn. Syst.* **18,** 101–129 (2012).
40. Nolfi, S. & Floreano, D. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines* (MIT Press, 2000).
41. Bongard, J. Evolutionary robotics. *Commun. ACM* **56,** 74–85 (2013).
42. Floreano, D. & Keller, L. Evolution of adaptive behavior in robots by means of Darwinian selection. *PLoS Biol.* **8,** e1000292 (2010).
43. Hinton, G. E. & Nowlan, S. J. How learning can guide evolution. *Complex Syst.* **1,** 495–502 (1987).
    This seminal paper showed that learning can guide evolution even though characteristics acquired by the phenotype are not communicated to the genotype.
44. Borenstein, E., Meilijson, I. & Ruppin, E. The effect of phenotypic plasticity on evolution in multipeaked fitness landscapes. *J. Evol. Biol.* **19,** 1555–1570 (2006).
45. Paenke, I., Jin, Y. & Branke, J. Balancing population and individual level of adaptation in changing environments. *Adapt. Behav.* **17,** 153–174 (2009).
46. Chen, X. S., Ong, Y. S., Lim, M. H. & Tan, K. C. A. Multi-facet survey on memetic computation. *IEEE Trans. Evol. Comput.* **15,** 591–607 (2011).
47. Krasnogor, N. & Smith, J. E. A tutorial for competent memetic algorithms: model, taxonomy and design issues. *IEEE Trans. Evol. Comput.* **9,** 474–488 (2005).
48. Smith, J. E., Clark, A. R., Staggemeier, A. T. & Serpell, M. C. A genetic approach to statistical disclosure control. *IEEE Trans. Evol. Comput.* **16,** 431–441 (2012).
49. Bentley, P. & Corne, D. *Creative Evolutionary Systems* (Morgan Kaufmann, 2002).
50. Romero, J. J. & Machado, P. (eds). *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music* (Springer, 2008).
51. Secretan, J. *et al*. Picbreeder: a case study in collaborative evolutionary exploration of design space. *Evol. Comput.* **19,** 373–403 (2011).
52. Bentley, P. *Evolutionary Design by Computers* (Morgan Kaufmann, 1999).
53. Hingston, P. F., Barone, L. C. & Michalewicz, Z. (eds). *Advances in Evolutionary Design* (Springer, 2008).
54. Koza, J. R. Human-competitive results produced by genetic programming. *Genet. Program. Evolvable Mach.* **11,** 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 artificial intelligence.
55. Eiben, A. E. & Rudolph, G. Theory of evolutionary algorithms: a bird's eye view. *Theor. Comput. Sci.* **229,** 3–9 (1999).
56. Wolpert, D. H. & Macready, W. G. No free lunch theorems for optimisation. *IEEE Trans. Evol. Comput.* **1,** 67–82 (1997).
    This paper reported game-changing results that supported the shift in focus in evolutionary computing and other fields away from the search for a 'super solver', and inspired insightful discussions that are still ongoing.
57. Rudolph, G. Convergence analysis of canonical genetic algorithms. *IEEE Trans. Neural Netw.* **5,** 96–101 (1994).
58. Lehre, P. R. & Yao, X. On the impact of mutation-selection balance on the runtime of evolutionary algorithms. *IEEE Trans. Evol. Comput.* **16,** 225–241 (2012).
59. Jansen, T. *Analyzing Evolutionary Algorithms: The Computer Science Perspective* (Springer, 2005).
60. Borenstein, Y. & Moraglio, A. (eds). *Theory and Principled Methods for Designing Metaheuristics* (Springer, 2014).
    This text provides good coverage of a range of recent approaches and results in the theory of evolutionary algorithms.
61. Eiben, A. E., Hinterding, R. & Michalewicz, Z. Parameter control in evolutionary algorithms. *IEEE Trans. Evol. Comput.* **3,** 124–141 (1999).
    This paper had a long-lasting effect by putting the issue of parameter calibration on the research agenda and establishing the corresponding conceptual framework.
62. Bartz-Beielstein, T. T. *Experimental Research in Evolutionary Computation: the New Experimentalism* (Springer, 2006).
63. Hutter, F., Hoos, H. H., Leyton-Brown, K. & Stützle, T. ParamILS: an automatic algorithm configuration framework. *J. Artif. Intell. Res.* **36,** 267–306 (2009).
64. Eiben, A. E. & Smit, S. K. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm Evol. Comput.* **1,** 19–31 (2011).
65. Bartz-Beielstein, T. & Preuss, M. in *Theory and Principled Methods for Designing Metaheuristics* (eds Borenstein, Y. & Moraglio, A.) 205–245 (Springer, 2014).
66. Lobo, F. J., Lima, C. F., Michalewicz, Z. (eds). *Parameter Setting in Evolutionary Algorithms* (Springer, 2007).

67. Serpell, M. & Smith, J. E. Self-adaption of mutation operator and probability for permutation representations in genetic algorithms. *Evol. Comput.* **18,** 491–514 (2010).
68. Fialho, A., Da Costa, L., Schoenauer, M. & Sebag, M. Analyzing bandit-based adaptive operator selection mechanisms. *Ann. Math. Artif. Intell.* **60,** 25–64 (2010).
69. Karafotias, G., Hoogendoorn, M. & Eiben, A. E. Parameter control in evolutionary algorithms: trends and challenges. *IEEE Trans. Evol. Comput.* **19,** 167–187 (2015).
This is a recent follow up to ref. 61.
70. Jin, Y. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Comput.* **9,** 3–12 (2005).
71. Jin, Y. Surrogate-assisted evolutionary computation: recent advances and future challenges. *Swarm Evol. Comput.* **1,** 61–70 (2011).
72. Loshchilov, I., Schoenauer, M. & Sebag, M. Self-adaptive surrogate-assisted covariance matrix adaptation evolution strategy. In *Proc. Conference on Genetic and Evolutionary Computation* (eds Soule, T. & Moore, J. H.) 321–328 (ACM, 2012).
73. Zaefferer, M. *et al*. Efficient global optimization for combinatorial problems. In *Proc. Conference on Genetic and Evolutionary Computation* (eds Igel, C. & Arnold, D. V.) 871–878 (ACM, 2014).
74. Deb, K. *Multi-objective Optimization Using Evolutionary Algorithms* (Wiley, 2001).
75. Zhang, Q. & Li, H. MOEA/D: a multi-objective evolutionary algorithm based on decomposition. *IEEE Trans. Evol. Comput.* **11,** 712–731 (2007).
76. Deb, K. & Jain, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE Trans. Evol. Comput.* **18,** 577–601 (2014).
77. Jain, H. & 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 Trans. Evol. Comput.* **18,** 602–622 (2014).
78. Branke, J., Greco, S., Slowinski, R. & Zielniewicz, P. Learning value functions in interactive evolutionary multiobjective optimization. *IEEE Trans. Evol. Comput.* **19,** 88–102 (2015).
79. Stanley, K. O. Compositional pattern producing networks: a novel abstraction of development. *Genet. Program. Evolvable Mach.* **8,** 131–162 (2007).
80. O'Reilly, U.-M. & Hemberg, H. Integrating generative growth and evolutionary computation for form exploration. *Genet. Program. Evolvable Mach.* **8,** 163–186 (2007).
81. Clune, J., Stanley, K. O., Pennock, R. & Ofria, C. On the performance of indirect encoding across the continuum of regularity. *IEEE Trans. Evol. Comput.* **15,** 346–367 (2011).
82. Jin, Y. & Meng, Y. Morphogenetic robotics: an emerging new field in developmental robotics. *IEEE Trans. Syst. Man Cybern. C* **41,** 145–160 (2011).
83. Doursat, R., Sayama, H. & Michel, O. (eds). *Morphogenetic Engineering: Toward Programmable Complex Systems* (Springer, 2013).
84. Doncieux, S., Bredeche, N. & Mouret, J.-B. (eds). *New Horizons in Evolutionary Robotics* (Springer, 2011).
85. Vargas, P. A., Di Paolo, E. A., Harvey, I. & Husbands, P. (eds). *The Horizons of Evolutionary Robotics* (MIT Press, 2014).
86. Harman, M. & McMinn, P. A theoretical and empirical study of search-based testing: local, global, and hybrid search. *IEEE Trans. Softw. Eng.* **36,** 226–247 (2010).
87. Preen, R. & Bull, L. Towards the coevolution of novel vertical-axis wind turbines. *IEEE Trans. Evol. Comput.* **19,** 284–294 (2015).
88. Banzhaf, W. *et al*. From artificial evolution to computational evolution: a research agenda. *Nature Rev. Genet.* **7,** 729–735 (2006).
89. Maynard Smith, J. Byte-sized evolution. *Nature* **355,** 772–773 (1992).
90. Waibel, M., Floreano, D. & Keller, L. A quantitative test of Hamilton's rule for the evolution of altruism. *PLoS Biol.* **9,** e1000615 (2011).
91. Long, J. *Darwin's Devices: What Evolving Robots Can Teach Us About the History of Life and the Future of Technology* (Basic Books, 2012).
92. Virgo, N., Fernando, C., Bigge, B. & Husbands, P. Evolvable physical self-replicators. *Artif. Life* **18,** 129–142 (2012).
93. Bongard, J. & Lipson, H. Evolved machines shed light on robustness and resilience. *Proc. IEEE* **102,** 899–914 (2014).
94. Bongard, J. Morphological change in machines accelerates the evolution of robust behavior. *Proc. Natl Acad. Sci. USA* **108,** 1234–1239 (2011).
This article demonstrated a hitherto unknown relationship between development, evolution, morphology and the neural control of behaviour, as phrased by the title.
95. Eiben, A. E. Grand challenges for evolutionary robotics. *Front. Robot. AI* **1,** http://dx.doi.org/10.3389/frobt.2014.00004 (2014).
96. Fernando, C., Kampis, G. & Szathmáry, E. Evolvability of natural and artificial systems. *Procedia Comput. Sci.* **7,** 73–76 (2011).

Acknowledgements The authors would like to thank the editors and reviewers, as well as A. Adamatzky, L. Bull, B. Filipic and M. Schoenauer for providing helpful insights on earlier versions of this manuscript.

Author Information Reprints and permissions information is available at www.nature.com/reprints. The author declares no competing financial interests. Readers are welcome to comment on the online version of this paper at go.nature.com/4r5hj1. Correspondence should be addressed to A.E.E. (a.e.eiben@vu.nl).